

Configuration Manual

MSc Practicum Part 2
MSc Cybersecurity

Anjana Unnikrishnan
Student ID: 23124865

School of Computing
National College of Ireland

Supervisor: Jawad Salahuddin

National College of Ireland
MSc Project Submission Sheet



School of Computing

Student Name: Anjana Unnikrishnan

Student ID: 23124865

Programme: Master of Science in Cybersecurity

Year: 2024

Module: MSc Practicum part 2

Lecturer: Jawad Salahuddin

Submission

Due Date: 12/12/2024

Project Title: Blockchain and Android: A data integrity preservation method

Word Count: Page Count:

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Anjana Unnikrishnan

Date: 28/01/2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	<input type="checkbox"/>
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	<input type="checkbox"/>
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	<input type="checkbox"/>

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Anjana Unnikrishnan
Student ID: 23124865

1 Introduction

The configuration manual includes the infrastructure setup, and tools required to build the project. This project builds a hybrid storage – PostgreSQL and Hyperledger fabric system with calculation of SHA-256 hash of user data record and comparing them to evaluate the integrity of the system. This document lists the hardware and software requirements, project setup and execution

2 Hardware Requirements

The hardware requirements for the development of this project are:

- Processor: 12th Gen Intel(R) Core(TM) i5-12500H 2.50GHz, 12 Cores
- RAM: 16.0GB
- Operating System: Microsoft Windows 11 (64-bit)
- GPU: Intel(R) Iris(R) Xe Graphics
- Storage: 512 GB SSD

3 Software Requirements

Tools used

- Visual Studio Code – v1.95.3 (Integrated Development Environment (IDE))
- Postman – Tool used for simulating and testing API requests.
- Windows Docker Container – v4.34.3, To run Hyperledger Fabric network separated from the infrastructure.
- Windows Subsystem for Linux – WSL2 For Hyperledger Fabric for smooth running as it is mostly “bash” based.

4 Project Setup

The steps for setting up the project include:

1. Install Docker Desktop container and WSL2 in the host machine.
2. Turn on the WSL integration in docker desktop settings.

3. Install Visual Studio Code in the system.
4. Install Python3 and Node.js libraries
5. Download Hyperledger Fabric samples, docker images and binaries as per official documentation. After this step, the “fabric-samples” folder with all the platformspecific Hyperledger Fabric CLI tools and config files.

