**Chapter 1**

**Introduction**

Digital display board is a common sight today. Advertisement is going digital in recent days. The use of digital display boards at railway station, bus stands, shopping malls, educational institutions and public places are becoming an effective mode of communication in providing information to the people. Notice Board is used in various institutes to display notices and these boards are managed manually. It is a long process to put up notices on the notice board. This wastes a lot of resources like paper, printer ink, man power and also loss of time and these off-the-shelf units are somewhat inflexible in terms of updating the message instantly. If the user wants to change the message it needs to be done using a computer and hence the person needs to be present at the location of the display board. It means message cannot be changed from wherever or whenever. Also, the display board cannot be placed anywhere because of complex and delicate wiring.

Digital notice board using IoT overcomes these drawbacks. The main objective is to design an automatic, self-enabled highly reliable electronic notice board. In this project we have proposed a system which will enable people to wirelessly transmit notices on notice board using Wi-Fi. Here we have proposed a system by which only authorized person can accesses the notice board. It requires less time due to fast data transmission through Wi-Fi. Less cost and save the resources like paper. A display connected to a server system should continuously listen for the incoming messages from user, process it and display it on LCD screen. Message displayed should be updated every time the user sends new information. Only authenticated people should update the data to be displayed on the monitor. Digital smart board is an automatic self-enabled, highly reliable wireless electronic board. The information can be text, images, audio and video. It makes use of a mini computer, which is commonly termed as Raspberry Pi.

**1.1 Related Work**

Several works have used GSM to monitor several applications such as cell phone operated robot, SMS based voting system, SMS based security system, GSM based automatic meter reading system using ARM controller, SMS based teaching and learning system and so on.

Also, several researches have been done on GSM based e-notice board, where, SMS sent from authorized mobile phones, via a GSM network, were displayed on a digital e-notice board. These several works have proven to be efficient and fast. With greater efficiency, messages were displayed with less error and less maintenance, though there is need for few modifications for better performance. In 2013 an SMS driven automatic display using ARM-LPC2148 to interface multiple graphical display was designed. With this technology, a single notice could be sent to several e-notice boards via ARM-LPC2148. Again in 2013 a GSM based multiple LED display boards using AT89S52 microcontroller, GSM module, LCD and several moving LED displays was designed and developed. Multiple moving LED displays were connected via different GSM modules at different geographical locations such that the same SMS sent was displayed on all the moving LED displays. Though with few limitations, this work proved to be cost-effective, secured and efficient as compared to previous works. In 2014 development of GSM based digital notice board was proposed.

The complete system would have a dual system in terms of changing message display, dual power supply switchable between solar power system and alternating current (AC) from the utility supply, and inbuilt motion detector that could automatically switch OFF the whole system after working hours and would automatically switch ON if any motion is sensed by the motion detector after the programmed working hours. This work would probably prove highly efficient in terms of ensuring better communication and continuous power supply.

**1.2 Proposed System**

This project, aims to increase the usability of electronic notice boards, deals with wireless reception and display of messages using Raspberry pi. Practically, all output resolutions are supported. It aims at designing a LED Monitor based message display controlled through a laptop. It is simple, easy to install, user-friendly system which can receive and display information in a particular manner with respect to date and time which will help the user to easily keep the track of information every day and each time he uses the system.

**Chapter 2**

**DETAILED LITERATURE SURVEY**

**2.1 Large Screen Wireless Notice Display System. [June 2015]**

**Authors: Yash Teckchandani, G. Siva Perumal, Radhika Mujumdar, Sridhar Lokanathan.**

Wireless electronic notice boards are a faster alternative to conventional pin-up type notice boards. A major constraint of the methods used so far is the small size of the 16x2 Liquid Crystal Displays (LCD) used to display the notices. This paper proposes a method in which large screens like computer monitors or televisions can be used for displaying notices sent as text messages from a mobile phone. The proposed method uses Hypertext Markup Language (HTML) to present the output since it offers many customization options. To give high resolution output, the credit-card sized computer Raspberry pi has been used. This HTML output can be displayed by a web browser running on the Raspberry pi. The notice to be displayed is sent as a Short Message Service (SMS), which is received by a Global System for Mobile Communications (GSM) modem, making the reception of the message wireless. The GSM modem is polled at regular intervals by the Raspberry pi to display the latest messages.

**Methodologies Used**

* Since GSM network is being used, the notice can be sent from practically any location on the globe and it will be displayed on the screen.
* The Global System for Mobile Communications (GSM) network is digital. This makes it immune to noise. Also, GSM networks are relatively free of errors.

**Drawback of the proposed system**

* Use of GSM modem limits the application to text messages.
* The use of GSM limits the range of connectivity to the remote location.

**2.2 GSM Based Wireless Electronic Notice Board Display through ARM7 and LED. [May 2016]**

**Author:** **P. Sampath Kumar, V. Priyanka, Lakshmi Surekha, Y. Harish Reddy.**

This paper is developed a GSM based notice board display using ARM7 controller along with LED array. The microcontrollers provide all the functionality of the display notices and wireless control. The Display is obtained on a 7X96 Light Emitting Diode (LED) dot matrix display. A desired text message from a mobile phone is sent via a Global System for Mobile Communication (GSM) to the GSM module located at the receiving end. The GSM modem is connected, through MAX 232 Integrated Circuit (MAX 32 IC), to the ARM7 microcontroller. The message that is stored in the Electrically Erasable Programmable Read Only Memory (EEPROM) is then displayed on the LED dot matrix display. This hardware uses regulated 5V, 500mA power supply. A three-terminal LM7805 is employed for regulation of the voltage. A bridge type full-wave rectifier is used to rectify the AC output of the secondary of 230/12V step down transformer. The system was tested to work according to specification.

**Methodologies Used**

* In this project we have proposed a system which will enable people to wirelessly transmit notices on a notice board using GSM.
* The project mainly focuses on transmission of textual data through air interface by the use of GSM through asynchronous serial communication.

**Drawback of the proposed system**

* The SMS is deleted from the SIM each time it is read, thus making room for the next SMS.
* The major constraints incorporated are the use of “\*” as the termination character of the SMS and the display of one SMS as a time.

**2.3 Wireless Electronic Notice Board Using Raspberry Pi 3. [June 2017]**

**Author:** **Er. G. Jalalu, Er. Polepogu Rajesh.**

Notice boards can change the way communication with each other, using notice boards is a constructive method of promoting important information to a large number of people. Notice board is ideally useful tool for organizing and displaying information, these are used in multitude of businesses such as schools, colleges, hospitals, railway station, bus station, hotels, shopping malls etc. As they can be used over and over again to display important notices or advertise forthcoming events or meeting. In this paper, we proposed an advanced wireless notice board in which at any time we can add or remove or alter the message according to our requirement. The main aim of this proposed project is to drastically reduce the cost involved, consume smaller amount of power and help in achieving quality of service. For this we need a computer/laptop as a transmitter, Raspberry PI 3 model B as a receiver, Wi-Fi for data transmission and a LED/LCD screen as a display.

**Methodologies Used**

* We have used the laptop as transmitter to send the notices and Raspberry Pi 3 model is used as receiver.
* When both the transmitter and receiver are connected to the same network, then the notices are displayed on the monitor.
* They are displayed one after the other after 5 seconds time gap.
* The software which is used in this project is FileZilla.

**Drawback of the proposed system**

* The proposed system limits the client and server to be confined to a single network.
* The use of Transmitter and Receiver limits the range of connectivity to the remote location.

**2.4** **An IOT Based Web Page Controlled Digital Notice Board. [August 2017]**

**Author: P. Bhaskara Chary, Dr. T. Srinivasulu.**

GSM – an advanced versatile communication framework, which is internationally gotten to by nearly 212 nations and domains. Worldwide framework for versatile work is totally upgraded for full duplex voice communication. At first produced for the substitution of real (1G) innovation, now GSM is accessible with heaps of hitting highlights with the consistent up degree of third era (3G) innovation Likewise, in trains and transports the data like stage number, ticket data is shown in computerized loads up. Individuals are presently adjusted to the possibility of the world readily available. The utilization cell phones have expanded definitely finished years. Control and correspondence have turned out to be imperative in every one of the parts of the world. This undertaking is a Web Controlled notice board with a GSM modem at the recipient’s end. So, if the client needs to show any message, he can send the data by Web server (Thingspeak.com) and hence refresh the LCD show appropriately.

