

IOT DEVICE SECURITY THROUGH MQTT PROTOCOL

*Minor project report submitted
in partial fulfillment of the requirement for award of the degree of*

**Bachelor of Technology
in
Computer Science & Engineering**

By

S. ROHITH	(20UECS0815)	(12235)
E. SAGIN SANDOZ FERNANDO	(20UECS0825)	(16803)
M. G. SANDEEP PRASAN KUMAR	(20UECS0836)	(16810)

*Under the guidance of
Mr.VASHOK KUMAR,M.TECH.,
ASSISTANT PROFESSOR*



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SCHOOL OF COMPUTING**

**VEL TECH RANGARAJAN DR. SAGUNTHALA R&D INSTITUTE OF
SCIENCE & TECHNOLOGY**

(Deemed to be University Estd u/s 3 of UGC Act, 1956)

**Accredited by NAAC with A Grade
CHENNAI 600 062, TAMILNADU, INDIA**

November , 2022

CERTIFICATE

It is certified that the work contained in the project report titled “IOT DEVICE SECURITY THROUGH MQTT PROTOCOL” by “S. ROHITH (20UECS0815), E. SAGIN SANDOZ FERNANDO (20UECS 0825), M. G. SANDEEP PRASAN KUMAR (20UECS0836)” has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

Signature of Supervisor

Mr. V. Ashok Kumar

Assistant Professor

Computer Science & Engineering

School of Computing

Vel Tech Rangarajan Dr.Sagunthala R&D

Institute of Science & Technology

November,2022

Signature of Head of the Department

Dr. V. Srinivasa Rao

Professor & Dean

Computer Science & Engineering

School of Computing

Vel Tech Rangarajan Dr.Sagunthala R&D

Institute of Science & Technology

November,2022

DECLARATION

We declare that this written submission represents my ideas in our own words and where others' ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea in our submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

(Signature)

(S. ROHITH)

Date: / /

(Signature)

(E. SAGIN SANDOZ FERNANDO)

Date: / /

(Signature)

(M. G. SANDEEP PRASAN KUMAR)

Date: / /

APPROVAL SHEET

This project report entitled IOT DEVICE SECURITY THROUGH MQTT PROTOCOL by S. ROHITH (20UECS0815), E. SAGIN SANDOZ FERNANDO (20UECS0825), M. G. SANDEEP PRASAN KUMAR (20UECS0836) is approved for the degree of B.Tech in Computer Science & Engineering.

Examiners

Supervisor

Mr. V. ASHOK KUMAR, M.TECH.,

Date: / /

Place:

ACKNOWLEDGEMENT

We express our deepest gratitude to our respected **Founder Chancellor and President Col. Prof. Dr. R. RANGARAJAN B.E. (EEE), B.E. (MECH), M.S (AUTO),D.Sc., Foundress President Dr. R. SAGUNTHALA RANGARAJAN M.B.B.S.** Chairperson Managing Trustee and Vice President.

We are very much grateful to our beloved **Vice Chancellor Prof. S. SALIVAHANAN**, for providing us with an environment to complete our project successfully.

We record indebtedness to our **Professor & Dean, Department of Computer Science & Engineering Dr.V.SRINIVASA RAO, M.Tech., Ph.D.,** for immense care and encouragement towards us throughout the course of this project.

We also take this opportunity to express a deep sense of gratitude to our Internal Supervisor **Mr. V. ASHOK KUMAR, M.TECH.,** for his cordial support, valuable information and guidance, he helped us in completing this project through various stages.

A special thanks to our **Project Coordinators Mr. V. ASHOK KUMAR, M.Tech., Mrs. C. SHYAMALA KUMARI, M.E.,** for their valuable guidance and support throughout the course of the project.

We thank our department faculty, supporting staff and friends for their help and guidance to complete this project.

S. ROHITH	(20UECS0815)
E. SAGIN SANDOZ FERNANDO	(20UECS0825)
M. G. SANDEEP PRASAN KUMAR	(20UECS0836)

ABSTRACT

This work discusses the development of an “IoT Device Security through MQTT Protocol” on ReactJS Platform. The work aims at the development of the protocol that helps to secure the communication between the Application and the household appliances or any IoT devices. The Internet of Things (IoT) can be described as a network of physical objects or things embedded with software, electronics, sensors and network connectivity that helps these objects collect and exchange data. The smart devices and sensors in home automation help collect (or sense) the physical experience and convert it into information data. The major element of home automation based on IoT is the Raspberry Pi. It collects the data from sensors which is connected through MQTT Protocol which interprets them to control and manage household devices like fan, light, heater, door, and opening and closing of curtains. For example, if there is no presence of a person in a certain room, the lights are automatically turned off for that room which the protocol ensures the security of the IoT devices interconnected through wireless medium which cannot be accessed by malicious users.

KEYWORDS : MQTT, ReactJS, IoT, Raspberry Pi

LIST OF FIGURES

4.1	Architecture Diagram	10
4.2	Data Flow Diagram	11
4.3	Use Case Diagram	12
4.4	Class Diagram	13
4.5	Sequence Diagram	14
4.6	Collaboration Diagram	15
4.7	Creating user interface – App.js	17
4.8	Connecting the UI and DB by php	18
4.9	Data transferred from DB to UI through MQTT Broker	19
5.1	Input	20
5.2	Output	21
5.3	Functional testing input	22
5.4	Functional testing output	23
8.1	Poster Presentation	29
9.1	Poster Presentation	33

LIST OF ACRONYMS AND ABBREVIATIONS

ABE	Attribute-Based Encryption
API	Application Programming Interface
BLE	Bluetooth Low Energy
C	Celsius
CM	Cache Miss
CoAP	Constrained Application Protocol
CSI	Camera Serial Interface
DC	Direct Current
DSI	Display Serial Interface
GL	Graphics Library
GPIO	General Purpose Input/Output
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
ICC	International Commercial Agency
IoT	Internet of Things
JS	JavaScript
JSON	JavaScript Object Notation
LAN	Local Area Network
LPDDR	Low-Power Double Data Rate
ML	Machine learning
MQ	Message Query
MQTT	Message Queuing Telemetry Transport
NPM	Node Package Manager

OS	Operating System
PAN	Personal Area Network
POE	Power over Ethernet
SCADA	Supervisory Control And Data Acquisition
SD	Secure Digital
SN	Say Nothing
SQL	Structured Query Language
SSL	Secure Sockets Layer
SDRAM	Synchronous Dynamic Random-Access Memory
TEA	Tiny Encryption Algorithm
TLS	Transport Layer Security
UI	User interface
USB-C	Universal Serial Bus Type-C
UX	User experience
WAN	Wireless Area Network
XAMPP	X-operating system, Apache, Mysql, Php, Perl

Chapter 1

INTRODUCTION

1.1 Introduction

IoT technology has been broadly used in diverse fields, including environmental perception, wearable medical, smart home, and smart city. IoT interconnected sensors and computers need to send data to cloud servers to perform computational activities in a standard IoT service. Message Queuing Telemetry Transport (MQTT) is a de facto standard for various Internet of Things (IoT) and industrial IoT applications. It is capable of transmitting data over low bandwidth or unreliable networks with very low consumption of power. MQTT is a bi-directional communication protocol. This helps in both sharing data, managing, and controlling devices. It is not required that the address in MQTT depends on the physical location as the IP address. Offered traffic in the network is reduced to transfer information. The data produced by a publisher are delivered to several subscribers via an MQTT broker.

A client is responsible for opening network connection to the server, creating messages to be published, publishing application messages to the server. Subscribing to request application messages that it is interested in receiving, unsubscribing to remove a request for application messages and closing network connection to the server. The data/message which transferred through MQTT Broker which were stored in the Database such as SQL/SQLite database which is a compact and in-process library, which do not have a separate server but reads the data and writes the data to disk files. It is also known as embedded SQL database engine. SQLite data set engine is installed on the Raspberry Pi and a python application is run on the Raspberry Pi which subscribes the data from MQTT Broker and gets the data from broker in JSON format. This data is formatted using HTML application and is stored in the SQLite database columns.

1.2 Aim of the project

The overall goal is to protect the entire system which represents an IoT installation. The more granular security requirements, often called security attributes, are confidentiality, availability, integrity, and privacy.

1.3 Project Domain

Internet of Things (IoT) security is the safeguards and protections for cloud-connected devices such as home automation, SCADA machines, security cameras, and any other technology that connects directly to the cloud. IoT technology is distinguished from mobile devices (e.g., smartphones and tablets) technology based on its automatic cloud connectivity in gadgets. IoT security involves securing traditionally poorly designed devices for data protection and cyber security. Recent data breaches have shown that IoT security should be a priority for most manufacturers and developers.

1.4 Scope of the Project

IoT security hacks can happen in anywhere and in any industry, from a smart home to a manufacturing plant to a connected car. The severity of impact depends greatly on the individual system, the data collected and/or the information it contains.

Chapter 2

LITERATURE REVIEW

F. De Rango, G. Potrino, M. Tropea, and P. Fazio, “Energy-aware dynamic Internet of Things security system based on Elliptic Curve Cryptography and Message Queue Telemetry Transport protocol for mitigating Replay attacks,” *Pervasive and Mobile Computing*, vol. 61, Article 101105, Jan. 2020. Protocol of MQTT - Message Queuing Telemetry Transport process is among the majority of lengthy procedures in IoT protocols. Nevertheless, this particular process doesn’t apply a solid protection pattern by default, and that doesn’t let a protected authentication mechanism between individuals within the reception. Little IoT products deliver additional and much smarter details within aspects of IoT and so on.[1]

B. Girgenti, P. Perazzo, C. Vallati, F. Righetti, G. Dini, and G. Anastasi, “On the Feasibility of Attribute-Based Encryption on Constrained IoT Devices for Smart Systems,” in *Proc. 2019 IEEE International Conference on Smart Computing*, pp. 225-232, Washington D.C., U.S.A., Jun. 12-15, 2020. IoT devices generate an uninterrupted flow of information that can be transmitted through an untrusted network and stored on an untrusted infrastructure. Attribute-Based Encryption (ABE) is a new type of public-key encryption that ensures a fine-grained access control mechanism on encrypted data based on flexible access policies.[2]

D. Dinculeană and X. Cheng, “Vulnerabilities and Limitations of MQTT Protocol Used between IoT Devices,” *Applied Science*, vol. 9, no.5, Article 848, Feb. 2019. With the proliferation of smart devices capable of communicating over a network using different protocols, each year more and more successful attacks are recorded against these, underlining the necessity of developing and implementing mechanisms to protect against such attacks. The method Value-to-Keyed-Hash Message Authentication Code (Value-to-HMAC) mapping, uses signatures to send messages, instead of encryption, by implementing a Keyed-Hash Message Authentication Code generation algorithm.[3]

O. Sadio, I. Ngom, and C. Lishou, “Lightweight Security Scheme for MQTT/MQTT -SN Protocol,” in Proc. the 6th Int’l conf. on Internet of Things: Systems, Management and Security, Granada, Spain, Oct. 22-25, 2020.[4]

M. Calabretta, R. Pecori, M. Vecchio, and L. Veltri, “MQTT-Auth: a Token-based Solution to Endow MQTT with Authentication and Authorization Capabilities,” Journal of Communications Software and Systems, vol. 14, no. 4, pp. 320-331, Dec. 2018.[5]

Chapter 3

PROJECT DESCRIPTION

3.1 Existing System

The token-based authentication of MQTT protocol in constrained IoT devices. They proposed a JSON Web token based authentication for MQTT Protocol. Recently a new paper was proposed by Katsikeas et al. In their work they highlighted the suitable properties of MQTT as a lightweight protocol and its suitability for industrial purposes. They evaluated different security options for MQTT nodes, namely payload encryption with AES, payload encryption with AES-CBC, payload authenticated encryption with AES-OCB and link layer encryption with AES-CCM. They concluded in their evaluation that all the options are attractive depending on the purpose. A very recent paper by Hernandez et al. investigated a new framework for MQTT which they call a novel fuzzing approach. They claim that their framework improves security of applications which implement MQTT. In other words, they designed a tool which perform security tests on MQTT

Disadvantages of existing system

- Slower transmit cycles compared to Constrained Application Protocol (CoAP).
- Fast cycles are critical for systems with more than 250 devices.
- Resource discovery. MQTT operates on a flexible topic subscription system. CoAP uses a stable resource discovery system.
- Lack of security encryption. While MQTT uses Transport Layer Security/Secure Sockets Layer (TLS/SSL), it is primarily unencrypted.

