INTRODUCTION

1.1 GENERAL

This project involves development of a Web Application which provides a platform to users to control display of a device which shows the information that includes calendar events, to-do tasks, etc. The ultimate goal of this project is to make a dynamic web-app which would display the contents specified by the user which would help the user to get updated with event and to work efficiently. The concept of Web Application is widely used and information is available from various sources but this information is not available at right time and at the right place.

1.2 WEB-APP

Web Application are not real applications; they are really websites that, in many ways, look and feel like native applications, but are not implemented as such. They are run by a browser and typically written in HTML5. Users first access them as they would access any web page: they navigate to a special URL and then have the option of “installing” them on their home screen by creating a bookmark to that page. Web apps became really popular when HTML5 came around and people realized that they can obtain native-like functionality in the browser. Today, as more and more sites use HTML5, the distinction between web apps and regular web pages has become blurry

Information can be obtained from various sources like Google, Facebook, Twitter, etc. The Web Application aims to collect this information from various sources, understand this information and provide user the relevant information. The user has the ability to control the Web Application, the Web Application provides user with a Graphical User Interface (GUI) which includes options to display specified content to the display device. [1]

1.3 WEB-SERVICES

A web service is a collection of open protocols and standards used for exchanging data between applications or systems. Software applications written in various programming languages and running on various platforms can use web services to exchange data over computer networks like the Internet in a manner similar to inter-process communication on a single computer. This interoperability (e.g., between Java and Python, or Windows and Linux applications) is due to the use of open standards.

We aim to use RESTful web services for the development of Digiframe. REST, an architectural style of shaping web services which describes the act of transferring a state of something by its representation. The representation of data can be done using XML or JSON which are two standardized and easy ways to handle, transfer and represent data. We will be using JSON representation in for Digiframe as JSON provides a more compact and easy representation of data. Another reason behind using JSON is, it is a subset of JavaScript and hence it will be more compatible with the web application.

The main advantage of REST is that it totally separates the user interface from the server and the data storage. The separation between client and server has one evident advantage, and that is that each development team can scale the product without too much problem.

1.4 HARDWARE DEVICES

The key element of this project is Raspberry Pi which acts as a server that receives the data from the Web Application, processes this data and provides content to display device which have a Universal Serial Bus (USB) or High-Definition Multimedia Interface (HDMI) port. The server receives the data in format encoded by MQTT. MQTT is an open message protocol for machine-to-machine (M2M) or Internet of Things (IoT) communications that enables the transfer of telemetry-style data (i.e. measurements collected in remote locations) in the form of messages from devices and sensors, along unreliable or constrained networks, to a server. Andy Stanford-Clark of IBM, and Arlen Nipper of Cirrus Link Solutions invented the protocol.

1.4.1 RASPBERRY PI

The Raspberry Pi is a series of credit card-sized single-board computers developed in the United Kingdom by the Raspberry Pi Foundation to promote the teaching of basic computer science in schools and developing countries.

The Raspberry Pi hardware has evolved through several versions that feature variations in memory capacity and peripheral-device support.

LITRATURE REVIEW

2.1 INTERNET OF THINGS

The internet of things is the network of physical objects or “things”. The IoT transforms these objects from being traditional too smart by exploiting its underlying technologies [1]. Internet of things allows objects to be sensed and controlled remotely across existing network infrastructure, creating opportunities for more direct integration between the physical world and computer based systems and resulting in improved efficiency, accuracy, and economic benefit.

Each thing is uniquely identifiable through its embedded computing system but is able to interoperate within the existing Internet infrastructure experts estimate that the IoT will consist of almost 50 billion objects by 2020.[2]

2.2 SMART DISPLAY SCREENS USING IOT

One of the several attempts in Internet of things applications include the development of a display panel powered by an Arduino device or a raspberry pi device. These type of applications typically include digital notice boards, digital wall calendars, digital photo frames also called as raspberry pi media panel. These application generally serve one of these purposes. The data to be displayed on such a screen is generally kept local to the screen in case it is powered by a raspberry pi device which can support a small amount of secondary memory. There are smart screen systems for which the data to be displayed on the screen is managed remotely. The values are set remotely through SMS based system incorporating the widely used GSM to facilitate the communication [3] or a traditional socket programming model is used for the communication [4].

2.3 MQTT PROTOCOL

MQTT is an open message protocol for machine-to-machine (M2M) or Internet of Things (IoT) communications that enables the transfer of telemetry-style data (i.e. measurements collected in remote locations) in the form of messages from devices and sensors, along unreliable or constrained networks, to a server.

MQTT protocol is created in the late 1990s by Andy Stanford-Clark of IBM and Arlen Nipper, then of Arcom Control Systems. It runs over TCP/IP. The current version of MQTT is 3.1.1. Its primary goals are:
to avoid polling of sensors, allowing data to be sent to interested parties the moment it is ready lightweight, so that it can be used on very low bandwidth connections

MQTT's strengths are simplicity, a compact binary packet payload (compressed headers, much less verbose than HTTP), and it makes a good fit for simple push messaging scenarios such as temperature updates, stock price tickers, oil pressure feeds or mobile notifications. It also works well connecting constrained or smaller devices and sensors to the enterprise, such as connecting an Arduino device to a web service, for example.[8] Features of the protocol include:

The publish/subscribe message pattern to provide one-to-many message distribution and decoupling of applications

A messaging transport that is agnostic to the content of the
payload The use of TCP/IP to provide basic network connectivity

MQTT supports three quality of service levels, "Fire and forget", "delivered at least once" and "delivered exactly once".

A small transport overhead (the fixed-length header is just 2 bytes), and protocol exchanges minimised to reduce network traffic

A mechanism to notify interested parties to an abnormal disconnection of a client using the Last Will and Testament feature.[5]

2.3.1 THE PUBLISH/SUBSCRIBE PATTERN

It follows a publish/subscribe architecture, as shown in Figure , where the system consists of three main components: publishers, subscribers, and a broker. From IoT point of view, publishers are basically the lightweight sensors that connect to the broker to send their data and go back to sleep whenever possible. Subscribers are applications that are interested in a certain topic, or sensory data, so they connect to brokers to be informed whenever new data are received. The brokers classify sensory data in topics and send them to subscribers interested in the topics. [6]

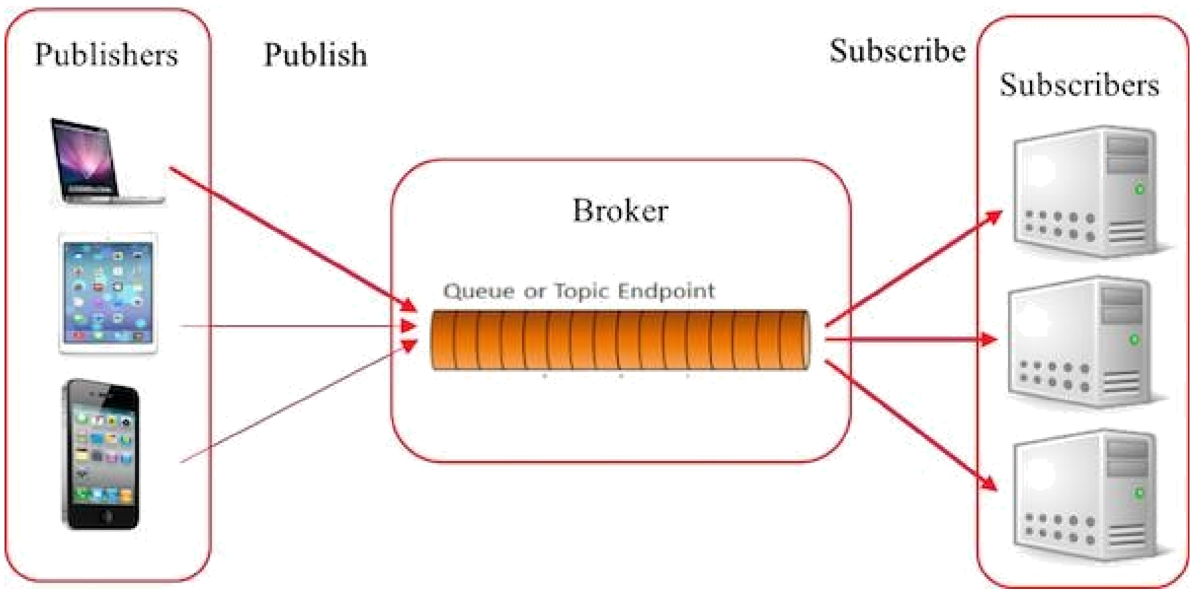


Fig 2.1: MQTT ARCHITECTURE. [9]