**Methodologies Used**

* Receiver section contains power supply, microcontroller, LCD display and GSM800L GSM module. Initially the GSM800L module programmed with AT commands and microcontroller is programmed by using embedded C language in KEIL software. A sim is inserted into the GSM module, once the power supply on AT commands executes one by one.
* The GSM800L module will download the data from the web server and transmitted to the microcontroller block. Now the microcontroller fetches and executes the information from the GSM module and displays in the LCD.

**Drawback of the proposed system**

* This proposed system uses the very old and slow technology i.e. C language in KEIL software.

**2.5** **IoT based web-controlled notice board. [April 2018]**

**Author:** **Divyashree M, Harinag Prasad S, Sandeep G T, Bhavya S N, Poornima S.**

IoT is the network of physical “things” or object that contain embedded technology to interface and sense to move with their internal states or the external setting. Automation is the most often spelled term within the field of electronics. The hunger for automation brought several revolutions within the existing technologies. Notice board could be a primary factor in any establishment or public places like bus stations, railway stations, colleges, malls etc. Sticking out numerous notices day to day could be a tough method. A separate person is needed to take care of this notice display. This project is regarding advanced wireless notice board. In IoT based Web Controlled Notice Board, Internet is employed to wirelessly send the message from Browser to the liquid crystal display. A local web server is created, this could be a global server over net. At the Raspberry Pi, LCD is used to display message and flask for receiving the message over network. Whenever Raspberry receives any wireless message from Web browser, it displays on the liquid crystal display.

**Methodologies Used**

* Client: Authorized user.
* Server: Raspberry pi.
* Raspberry pi interfaces with router using a Wi-Fi adapter.
* Users enters SSID (router name) a password of router.
* Routers allot IP address to raspberry pi.
* TCP server is made on raspberry pi which listens for incoming calls.
* A TCP client is made on PC which interfaces with TCP server.
* When a connection is established the client sends message to server the message sent to by the client is stored in a text file on raspberry pi and hard disk (SD card).
* The text file is prepared by another program which displays the text on LCD screen connected on HDMI interface.

**Drawback of the proposed system**

* Remote activities allow administrations, for example, long-go interchanges, that are inconceivable or illogical to execute with the utilization of wires
* Electronic Notice Board is one of the applications where WIFI and Raspberry pi can be utilized successfully but this system is limited to the specific uses.

**2.6 Disadvantages of the Existing System.**

* Use of GSM modem limits the application to text messages.
* The use of GSM limits the range of connectivity to the remote location.
* The SMS is deleted from the SIM each time it is read, thus making room for the next SMS.
* The major constraints incorporated are the use of „\*‟ as the termination character of the SMS and the display of one SMS as a time.
* The proposed system limits the client and server to be confined to a single network.
* The use of Transmitter and Receiver limits the range of connectivity to the remote location.
* This proposed system uses the very old and slow technology i.e. C language in KEIL software.

**Chapter 3**

**OBJECTIVES AND METHODOLOGY OF THE PROPOSED SYSTEM**

**3.1 Proposed system**

The proposed system will be a moving message display, which might be utilized as the digital notice board, and moreover a Wi-Fi transceiver, that will be that the most recent innovation utilized for communication between the mobile and the embedded devices. System can work like once the user desires to display or update the notice board that is unimaginably useful to show the circulars, day by day occasions, plans are to be shown.

At that point the WI-FI will receive the message in notice board system, the Raspberry Pi chip has been inside the system is programmed in such a way that when the coding is written in embedded system Language receives any message it will browse the message form serial port through WI-FI transceiver, if the message is writing in any PC then it will begin displaying the information within the display system.

The messages are displayed on the liquid crystal display. This system is to cut back the time wastage and update with any time is to terribly simply. The serial WI-FI has been utilized it can be used to transmit an information from serial port communication. It implies that to display the information from to a tiny bit at a time to get the notice load up then stores it, messages are then shows it in the LCD/LED module.

**3.2 Objectives of the proposed system**

The main objective is to design an automatic, self-enabled highly reliable electronic notice board. A display connected to a server system should continuously listen for the incoming messages from user, process it and display it on LCD screen.

To develop a wireless notice board that display message sent from the user and to design a simple, easy to install, user friendly system, which can receive and display notice in a particular manner with respect to date and time which will help the user to easily keep the track of notice board every day and each time he uses the system. Wi-Fi is the wireless technology used. Message displayed should be updated every time the user sends new information. Only authenticated people should update the data to be displayed on the monitor.

**3.3 Advantages of the proposed system**

* Data is more secure.
* It provides faster and dynamic displaying of messages.
* User can send data anywhere in the world.
* IT is Eco Friendly, using IOT notice board we can reduce paper usage
* It can likewise be utilized as a part of Malls and Highways for Advertisement reason.
* A moving showcase with variable speed can likewise be utilized as a part of place of static display.
* IOT notice board can be used in various organisations like hospitals, schools, colleges, Offices etc.
* IOT notice board can be used in various organisations like hospitals, schools, colleges, Offices etc.

**3.4 Methodology of the Proposed System**

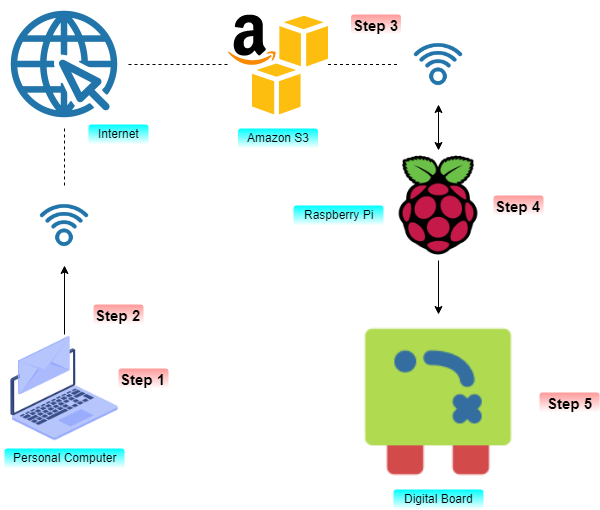
****

Fig 3.1: Methodology of the Proposed System.

* **Step1:** The user will open the bitwise SSH client and will enter the IP address of the raspberry pi along with the username and password of the pi. This provides authorization of the User and on correct username and password the user will connect to the pi over the network otherwise the bitwise SSH client will tell the user that he has entered wrong credentials.
* **Step 2:** After successfully logging in the user will open that ip address with the port number using the browser in the format ipaddress: portnumber and the home page will appear in which the user can upload the files to be displayed on the digital board. The user will select the file such as text, image, and video and will click the upload button.
* **Step 3:** After successful upload, the processing screen will be displayed during which a S3 bucket will be created and the data will be stored on the cloud and a copy of the data will be send to the pi and in case of data loss the user can go to the cloud and download the data and again upload the data.
* **Step 4:** After receiving the data, Raspberry Pi will send the data to the digital board through the HDMI cable. The data will be displayed based upon the FIFO (First-in-First-Out) principle.
* **Step 5:** The Digital board will receive the data from the Raspberry Pi through the HDMI port and the same will be displayed on the board.

**Chapter 4**

**SYSTEM ANALYSIS**

**4.1 Functional requirements**

It deals with the functionalities required from the system.

This proposed application will help many organizations such as railway stations, shopping malls, educational institutions to display any sort of information and will have the following requirements:

* Only an authorized person can access the application.
* All personnel using the system will be trained and given information about the accessibility of the application.
* The user can push any content that he wants to display on the board through the Dataplicity portal.
* The content can be any information such as text, audio, video, images etc.

**4.2 Non**-**Functional requirements**

There are some nonfunctional requirements such as:

**4.2.1 Security**

Security is very useful feature of Dataplicity and Raspberry pi which our system requires. To reduce the scam and unauthorized access to the digital screen.

**4.2.2 Availability**

Users can access their digital screen by portal website provided to them. This application uses internet as the medium of connection to raspberry pi. So, this system is available 24/7 without caring about handling servers.

**4.2.3 Reliability**

It means the extent to which program performs with require precision. The application developed should be extremely reliable and secure, hence the information displayed on the board must be correct and as per the user’s requirement.

**4.2.4 Usability**

This system provides the interface which is very user friendly and it requires least effort to operate. The application will provide services like session management to maintain sessions in order to display the content on the board through our application.

**4.2.5 Portability**

This system is built on modular design, each component of the system is independent of other. These different components of the system are connected using internet and raspberry pi. The application makes use of Raspbian and Python libraries etc., which are again platform independent.

**4.2.6 Transparency**

The whole application should be made using independent modules so that any changes done in one module should not affect the other one and new modules can be added easily to increase functionality.