3.2 Proposed System

MQTT communication involves three entities publisher, subscriber, and a broker. In the proposed model, we consider that the user wants to access the data from the

sensing device. For that, both user device and sensing device do the registration with the broker device and later on user device authenticate with the broker device for to get the sensor data. The user provides user credentials, and the user device computes the necessary credentials. The user device encrypts the computed credentials and creates a JSON object. The user device publishes this JSON object to the broker. The broker device retrieves the user credentials and validate the identity of the user in the database. The broker allocates the valid topics to the user to receive the data from the sensing devices. The broker also computes the session key credentials that may be needed by the user for secure communication with the sensing device. The broker creates a JSON object with the confirm message, session key, and topic of communication and encrypts it with the public key of the user device. The broker publishes this JSON object to the user device. The broker also shares user identity and session keys with the sensing devices in a secured manner. The user retrieves the valid topic to subscribe to each sensor and the session key. The user subscribes to the allocated topics and stores the session key. It starts receiving the data from the sensor in the session key encrypted manner. The user encrypts those data and stores it into either a local database or into a broker database through the public channel.

Advantages of Proposed system

- Efficient data transmission and quick to implement, due to its being a lightweight protocol.
- Low network usage, due to minimized data packets.
- Efficient distribution of data.
- Successful implementation of remote sensing and control

3.3 Feasibility Study

3.3.1 Technical Feasibility

The study was done to compare the power usage of MQTT (Message Queueing Telemetry Transport) protocol with other several lightweight Internet of Things protocols, which are CoAP (Constrained Application Protocol) and HTTP (Hypertext Transfer Protocol). The comparison was done by measuring the current consumption

in each protocols each time the ESP8266 board sends a message to the corresponding server of the protocol in an interval. To conclude, we studied the comparison of power usage in the ESP8266 board with the MQTT protocol and compared it with other protocols.

3.4 System Specification

3.4.1 Hardware Specification

Raspberry Pi 4

- Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
1GB, 2GB, 4GB or 8GB LPDDR4-3200 SDRAM (depending on model)
- 2.4 GHz and 5.0 GHz IEEE 802.11ac wireless, Bluetooth 5.0, BLE Gigabit Ethernet
- 2 USB 3.0 ports; 2 USB 2.0 ports.
- Raspberry Pi standard 40 pin GPIO header (fully backwards compatible with previous boards)
- 2 × micro-HDMI ports (up to 4kp60 supported)
- 2-lane MIPI DSI display port
- 2-lane MIPI CSI camera port
- 4-pole stereo audio and composite video port
- H.265 (4kp60 decode), H264 (1080p60 decode, 1080p30 encode)
- OpenGL ES 3.1, Vulkan 1.0
- Micro-SD card slot for loading operating system and data storage
- 5V DC via USB-C connector (minimum 3A*)
- 5V DC via GPIO header (minimum 3A*)
- Power over Ethernet (PoE) enabled
- Operating temperature: 0 – 50 degrees C ambient

3.4.2 Software Specification

Visual code

- OS X El Capitan (10.11+)
- Windows 8.0, 8.1 and 10, 11 (32-bit and 64-bit)
- Linux (Debian): Ubuntu Desktop 16.04, Debian 9
- Linux (Red Hat): Red Hat Enterprise Linux 7, CentOS 7, Fedora 34

Mosquito

- mosquito-2.0.15-install-windows-x64.exe (64-bit build, Windows Vista and up, built with Visual Studio Community 2019)
- mosquito-2.0.15-install-windows-x32.exe (32-bit build, Windows Vista and up, built with Visual Studio Community 2019)

XAMPP

- Windows IA-32
- Linux x64

3.4.3 Standards and Policies

Wireless PAN

A Personal Area Network (PAN) is a computer network for interconnecting electronic devices within an individual person's workspace. A PAN provides data transmission among devices such as computers, smartphones, tablets and personal digital assistants.

Standard Used: IEEE 802.15

Wireless LAN

A Local Area Network (LAN) is a computer network that interconnects computers within a limited area such as a residence, school, laboratory, university campus or office building. By contrast, a Wide Area Network (WAN) not only covers a larger geographic distance, but also generally involves leased telecommunication circuits.

Standard Used: IEEE 802.11

Ethernet

Ethernet is a family of wired computer networking technologies commonly used in Local Area Networks (LAN), Metropolitan Area Networks (MAN) and Wide Area Networks (WAN). It was commercially introduced in 1980 and first standardized in 1983 as IEEE 802.3. Ethernet has since been refined to support higher bit rates, a greater number of nodes, and longer link distances, but retains much backward compatibility.

Standard Used: IEEE 802.3

Overview and Architecture

The IEEE 802 standards are restricted to computer networks carrying variable-size packets, unlike cell relay networks, for example, in which data is transmitted in short, uniformly sized units called cells. Isochronous signal networks, in which data is transmitted as a steady stream of octets, or groups of octets, at regular time intervals, are also outside the scope of the IEEE 802 standards.

Standard Used: IEEE 802(R)

Chapter 4

METHODOLOGY

4.1 General Architecture

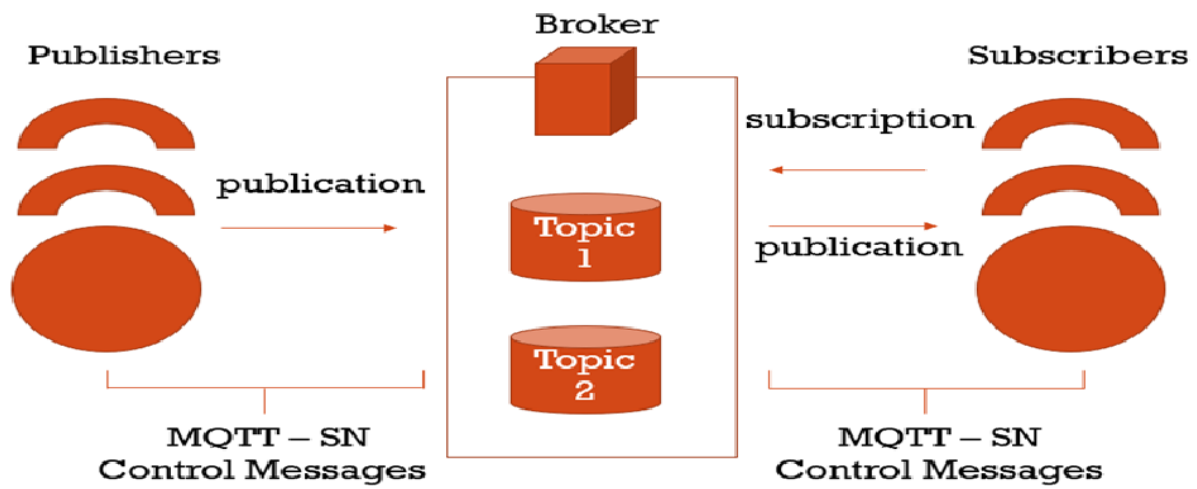


Figure 4.1: Architecture Diagram

Description

The architecture is based on a broker, which serves as the primary system, and which contains servers, communication networks, workstations, and internal linkage to the marketing system. Thus, a broker serves as a gateway, receiver, and server, and thus is required in some situations. It is possible for clients to submit very short one-hop messages to the broker and also receive messages if they have subscribed to a certain subject through the use of the MQTT server. The MQTT broker will sit in the middle and then allow the client to communicate.

4.2 Design Phase

4.2.1 Data Flow Diagram



Figure 4.2: Data Flow Diagram

Description

The data has to be collected from IoT devices and it should be under pre-processing. Then it should be transferred securely through MQTT protocol . Finally, the results should be displayed on the web page.