2.3.1.1 SCALABILITY

Pub/Sub also provides a greater scalability than the traditional client-server approach**.** This is because operations on the broker can be highly parallelized and processed event-driven. Also often message caching and intelligent routing of messages is decisive for improving the scalability. But it is definitely a challenge to scale publish/subscribe to millions of connections. This can be achieved using clustered broker nodes in order to distribute the load over more individual servers with load balancers. [6]

2.3.2 CLIENT, BROKER AND CONNECTION ESTABLISHMENT

2.3.2.1 CLIENT

A MQTT client can be both a publisher & subscriber at the same time. A MQTT client is any device from a micro controller up to a full-fledged server, that has a MQTT library running and is connecting to an MQTT broker over any kind of network. This could be a really small and resource constrained device that is connected over a wireless network and has a library strapped to the minimum or a typical computer running a graphical MQTT client for testing purposes, basically any device that has a TCP/IP stack and speaks MQTT over it. MQTT client libraries are available for a huge variety of programming languages, for example Android, Arduino, C, C++, C#, Go, iOS, Java, JavaScript, .NET.[7]

2.3.2.2 BROKER

A broker (Server) can handle up to thousands of concurrently connected MQTT clients. The broker is primarily responsible for receiving all messages, filtering them, decide who is interested in it and then sending the message to all subscribed clients. It also holds the session of all persisted clients including subscriptions and missed messages . Another responsibility of the broker is the authentication and authorization of clients. And at most of the times a broker is also extensible, which allows to easily integrate custom authentication, authorization and integration into backend systems.[7]

2.3.2.3 MQTT CONNECTION

The MQTT protocol is based on top of TCP/IP and both client and broker need to have a TCP/IP stack.

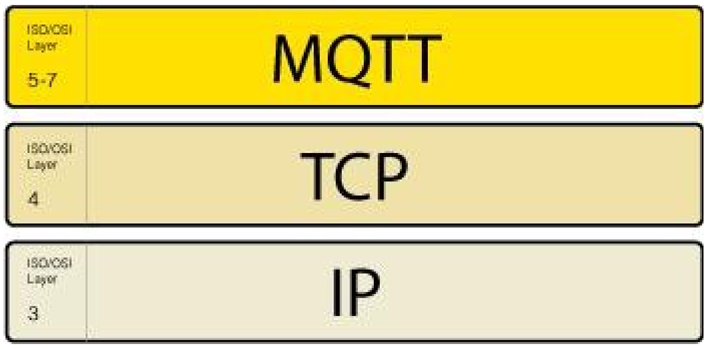


Fig 2.2: IoT LAYERS (MQTT)

The MQTT connection itself is always between one client and the broker, no client is connected to another client directly. **The connection is initiated through a client sending a CONNECT message to the broker. The broker response with a CONNACK** and a status code. Once the connection is established, the broker will keep it open as long as the client doesn't send a disconnect command or it loses the connection.[7]

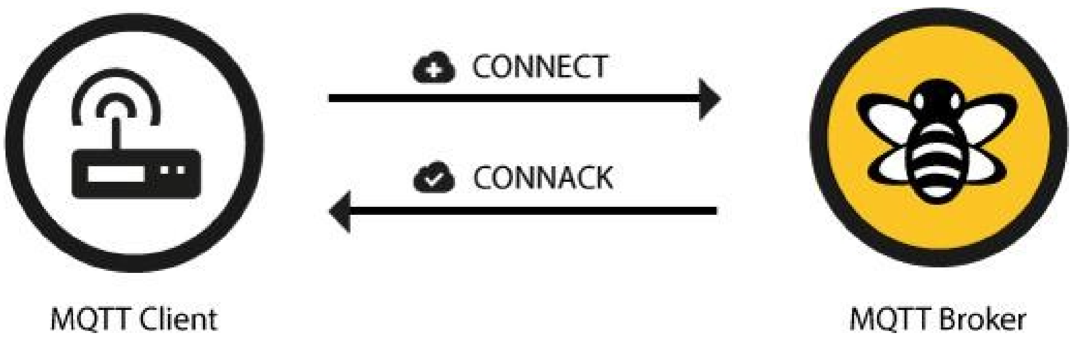


Fig 2.3: MQTT CONNECTION

2.3.3 TOPICS

A topic is a UTF-8 string, which is used by the broker to filter messages for each connected client. A topic consists of one or more topic levels. Each topic level is separated by a forward slash (topic level separator).

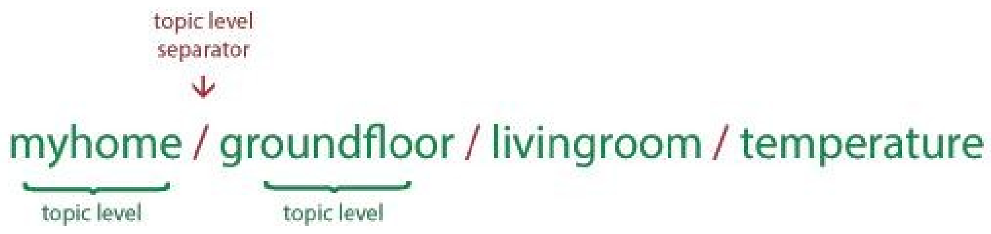


Fig 2.4: TOPICS IN MQTT

In comparison to a message queue a topic is very lightweight. There is no need for a client to create the desired topic before publishing or subscribing to it, because a broker accepts each valid topic without any prior initialization.

2.3.4 APPLICATION LEVEL QoS

Three qualities of service for message delivery:

"At most once", where messages are delivered according to the best efforts of the underlying TCP/IP network. Message loss or duplication can occur. This level could be used, for example, with ambient sensor data where it does not matter if an individual reading is lost as the next one will be published soon after.

"At least once", where messages are assured to arrive but duplicates may occur.

"Exactly once", where message are assured to arrive exactly once. This level could be used, for example, with billing systems where duplicate or lost messages could lead to incorrect charges being applied.[5]

2.3.5 SECURITY

MQTT brokers may require username and password authentication from clients to connect. To ensure privacy, the TCP connection may be encrypted with SSL/TLS.

2.4 REST

REST stands for Representational State Transfer. It relies on a stateless, client-server, cacheable communications protocol -- and in virtually all cases, the HTTP protocol is used. REST is an architecture style for designing networked applications. The idea is that, rather than using complex mechanisms such as CORBA, RPC or SOAP to connect between machines, simple HTTP is used to make calls between machines.

RESTful applications use HTTP requests to post data (create and/or update), read data (e.g., make queries), and delete data. Thus, REST uses HTTP for all four CRUD (Create/Read/Update/Delete) operations.

REST is a lightweight alternative to mechanisms like RPC (Remote Procedure Calls) and Web Services (SOAP, WSDL, et al.). Later, we will see how much more simple REST is. As a programming approach, REST is a lightweight alternative to Web Services and RPC. Much like Web Services, a REST service is:

Platform-independent (you don't care if the server is Unix, the client is a Mac, or anything else),

Language-independent (C# can talk to Java, etc.),

Standards-based (runs on top of HTTP), and

Can easily be used in the presence of firewalls. [14]

PROPOSED WORK

3.1 OBJECTIVES

To create an interface in hybrid app for generating events.

To create an interface in hybrid app for adding images.

To prepare the REST services in Java.

To create message communication between mobile app and Raspberry Pi.

Consume calendar data on Raspberry Pi.

Consume image(s) data on Raspberry Pi.

Process and display calendar data on remote display device.

Process and display image(s) data on remote display device.