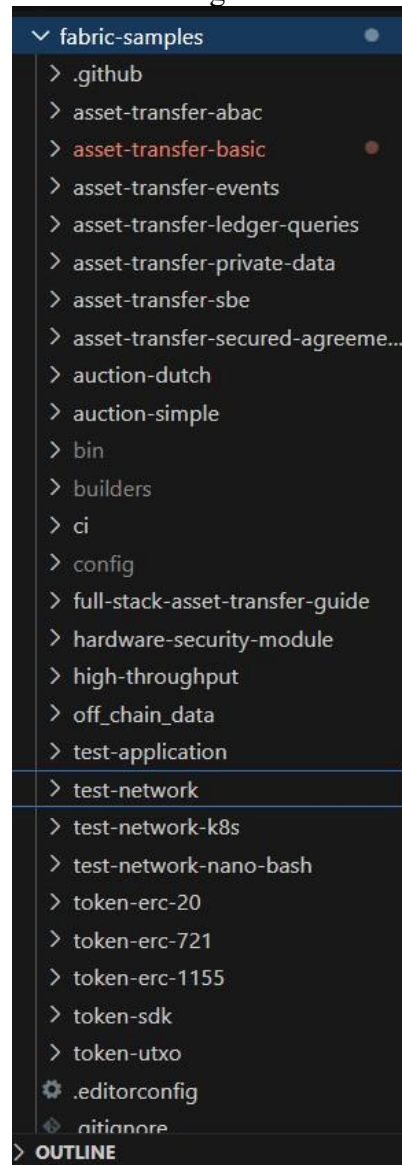
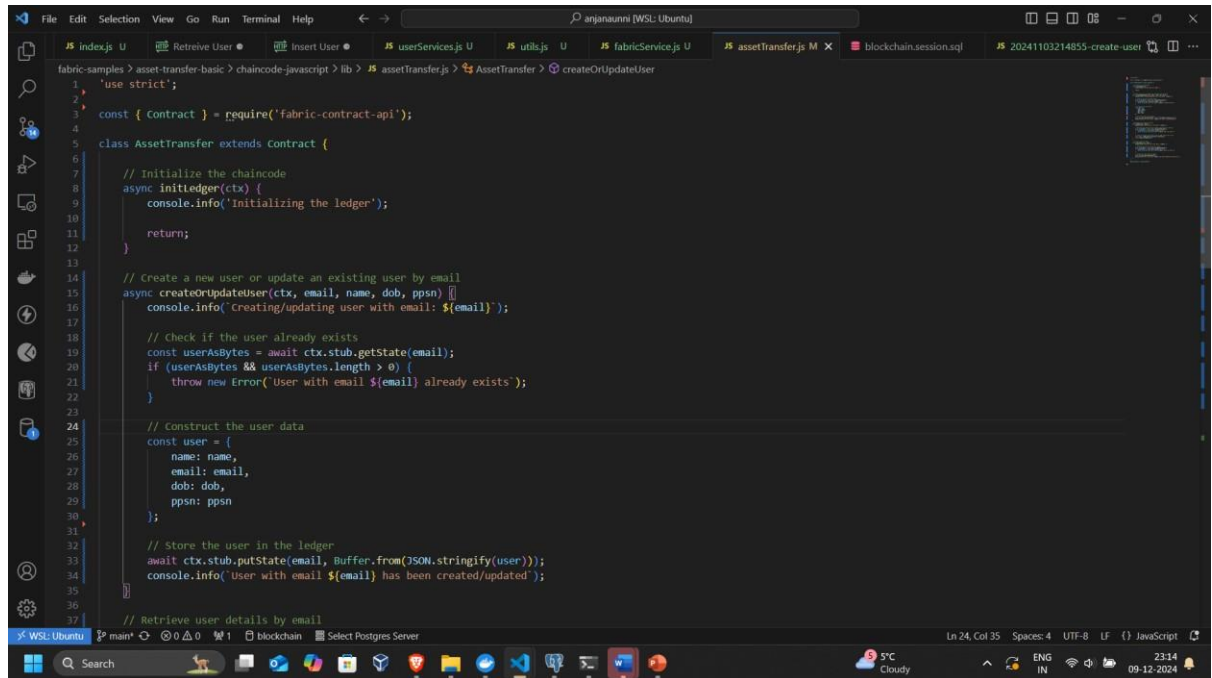


Figure 1: Project Structure of Fabric Samples

5 Execution Steps

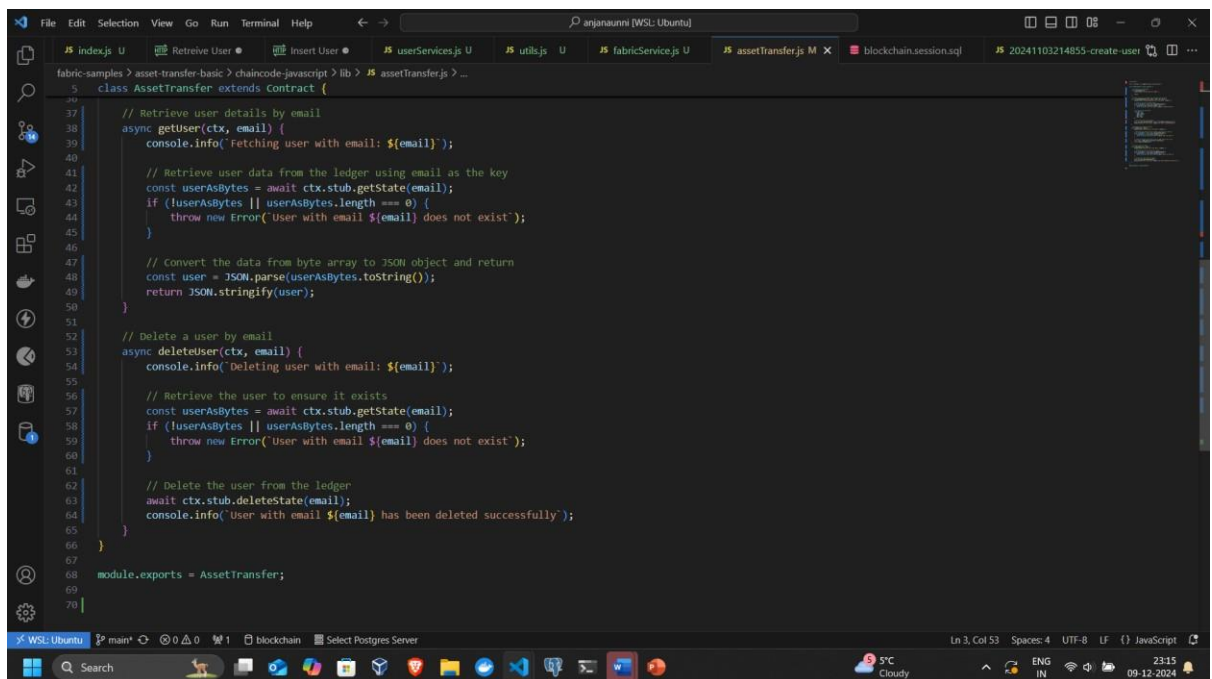
Steps to start the fabric network and package the chaincode(smart contract):