**4.3 Specific Requirements**

Mainly there are two subjects for this system – Admin/ Users. Some important functions for them are: -

**Admin/ Users:**

* Can add the information or content according to the requirements of the users.
* Can view all the contents to be displayed on the smart board.
* The information to be displayed includes text, images, audio, video, advertisements, notifications etc.
* Can set the sessions of display for each content to be displayed on the smart board.
* Can view the information displayed on the smart board.

**4.4 External Interface Requirements**

It includes the following interfaces

* User Interfaces
* Software Interfaces
* Hardware Interfaces

**4.4.1 User Interfaces**

The interface must be easy to understand. The user interface includes

* **User Registration and Login:** The screen will be the first to be displayed which will allow the users to do the user registration and login.
* **User Query and data upload:** User can upload their data or information with time and synchronization they wish to display on the digital board.
* **End messages:** The information and data entered by user is then displayed on the smart board accordingly.

**4.4.2 Hardware Interfaces**

**Server-side hardware**

* Hardware recommended by all the software needed.
* Communication hardware to serve client requests.

**Client-side hardware**

* Hardware recommended by respective client’s operating system and web browser.
* Compatible operating system: Windows/Linux/IOS

**Client-side software**

* Web browsers.

**4.4.3 Software Interface**

The software interface is provided by the Dataplicity portal. The two main programming languages used are Python and Raspbian:

**Python**

Python is an [interpreted](https://en.wikipedia.org/wiki/Interpreted_language) [high-level programming language](https://en.wikipedia.org/wiki/High-level_programming_language) for [general-purpose programming](https://en.wikipedia.org/wiki/General-purpose_programming_language). Created by [Guido van Rossum](https://en.wikipedia.org/wiki/Guido_van_Rossum) and first released in 1991, Python has a design philosophy that emphasizes [code readability](https://en.wikipedia.org/wiki/Code_readability), notably using [significant whitespace](https://en.wikipedia.org/wiki/Significant_whitespace). It provides constructs that enable clear programming on both small and large scales. In July 2018, Van Rossum stepped down as the leader in the language community

Python uses [dynamic typing](https://en.wikipedia.org/wiki/Dynamic_typing), and a combination of [reference counting](https://en.wikipedia.org/wiki/Reference_counting) and a cycle-detecting garbage collector for [memory management](https://en.wikipedia.org/wiki/Memory_management). It also features dynamic [name resolution](https://en.wikipedia.org/wiki/Name_resolution_(programming_languages)) ([late binding](https://en.wikipedia.org/wiki/Late_binding)), which binds method and variable names during program execution.

## **Advantages of Python**

The diverse application of the Python language is a result of the combination of features which give this language an edge over others. Some of the benefits of programming in Python include:

* **Open Source and Community Development**

Python language is developed under an OSI-approved open source license, which makes it free to use and distribute, including for commercial purposes. Further, its development is driven by the community which collaborates for its code through hosting conferences and mailing lists and provides for its numerous modules.

* **Extensive Support Libraries**

Python provides alarge standard library which includes areas like internet protocols, string operations, web services tools and operating system interfaces

* **Learning Ease and Support Available**

Python offers excellent readability and uncluttered simple-to-learn syntax which helps beginners to utilize this programming language. Additionally, the wide base of users and active developers has resulted in a rich internet resource bank to encourage development and the continued adoption of the language.

* **User-friendly Data Structures**

Python has built-in list and dictionary data structures which can be used to construct fast runtime data structures. Further, Python also provides the option of dynamic high-level data typing which reduces the length of support code that is needed.

* **Productivity and Speed**

Python has cleaned object-oriented design, provides enhanced process control capabilities, and possesses strong integration and text processing capabilities and its own unit testing framework, all of which contribute to the increase in its speed and productivity.

**Raspbian**

**Raspbian** is a Debian-based computer operating system for Raspberry Pi. There are several versions of Raspbian including Raspbian Stretch and Raspbian Jessie. Since 2015 it has been officially provided by the Raspberry Pi Foundation as the primary operating system for the family of Raspberry Pi single-board computers. Raspbian was created by Mike Thompson and Peter Green as an independent project. The initial build was completed in June 2012. The operating system is still under active development. Raspbian is highly optimized for the Raspberry Pi line's low-performance ARM CPUs.

Raspbian uses PIXEL, **P**i **I**mproved **X**-Window **E**nvironment, **L**ightweight as its main desktop environment as of the latest update. It is composed of a modified LXDE desktop environment and the Open box stacking window manager with a new theme and few other changes. This distribution is shipped with a copy of computer algebra program Mathematica and a version of Minecraft called Minecraft Pi as well as lightweight version of Chromium as of the latest version.

**4.5 Hardware Requirements**

* Laptop
* Raspberry Pi 3B+
* LCD Screen
* Routers
* WIFI Adapter- TL-WN725n
* Speakers

**4.6 Software Requirements**

* Windows
* Python
* HTML/CSS/JavaScript
* Raspbian
* AWS S3 Bucket

**Chapter 5**

**DETAILED DESIGN**

**5.1 Flow Diagram**

Flow diagram is a collective term for a [diagram](https://en.wikipedia.org/wiki/Diagram) representing a flow or set of dynamic relationships in a system. The term flow diagram is also used as a synonym for [flowchart](https://en.wikipedia.org/wiki/Flowchart) and sometimes as a counterpart of the flowchart. Flow diagrams are used to structure and order a [complex system](https://en.wikipedia.org/wiki/Complex_system), or to reveal the underlying structure of the elements and their interaction.

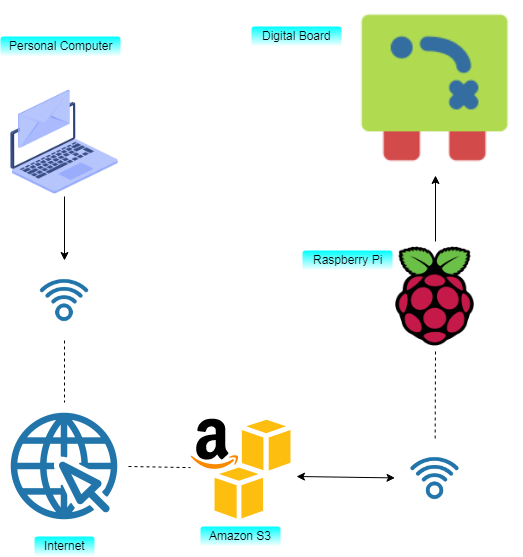
****

Fig 5.1: Flow Diagram

This diagram shows that how the components are connected to each other in the local in the proposed system. Wherein the data is pushed from Laptop and is received by the AWS cloud through the internet and the same is pushed by the raspberry pi to the digital board.

The user will push the data such as text, image, audio or video from the user interface by logging into the IP address provided to the user. The user interface contains options like upload tab in which the user can upload the data based upon the categories like text, multimedia and pdf .The user will choose the appropriate category and then upload the file from his system and the will click the upload button and on successful upload a message will pop up stating the file has been uploaded.

The log button helps us with the data that has been already uploaded. Using the log button we can get to know the data that is already uploaded or is current there to be displayed. This helps us keep track of the things that we have uploaded. As the data is uploaded the count on the button is updated. The deletion button in the log tab helps us in deleting the data that is already being displayed after deleting the data we need to reboot the server so that our changes are saved.

The Reboot button is used to restart the raspberry pi. The main use of this button is to refresh the server and restart the raspberry pi .When we restart the raspberry pi the configuration files are executed again and the pi is rebooted.

The user Interface also contains a button called “create bucket” which is used for creating the S3 bucket in the AWS cloud in which the data is stored as it is pushed from the Web application handled by the user. The complete transmission and retrieval is handled by S3 bucket. The main advantage of using the AWS cloud is that we have a proper authenticated system an in case of data loss the user can ask the admin to send him the data which was lost and the user can proceed with the same procedure again .The cloud access is limited only to the admin and can be manipulated only by him.

After the user uploads the data, the data gets uploaded into the cloud and resides in the S3 bucket, then a copy of the data is sent to the raspberry pi. The main advantage of sending a copy is that in case of data loss the data can be retrieved from the cloud by downloading it and again uploading it to the raspberry pi. When the raspberry pi receives the data it displays it on the digital screen. The swapping speed can be controlled by changing the time delay.

**5.2 Low Level Design**

Low-level design (LLD) is a component-level design process that follows a step-by-step [refinement](https://en.wikipedia.org/wiki/Refinement_(computing)) process. This process can be used for designing data structures, required software architecture, source code and ultimately, performance algorithms. Overall, the data organization may be defined during requirement analysis and then refined during data design work. Post-build, each component is specified in detail.