3.2 METHODOLOGY

The DIGIFRAME system flow can be bifurcated into the following components:

1 **User Web-App**

The user Web-App accepts data from user in the form of event(s) and/or image(s). The user has multiple options which would give access to various features (calendar, to-do list etc.) that would be shown on the display device. The user can create and update tasks using the Web-App.

2 **MQTT Broker**

The MQTT broker which resides on raspberry pi is responsible for accepting the data under various topics which calendar and images in our project. Whenever the subscriber subscribes to a topic, the broker forwards the message under that topic to the subscriber.

3 **Display Device**

The display device is connect to raspberry pi circuit. The subscriber receives the data from the MQTT broker, processes it and displays it on remote display device.

3.3 FEATURES

Real-time transfer of data.

No size restriction on amount of data.

Uniform display on remote display device irrespective of screen size.

SYSTEM DESIGN

4.1 OVERVIEW

We present the architecture and design of the proposed system which is based on the publish-subscribe pattern of MQTT protocol which works at real-time. The data does not get sored anywhere it is just transmitted from the publisher to the subscriber.

4.2 ARCHITECTURE

From a top view, Dexter has been proposed as a system consisting of three main components

1. User Web-App (Publisher)
2. Remote Display (subscriber)

4.2.1 USER WEB APP

The User Web-App provides a Graphical User Interface (GUI) that has multiple tabs which would give access to various features (calendar, to-do list etc.) that would be shown on the display device. The user can create and update tasks using the Web-App. User app is responsible for accepting data from the user and providing the interface to the user in order to manipulate the data to control what is going to be displayed on the screen. The user app can be accessed from any computing device connected to the internet so as to control what is going to be displayed on the screen.

The web services deployed on the tomcat server are responsible to handle the user requests and perform the operation on the resources. The RESTful web services represent the user data as resources and performs the CRUD operations on them. The state of resources is transferred using the JSON format through MQTT protocol.

4.2.2 REMOTE DISPLAY

In the MQTT terms, the data is published from the user end and it is subscribed at the other end that is the remote display. The remote display is powered by a raspberry pi where the MQTT Broker runs. The broker is responsible to consume the data.

The subscriber connects to the broker and subscribes to certain topics so as to access the data under that topic and process that data in order to display it on the screen.

4.3 DATA FLOW DIAGRAM

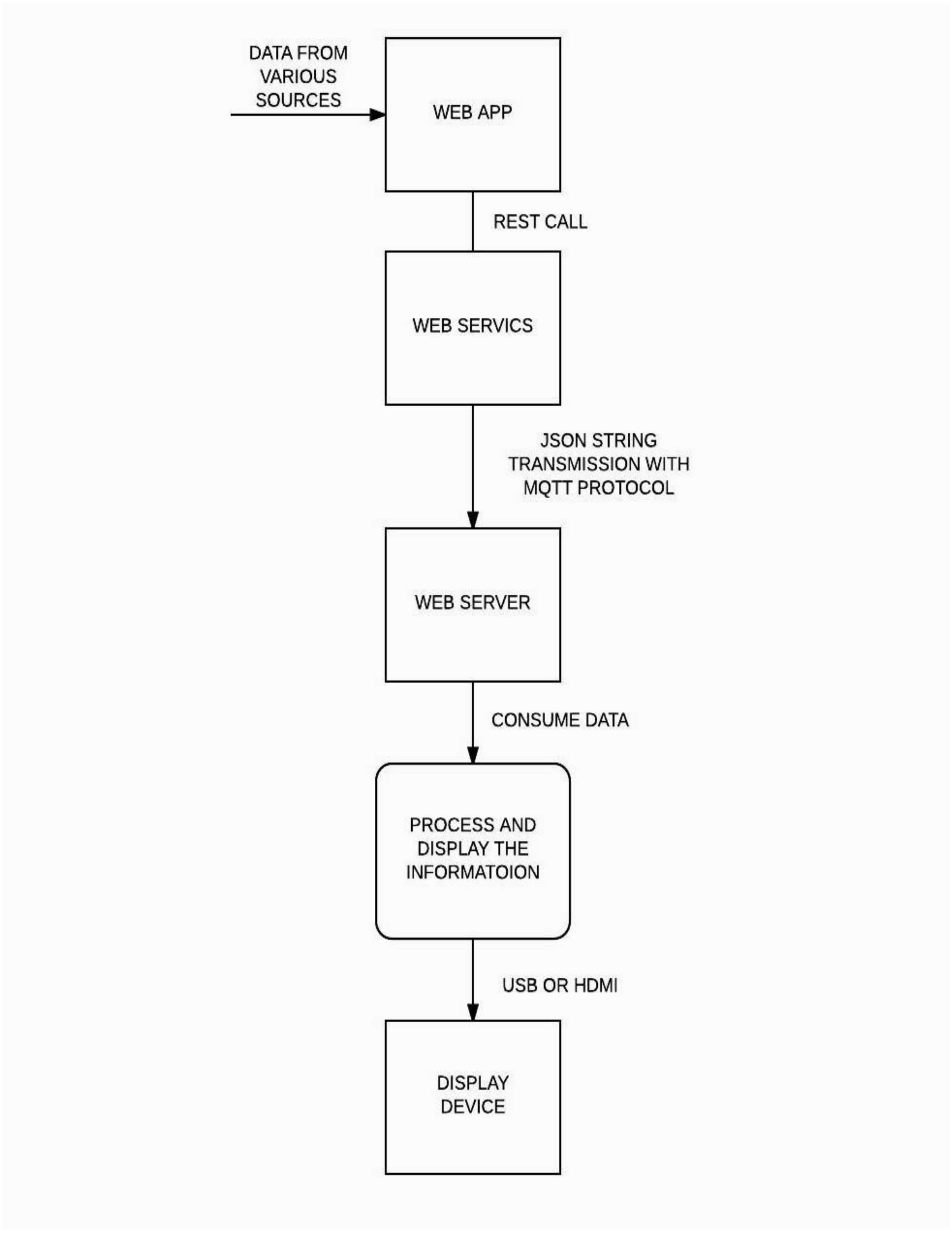


Fig 4.1: DATA FLOW DIAGRAM

IMPLEMENTATION

5.1 HARDWARE REQUIREMENTS

5.1.1 RASPBERRY PI

The Raspberry Pi is a series of credit card-sized single-board computers developed in the United Kingdom by the Raspberry Pi Foundation to promote the teaching of basic computer science in schools and developing countries.[12]