4.2.2 Use Case Diagram

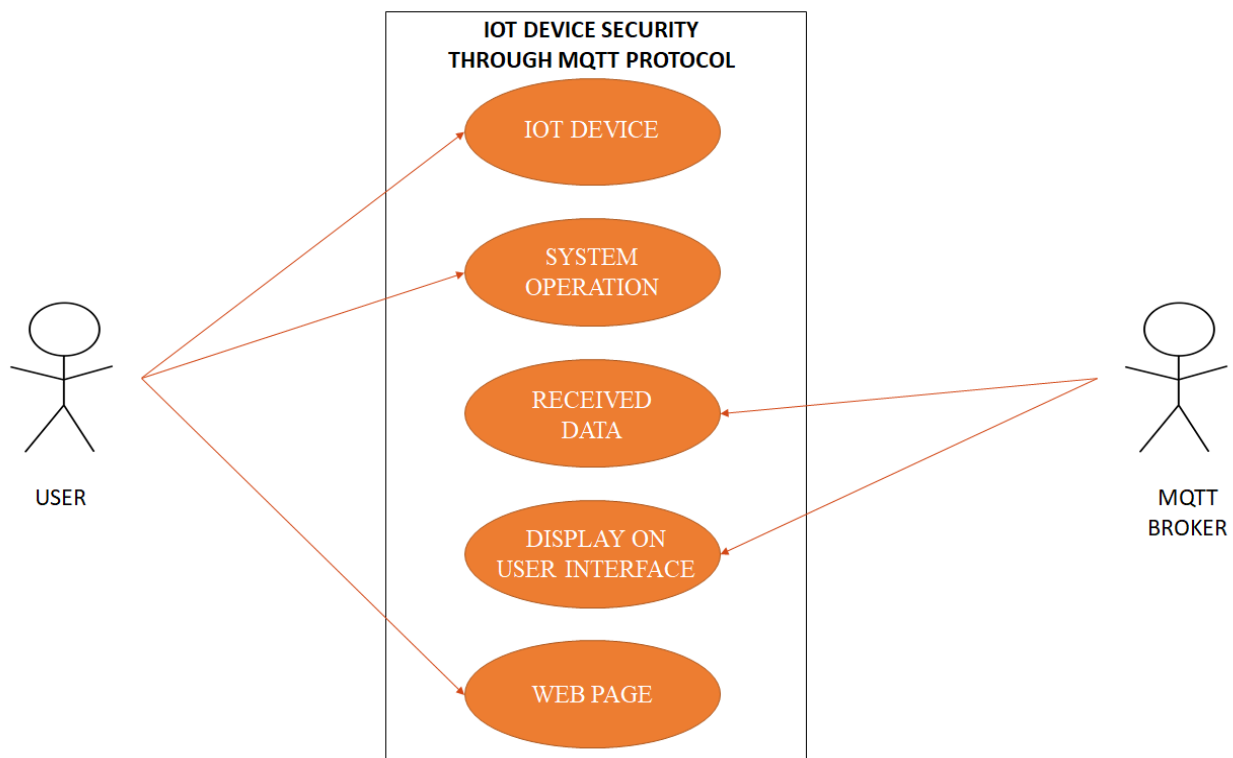


Figure 4.3: Use Case Diagram

Description

It shows the example of the usage of MQTT protocol. Publish and subscribe operations can be analogized like client and server models. The central server in MQTT is named broker that acts as the recipient of the message from the client which is, essentially, the entire node involved in the communication process. The message itself can be in the form of publish or subscribe topic. Furthermore, all the devices connected using this protocol can become publishers and subscribers. Usually, in MQTT architecture, several sensors periodically publish the results of their measurements (i.e. payload data) to a topic address. Every device that has been registered as a subscriber to a specific topic will receive a message from the broker each time the topic is updated.

4.2.3 Class Diagram

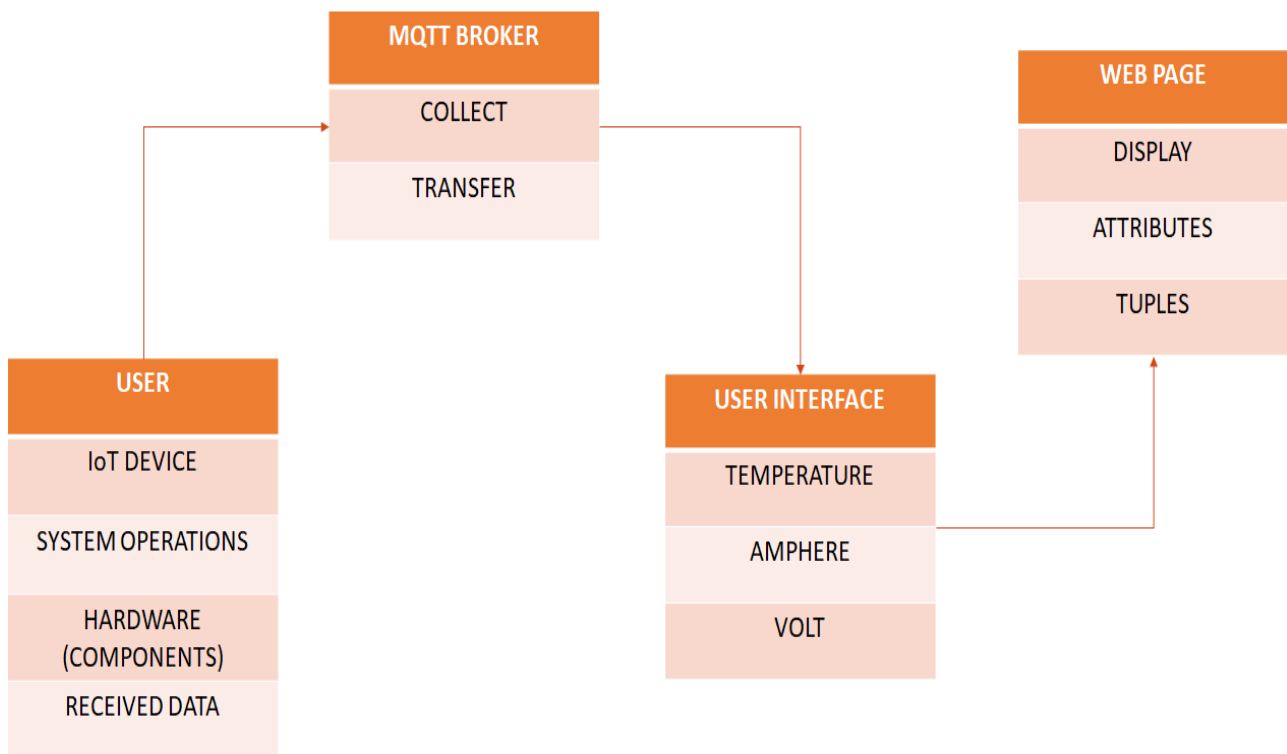


Figure 4.4: **Class Diagram**

Description

Here the broker is central hub that receives messages, filters them, and distributes them to appropriate clients, such that both message publishers, as well as subscribers, are clients.

4.2.4 Sequence Diagram

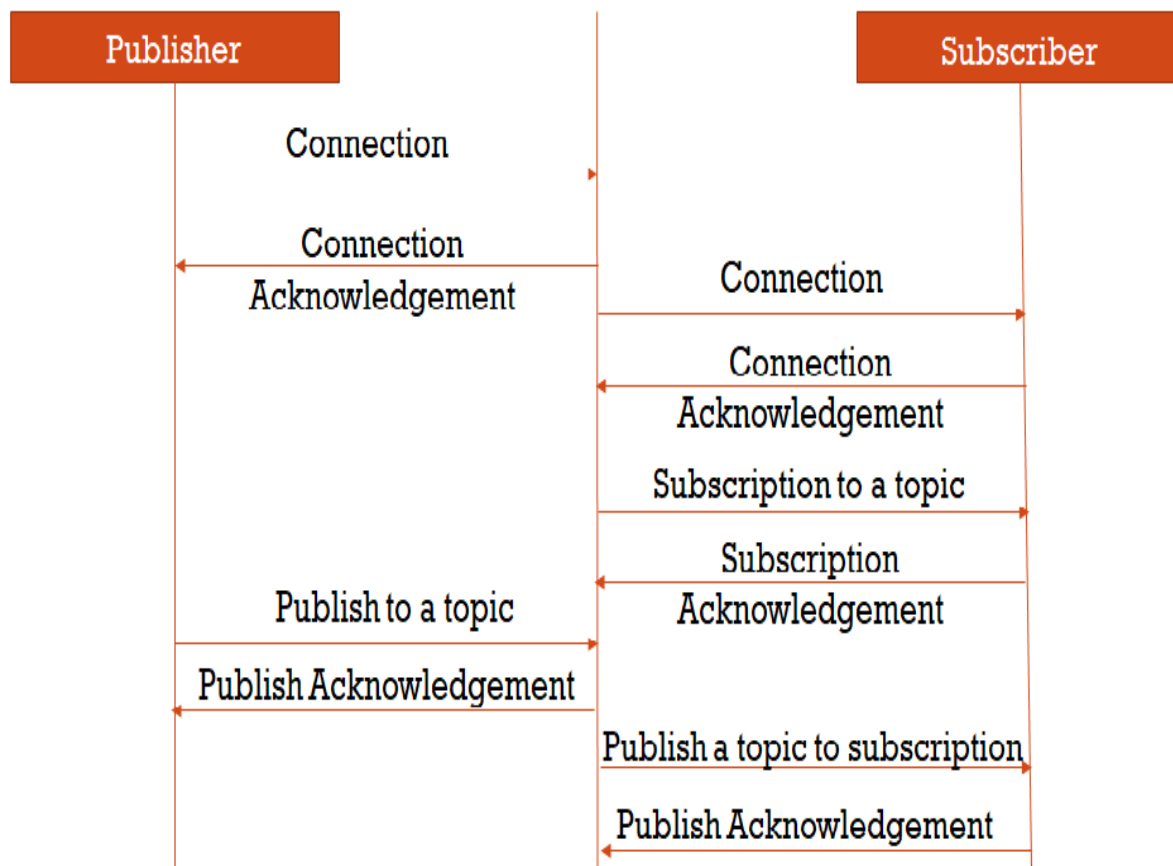


Figure 4.5: Sequence Diagram

Description

This allows an efficient way of transferring the domain-specific ML model to whichever device requires it from the main Cloud processing facility. However, normally trained ML model represents a set of weights or numbers arranged in a specific spatial object that defines this model. In order to transmit this model over the network, it should be serialized (or marshalled) before the transmission and then unserialized (or unmarshalled) back into abstract data type that defines ML model. The suggested data flow and corresponding MQTT-based protocol are presented in the Figure 4.5.

4.2.5 Collaboration diagram

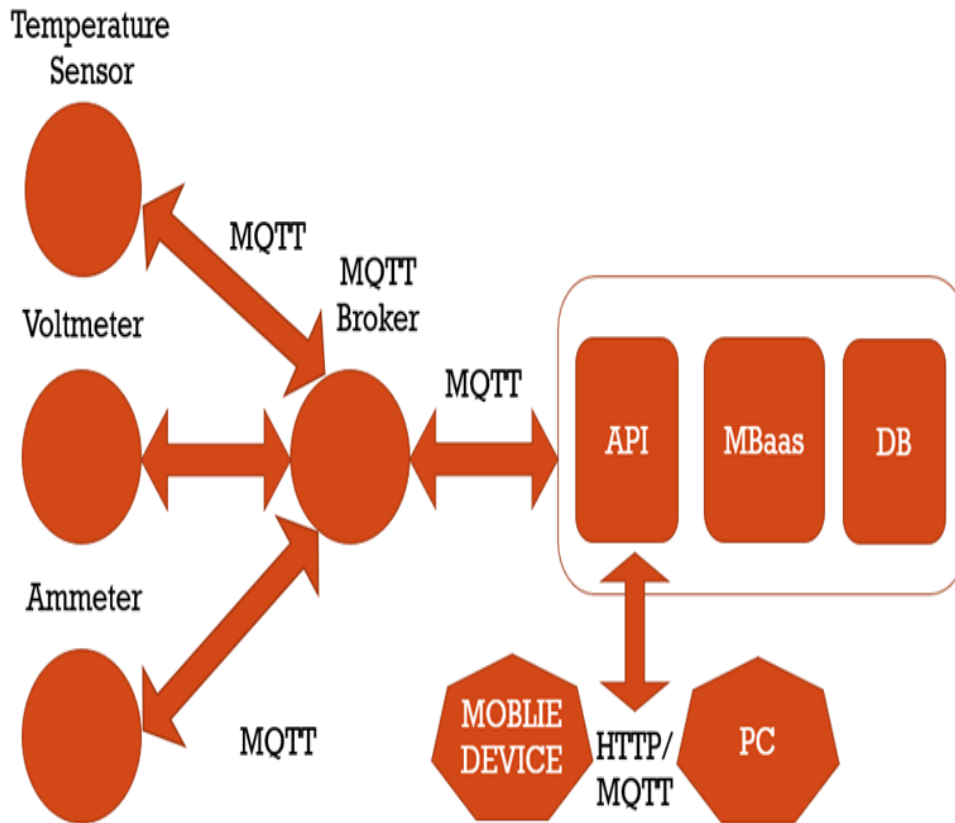


Figure 4.6: Collaboration Diagram

Description

This sequence is performed only if ICC finds the asked resource in cache (cache hit). On contrary, if nothing found (cache miss), an actual interaction with a MQTT server is required. This is done after ICC passes the request to CM which forwards it to target plugin (MQTT in this case). The plugin makes a request to server, and when response arrives the plugin passes it to CM that sends it to ICC which, finally, stores it in cache.