The LLD phase is the stage where the actual software components are designed. During the detailed phase the logical and functional design is done and the design of application structure is developed during the high-level design phase.

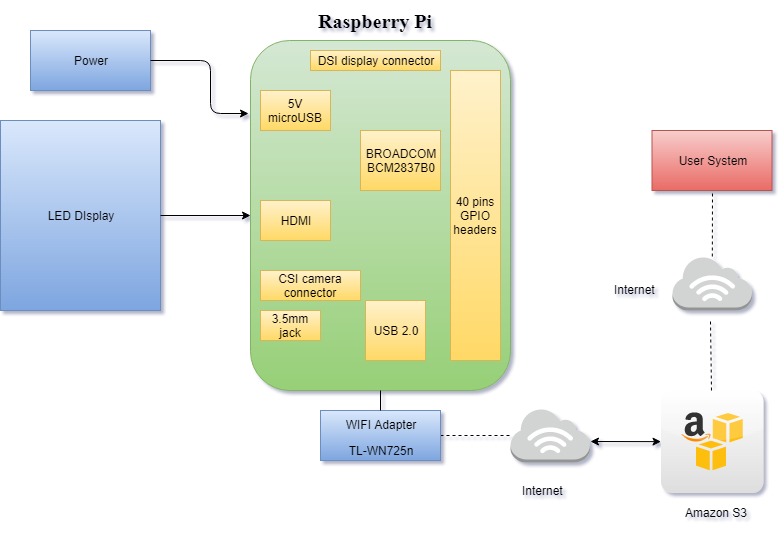
****

Fig 5.2: Low level design

This diagram shows the Low-level design of the proposed system containing Raspberry Pi, LED/LCD display, Router and User Personal Computer and also shows that how these components are connected to each other and transferring the data from one unit to other in the network. The complete connection is shown in the above diagram.

**Chapter 6**

**IMPLEMENTATION**

Objective of implementation phase is to convert the design of the project into executable code written in programming language. This code will then be fed into the computer for execution which will perform the computation specified by the design. This chapter will be dealing with their modules and their description.

**6.1 Introduction**

Implementation is the realization of an application, or execution of a plan, idea, model, design, specification, standard, algorithm or policy. In other words, an implementation is the realization of a technical specification or algorithm as a program, software component, or other computer system through deployment and programming. Many implementations may exist for a given specification or standard.

Implementation is one of the most important phases of the software Development Life Cycle (SDLC). It encompasses all the processes involved in getting new software or hardware operating properly in its environment, including installation, configuration, and running, testing and making necessary changes. Specifically, it involves coding a system using a particular programming language and transferring the design into an actual working system. This phase of the system is conducted with the idea that whatever that is designed should be Implemented; keeping in mind that it fulfills the user requirement, objective and scope of the system. The implementation phase produces solution to the user problem.

**6.2** **Hardware Implementation**

**6.2.1 Raspberry Pi 3B+**

Raspberry Pi can be configured TCP server to listen for incoming calls. The **Raspberry Pi** is a series of credit card–sized single-board computers developed in England, United Kingdom by the Raspberry Pi Foundation with the intent to promote the teaching of basic computer science in schools and developing countries.

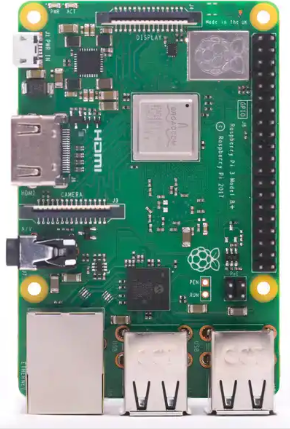


Fig 6.1: Raspberry Pi 3B+

The [Broadcom](https://en.wikipedia.org/wiki/Broadcom) BCM2835 SoC used in the first generation Raspberry Pi[[20]](https://en.wikipedia.org/wiki/Raspberry_Pi#cite_note-Broadcom-BCM2835-Website-20) includes a 700 [MHz](https://en.wikipedia.org/wiki/Hertz) [ARM11](https://en.wikipedia.org/wiki/ARM11) 76JZF-S processor, [Video Core](https://en.wikipedia.org/wiki/VideoCore) IV [graphics processing unit](https://en.wikipedia.org/wiki/Graphics_processing_unit) (GPU) and RAM. It has a level 1 (L1) [cache](https://en.wikipedia.org/wiki/CPU_cache) of 16 [KB](https://en.wikipedia.org/wiki/Kibibyte) and a level 2 (L2) cache of 128 KB. The [level 2 cache](https://en.wikipedia.org/wiki/Level_2_cache) is used primarily by the GPU. The SoC is [stacked](https://en.wikipedia.org/wiki/Package_on_package) underneath the RAM chip, so only its edge is visible. The 1176JZ(F)-S is the same CPU used in the [original iPhone](https://en.wikipedia.org/wiki/IPhone_(1st_generation)), although at a higher [clock rate](https://en.wikipedia.org/wiki/Clock_rate), and mated with a much faster GPU.

The earlier V1.1 model of the Raspberry Pi 2 used a Broadcom BCM2836 SoC with a 900 MHz [32-bit](https://en.wikipedia.org/wiki/32-bit), [quad-core](https://en.wikipedia.org/wiki/Quad_Core) [ARM Cortex-A7](https://en.wikipedia.org/wiki/ARM_Cortex-A7) processor, with 256 KB shared L2 cache. The Raspberry Pi 2 V1.2 was upgraded to a Broadcom BCM2837 SoC with a 1.2 GHz [64-bit](https://en.wikipedia.org/wiki/64-bit_computing) quad-core [ARM Cortex-A53](https://en.wikipedia.org/wiki/ARM_Cortex-A53) processor, the same SoC which is used on the Raspberry Pi 3, but [underclocked](https://en.wikipedia.org/wiki/Underclocking) (by default) to the same 900 MHz CPU clock speed as the V1.1. The BCM2836 SoC is no longer in production as of late 2016.

The Raspberry Pi 3+ uses a Broadcom BCM2837B0 SoC with a 1.4 GHz 64-bit quad-core [ARM Cortex-A53](https://en.wikipedia.org/wiki/ARM_Cortex-A53) processor, with 512 KB shared L2 cache. The Raspberry Pi Zero and ZeroW use the same Broadcom BCM2835 SoC as the first-generation Raspberry Pi, although now running at 1GHz CPU clock speed.

**6.2.2 WIFI Adapter TL-WN725n**

Hardware Features

|  |  |
| --- | --- |
| Interface | USB 2.0 |
| Dimensions | 0.73x0.59x0.28in.(18.6x15x7.1mm) |
| Antenna | Internal antenna |
| LED | Status |
| Weight | 2.2grams |
| Wireless Standards | IEEE 802.11b, IEEE 802.11g, IEEE 802.11n |
| Frequency | 2.400-2.4835ghz |
| Signal Rate | 11b: Up to 11Mbps (dynamic) |
|  | 11g: Up to 54Mbps (dynamic) |
| Reception Sensitivity | 130M: -68dbm@10% PER |
| Transmit Power | <20dbm |
| Wireless Modes | Ad-Hoc / Infrastructure mode |
| Wireless Security | Supports 64/128 WEP, WPA/WPA2, |
|  | WPA-PSK/WPA2-PSK (TKIP/AES), |
| Modulation Technology | DBPSK, DQPSK, CCK, OFDM, 16-QAM, |

Table 6.1: WIFI Adapter TL-WN725n Specifications



Fig 6.2: TL-WN725n

**6.3 Software Implementation**

**6.3.1 Python**

**Python** is an [interpreted](https://en.wikipedia.org/wiki/Interpreted_language), [high-level](https://en.wikipedia.org/wiki/High-level_programming_language), [general-purpose](https://en.wikipedia.org/wiki/General-purpose_programming_language) [programming language](https://en.wikipedia.org/wiki/Programming_language). Created by [Guido van Rossum](https://en.wikipedia.org/wiki/Guido_van_Rossum) and first released in 1991, Python's design philosophy emphasizes [code readability](https://en.wikipedia.org/wiki/Code_readability) with its notable use of [significant whitespace](https://en.wikipedia.org/wiki/Off-side_rule). Its language constructs and [object-oriented](https://en.wikipedia.org/wiki/Object-oriented_programming) approach aims to help programmers write clear, logical code for small and large-scale projects.