5.1.1.1 Hardware

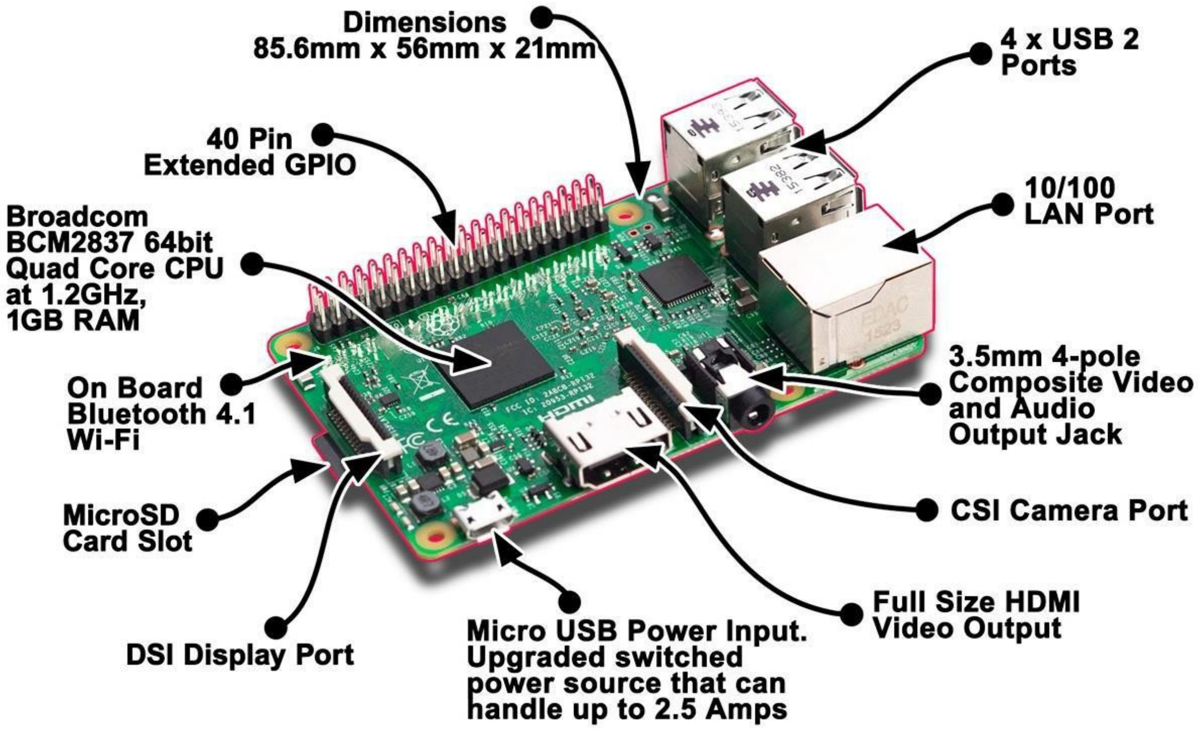

Dimensions 85.6mm x 56mm x 21mm
4 x USB 2 Ports
40 Pin Extended GPIO
10/100 LAN Port
Broadcom BCM2837 64bit Quad Core CPU at 1.2GHz, 1GB RAM
3.5mm 4-pole Composite Video and Audio Output Jack
On Board Bluetooth 4.1 Wi-Fi
CSI Camera Port
MicroSD Card Slot
Full Size HDMI Video Output
DSI Display Port
Micro USB Power Input. Upgraded switched power source that can handle up to 2.5 Amps

Fig 5.1: RASPBERRY PI SPECIFICATIONS

5.1.1.2 Processor

The Raspberry Pi 3 uses a Broadcom BCM2837 system on chip with a 1.2 GHz 64-bit quad-core ARM Cortex-A53 processor, with 512 KB shared L2 cache. [12]

5.1.1.3 RAM

The Raspberry Pi 2 and the Raspberry Pi 3 have 1 GB of RAM. The Raspberry Pi Zero has 512 MB of RAM. [12]

5.1.1.4 NETWORKING

On the Model B and B+ the Ethernet port is provided by a built-in USB Ethernet adapter using the SMSC LAN9514 chip. The Raspberry Pi 3 is equipped with 2.4 GHz WiFi 802.11n (150 Mbit/s) and Bluetooth 4.1 (24 Mbit/s) in addition to the 10/100 Ethernet port.[12]

5.1.1.5 PERIPHERALS

The Raspberry Pi may be operated with any generic USB computer keyboard and mouse. [12]

5.1.1.6 VIDEO

The video controller can emit standard modern TV resolutions, such as HD and Full HD, and higher or lower monitor resolutions and older standard CRT TV resolutions. As shipped (i.e., without custom overclocking) it can emit these: 640×350 EGA; 640×480 VGA; 800×600 SVGA; 1024×768 XGA; 1280×720 720p HDTV; 1280×768 WXGA variant; 1280×800 WXGA variant; 1280×1024 SXGA; 1366×768 WXGA variant; 1400×1050 SXGA+; 1600×1200 UXGA; 1680×1050 WXGA+; 1920×1080 1080p HDTV; 1920×1200 WUXGA.[12]General purpose input-output (GPIO) connector

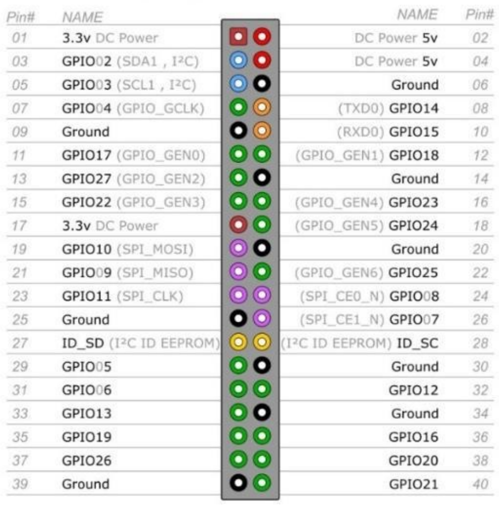


Fig 5.2: General purpose input-output

5.1.1.7 POWER SUPPLY

The Raspberry Pi 3 is powered by a +5.1V micro USB supply. Typically, the model B uses between 700-1000mA depending on what peripherals are connected. The maximum power the Raspberry Pi can use is 1 Amp. The power requirements of the Raspberry Pi increase as more interfaces are used on the Raspberry Pi. The GPIO pins can draw 50mA safely, distributed across all the pins; an individual GPIO pin can only safely draw 16mA. The HDMI port uses 50mA, the camera module requires 250mA, and keyboards and mice can take as little as 100mA.[13]

5.1.1.8 OPERATING SYSTEM

The Raspberry Pi primarily uses Raspbian, a Debian-based Linux operating system. Other third party operating systems available via the official website include Android Things(by Google),Ubuntu MATE, Snappy Ubuntu Core, Windows 10 IoT Core and RISC OS. Many other operating systems can also run on the Raspberry Pi.[13]

5.2 SOFTWARE REQUIREMENTS

5.2.1 RASPBIAN OS

Raspbian is a free operating system based on Debian optimized for the Raspberry Pi hardware. An operating system is the set of basic programs and utilities that make the Raspberry Pi run. However, Raspbian provides more than a pure OS: it comes with over 35,000 packages, pre-compiled software bundled in a format for easy installation on the Raspberry Pi. [13] Raspbian is highly optimized for the Raspberry Pi line's low-performance ARM CPUs. Raspbian uses PIXEL, Pi Improved Xwindows Environment, and Lightweight as its main desktop environment.

5.2.1 MOSQUITTO BROKER

Mosquitto is an open source message broker that implements the MQ Telemetry Transport (MQTT) protocol. A broker in MQTT handles receiving published messages and sending them on to any clients who have subscribed.

5.2.1.1 SCOPE

The Mosquitto project provides a small server implementation of the MQTT and MQTT-SN protocols. Small means that: only necessary function is included (conditional compilation can be used to be able to omit unneeded function for a particular application), that function is coded as efficiently as possible, the externals are as simple as possible for the function provided. The server has the following features, which are not described in the MQTT specification:

An MQTT bridge, to allow Mosquitto to connect to other MQTT servers.

The ability to secure communications using SSL/TLS.

User authorization - the ability to restrict user access to MQTT topics.[11]

5.2.1.2 DESCRIPTION

Mosquitto provides a lightweight server implementation of the MQTT and MQTT-SN protocols, written in C. The reason for writing it in C is to enable the server to run on machines which do not even have capacity for running a JVM. Sensors and actuators, which are often the sources and destinations of MQTT and MQTT-SN messages, can be very small and lacking in power. This also applies to the embedded machines to which they are connected, which is where Mosquitto could be run.

As well as accepting connections from MQTT client applications, Mosquitto has a bridge which allows it to connect to other MQTT servers, including other Mosquitto instances. This allows networks of MQTT servers to be constructed, passing MQTT messages from any location in the network to any other, depending on the configuration of the bridges. [11]

5.2.3 APACHE TOMCAT SERVER

Apache Tomcat, often referred to as Tomcat Server, is an open-source Java Servlet Container developed by the Apache Software Foundation (ASF). Tomcat implements several Java EE specifications including Java Servlet, JavaServer Pages (JSP), Java EL, and WebSocket, and provides a "pure Java" HTTP web server environment in which Java code can run. It basically make our Java Web applications to run on host and server based system and it is configured on local host port 8080.

There is a built in web container called Catalina in the tomcat bin directory. It loads all http related request and has privilege to instantiate the GET and POST method's object.

It also uses cynote I.e an http connector through network layer of the computer. All the execution is managed by JSP engine.

5.3 IMPLEMENTATIONN DETAILS

5.3.1 USER WEB-APP

Here is the view of the user app of Dexter which the user might see the first time he uses this system. It gives the user the option to choose what the user wants to send to the remote screen, which can either be images or calendar events.