4.3 Algorithm & Pseudo Code

4.3.1 Algorithm

Tiny Encryption Algorithm (TEA) is a block cipher notable for its simplicity of description and implementation, typically a few lines of code. TEA operates on two 32-bit unsigned integers (could be derived from a 64-bit data block) and uses a 128-bit key. It has a Feistel structure with a suggested 64 rounds, typically implemented in pairs termed cycles. It has an extremely simple key schedule, mixing all of the key material in exactly the same way for each cycle. Different multiples of a magic constant are used to prevent simple attacks based on the symmetry of the rounds. The magic constant, 2654435769 or 0x9E3779B9 is chosen to be $2^{32}/\phi$, where ϕ is the golden ratio (as a Nothing-up-my-sleeve number).

4.3.2 Pseudo Code

```
1 #include <stdint.h>
2
3 void encrypt (uint32_t v[2], const uint32_t k[4]) {
4     uint32_t v0=v[0], v1=v[1], sum=0, i;          /* set up */
5     uint32_t delta=0x9E3779B9;                     /* a key schedule constant */
6     uint32_t k0=k[0], k1=k[1], k2=k[2], k3=k[3];  /* cache key */
7     for (i=0; i<32; i++) {                          /* basic cycle start */
8         sum += delta;
9         v0 += ((v1<<4) + k0) ^ (v1 + sum) ^ ((v1>>5) + k1);
10        v1 += ((v0<<4) + k2) ^ (v0 + sum) ^ ((v0>>5) + k3);
11    }                                                  /* end cycle */
12    v[0]=v0; v[1]=v1;
13 }
14
15 void decrypt (uint32_t v[2], const uint32_t k[4]) {
16     uint32_t v0=v[0], v1=v[1], sum=0xC6EF3720, i; /* set up; sum is (delta << 5) & 0xFFFFFFFF */
17     uint32_t delta=0x9E3779B9;                     /* a key schedule constant */
18     uint32_t k0=k[0], k1=k[1], k2=k[2], k3=k[3];  /* cache key */
19     for (i=0; i<32; i++) {                          /* basic cycle start */
20         v1 -= ((v0<<4) + k2) ^ (v0 + sum) ^ ((v0>>5) + k3);
21         v0 -= ((v1<<4) + k0) ^ (v1 + sum) ^ ((v1>>5) + k1);
22         sum -= delta;
23     }                                                  /* end cycle */
24     v[0]=v0; v[1]=v1;
25 }
```

4.4 Module Description

4.4.1 Creating user interface – App.js

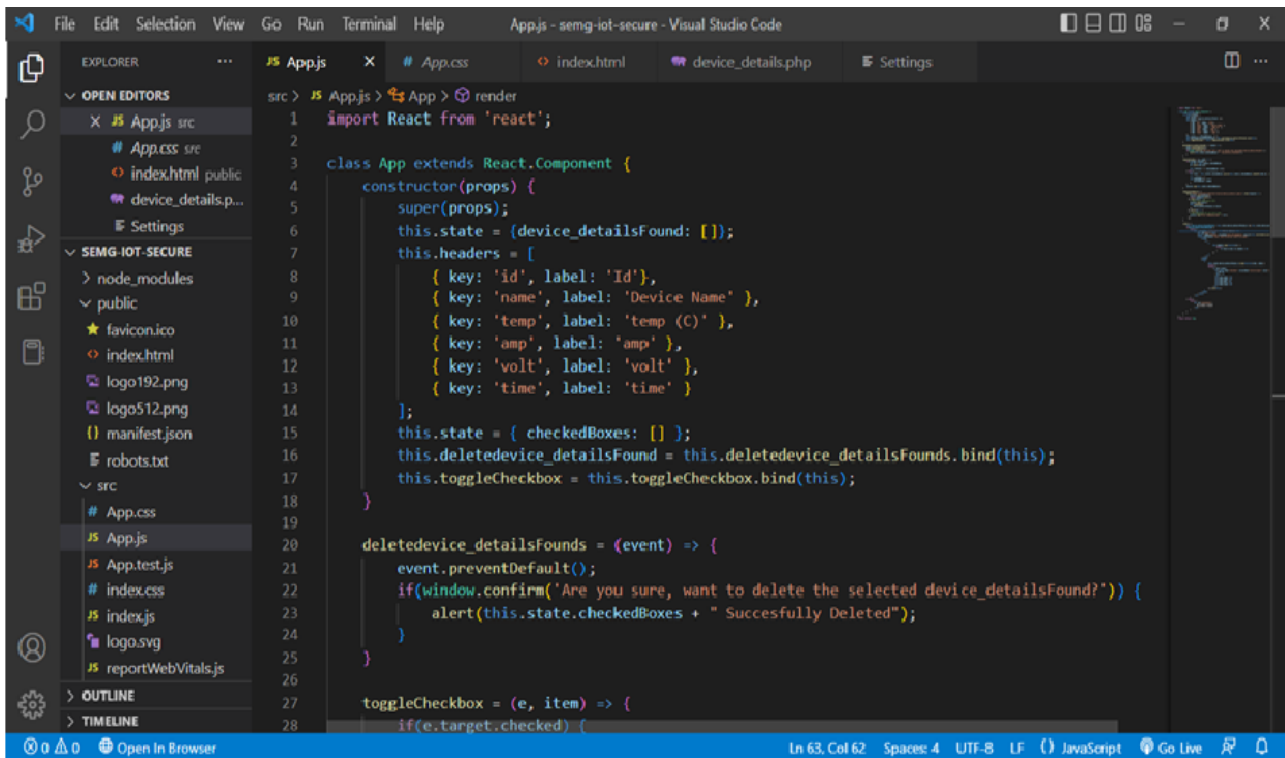
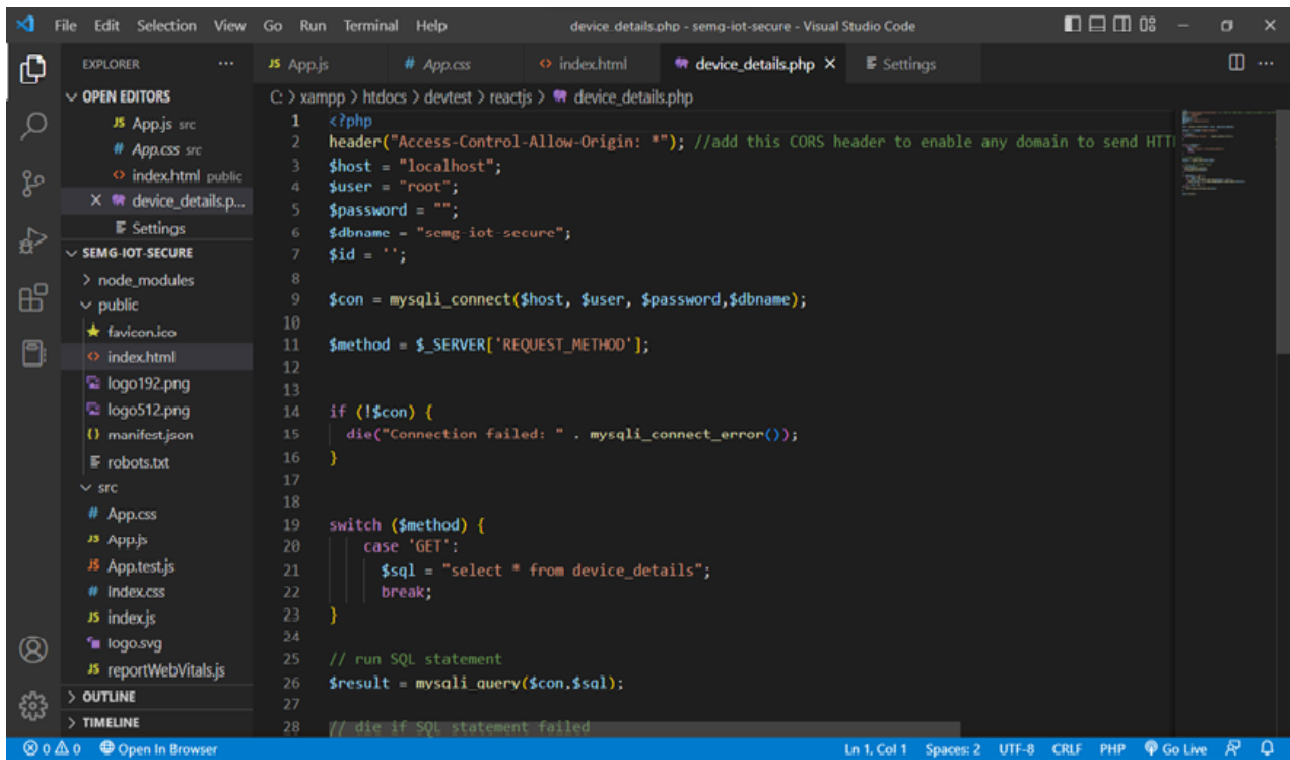


Figure 4.7: Creating user interface – App.js

4.4.2 Connecting the UI and DB by php



```
1 <?php
2 header("Access-Control-Allow-Origin: *"); //add this CORS header to enable any domain to send HTTP
3 $host = "localhost";
4 $user = "root";
5 $password = "";
6 $dbname = "semg-iot-secure";
7 $id = '';
8
9 $con = mysqli_connect($host, $user, $password,$dbname);
10
11 $method = $_SERVER['REQUEST_METHOD'];
12
13
14 if (!$con) {
15     die("Connection failed: " . mysqli_connect_error());
16 }
17
18
19 switch ($method) {
20     case 'GET':
21         $sql = "select * from device_details";
22         break;
23 }
24
25 // run SQL statement
26 $result = mysqli_query($con,$sql);
27
28 // die if SQL statement failed
```