1. Redirect to “fabric-samples/test-network” and execute “./network.sh up createChannel”
2. Now, create a chain code, move to “fabric-samples/asset-transfer-basic/chaincodejavascript/lib” and create the chaincode.



```
1 'use strict';
2
3 const { Contract } = require('fabric-contract-api');
4
5 class AssetTransfer extends Contract {
6
7   // Initialize the chaincode
8   async initLedger(ctx) {
9     console.info('Initializing the ledger');
10
11     return;
12   }
13
14   // Create a new user or update an existing user by email
15   async createOrUpdateUser(ctx, email, name, dob, ppsn) {
16     console.info('Creating/updating user with email: ${email}');
17
18     // Check if the user already exists
19     const userAsBytes = await ctx.stub.getState(email);
20     if (userAsBytes && userAsBytes.length > 0) {
21       throw new Error('User with email ${email} already exists');
22     }
23
24     // Construct the user data
25     const user = {
26       name: name,
27       email: email,
28       dob: dob,
29       ppsn: ppsn
30     };
31
32     // Store the user in the ledger
33     await ctx.stub.putState(email, Buffer.from(JSON.stringify(user)));
34     console.info('User with email ${email} has been created/updated');
35
36     // Retrieve user details by email
```

Figure 2: Chaincode part 1



```
37
38   // Retrieve user details by email
39   async getUser(ctx, email) {
40     console.info('Fetching user with email: ${email}');
41
42     // Retrieve user data from the ledger using email as the key
43     const userAsBytes = await ctx.stub.getState(email);
44     if (!userAsBytes || userAsBytes.length === 0) {
45       throw new Error('User with email ${email} does not exist');
46     }
47
48     // Convert the data from byte array to JSON object and return
49     const user = JSON.parse(userAsBytes.toString());
50     return JSON.stringify(user);
51   }
52
53   // Delete a user by email
54   async deleteUser(ctx, email) {
55     console.info('Deleting user with email: ${email}');
56
57     // Retrieve the user to ensure it exists
58     const userAsBytes = await ctx.stub.getState(email);
59     if (!userAsBytes || userAsBytes.length === 0) {
60       throw new Error('User with email ${email} does not exist');
61     }
62
63     // Delete the user from the ledger
64     await ctx.stub.deleteState(email);
65     console.info('User with email ${email} has been deleted successfully');
66   }
67
68   module.exports = AssetTransfer;
69
70 }
```

Figure 3: Chaincode part 2

3. Now, move to “fabric-samples/asset-transfer-basic/chaincode-javascript” and do “npm install”.
4. Move to test-network folder and package the chaincode using: “peer lifecycle chaincode package basic.tar.gz --path ../asset-transfer-basic/chaincode-javascript/ -lang node --label basic_1.0”.

5. Now, install the chaincode on the peers of each organization: “peer lifecycle chaincode install basic.tar.gz”.
6. Approve the chaincode by both organizations: “peer lifecycle chaincode approveformyorg -o localhost:7050 --ordererTLSTLSHostnameOverride orderer.example.com --channelID mychannel --name basic --version 1.0 --package-id \$CC_PACKAGE_ID --sequence 1 --tls --cafile “\${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem””.
7. Then, the chaincode is committed to the channel of communication: “peer lifecycle chaincode checkcommitreadiness --channelID mychannel --name basic --version 1.0 - -sequence 1 --tls --cafile “\${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem” --output json”.