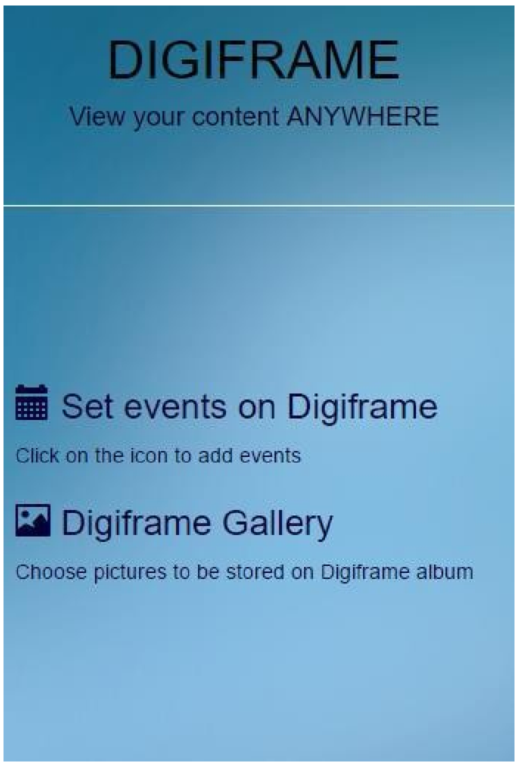

DIGIFRAME
View your content ANYWHERE
Set events on Digiframe
Click on the icon to add events
Digiframe Gallery
Choose pictures to be stored on Digiframe album

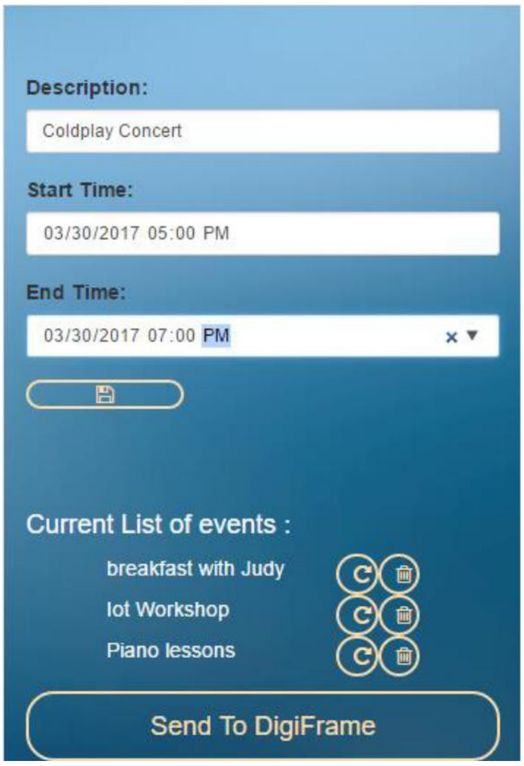
Fig 5.3 USER WEB-APP

5.3.2 CALENDAR EVENTS

The user can create calendar events by specifying 3 parameters that is event description, start time of the event and the end time of the event. The user can view the existing events right below the event creation interface. The user can update or delete existing events and once satisfied he can hit the send button.

In the backend as the REST architecture is used, the calendar event is handled as a resource which is a plain java object. The Create, read, update and delete operations are performed on this resource using various http methods. The resource class consumes and produces the data whereas the service class performs the actual operation on the resource.

The event resource is represented in json format in order to be used during communication. When the user hits the send button, this json representation is converted into bytes and it is published via MQTT protocol.



Description:
Coldplay Concert

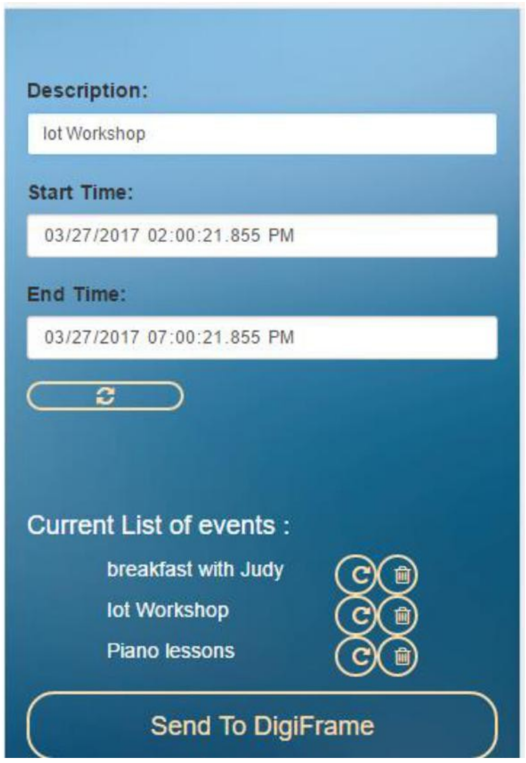
Start Time:
03/30/2017 05:00 PM

End Time:
03/30/2017 07:00 PM

Current List of events :
breakfast with Judy
Iot Workshop
Piano lessons

Send To DigiFrame

Fig 5.4: CREATING A NEW EVENT



Description:
Iot Workshop

Start Time:
03/27/2017 02:00:21.855 PM

End Time:
03/27/2017 07:00:21.855 PM

Current List of events :
breakfast with Judy
Iot Workshop
Piano lessons

Send To DigiFrame

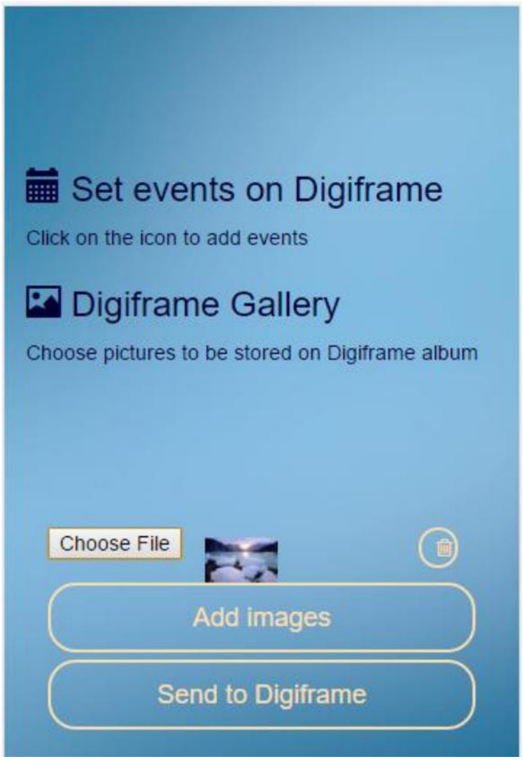
Fig 5.5: UPDATING AN EXISTING EVENT

5.3.3 IMAGES

The user can select one or more images to be displayed on the display screen. User can add as many images as needed also the images can be deleted before sending if required. When the user is satisfied with the image input, he can hit the send button.

The images are then handed over to a java servlet which responsible to convert the images into bytes and publish them via MQTT protocol.

It is important to convert the data to be published into bytes because it needs to be set as the payload for MQTT message and MQTT requires it payload to be in bytes only.