Figure 4.8: Connecting the UI and DB by php

4.4.3 Data transferred from DB to UI through MQTT Broker

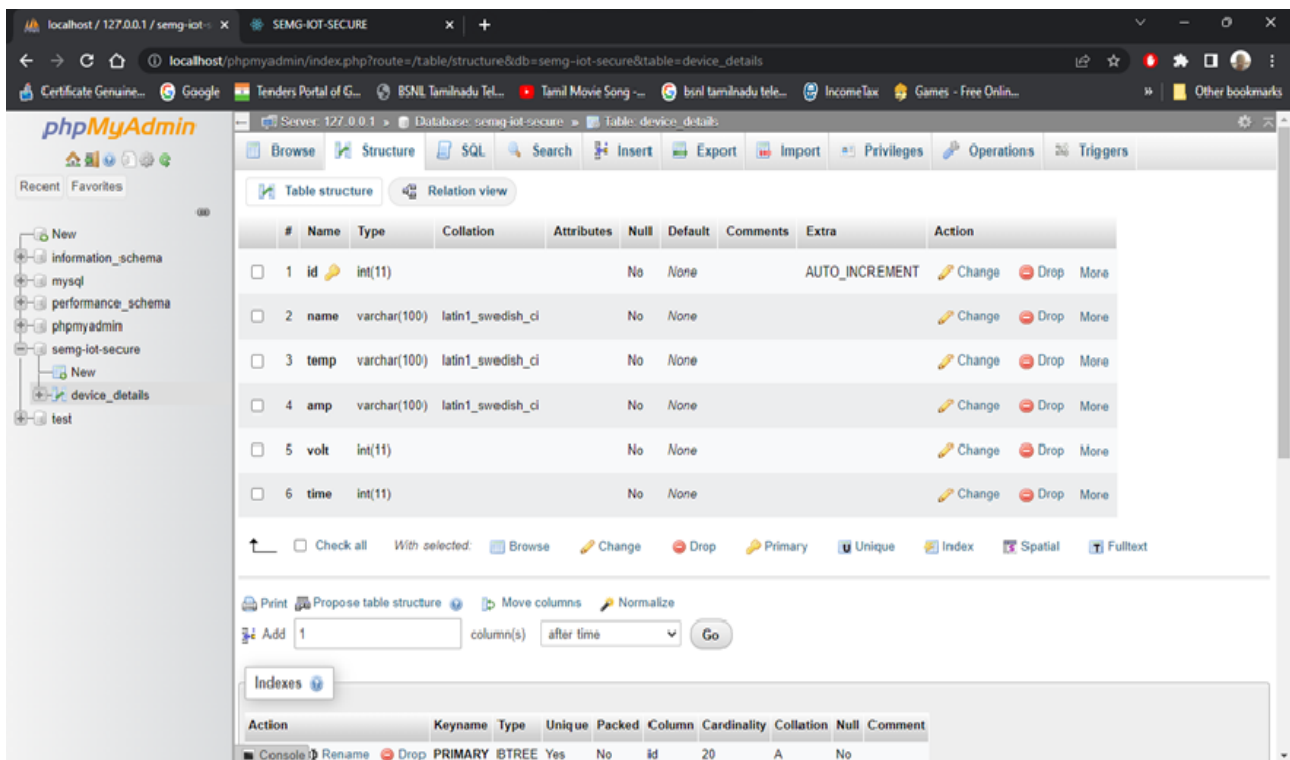


Figure 4.9: Data transferred from DB to UI through MQTT Broker

4.5 Steps to execute/run/implement the project

4.5.1 Step1

Create a React App

- `npx create-react-app ./`

4.5.2 Step2

Execute the App

- `npm start`

4.5.3 Step3

Implementing the App

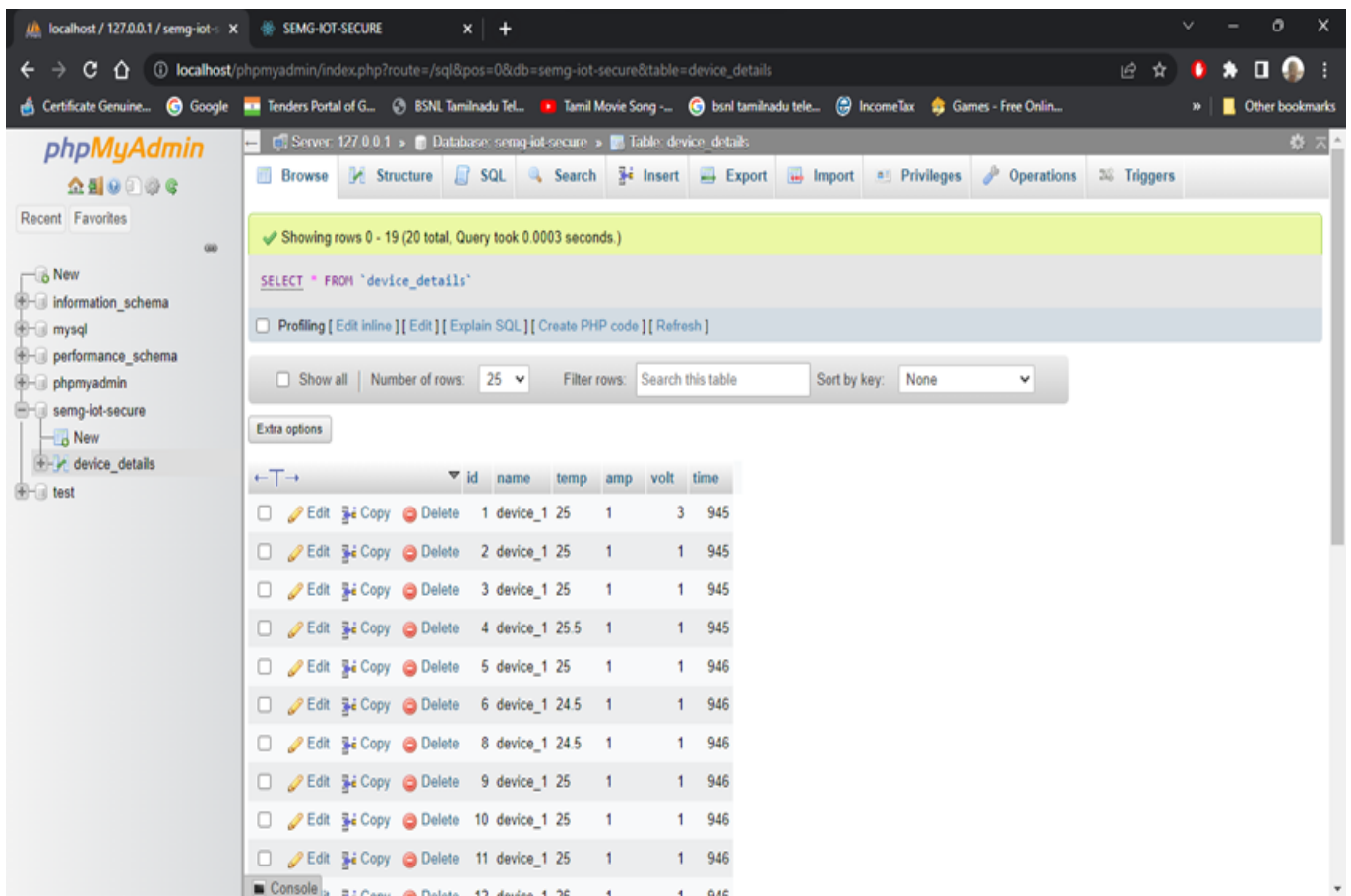
- `npm build`

Chapter 5

IMPLEMENTATION AND TESTING

5.1 Input and Output

5.1.1 Input Design



Showing rows 0 - 19 (20 total. Query took 0.0003 seconds.)

```
SELECT * FROM `device_details`
```

☐ Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

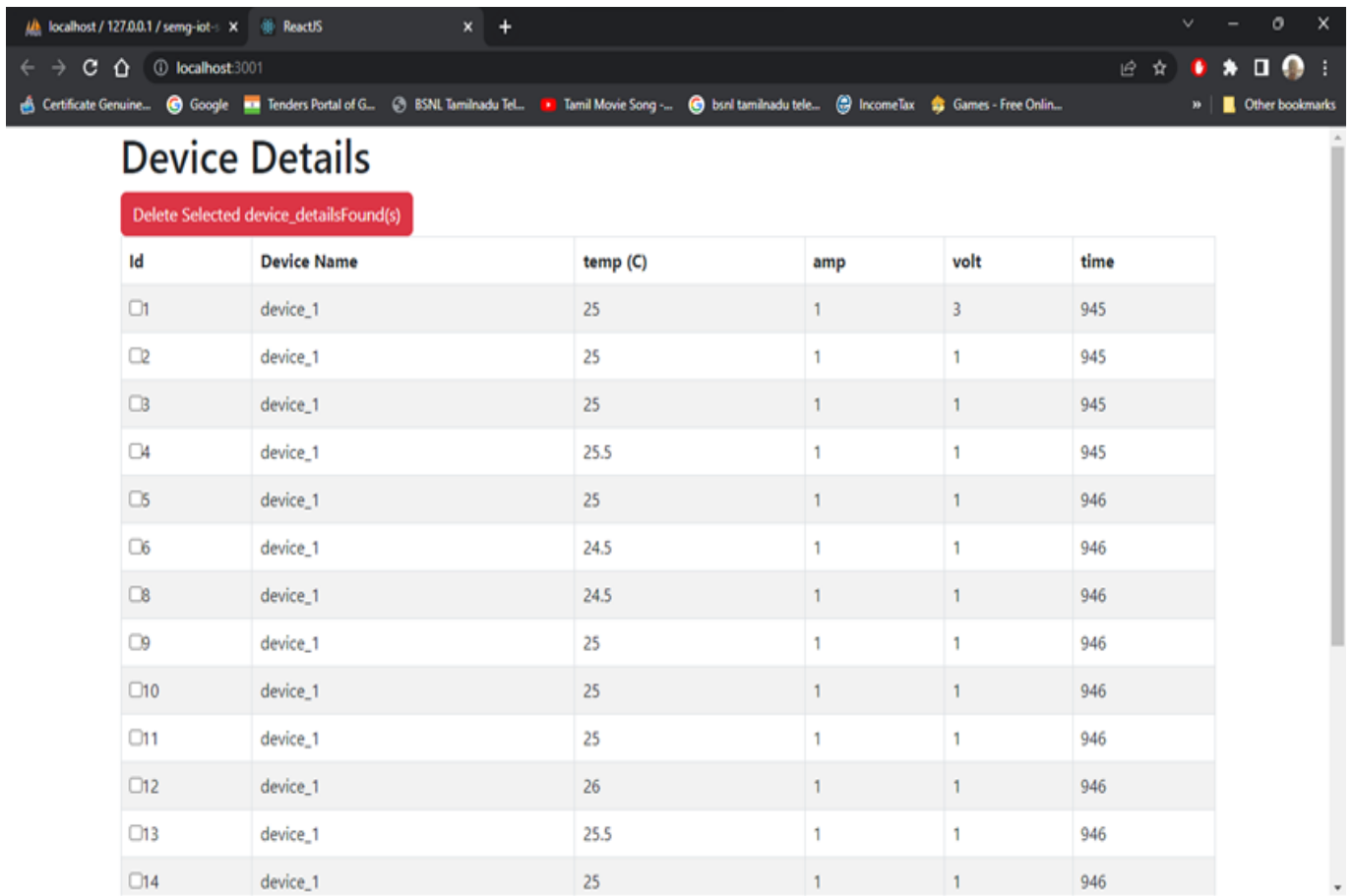
☐ Show all | Number of rows: 25 | Filter rows: Search this table | Sort by key: None