Now, the Node.js application is built with below features:

1. API request with user details in the body to store data in PostgreSQL and Hyperledger Fabric network.
 - a. For PostgreSQL:

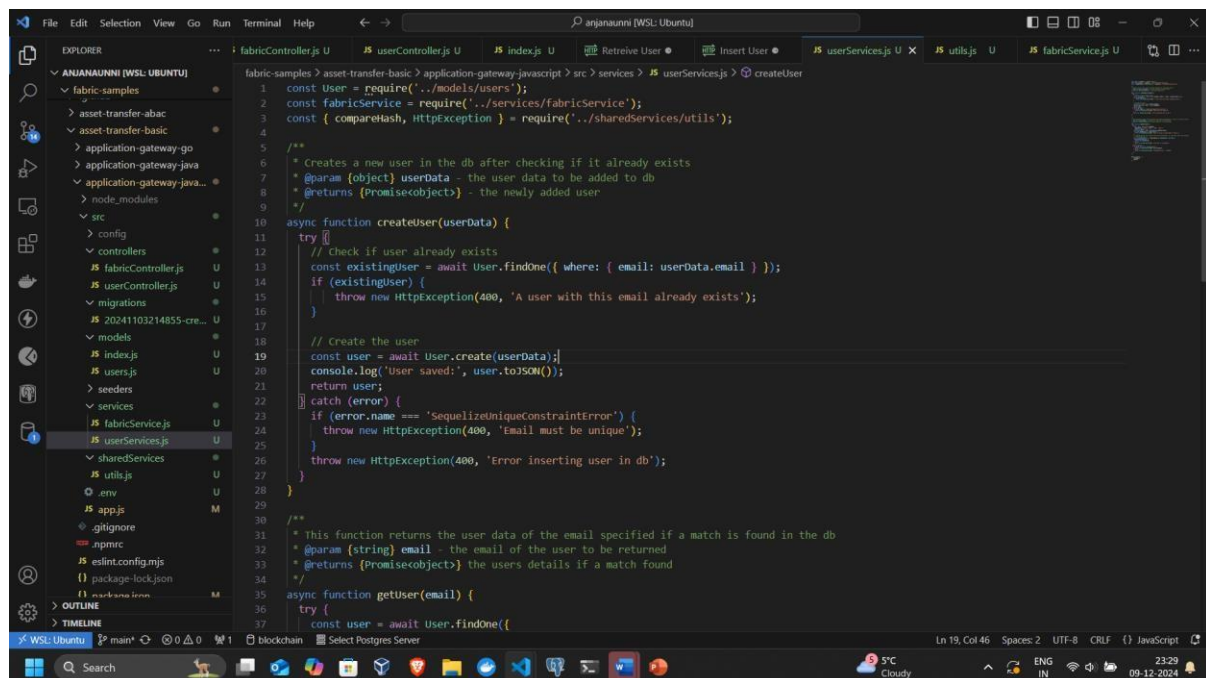


Figure 4: Store user to PostgreSQL

- b. For Hyperledger Fabric network

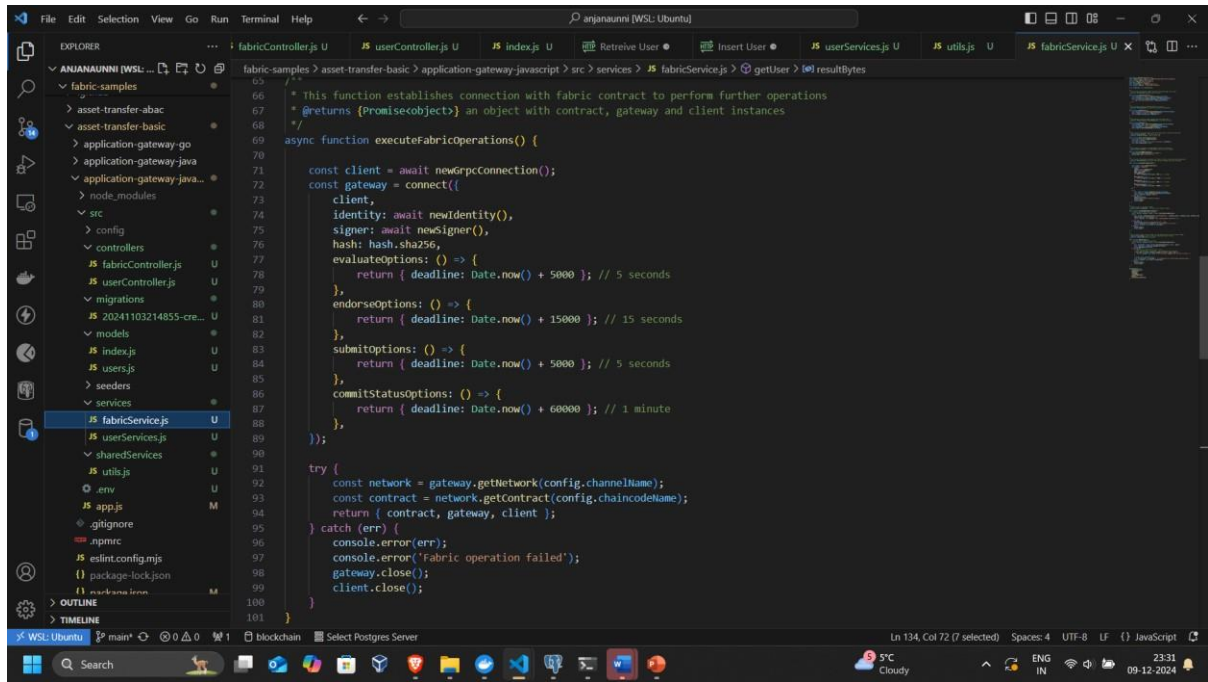


Figure 5: Node.js server establishing connection with fabric network

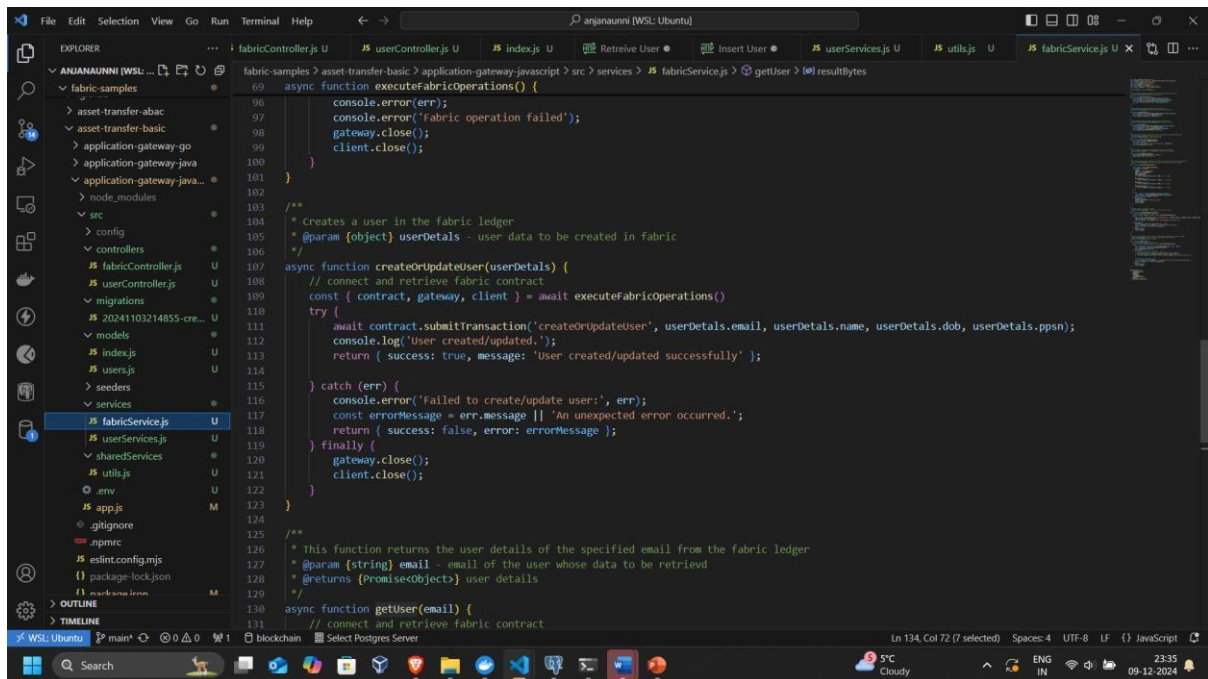


Figure 6: Function to store user data to blockchain

2. API request to retrieve the details of the user with the specified email from both PostgreSQL and blockchain.
 - a. PostgreSQL


```

30 /**
31  * This function returns the user data of the email specified if a match is found in the db
32  * @param {string} email - the email of the user to be returned
33  * @returns {Promise<object>} the users details if a match found
34  */
35 async function getUser(email) {
36   try {
37     const user = await User.findOne({
38       attributes: ['name', 'email', 'dob', 'ppsn'],
39       where: { email } });
40     const fabricUser = await FabricService.getUser(email)
41     if (user.dataValues || !fabricUser) {
42       throw new HttpException(404, 'User not found in PostgreSQL or Fabric');
43     }
44     // compare hash of user details from db and blockchain, if true then return user details
45     // else throw error
46     const hashCompareResult = compareHash(user.dataValues, fabricUser)
47     if (hashCompareResult) {
48       return user.dataValues;
49     } else {
50       throw new HttpException(404, 'User data is corrupted');
51     }
52   } catch (error) {
53     if (error instanceof HttpException) {
54       throw new HttpException(404, error.message);
55     } else {
56       throw new HttpException(404, 'Unexpected error: ' + error);
57     }
58   }
59 }
60
61 module.exports = {
62   createUser,
63   getUser
64 };
65