Python's large [standard library](https://en.wikipedia.org/wiki/Standard_library), commonly cited as one of its greatest strengths, provides tools suited for many tasks. For Internet-facing applications, many standard formats and protocols such as [MIME](https://en.wikipedia.org/wiki/MIME) and [HTTP](https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol) are supported. It includes modules for creating [graphical user interfaces](https://en.wikipedia.org/wiki/Graphical_user_interface), connecting to [relational databases](https://en.wikipedia.org/wiki/Relational_database), [generating pseudorandom numbers](https://en.wikipedia.org/wiki/Pseudorandom_number_generator), arithmetic with arbitrary precision decimals, manipulating [regular expressions](https://en.wikipedia.org/wiki/Regular_expression), and [unit testing](https://en.wikipedia.org/wiki/Unit_testing).

Some parts of the standard library are covered by specifications (for example, the [Web Server Gateway Interface](https://en.wikipedia.org/wiki/Web_Server_Gateway_Interface) (WSGI) implementation wsgiref follows PEP 333), but most modules are not. They are specified by their code, internal documentation, and test suites (if supplied). However, because most of the standard library is cross-platform Python code, only a few modules need altering or rewriting for variant implementations.

The raspberry pi in our project is used for receiving and transmitting the data from the user’s system to the digital board in an appropriate manner. The python is used for scripting the raspberry pi and for creating the S3 buckets.

**6.3.2 S3 Bucket**

Amazon S3 is cloud storage for the internet. To upload your data (photos, videos, documents etc.), you first create a bucket in one of the AWS Regions. You can then upload any number of objects to the bucket.

In terms of implementation, buckets and objects are resources, and Amazon S3 provides APIs for you to manage them. For example, you can create a bucket and upload objects using the Amazon S3 API. You can also use the Amazon S3 console to perform these operations. The console uses the Amazon S3 APIs to send requests to Amazon S3.This section explains how to work with buckets. For information about working with objects, see [Working with Amazon S3 Objects](https://docs.aws.amazon.com/AmazonS3/latest/dev/UsingObjects.html).

An Amazon S3 bucket name is globally unique, and the namespace is shared by all AWS accounts. This means that after a bucket is created, the name of that bucket cannot be used by another AWS account in any AWS Region until the bucket is deleted. You should not depend on specific bucket naming conventions for availability or security verification purposes. For bucket naming guidelines, see Limitations. Amazon S3 creates buckets in a Region you specify. To optimize latency, minimize costs, or address regulatory requirements, choose any AWS Region that is geographically close to you. For example, if you reside in Europe, you might find it advantageous to create buckets in the EU (Ireland) or EU (Frankfurt) Regions. For a list of Amazon S3 Regions, see [Regions and Endpoints](https://docs.aws.amazon.com/general/latest/gr/rande.html#s3_region) in the AWS General Reference.

**6.3.2.1 Creating the S3 bucket:**

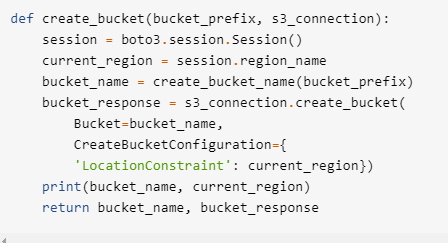
****

Fig 6.3: S3 Bucket Creation

**6.3.2.2 Bucket Downloading**

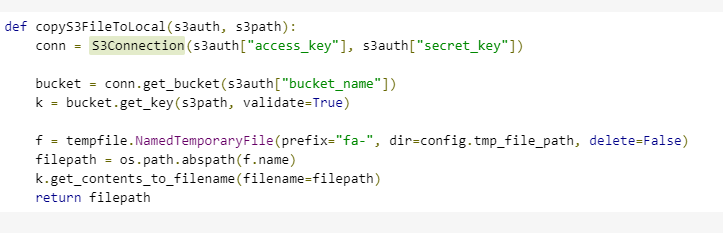
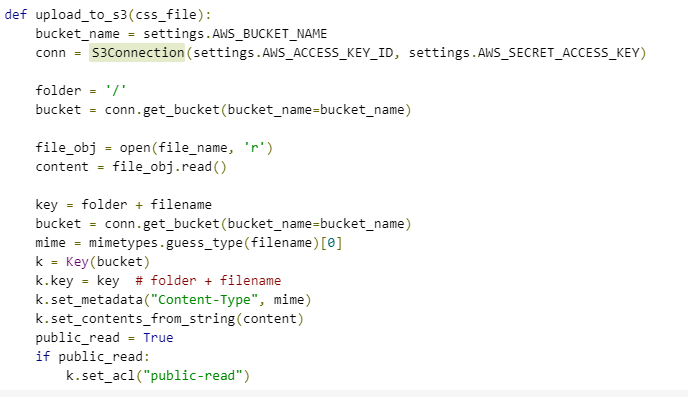
****

Fig 6.4: Bucket Downloading

**6.3.2.3 Uploading file to S3**

****Fig 6.5: Uploading file to S3

**6.3.2.4 The Main Calling Function**

import json

import os

from app import app

from flask import Flask, flash, request, redirect, render\_template, jsonify

from werkzeug.utils import secure\_filename

import subprocess

from pdf2image import convert\_from\_path

from PIL import Image, ImageDraw

import time

img = Image.new('RGB', (640, 480), color = (73, 109, 137))

ALLOWED\_EXTENSIONS = set(['txt', 'pdf', 'png', 'jpg', 'jpeg', 'gif'])

def allowed\_file(filename):

return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED\_EXTENSIONS

@app.route('/')

def upload\_form():

return render\_template('index.html')

@app.route('/', methods=['POST'])

def upload\_file():

if request.method == 'POST':

# check if the post request has the file part

if 'file' not in request.files:

flash('No file part')

return redirect(request.url)

file = request.files['file']

if file.filename == '':

flash('No file selected for uploading')

return redirect(request.url) if(fileandallowed\_file(file.filename)and(int(round(time.time()))<1568172850)):

filename = secure\_filename(file.filename)

print(filename)

# print([filename])

if filename.endswith('.pdf'):

print("pdf")

file.save(os.path.join('/home/pi/flask/uploads\_2', filename))

pages = convert\_from\_path('/home/pi/flask/uploads\_2/'+filename, 500)

for page in pages:

page.save('/home/pi/flask/uploads/'+filename[:-4]+'.jpg','JPEG')

cmd = "sudo rm /home/pi/flask/uploads\_2/\*"

subprocess.Popen(cmd, shell=True)

else:

file.save(os.path.join(app.config['UPLOAD\_FOLDER'], filename))

flash('File(s) successfully uploaded')

return redirect('/')

@app.route('/up', methods=['GET'])

def up():

if request.method == 'GET':

imageList = sorted(os.listdir("/home/pi/flask/uploads"), reverse=True)

print(imageList)

return jsonify(imageList)

@app.route('/del', methods=['POST'])

def deleteAllImages():

if request.method == 'POST':

data = request.get\_json()

print(data)

print(data['dataa'])

cmd = "sudo rm /home/pi/flask/uploads/"+str(data['dataa'])

subprocess.Popen(cmd, shell=True)

return jsonify(msg="Deleted File"), 200

@app.route('/text', methods=['POST','GET'])

def texts():

if request.method == 'POST':

data = request.get\_json()

print(data)

print(data['dataa'])

msg=data['dataa']

d = ImageDraw.Draw(img)

d.text((50,50), msg, fill=(255,255,0))

# myFont = ImageFont.truetype("/home/pi/flask/TIMES.ttf", 16)

# d.textsize(msg, font=myFont)

img.save('/home/pi/flask/uploads/pil\_text.png')

#cmd = "sudo rm /home/pi/flask/uploads/"+str(data['dataa'])

#subprocess.Popen(cmd, shell=True)

return jsonify(msg="Added text"), 200

@app.route('/reboot', methods=['POST','GET'])

def reboot():

if request.method == 'POST':

data = request.get\_json()

print(data)

print(data['dataa'])

cmd = "sudo reboot now"

subprocess.Popen(cmd, shell=True)

return jsonify(msg="Rebooted"), 200

if \_\_name\_\_ == "\_\_main\_\_":

app.run(host='0.0.0.0', port=8000)

**6.3.3 Flask**

Flask is a micro [web framework](https://en.wikipedia.org/wiki/Web_framework) written in [Python](https://en.wikipedia.org/wiki/Python_(programming_language)). It is classified as a [micro framework](https://en.wikipedia.org/wiki/Microframework) because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Extensions exist for object-relational mappers, form validation, and upload handling, various open authentication technologies and several common framework related tools. Extensions are updated far more regularly than the core Flask program.It includes modules for creating [graphical user interfaces](https://en.wikipedia.org/wiki/Graphical_user_interface), connecting to [relational databases](https://en.wikipedia.org/wiki/Relational_database), [generating pseudorandom numbers](https://en.wikipedia.org/wiki/Pseudorandom_number_generator), arithmetic with arbitrary precision decimals, manipulating [regular expressions](https://en.wikipedia.org/wiki/Regular_expression) and [unit testing](https://en.wikipedia.org/wiki/Unit_testing).