Set events on Digiframe
Click on the icon to add events
Digiframe Gallery
Choose pictures to be stored on Digiframe album
Choose File
Add images
Send to Digiframe

Fig 5.6: ADDING ONE IMAGE

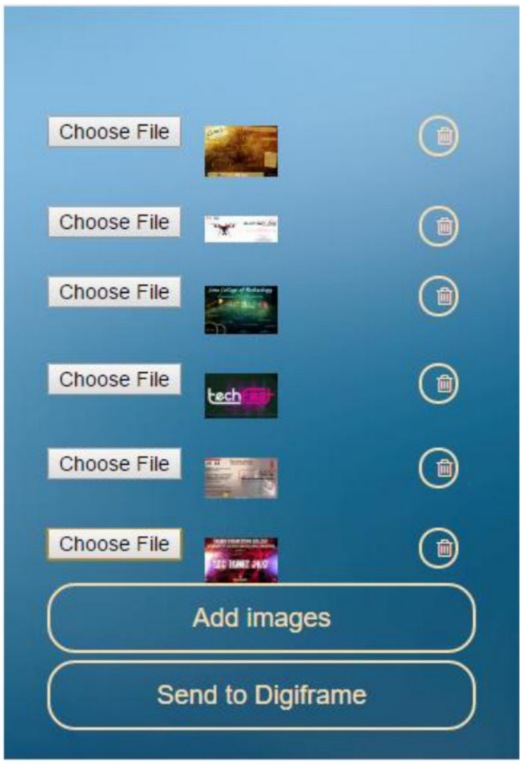


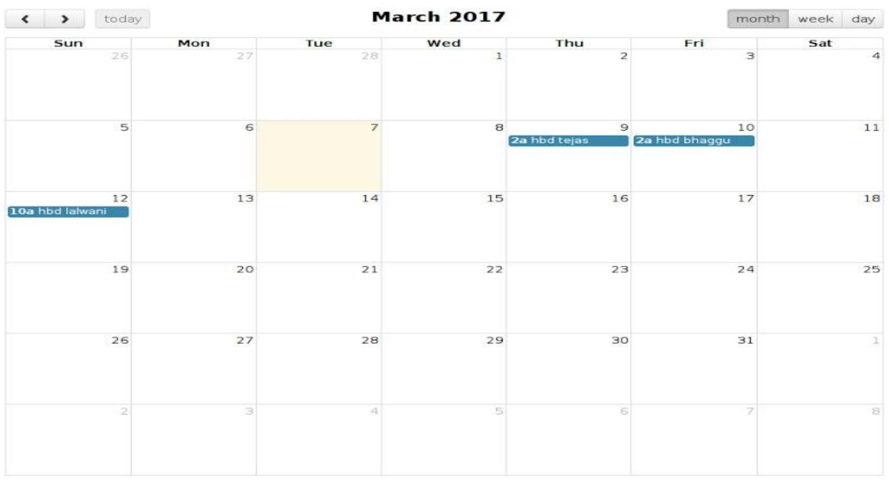
Fig 5.7: ADDING MULTIPLE IMAGES

5.3.4 RASPBERRY PI AND REMOTE DISPLAY DEVICE

The mosquitto (MQTT broker) resides in the raspberry pi which consumes the data sent by the publisher under a topic (image/calendar) in the form of byte array which is payload of MQTT messages and displays it on remote display device.

5.3.4.1 CALENDAR EVENTS

The publisher publishes the calendar events in the form of JSON object, which is in turn converted into byte array. This byte array is then set as payload of the MQTT message and then sent to the mosquitto broker by the publisher. The subscribe extracts the payload of the MQTT message which is a JSON object (calendar events) encoded as byte array. This JSON object is stored in a JSON file which is read by the javascript file present on raspberry pi. After the data is processed by the javascript program it is displayed on the remote display device.



Sun	Mon	Tue	Wed	Thu	Fri	Sat
26	27	28	1	2	3	4
5	6	7	8	9 2a hbd tejas	10 2a hbd bhaggu	11
12 10a hbd lalwani	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1
2	3	4	5	6	7	8

FIG 5.8: CALENDAR VIEW

5.3.4.1 IMAGES

The publisher reads the image(s) uploaded in the form of Buffered image which is converted into byte array. . This byte array is then set as payload of the MQTT message and then sent to the mosquitto broker by the publisher. The subscribe extracts the payload of the MQTT message which is a byte array.

This byte array is converted into Buffered image, stored on the raspberry pi and then displayed on remote display device.

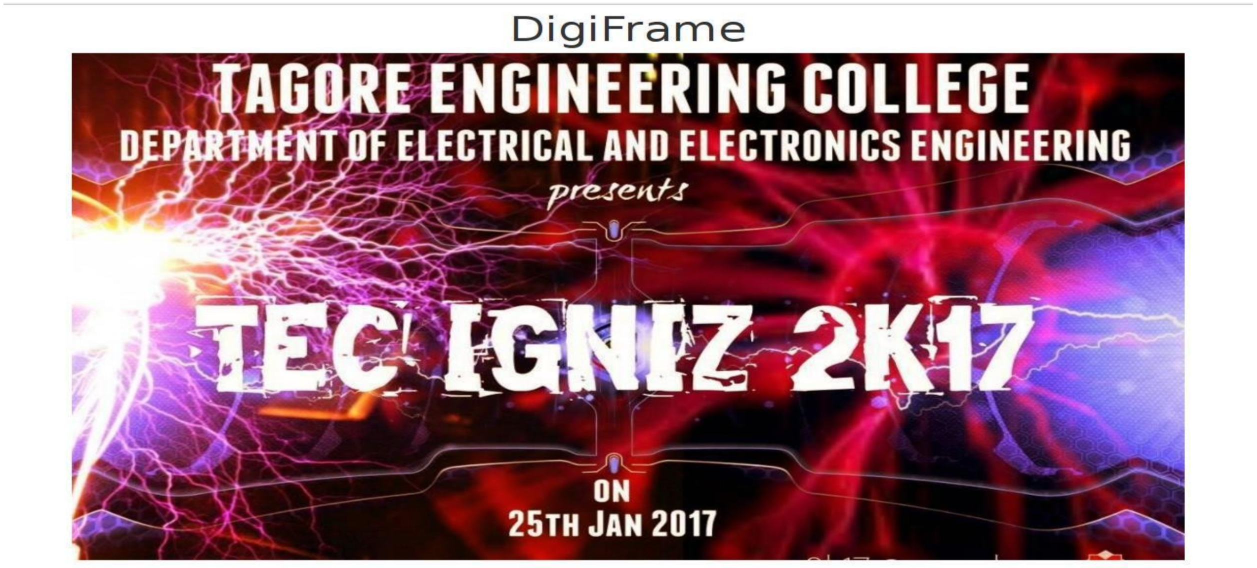

DigiFrame
TAGORE ENGINEERING COLLEGE
DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING
presents
TEC IGNIZ 2K17
ON
25TH JAN 2017

Fig 5.9: IMAGE VIEW

5.4 METHODS IMPLEMENTED

5.4.1 ECLIPSE PAHO JAVA CLIENT

The Paho Java Client is an MQTT client library written in Java for developing applications that run on the JVM or other Java compatible platforms such as Android.

The Paho Java Client provides two APIs: MqttAsyncClient provides a fully asychronous API where completion of activities is notified via registered callbacks. MqttClient is a synchronous wrapper around MqttAsyncClient where functions appear synchronous to the application. [10]

org.eclipse.paho.client.mqttv3 Description

The package contains a programming interface enabling applications to communicate with an MQTT server.

The basic means of operating the client is:

1. Create an instance of **MqttClient** or **MqttAsyncClient**, providing the address of an MQTT server and a unique client identifier
2. Connect to the server.
3. Exchange messages with the server:

   Publish messages to the server specifying a topic as the destination on the server

   Subscribe to one more topics. The server will send any messages it receives on those topics to the client. The client will be informed when a message arrives via a callback
4. disconnect from the server.[5]

The programming model and key concepts:

Every client instance that connects to an MQTT server must have a unique client identifier. If a second instance of a client with the same ID connects to a server the first instance will be disconnected.

When subscribing for messages the subscription can be for an absolute topic or a wildcarded topic.

When unsubscribing the topic to be unsubscribed must match one specified on an earlier subscribe.

There are two MQTT client libraries to choose from:

- **MqttAsyncClient** which provides a non-blocking interface where methods return before the requested operation has completed. The completion of the operation can be monitored by in several ways:
    - Use the **waitForCompletion** call on the token returned from the operation. This will block until the operation completes.
    - Pass a **IMqttActionListener** to the operation. The listener will then be called back when the operation completes.
    - Set a **MqttCallback** on the client. It will be notified when a message arrives, a message have been delivered to the server and when the connection to the server is lost.
- **MqttClient** where methods block until the operation has completed.

For both the blocking and non-blocking clients some operations are asynchronous. This includes:

- Notification that a new message has arrived: **messageArrived**.
- Notification that the connection to the server has broken: **connectionLost**.
- Notification that a message has been delivered to the server**: deliveryComplete**.
- A client registers interest in these notifications by registering a **MqttCallback** on the client.