```

Figure 7: Function to retrieve user data from PostgreSQL

b. Hyperledger Fabric network

```

124 /**
125  * This function returns the user details of the specified email from the fabric ledger
126  * @param {string} email - email of the user whose data to be retrieved
127  * @returns {Promise<object>} user details
128  */
129
130 async function getUser(email) {
131   // connect and retrieve fabric contract
132   const { contract, gateway, client } = await executeFabricOperations()
133   try {
134     const resultBytes = await contract.evaluateTransaction('getUser', email);
135     const resultJson = utf8Decoder.decode(resultBytes);
136     return JSON.parse(resultJson);
137   } catch (err) {
138     // console.error('Failed to get user:', err);
139     // const errorMessage = err.message || 'An unexpected error occurred.';
140     if (err.code === 2 && err.details.includes('chaincode response 500')) {
141       return { success: false, error: 'User with email $(email) does not exist in Fabric' };
142     }
143     // Handle other unexpected errors
144     const errorMessage = err.message || 'An unexpected error occurred while querying Fabric.';
145     return { success: false, error: errorMessage };
146   } finally {
147     gateway.close();
148     client.close();
149   }
150 }
151
152 module.exports = {
153   newGrpcConnection,
154   newIdentity,
155   newSigner,
156   initLedger,
157   createOrUpdateUser,
158   getUser,
159 }

```

Figure 8: Function to retrieve user data from fabric network

3. The SHA-256 hash of the user records retrieved from PostgreSQL and Hyperledger Fabric are calculated and compared.

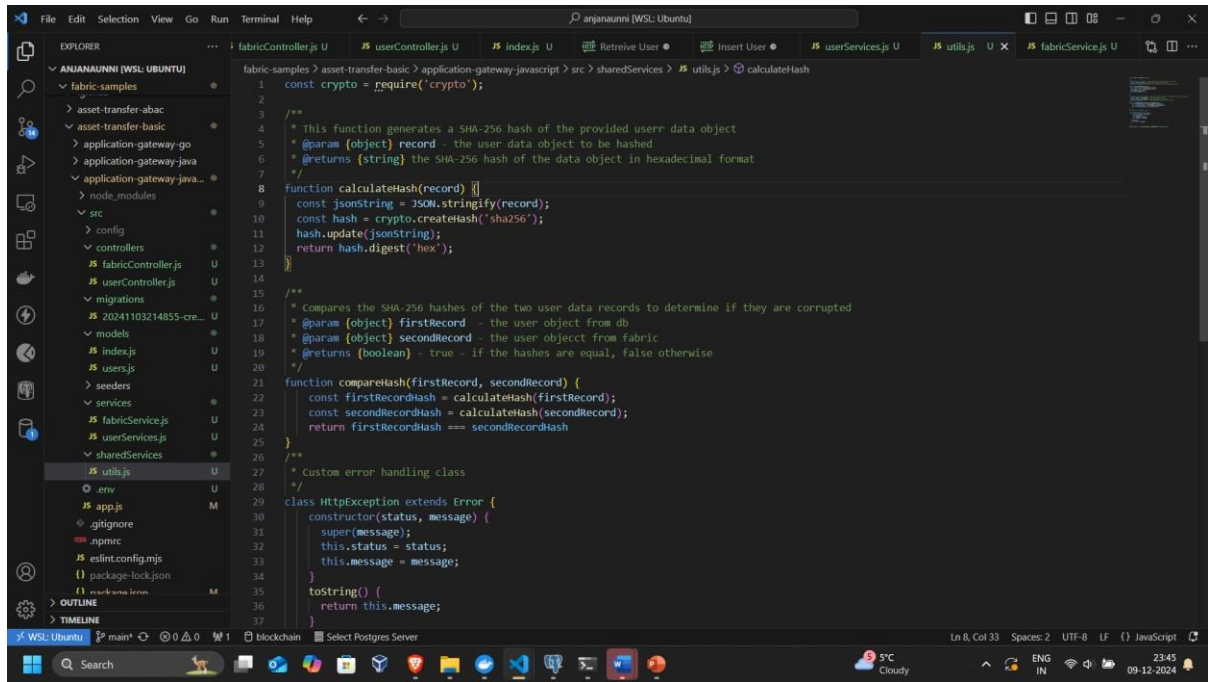


Figure 9: SHA-256 hash calculation and comparison

Outputs:

1. The insertion of user into storage

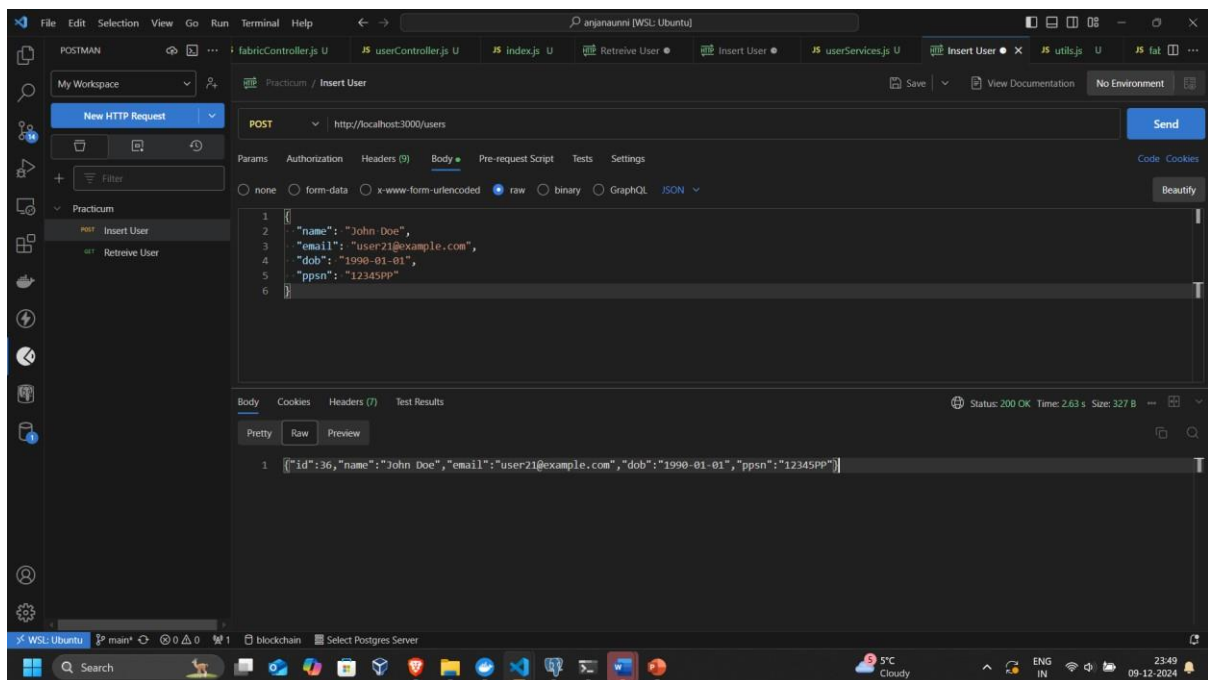


Figure 10: Postman testing to insert user to storage systems

2. The retrieval of user data after integrity checking