**6.3.4 Boto3**

Boto3 is the Amazon Web Services (AWS) SDK for Python. It enables Python developers to create, configure, and manage AWS services, such as EC2 and S3. Boto provides an easy to use, object-oriented API, as well as low-level access to AWS services.

**6.3.5 FBI**

Frame Buffer Image Viewer (FBI) is used for image rendering. It displays the specified file on the console using the framebuffer device. PhotoCD, jpeg, ppm, gif, tiff, xpm, xwd, bmp, png and webp formats are supported natively.

**6.3.6 Omx player**

[Omxplayer](https://github.com/popcornmix/omxplayer) is a video player specifically made for the Raspberry Pi's GPU made by Edgar (gimli) Hucek from the XBMC/Kodi project. It relies on the [OpenMAX](https://en.wikipedia.org/wiki/OpenMAX) hardware acceleration API, which is the [Broadcom's VideoCore](https://en.wikipedia.org/wiki/VideoCore) officially supported API for GPU video/audio processing. Raspberry Pi forum user spenning made precompiled binaries available on the forum. See [here](http://www.raspberrypi.org/phpBB3/viewtopic.php?f=2&t=5543).

**6.3.7 Bashrc**

The. bashrc file is a shell script which is run every time a user opens a new shell. Every time you open a terminal window the bashrc file is performed. The. bashrc file is commonly used to set aliases to commonly used commands so that you don't have to remember long commands.

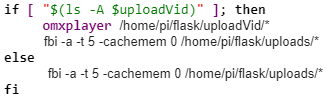
****

Fig 6.6: bashrc Script

**6.3.8 Html, CSS and JavaScript**

**Hypertext Markup Language (HTML)** is the standard [mark-up language](https://en.wikipedia.org/wiki/Markup_language) for creating [web pages](https://en.wikipedia.org/wiki/Web_page) and [web applications](https://en.wikipedia.org/wiki/Web_application). With [Cascading Style Sheets](https://en.wikipedia.org/wiki/Cascading_Style_Sheets) (CSS) and [JavaScript](https://en.wikipedia.org/wiki/JavaScript), it forms a triad of [cornerstone](https://en.wikipedia.org/wiki/Cornerstone) technologies for the [World Wide Web](https://en.wikipedia.org/wiki/World_Wide_Web).

[Web browsers](https://en.wikipedia.org/wiki/Web_browser) receive HTML documents from a [web server](https://en.wikipedia.org/wiki/Web_server) or from local storage and [render](https://en.wikipedia.org/wiki/Browser_engine) the documents into multimedia web pages. HTML describes the structure of a web page [semantically](https://en.wikipedia.org/wiki/Semantic_Web) and originally included cues for the appearance of the document.

**Cascading Style Sheets (CSS)** is a [style sheet language](https://en.wikipedia.org/wiki/Style_sheet_language) used for describing the [presentation](https://en.wikipedia.org/wiki/Presentation_semantics) of a document written in a [markup language](https://en.wikipedia.org/wiki/Markup_language) like [HTML](https://en.wikipedia.org/wiki/HTML). CSS is a cornerstone technology of the [World Wide Web](https://en.wikipedia.org/wiki/World_Wide_Web), alongside HTML and [JavaScript](https://en.wikipedia.org/wiki/JavaScript).

CSS is designed to enable the separation of presentation and content, including [layout](https://en.wikipedia.org/wiki/Page_layout), [colors](https://en.wikipedia.org/wiki/Color), and [fonts](https://en.wikipedia.org/wiki/Typeface). This separation can improve content [accessibility](https://en.wikipedia.org/wiki/Accessibility), provide more flexibility and control in the specification of presentation characteristics, enable multiple [web pages](https://en.wikipedia.org/wiki/Web_page) to share formatting by specifying the relevant CSS in a separate .css file, and reduce complexity and repetition in the structural content.

**JavaScript,** often abbreviated as **JS**, is a [high-level](https://en.wikipedia.org/wiki/High-level_programming_language), [interpreted](https://en.wikipedia.org/wiki/Interpreted_language) [programming language](https://en.wikipedia.org/wiki/Programming_language) that conforms to the [ECMAScript](https://en.wikipedia.org/wiki/ECMAScript) specification. JavaScript has [curly-bracket syntax](https://en.wikipedia.org/wiki/List_of_programming_languages_by_type#Curly-bracket_languages), [dynamic typing](https://en.wikipedia.org/wiki/Dynamic_programming_language), [prototype-based](https://en.wikipedia.org/wiki/Prototype-based_programming) [object-orientation](https://en.wikipedia.org/wiki/Object-oriented_programming), and [first-class functions](https://en.wikipedia.org/wiki/First-class_function).

Alongside [HTML](https://en.wikipedia.org/wiki/HTML) and [CSS](https://en.wikipedia.org/wiki/CSS), JavaScript is one of the core technologies of the [World Wide Web](https://en.wikipedia.org/wiki/World_Wide_Web).[[9]](https://en.wikipedia.org/wiki/JavaScript#cite_note-9) JavaScript enables interactive [web pages](https://en.wikipedia.org/wiki/Web_page) and is an essential part of [web applications](https://en.wikipedia.org/wiki/Web_application). The vast majority of [websites](https://en.wikipedia.org/wiki/Website) use it, and major [web browsers](https://en.wikipedia.org/wiki/Web_browser) have a dedicated [JavaScript engine](https://en.wikipedia.org/wiki/JavaScript_engine) to execute it. As multi-paradigm language, JavaScript supports [event-driven](https://en.wikipedia.org/wiki/Event-driven_programming), [functional](https://en.wikipedia.org/wiki/Functional_programming), and [imperative](https://en.wikipedia.org/wiki/Imperative_programming) (including [object-oriented](https://en.wikipedia.org/wiki/Object-oriented_programming) and [prototype-based](https://en.wikipedia.org/wiki/Prototype-based_programming)) [programming styles](https://en.wikipedia.org/wiki/Programming_paradigm).

It has [APIs](https://en.wikipedia.org/wiki/Application_programming_interface) for working with text, [arrays](https://en.wikipedia.org/wiki/Array_data_type), dates, [regular expressions](https://en.wikipedia.org/wiki/Regular_expression), and the [DOM](https://en.wikipedia.org/wiki/Document_Object_Model), but the language itself does not include any [I/O](https://en.wikipedia.org/wiki/Input/output), such as [networking](https://en.wikipedia.org/wiki/Computer_network), [storage](https://en.wikipedia.org/wiki/Data_storage), or [graphics](https://en.wikipedia.org/wiki/Computer_graphics) facilities. It relies upon the host environment in which it is embedded to provide these features.

The code of the client’s user interface is as follows:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>IOT BASED CLOUD ENABLED DIGITAL NOTICE BOARD</title>

<title></title>

<!-- common css -->

<link rel="icon" type="text/html" href="static/home-bg.jpg">

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.1.0/css/bootstrap.min.css">

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/responsive/1.0.5/responsive.min.js">

<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.0/umd/popper.min.js"></script>

<script type="text/javascript" src="{{ url\_for('static', filename='script.js') }}"></script>

<link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.8.2/css/all.css">

<link rel="stylesheet" type="text/css" href="/static/style.css">

</head>

<body>

<!-- header area-->

<header>

<div class="header-logo-area">

<div class="container">

<div class="row">

<div class="col-md-3 hidden-xs">

<div class="call-us pull-left">

<p><i class="fa fa-phone"></i>(xx)-xx-xxx-xxx</p>

<p><i class="fa fa-envelope"></i>info@acharya.ac.in</p>

</div>

</div>

<div class="col-md-6">

<div class="logo text-center">

<a href="index.html">

<img src="https://www.acharya.ac.in/a/public/images/logo/logo.png" alt=""></a>

</div>

</div>

<div class="col-md-3">

<div class="pull-right user-profile-link cart-total">

<ul>

<li>

<div id="list" class="shopping-cart">

<a href="">

<i class="fa fa-bell"></i>

<span class="cart-count"></span>

</a>

<div class="mini-cart-content">

<div class="cart-img-details">

<div id="demo">

</div>

</div>

<div class="clear"></div>

</div>

</div>

<div class="clear"></div>

</li>

<li>

<div data-toggle="tooltip" title="Upload Text, Video, Image, PDF">

<a href="">

<i class="fa fa-link"></i>

<span class="">Upload</span>

</a>

<div class="mini-cart-content" style="top: 57px;">

<div class="cart-img-details">

<form method="" action="" enctype="">

<h5>Enter your text</h5>

<p>

<textarea id="textarea" rows="4" cols="25" name="comment"form="usrform"></textarea>