Extra options

				id	name	temp	amp	volt	time
<input type="checkbox"/>	Edit	Copy	Delete	1	device_1	25	1	3	945
<input type="checkbox"/>	Edit	Copy	Delete	2	device_1	25	1	1	945
<input type="checkbox"/>	Edit	Copy	Delete	3	device_1	25	1	1	945
<input type="checkbox"/>	Edit	Copy	Delete	4	device_1	25.5	1	1	945
<input type="checkbox"/>	Edit	Copy	Delete	5	device_1	25	1	1	946
<input type="checkbox"/>	Edit	Copy	Delete	6	device_1	24.5	1	1	946
<input type="checkbox"/>	Edit	Copy	Delete	8	device_1	24.5	1	1	946
<input type="checkbox"/>	Edit	Copy	Delete	9	device_1	25	1	1	946
<input type="checkbox"/>	Edit	Copy	Delete	10	device_1	25	1	1	946
<input type="checkbox"/>	Edit	Copy	Delete	11	device_1	25	1	1	946
<input type="checkbox"/>	Edit	Copy	Delete	12	device_1	26	1	1	946

Figure 5.1: Input

5.1.2 Output Design



localhost / 127.0.0.1 / semg-iot - x ReactJS

localhost:3001

Certificate Genuine... Google Tenders Portal of G... BSNL Tamilnadu Tel... Tamil Movie Song -... bsnl tamilnadu tele... IncomeTax Games - Free Onlin... Other bookmarks

Device Details

Delete Selected device_detailsFound(s)

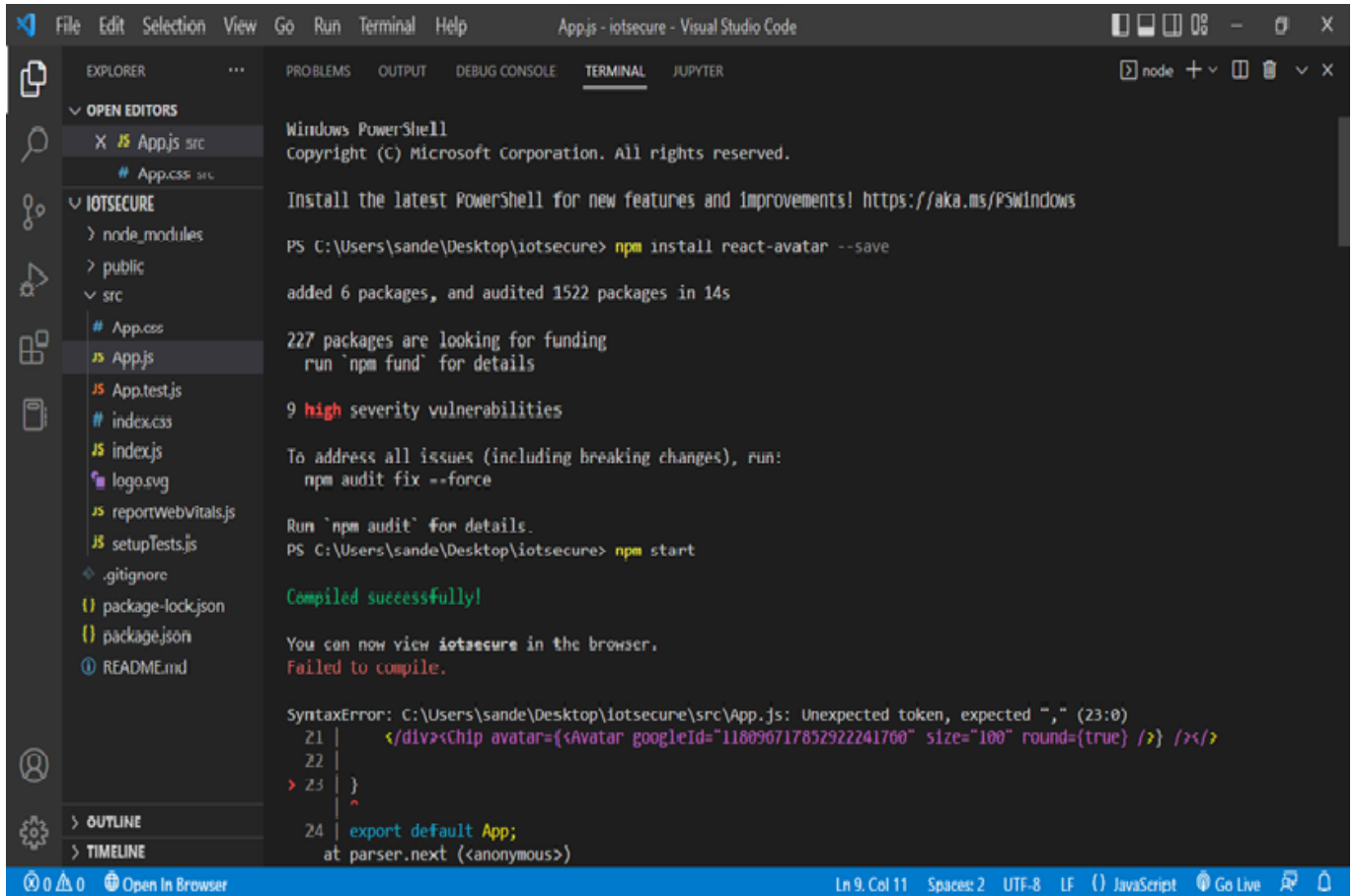
Id	Device Name	temp (C)	amp	volt	time
<input type="checkbox"/> 1	device_1	25	1	3	945
<input type="checkbox"/> 2	device_1	25	1	1	945
<input type="checkbox"/> 3	device_1	25	1	1	945
<input type="checkbox"/> 4	device_1	25.5	1	1	945
<input type="checkbox"/> 5	device_1	25	1	1	946
<input type="checkbox"/> 6	device_1	24.5	1	1	946
<input type="checkbox"/> 8	device_1	24.5	1	1	946
<input type="checkbox"/> 9	device_1	25	1	1	946
<input type="checkbox"/> 10	device_1	25	1	1	946
<input type="checkbox"/> 11	device_1	25	1	1	946
<input type="checkbox"/> 12	device_1	26	1	1	946
<input type="checkbox"/> 13	device_1	25.5	1	1	946
<input type="checkbox"/> 14	device_1	25	1	1	946

Figure 5.2: Output

5.2 Types of Testing

5.2.1 Functional testing

Input



The screenshot shows the Visual Studio Code interface with a terminal window open. The terminal is running a Windows PowerShell session. The commands and output are as follows:

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\sande\Desktop\iotsecure> npm install react-avatar --save

added 6 packages, and audited 1522 packages in 14s

227 packages are looking for funding
  run 'npm fund' for details

9 high severity vulnerabilities

To address all issues (including breaking changes), run:
  npm audit fix --force

Run 'npm audit' for details.
PS C:\Users\sande\Desktop\iotsecure> npm start

Compiled successfully!

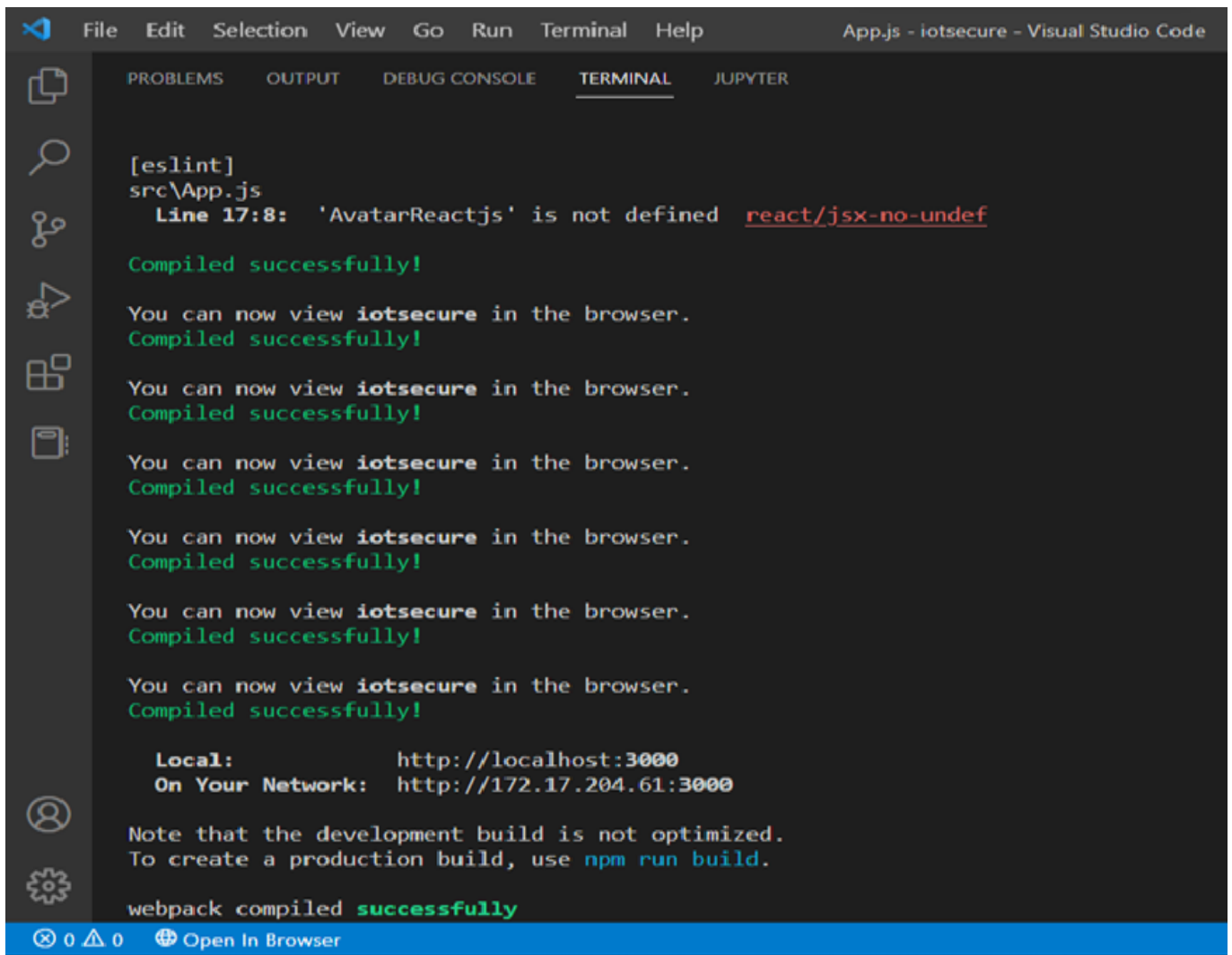
You can now view iotsecure in the browser.
Failed to compile.