a. Case 1 – no corruption

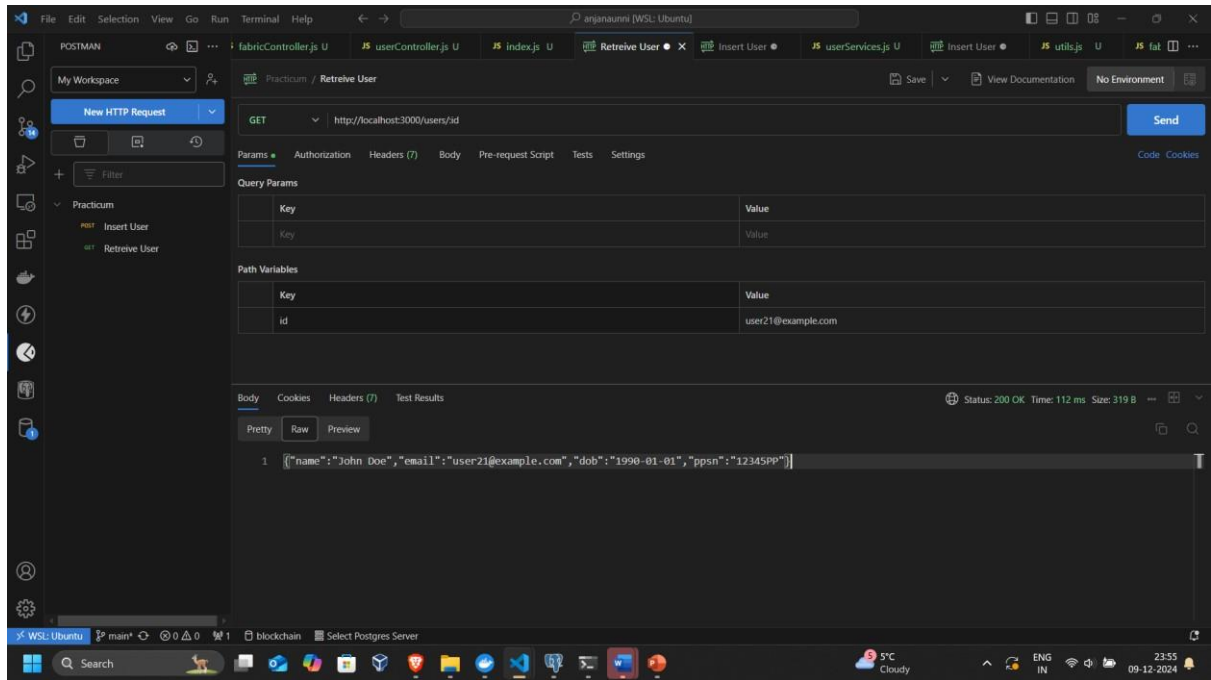


Figure 11: Data with no corruption is retrieved

b. Case 2 – corrupted user

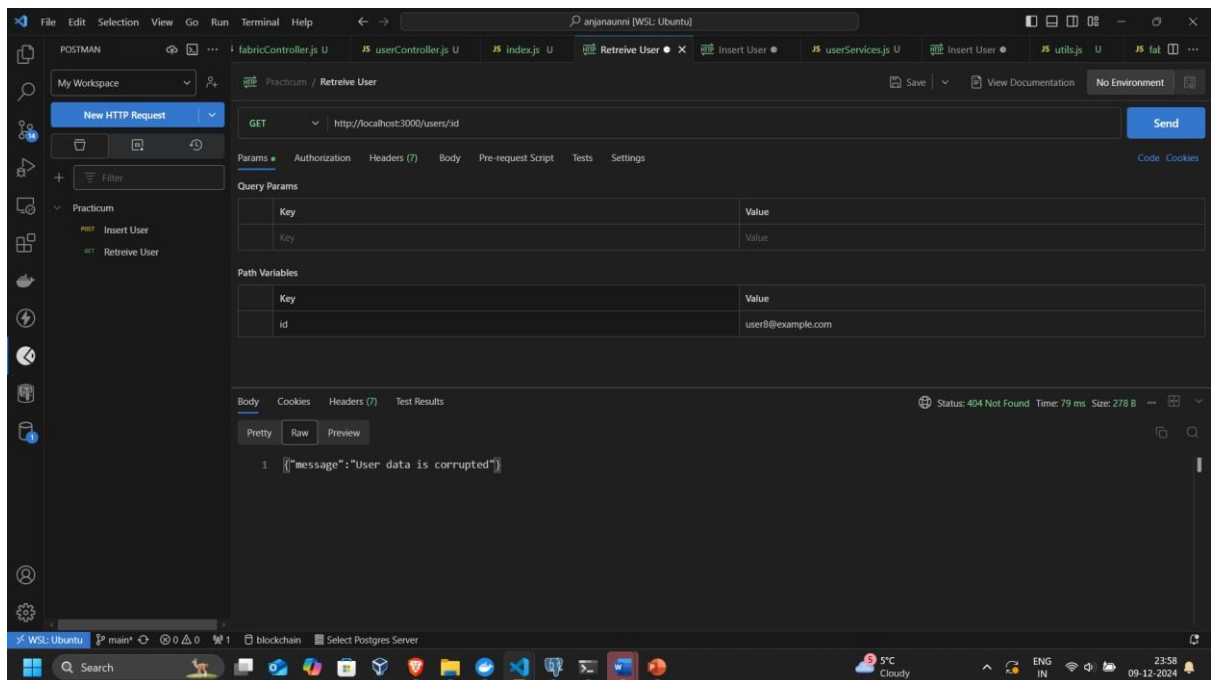


Figure 12: User data is corrupted