</p>

<p>

<input id="text" type="submit" class="btn btn-success" value="Submit text">

</p>

</form>

<form method="post" action="/" enctype="multipart/form-data">

<dl>

<h5>Upload your multimedia here</h5>

<p>

<input type="file" name="file" multiple="true" autocomplete="off">

</p>

</dl>

<p>

<input type="submit" class="btn btn-success" value="Upload Multimedia">

</p>

</form>

<form method="post" action="/" enctype="multipart/form-data">

<dl>

<h5>Upload your PDFs here</h5>

<p>

<input type="file" name="file" multiple="true" autocomplete="off">

</p>

</dl>

<p>

<input type="submit" class="btn btn-success" value="Upload PDF">

</p>

</form>

<div class="clear"></div>

</div>

</div>

</div>

</li>

</ul>

</div>

</div>

</div>

</div>

</div>

</header>

<!-- home area-->

<section id="home">

<div class="container">

<div class="row">

<div class="col-md-offset-1 col-md-3">

<div class="home-content">

<h2 class="text-uppercase">Notice Board</h2>

<h5 class="text-uppercase">Important Information<span></span></h5>

<p>

{% with messages = get\_flashed\_messages(with\_categories = true) %} {% if messages %}

<ul class=flashes>

{% for category,message in messages %}

<li class="{{category}}">{{ message }}</li>

{% endfor %}

</ul>

{% endif %} {% endwith %}

</p>

<a id="refresh" href="" class="text-uppercase" value="Refresh Notice board">Refresh Board</a>

</div>

<div class="home-content">

<a id="createBucket" href="" class="text-uppercase" value="createBucket">Create Bucket</a>

</div>

</div>

</div>

</div>

</section>

<footer class="copy text-center">

<p>&copy; 2019 Team 5. All Right Reserved</p>

</footer>

<script type="text/javascript" src="{{ url\_for('static', filename='amazon.js') }}"></script>

</body>

</html>

**Chapter 7**

**TESTING**

**Testing**

Software testing is defined as an activity to check whether the actual results match the expected results and to ensure that the software system is defect free. It involves execution of a software component or system component to evaluate one or more properties of interest.

Software testing also helps to identify errors, gaps or missing requirements in contrary to the actual requirements. It can be either done manually or using automated tools. Some prefer saying Software testing as a White Box and Black Box Testing.

**White Box Testing:**It is also called as Glass Box, Clear Box, and Structural Testing. White Box Testing is based on applications internal code structure. In white-box testing, an internal perspective of the system, as well as programming skills, are used to design test cases. This testing is usually done at the unit level.

**Black Box Testing:**It is also called as Behavioural/Specification-Based/Input-Output Testing. Black Box Testing is a software testing method in which testers evaluate the functionality of the software under test without looking at the internal code structure.

**Levels of Testing**

A level of software testing is a process where every unit or component of a software/system is tested. The main goal of system testing is to evaluate the system's compliance with the specified needs.

There are many different testing levels which help to check behaviour and performance for software testing. These testing levels are designed to recognize missing areas and reconciliation between the development lifecycle states. In SDLC models there are characterized phases such as requirement gathering, analysis, design, coding or execution, testing, and deployment.

All these phases go through the process of software testing levels.

There are mainly four testing levels are:

1. Unit Testing
2. Integration Testing
3. System Testing

**7.1 Unit Testing**

A Unit is a smallest testable portion of system or application which can be compiled, liked, loaded, and executed. This kind of testing helps to test each module separately. The aim is to test each part of the software by separating it. It checks that component is fulfilling functionalities or not. This kind of testing is performed by developers.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sl. No** | **Description** | **Input** | **Expected Output** | **Actual Output** | **Status** |
| 1 | Checking if Raspberry Pi works | Power Supply | Red light Indicator glows | Red light Indicator glowing | Pass |
| 2 | To check whether Raspbian OS is Installed Successfully | Insert bootable SD card | Dashboard will appear | Dashboard appeared | Pass |
| 3 | Function to test upload\_file | File with invalid extension | Uploading fail | Successfully Uploaded | Fail |
| 4 | Function to test upload\_file | Valid File | Successful Upload | Successfully Uploaded | Pass |
| 5 | Function to test | Click the refresh notice board button | Home screen will appear | Home screen appeared | Pass |
| reboot |

Table 7.1: Unit Testing

**7.2 Integration Testing**

Integration means combining. For Example, in this testing phase, different software modules are combined and tested as a group to make sure that integrated system is ready for system testing. Integration testing checks the data flow from one module to other modules. This kind of testing is performed by testers.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sl. No** | **Description** | **Input** | **Expected Output** | **Actual Output** | **Status** |
| 1 | Integrate PC & RPi for Authentication | Input Login Id & Password | Secured remote login using SSH protocol | Secured remote login successful | Pass |
| 2 | Integrate Web App & RPi to check web page | Open the web application in browser | Loading of web page when the correct IP address is provided. | Web page loading | Pass |
| 3 | Integrating RPi & Board to push data into display | Push invalid data from Raspberry Pi to board | Do not display data | Data displayed on Board | Fail |
| 4 | Integrating RPi & Board to push data to display | Push data from Raspberry Pi to board | Data to be displayed on Board | Data displayed on Board | Pass |

Table 7.2: Integration Testing

**7.3 System Testing**

System testing is performed on a complete, integrated system. It allows checking system's compliance as per the requirements. It tests the overall interaction of components. It involves load, performance, reliability and security testing. System testing most often the final test to verify that the system meets the specification. It evaluates both functional and non-functional need for the testing.

System Testing is a level of software testing where a complete and integrated software is tested. The purpose of this test is to evaluate the system's compliance with the specified requirements.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sl. No** | **Description** | **Input** | **Expected Output** | **Actual Output** | **Status** |
| 1 | Collect text message from user | Input text message into the dialog box | Allow the user to enter text message | User allowed to enter text message | Pass |
| 2 | Display the text message | Send the text message to the Raspberry Pi | Convert the message received from user into .jpeg format and display on the screen | Message converted into .jpeg format and displayed on the screen | Pass |
| 3 | Collect image message from user | Select the image from the system | Allow the user to browse any image file. | User allowed to browse any image file. | Pass |
| 4 | Display image | Send the image to the Raspberry Pi | Load images browsed by user and display on the screen | Loaded images by user are displayed on the screen | Pass |
| 5 | Collect pdf files from user | Select the pdf from the system | Allow the user to browse any number of pdf files. | User allowed to browse any number of pdf files. | Pass |
| 6 | Display pdf files | Send the pdf to the Raspberry Pi | Convert pdf files into .jpeg files and display on screen. | Pdf converted into .jpeg and displayed on screen. | Pass |