SyntaxError: C:\Users\sande\Desktop\iotsecure\src\App.js: Unexpected token, expected ",", (23:0)
  21 |     </div><Chlp avatar={<Avatar googleId="118096717852922241700" size="100" round={true} /> } /></>
  22 |
> 23 |   }
     |     ^
  24 |   export default App;
     |   at parser.next (<anonymous>)
```

The error message indicates a syntax error in the `App.js` file at line 23, column 0. The error is related to an unexpected token in the JSX element.

Figure 5.3: Functional testing input

5.2.2 Test Result



The screenshot shows the Visual Studio Code interface with the 'TERMINAL' tab selected. The terminal output is as follows:

```
[eslint]
src\App.js
  Line 17:8:  'AvatarReactjs' is not defined  react/jsx-no-undef

Compiled successfully!

You can now view iotsecure in the browser.
Compiled successfully!

You can now view iotsecure in the browser.
Compiled successfully!

You can now view iotsecure in the browser.
Compiled successfully!

You can now view iotsecure in the browser.
Compiled successfully!

You can now view iotsecure in the browser.
Compiled successfully!

You can now view iotsecure in the browser.
Compiled successfully!

Local:      http://localhost:3000
On Your Network:  http://172.17.204.61:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

At the bottom of the terminal window, there is a status bar showing '0 errors, 0 warnings' and a button to 'Open In Browser'.

Figure 5.4: Functional testing output

Chapter 6

RESULTS AND DISCUSSIONS

6.1 Efficiency of the Proposed System

MQTT is often called a protocol for the Internet of Things. Which means that it must be more lightweight for network usage. The experts in MQTT solutions also note that it's especially efficient in wired data transmission. Let's see what network-related data we can get from packet sniffers to compare MQTT over SSL and HTTPS. MQTT service part requires only 10 % less traffic than HTTP. The advantage of MQTT service part over Ethernet vs Wireless is negligible. MQTT client keeps connection and publishes each piece of data to MQTT broker, HTTP keep-alive connection with POST request for each data piece, A single HTTP request with the entire pack of data. MQTT is 20 times faster and requires 50 times less traffic on the task of posting consistent time-valuable data. MQTT over SSL connecting to the HiveMQ MQTT broker.

6.2 Comparison of Existing and Proposed System

Existing system:(Transport Layer Security)

Transport Layer Security (TLS) is a cryptographic protocol designed to provide communications security over a computer network. The protocol is widely used in applications such as email, instant messaging, and voice over IP, but its use in securing HTTP remains the most publicly visible.

Proposed system:(Tiny Encryption Algorithm)

Tiny Encryption Algorithm (TEA) is a block cipher notable for its simplicity of description and implementation, typically a few lines of code. TEA operates on two 32-bit unsigned integers (could be derived from a 64-bit data block) and uses a 128-bit key. It has a Feistel structure with a suggested 64 rounds, typically implemented in pairs termed cycles. It has an extremely simple key schedule, mixing all of the

key material in exactly the same way for each cycle. Different multiples of a magic constant are used to prevent simple attacks based on the symmetry of the rounds. The magic constant, 2654435769 or 0x9E3779B9 is chosen to be $2^{32}/\phi$, where ϕ is the golden ratio (as a Nothing-up-my-sleeve number).

6.3 Sample Code

```
1 import React from 'react';
2
3 class App extends React.Component {
4   constructor(props) {
5     super(props);
6     this.state = { device_detailsFound: [] };
7     this.headers = [
8       { key: 'id', label: 'Id' },
9       { key: 'name', label: 'Device Name' },
10      { key: 'temp', label: 'temp (C)' },
11      { key: 'amp', label: 'amp' },
12      { key: 'voltage', label: 'voltage' },
13      { key: 'time', label: 'time' }
14    ];
15    this.state = { checkedBoxes: [] };
16    this.deletedevice_detailsFound = this.deletedevice_detailsFounds.bind(this);
17    this.toggleCheckbox = this.toggleCheckbox.bind(this);
18  }
19
20  deletedevice_detailsFounds = (event) => {
21    event.preventDefault();
22    if(window.confirm('Are you sure, want to delete the selected device_detailsFound?')) {
23      alert(this.state.checkedBoxes + " Successfully Deleted");
24    }
25  }
26
27  toggleCheckbox = (e, item) => {
28    if(e.target.checked) {
29      let arr = this.state.checkedBoxes;
30      arr.push(item.id);
31
32      this.setState = { checkedBoxes: arr };
33    } else {
34      let items = this.state.checkedBoxes.splice(this.state.checkedBoxes.indexOf(item.id), 1);
35
36      this.setState = {
37        checkedBoxes: items
38      }
39    }
40  }
41 }
```

```

39     }
40     console.log(this.state.checkedBoxes);
41 }
42
43 componentDidMount() {
44     fetch('http://localhost/devtest/reactjs/device_details.php/').then(response => {
45         console.log(response);
46         return response.json();
47     }).then(result => {
48         // Work with JSON data here
49         console.log(result);
50         this.setState({
51             device_detailsFound_rs: result
52         });
53     }).catch(err => {
54         // Do something for an error here
55         console.log("Error Reading data " + err);
56     });
57 }
58
59 render() {
60     const device_detailsFoundFound = this.state.device_detailsFound_rs && this.state.
        device_detailsFound_rs.length;
61     if(device_detailsFoundFound) {
62         return (
63             <div className="container"><h1>Device Details </h1>
64             <div id="msg"></div>
65             <button type="button" className="btn btn-danger" onClick={this.
                deletedevice_detailsFounds}>Delete Selected device_detailsFound(s)</button>
66             <table className="table table-bordered table-striped">
67                 <thead>
68                     <tr>
69                         {
70                             this.headers.map(function(h) {
71                                 return (
72                                     <th key={h.key}>{h.label}</th>
73                                 )
74                             })
75                         }
76                     </tr>
77                 </thead>
78                 <tbody>
79                     {
80                         this.state.device_detailsFound_rs.map(function(item, index) {
81                             return (
82                                 <tr key={index}>
83                                     <td><input type="checkbox" className="selectsingle" value="{
                                        item.id}" checked={this.state.checkedBoxes.find((p) => p.
                                            id === item.id)} onChange={(e) => this.toggleCheckbox(e,
                                                item)}>

```

```

84         {item.id}
85     </td>
86     <td>{item.name}</td>
87     <td>{item.temp}</td>
88     <td>{item.amp}</td>
89     <td>{item.volt}</td>
90     <td>{item.time}</td>
91 </tr>
92     }).bind(this))
93     }
94 </tbody>
95 </table>
96 </div>
97 )
98 } else {
99     return (
100         <div id="container">
101             No product found
102         </div>
103     )
104 }
105 }
106 }
107 export default App;

```

Chapter 7

CONCLUSION AND FUTURE ENHANCEMENTS

7.1 Conclusion

The UI/UX Interface can be used to control and monitor the various IoT Devices and embedded technologies through Wireless technology with the help of MQTT software. The most important feature in the implementation is Raspberry Pi module acts as both MQTT broker to get sensor data from clients, to distribute data to subscribers and also SQ database installed on the Raspberry Pi stores the sensor information sent by the MQTT Clients by subscribing the MQTT broker. The system can be implemented for accessing the temperature and humidity of anyplace remotely. It can also be implemented in offices, industries and homes for security purposes. Protocol can be used in home or industrial networks, as it provides bidirectional channels, which allows the subscription and publication of topics / messages and discovery of resources and endpoints in a typical IoT environment. It ensures the security of the IoT devices interconnected though wireless medium which cannot be accessed by malicious users.

7.2 Future Enhancements

Future work includes developing analytic tools to estimate upper bounds for the time-sensitive traffic latency and its incorporation in the framework within an Admission Control service. Furthermore, complementary performance evaluation campaigns will be carried out, using concrete use cases and considering additional metrics, such as reliability, resource utilization, and scalability. In addition, the integration of security features will also be analyzed, considering both eventual conflicts, limitations, and impact on resource utilization and real-time performance.

Chapter 8

PLAGIARISM REPORT



Figure 8.1: Poster Presentation

Chapter 9

SOURCE CODE & POSTER PRESENTATION

9.1 Source Code

```
1 import React from 'react';
2
3 class App extends React.Component {
4   constructor(props) {
5     super(props);
6     this.state = { device_detailsFound: [] };
7     this.headers = [
8       { key: 'id', label: 'Id' },
9       { key: 'name', label: 'Device Name' },
10      { key: 'temp', label: 'temp (C)' },
11      { key: 'amp', label: 'amp' },
12      { key: 'volt', label: 'volt' },
13      { key: 'time', label: 'time' }
14    ];
15    this.state = { checkedBoxes: [] };
16    this.deletedevice_detailsFound = this.deletedevice_detailsFounds.bind(this);
17    this.toggleCheckbox = this.toggleCheckbox.bind(this);
18  }
19
20  deletedevice_detailsFounds = (event) => {
21    event.preventDefault();
22    if(window.confirm('Are you sure, want to delete the selected device_detailsFound?')) {
23      alert(this.state.checkedBoxes + " Succesfully Deleted");
24    }
25  }
26
27  toggleCheckbox = (e, item) => {
28    if(e.target.checked) {
29      let arr = this.state.checkedBoxes;
30      arr.push(item.id);
31
32      this.setState = { checkedBoxes: arr };
33    } else {
34      let items = this.state.checkedBoxes.splice(this.state.checkedBoxes.indexOf(item.id), 1);
35    }
```

```

36         this.setState = {
37             checkedBoxes: items
38         }
39     }
40     console.log(this.state.checkedBoxes);
41 }
42
43 componentDidMount() {
44     fetch('http://localhost/devtest/reactjs/device-details.php/').then(response => {
45         console.log(response);
46         return response.json();
47     }).then(result => {
48         // Work with JSON data here
49         console.log(result);
50         this.setState({
51             device_detailsFound_rs: result
52         });
53     }).catch(err => {
54         // Do something for an error here
55         console.log("Error Reading data " + err);
56     });
57 }
58
59 render() {
60     const device_detailsFoundFound = this.state.device_detailsFound_rs && this.state.
        device_detailsFound_rs.length;
61     if(device_detailsFoundFound) {
62         return (
63             <div className="container"><h1>Device Details </h1>
64                 <div id="msg"></div>
65                 <button type="button" className="btn btn-danger" onClick={this.
                    deletedevice_detailsFounds}>Delete Selected device_detailsFound(s)</button>
66                 <table className="table table-bordered table-striped">
67                     <thead>
68                         <tr>
69                             {
70                                 this.headers.map(function(h) {
71                                     return (
72                                         <th key={h.key}>{h.label}</th>
73                                     )
74                                 })
75                             }
76                         </tr>
77                     </thead>
78                     <tbody>
79                         {
80                             this.state.device_detailsFound_rs.map(function(item, index) {
81                                 return (
82                                     <tr key={index}>