MqttConnectOptions can be used to override the default connection options. This includes:

- Setting the cleansession flag
- Specifying a list of MQTT servers that the client can attempt to connect to
- Set a keepalive interval
- Setting the last will and testament
- Setting security credentials.[5]

TESTING AND RESULTS

6.1 AIM OF TESTING

The main aim of testing is to analyze the performance and to evaluate the errors that occur while the program is executed with different input sources and running in different operating environment the meaning of testing in this project is to find if it works well with different shapes and sizes of inputs while publishing as well as well subscribing the data

6.2 ARTIFACTS OF TESTING

For testing the application the testing process produces several Artifacts the different Artifacts are

6.2.1 TEST PLAN

Test plan gives us the process of testing the subject of application called as the test process the developers executed test plan and the results are used for the purpose of Management and future development.

6.6.2 TRACEABILITY MATRIX

Traceability matrix is a table that links design documents to the text documents. This also changes the test processes when the source documents are changed.

6.2.3 TEST CASE

A test case consists of unique identifier identifies the requirements of the project from the design phase have information about the series of steps like input output expected result and actual result the series of steps are stored in the text document or Excel spreadsheet.

6.2.4 TEST SUITE

Test Suite is a collection of test cases the test case Suite answers of the detailed instruction and the role of each collection of test cases

6.2.5 TEST DATA

The place where the test values and components can be modified is known as test data

6.2.6 TEST HARNESS

The test harness is a collection of hardware software input output and configuration used for the application

6.3 TYPES OF TESTING USED

There are two types of testing basically Alpha testing and beta testing

6.3.1 ALPHA TESTING

Alpha testing performed by testers at developers site it involves both white and black box testing mostly critical issues of excess can be addressed immediately in an Alpha testing

6.3.2 BETA TESTING

Beta testing is performed by Client or end users at client location it uses only black box testing beta testing will be implemented in future versions of the product by using the end user's feedback

We have used Alpha testing for testing for evaluating our project

6.4 LEVELS OF TESTING

6.4.1 UNIT TESTING

This level of testing has been used in our project where our concentration was on each component of the software has implemented in the source code

6.4.2 INTEGRATION TESTING

This type of testing focuses on design and construction of the software architecture and it’s all components and checked the working this we used at the time of design and implementation

6.4.3 VALIDATION TESTING

This type of testing provides final Assurance that the software meets all functional behavioral and performance requirements.

6.4.4 SYSTEM TESTING

Verify the software and other system and events as a whole

6.5 TYPES OF TESTING

Black box testing is the testing approach which tells about the possible combinations of the end user action black box testing doesn't need the knowledge about interior connections or programming course in the black box testing the end user's test the application by giving different sources and checks whether the output for the specified input is appropriate or not

White box testing is also known as glass box or clean box or open box testing it is opposite to the black box testing and whitebox testing we can create test cases by checking the court and executing in certain intervals and know the potential errors the analysis of the code can be done by giving suitable inputs for specified applications and using the source code for application blocks the white box testing applies to unit testing system testing and integration testing these are approaches are used in final implementation and validation of our project basically white box testing was used in our project that is Dexter where we created our test cases and check the code at each interval

6.6 TEST CASES

6.6.1 TEST CASE: CALENDAR MODULE

In the calendar module, when we need to add any event we need to add description to the event, the start time of the event and the end time of the event.

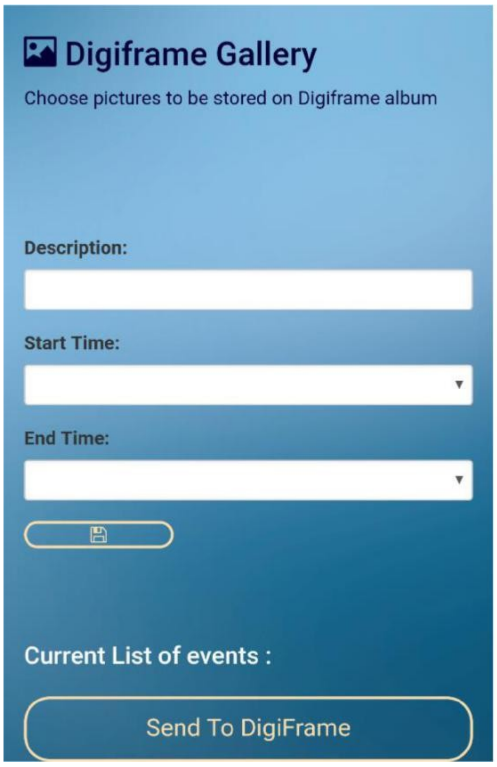
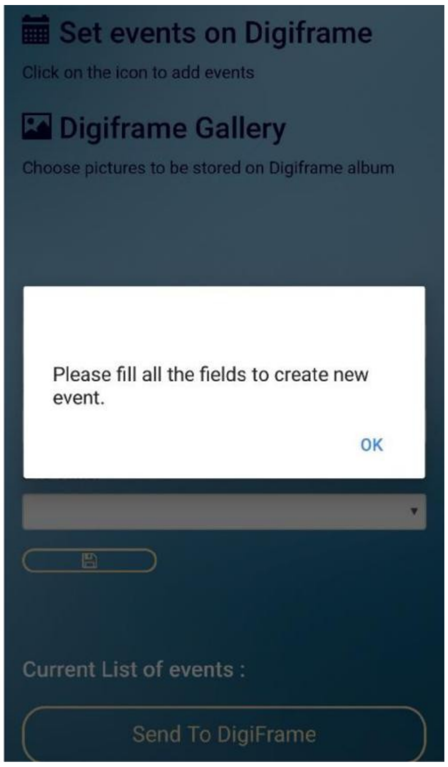


Fig 6.1: NO CALENDAR DATA



Please fill all the fields to create new event.

Fig 6.2: CALENDAR ALERT

6.6.1.1 EXPECTED RESULT

The user must be informed that he or she has left the some information unfiled.

6.6.1.2 ACTUAL RESULT

If any information is left blank then an alert is generated indicating that the field is left blank.

6.6.2 TEST CASE: IMAGE MODULE

In the image module the user need to select an image file to be sent to the remote display device, a user may select one or more images.