```




```

83         <td><input type="checkbox" className="selectsingle" value="{
            item.id}" checked={this.state.checkedBoxes.find((p) => p.
            id === item.id)} onChange={(e) => this.toggleCheckbox(e,
            item)}>/>
84         {item.id}
85     </td>
86     <td>{item.name}</td>
87     <td>{item.temp}</td>
88     <td>{item.amp}</td>
89     <td>{item.volt}</td>
90     <td>{item.time}</td>
91 </tr>
92     )}.bind(this))
93     }
94 </tbody>
95 </table>
96 </div>
97 )
98 } else {
99     return (
100         <div id="container">
101             No product found
102         </div>
103     )
104 }
105 }
106 }
107 export default App;

```

9.2 Poster Presentation



Vel Tech
Rangarajan Dr. Sagunthala
Vellore Institute of Technology
Chennai-600 026, India

IOT DEVICE SECURITY THROUGH MQTT PROTOCOL

Department of Computer Science & Engineering
School of Computing
1156CS601 – MINOR PROJECT
SUMMER SEMESTER 22-23

ABSTRACT

This work discusses the development of an "IoT Device Security through MQTT Protocol" on Python Platform. The work aims at the development of the protocol that helps to secure the communication between the Application and the household appliances or any IoT devices. The Internet of Things (IoT) can be described as a network of physical objects or "things" embedded with software, electronics, sensors and network connectivity that helps these objects collect and exchange data. The smart devices and sensors in home automation help collect (or sense) the physical experience and convert it into information data. The major element of home automation based on IoT is the Raspberry Pi. It collects the data from sensors which is connected through MQTT Protocol which interprets them to control and manage household devices like fan, light, heater, door, and opening and closing of curtains. For example, if there is no presence of a person in a certain room, the lights are automatically turned off for that room which the protocol ensures the security of the IoT devices interconnected through wireless medium which cannot be accessed by malicious users.

KEYWORDS : MQTT, Python, IoT, Raspberry Pi.

INTRODUCTION

IoT technology has been broadly used in diverse fields, including environmental perception, wearable medical, smart home, and smart city. IoT interconnected sensors and computers need to send data to cloud servers to perform computational activities in a standard IoT service. Message Queuing Telemetry Transport (MQTT) is a de facto standard for various Internet of Things (IoT) and industrial IoT applications. It is capable of transmitting data over low bandwidth or unreliable networks with very low consumption of power. MQTT is a bi-directional communication protocol. This helps in both sharing data, managing, and controlling devices. It is not required that the address in MQTT depends on the physical location as the IP address. Offered traffic in the network is reduced to transfer information. The data produced by a publisher are delivered to several subscribers via an MQTT broker. A client is responsible for opening network connection to the server, creating messages to be published, publishing application messages to the server. Subscribing to request application messages that it is interested in receiving, unsubscribing to remove a request for application messages and closing network connection to the server. The data/message which transferred through MQTT Broker which were stored in the Database such as SQL SQLite database which is a compact and in-process library, which do not have a separate server but reads the data and writes the data to disk files. It is also known as embedded SQL database engine. SQLite dataset engine is installed on the Raspberry Pi and a python application is run on the Raspberry Pi which subscribes the data from MQTT Broker and gets the data from broker in JSON format. This data is formatted using HTML application and is stored in the SQLite database columns.

RESULTS

The overhead expressed in percentage that other data sent in the communication between the IoT device and the MQTT broker imposes over the actual MQTT payload sent by the IoT device. From this figure we can see that it is experiment in which less overhead over the MQTT payload experienced. This is the case where standard MQTT was used, in other words the most lightweight case. We can see that the experiment which best approximates to the overhead values of experiment 1 is experiment 4. This is the case where AES-GCM payload encryption is used.

STANDARDS AND POLICIES

Wireless PAN: Personal Area Network (PAN) is a computer network for interconnecting electronic devices within an individual person's workspace. A PAN provides data transmission among devices such as computers, smartphones, tablets and personal digital assistants. **Standard Used: IEEE 802.15**

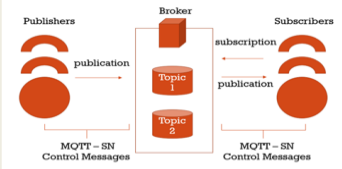
Wireless LAN: Local Area Network (LAN) is a computer network that interconnects computers within a limited area such as a residence, school, laboratory, university campus or office building. By contrast, a wide area network (WAN) not only covers a larger geographic distance, but also generally involves leased telecommunication circuits. **Standard Used: IEEE 802.11**

Ethernet is a family of wired computer networking technologies commonly used in Local Area Networks (LAN), Metropolitan Area Networks (MAN) and Wide Area Networks (WAN). It was commercially introduced in 1980 and first standardized in 1983 as IEEE 802.3. Ethernet has since been refined to support higher bit rates, a greater number of nodes, and longer link distances, but retains much backward compatibility. **Standard Used: IEEE 802.3**

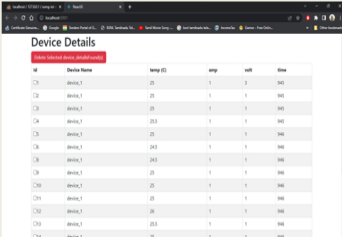
METHODOLOGIES

It is based on a broker, which serves as the primary system, and which contains servers, communication networks, workstations, and internal linkage to the marketing system. Thus, a broker serves as a gateway, receiver, and server, and thus is required in some situations. It is possible for clients to submit very short one-hop messages to the broker and also receive messages if they have subscribed to a certain subject through the use of the MQTT server. The MQTT broker will sit in the middle and then allow the client to communicate. Publish and subscribe operations can be analogized like client and server models. The central server in MQTT is named broker that acts as the recipient of the message from the client which is, essentially, the entire node involved in the communication process. The message itself can be in the form of publish or subscribe topic. Furthermore, all the devices connected using this protocol can become publishers and subscribers. Usually, in MQTT architecture, several sensors periodically publish the results of their measurements (i.e. payload data) to a topic address. Every device that has been registered as a subscriber to a specific topic will receive a message from the broker each time the topic is updated.

ARCHITECTURE DIAGRAM



OUTPUT SCREENSHOT



ID	Device Name	Temp (C)	Humid	Volts	Time
C1	Arduino1	25	1	1	140
C2	Arduino1	25	1	1	140
C3	Arduino1	25	1	1	140
C4	Arduino1	25.5	1	1	140
C5	Arduino1	25	1	1	140
C6	Arduino1	24.5	1	1	140
C7	Arduino1	24.5	1	1	140
C8	Arduino1	25	1	1	140
C9	Arduino1	25	1	1	140
C10	Arduino1	25	1	1	140
C11	Arduino1	25	1	1	140
C12	Arduino1	26	1	1	140
C13	Arduino1	25.5	1	1	140
C14	Arduino1	25	1	1	140

CONCLUSIONS

The UI/UX Interface can be used to control and monitor the various IoT Devices and embedded technologies through Wireless technology with the help of MQTT software. The most important feature in the implementation is Raspberry Pi module acts as both MQTT broker to get sensor data from clients, to distribute data to subscribers and also SQ database installed on the Raspberry Pi stores the sensor information sent by the MQTT Clients by subscribing the MQTT broker. The system can be implemented for accessing the temperature and humidity of anyplace remotely. It can also be implemented in offices, industries and homes for security purposes. Protocol can be used in home or industrial networks, as it provides bidirectional channels, which allows the subscription and publication of topics / messages and discovery of resources and endpoints in a typical IoT environment. It ensures the security of the IoT devices interconnected through wireless medium which cannot be accessed by malicious users.

ACKNOWLEDGEMENT

- Mr. V. Ashok Kumar, M.TECH., Assistant Professor
- +91 7010890795
- vashokkumar@veltech.edu.in

Figure 9.1: Poster Presentation

References

- [1] F. De Rango, G. Potrino, M. Tropea, and P. Fazio, “Energy-aware dynamic Internet of Things security system based on Elliptic Curve Cryptography and Message Queue Telemetry Transport protocol for mitigating Replay attacks,” *Pervasive and Mobile Computing*, vol. 61, Article 101105, Jan. 2020.
- [2] D. Dinculeană and X. Cheng, “Vulnerabilities and Limitations of MQTT Protocol Used between IoT Devices,” *Applied Science*, vol. 9, no.5, Article 848, Feb. 2019.
- [3] O. Sadio, I. Ngom, and C. Lishou, “Lightweight Security Scheme for MQTT/MQTT-SN Protocol,” in *Proc. the 6th Int’l conf. on Internet of Things: Systems, Management and Security*, Granada, Spain, Oct. 22-25, 2020.
- [4] B. Girgenti, P. Perazzo, C. Vallati, F. Righetti, G. Dini, and G. Anastasi, “On the Feasibility of Attribute-Based Encryption on Constrained IoT Devices for Smart Systems,” in *Proc. 2019 IEEE International Conference on Smart Computing*, pp. 225-232, Washington D.C., U.S.A., Jun. 12-15, 2020.
- [5] M. Calabretta, R. Pecori, M. Vecchio, and L. Veltri, “MQTT-Auth: a Token-based Solution to Endow MQTT with Authentication and Authorization Capabilities,” *Journal of Communications Software and Systems*, vol. 14, no. 4, pp. 320-331, Dec. 2018.
- [6] Emqx X Broker—High Performance MQTT Message Broker Documentation. Accessed: Aug. 23, 2020. [Online]. Available: <https://docs.emqx.io/broker/latest/en/>
- [7] HiveMQ Community Edition. Accessed: Aug. 23, 2020. [Online]. Available: <https://github.com/680hivemq/hivemq-community-edition>

- [8] HiveMQ Documentation. Accessed: Aug. 23, 2020. [Online]. Available: <https://www.hivemq.com/docs/hivemq/4.3/>
- [9] Curtin University. Introduction to the Internet of Things. Accessed: Aug. 23, 2020. [Online]. Available: <https://study.curtin.edu.au/offering/mooc-introduction-to-the-internet-of-things-iot1x>
- [10] IBM Knowledge Center. Accessed: Aug. 24, 2020. [Online]. Available: [https://www.ibm.com/support/knowledgecenter/en/SSWMAJ₅.0.0.1/.html](https://www.ibm.com/support/knowledgecenter/en/SSWMAJ_5.0.0.1/.html)