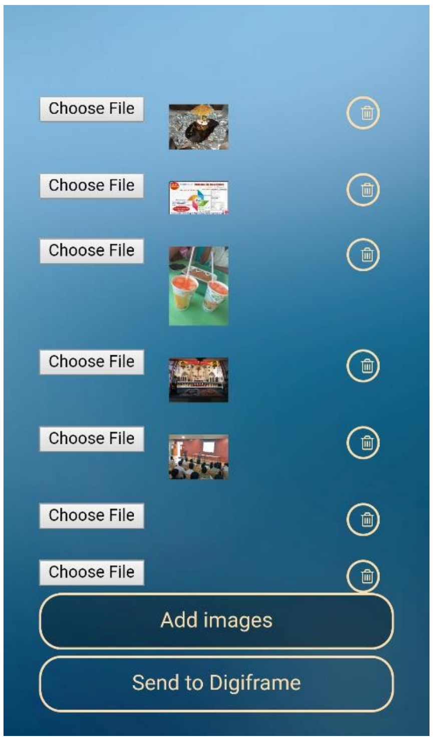


Fig 6.3: MULTIPLE IMAGES

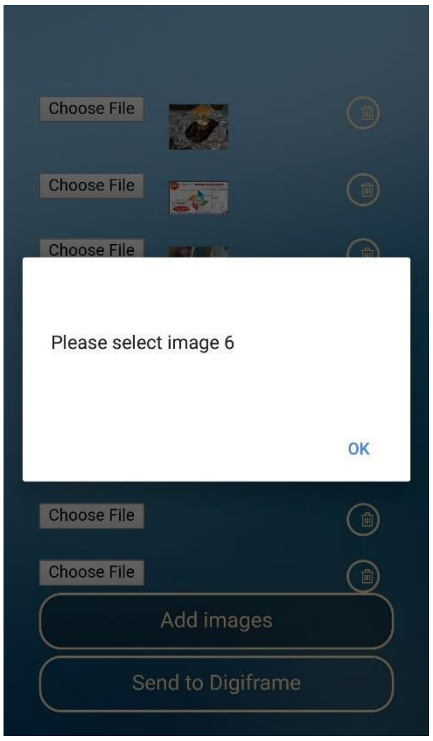

Please select image 6

Fig 6.4: IMAGE ALERT

6.6.2.1 EXPECTED RESULT

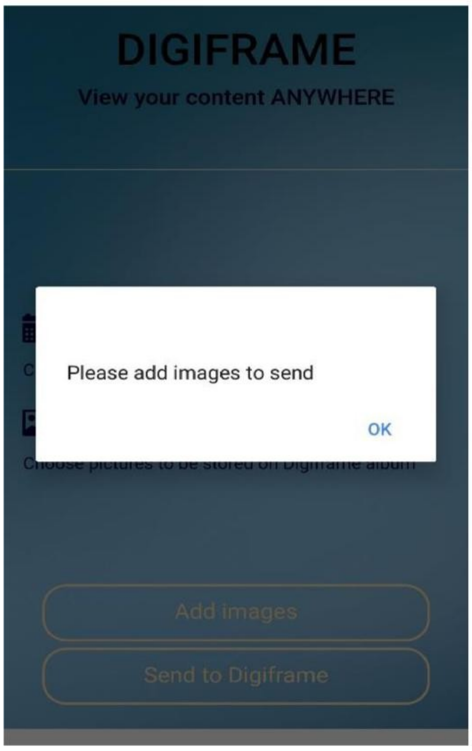
The user must be informed that he or she has not selected any image.

6.6.2.2 ACTUAL RESULT

If the image is not selected then an alert is generated to select an image.

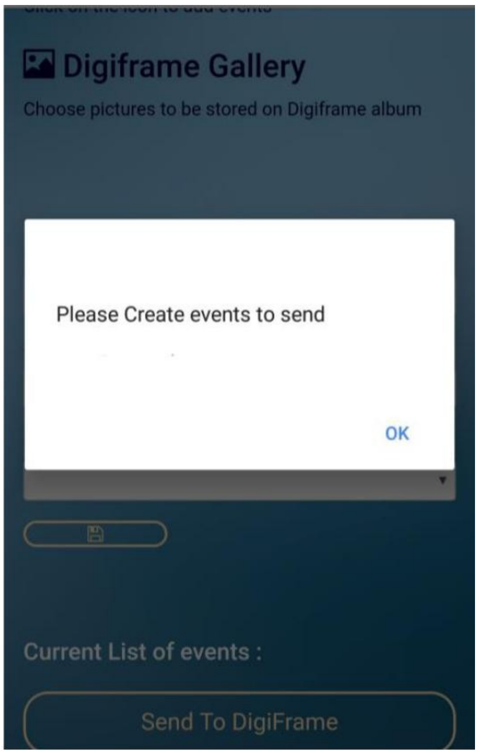
6.6.3 TEST CASE: SENDING DATA

When data is sent to remote display device, the input from the user must be checked whether the data is present or not.


DIGIFRAME
View your content ANYWHERE
Choose pictures to be stored on Digiframe album
Add images
Send to Digiframe

Please add images to send

OK



Please Create events to send

OK

Fig 6.5 SENDING ALERT1 Fig 6.6 SENDING ALERT2

6.6.3.1 EXPECTED RESULT

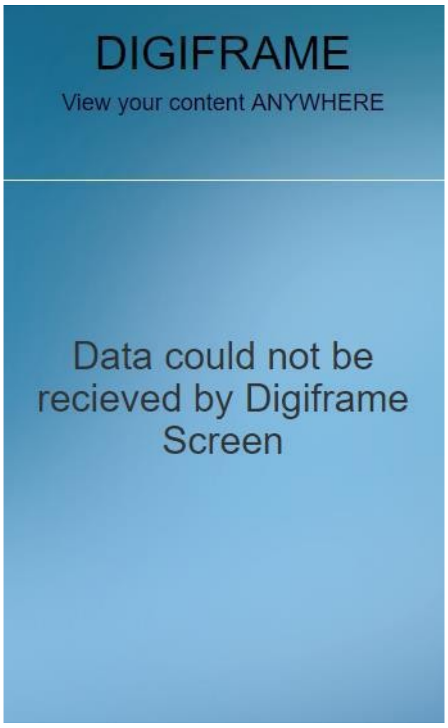
The user must be informed that he or she has not selected any data to send.

6.6.3.2 ACTUAL RESULT

If the data is not present then an alert is generated which tells the user to input the data.

6.6.4 TEST CASE: PUBLISHING DATA

When the user wants to publish the data to remote display device, the raspberry pi must be active (connected to broker). If the receiver (raspberry pi) is offline or it can’t receive data then a webpage is opened which tells remote display device can’t receive data.


DIGIFRAME

View your content ANYWHERE

Data could not be recieved by Digiframe Screen

Fig 6.7 PUBLISHING ALERT

6.6.4.1 EXPECTED RESULT

The user must be informed that the receiver (raspberry pi)) is offline or it can’t receive data.

6.6.4.2 ACTUAL RESULT

If the receiver (raspberry pi) is offline or it can’t receive data then a webpage is opened which tells remote display device can’t receive data.

.

CONCLUSION AND FUTURE SCOPE

7.1 CONCLUSION

DIGIFRAME helps the user to be updated and efficient as:

DIGIFRAME gives the user information at the right time and at the right place. The user is able to control a remote display device with a Hybrid App.

The user can get real-time updates of the data.

The content can be viewed uniformly on any display device.

7.2 FUTURE SCOPE

1. Multiuser support

   A login id and password will be provided to the user. Which would be needed to access the mobile app, this would add security features to DIGIFRAME as well as multiple users can access as unauthenticated user can’t use the hybrid app.

2. Various file type support

   Currently for image module, only JPEG format is supported. Images come in various format that is PNG, SVG, BMP, PSD, etc., to display these types of images provisions could be made.

3. Preview before sending data

   The actual display of data on remote display device is decided by the code written inside raspberry pi, which can only be scene once the data is sent to the raspberry pi. We can add the functionality to get a preview of data that would be displayed on the remote display device.

REFERENCES

[1]	www.academia.edu/download/45881427/Internet_of_Things_A_Survey_on_Enabling_Technologies__Protocols__and_Applications.pdf
[2]	http://www.cisco.com/c/dam/en_us/solutions/industries/docs/gov/everything-for-cities.pdf
[3]	https://www.slideshare.net/GaneshGani5/wireless-e-notice-board
[4]	https://www.slideshare.net/Brijenderk/electronic-notice-board-using-raspberry-pi-and-android-phone
[5]	http://www.eclipse.org/paho/files/javadoc/index.html
[6]	http://www.hivemq.com/blog/mqtt-essentials-part2-publish-subscribe
[7]	http://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment
[8]	http://programmingwithreason.com/article-mqtt-in-depth.html
[9]	http://www.cs.wustl.edu/~jain/cse570-15/ftp/iot_prot/index.html#MQTT
[10]	https://eclipse.org/paho/clients/java/
[11]	https://www.eclipse.org/proposals/technology.mosquitto/
[12]	https://en.wikipedia.org/wiki/Raspberry_Pi
[13]	https://www.raspberrypi.org/documentation
[14]	http://rest.elkstein.org/

PROJECT MEMBERS AND GUIDE INFORMATION

PROJECT TEAM MEMBERS

SR. NO.	NAME OF STUDENT	CONTACT NO.	EMAIL-ID
1	K. BHAGYASHREE RAO	9561471259	kbhagyashree.rao@gmail.com
2	TEJAS WAJE	8888444854	wajetejas@gmail.com
3	VAISHNAVI DHOKE	8805612838	vaishnavidhoke@gmail.com
4	ADITYA LALWANI	9960272263	adityalalwani05@gmail.com

PROJECT GUIDE

NAME	CONTACT NO.	E-MAIL ID
PROF. VAIBHAV DESHPANDE	+919890977613	vabartday@gmail.com

PROJECT MENTOR

NAME	CONTACT NO.	E-MAIL ID
RAHUL DHOTE	+919096569507	r.rahuldhote@gmail.com