Table 7.3: System testing

**Chapter 8**

**Results**

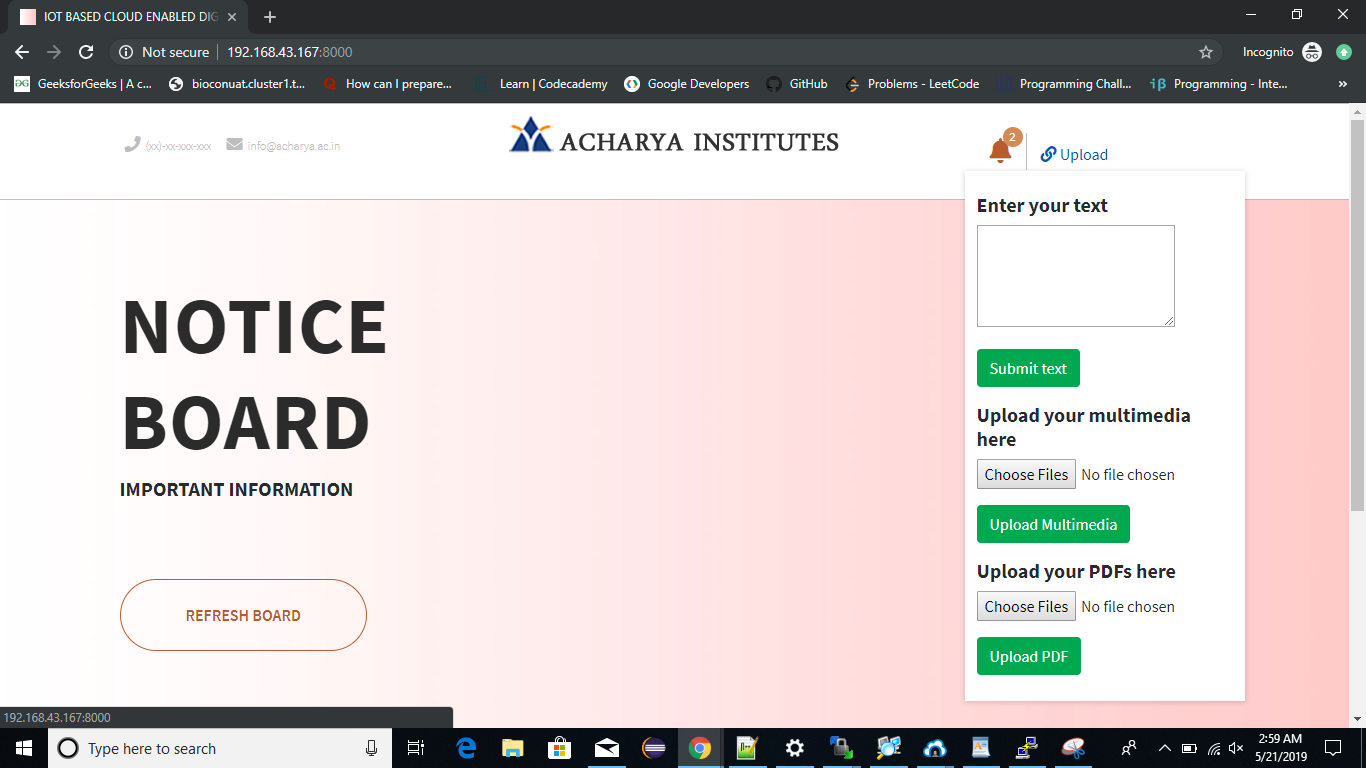
****

Fig 8.1: User’s Home Page

The above screenshot displays the user’s side home page from where the user will upload the data.

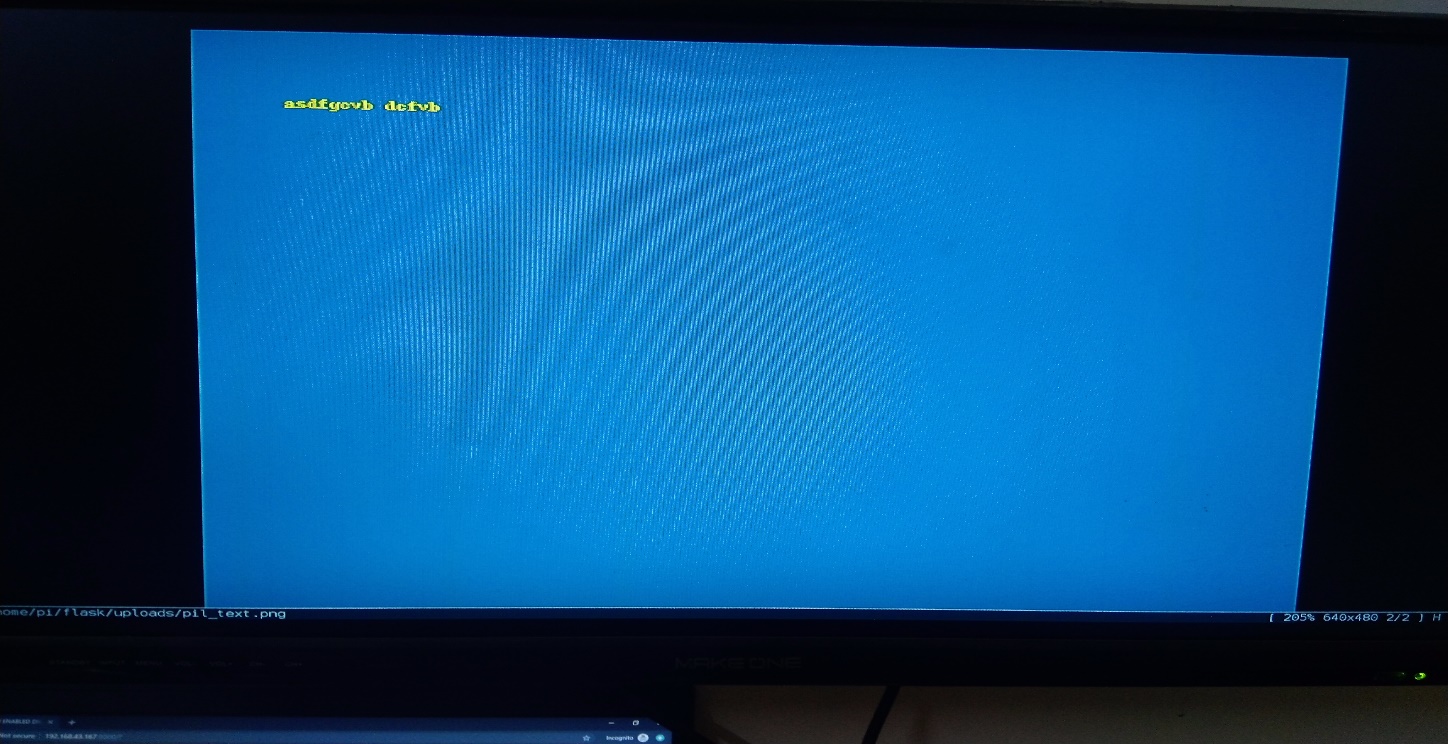


Fig 8.2: Output as Text

The above screenshot shows the text that will be displayed on the board when user uploads text.

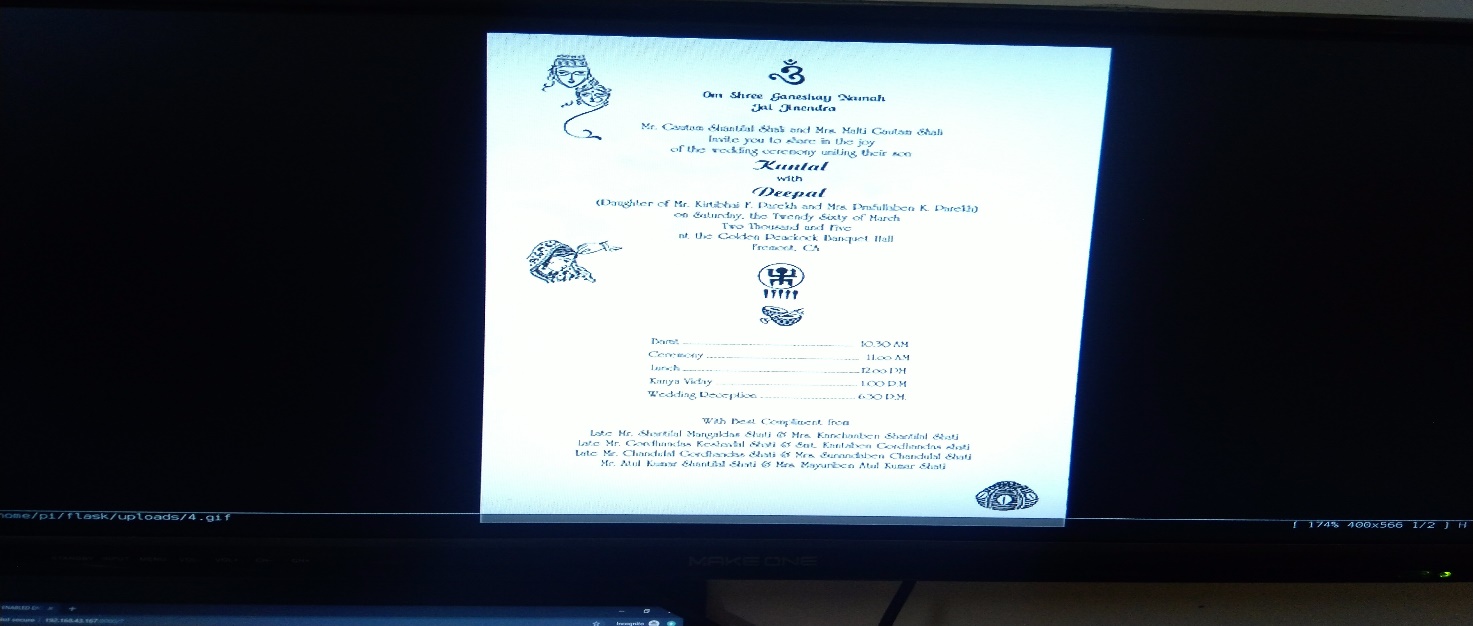


Fig 8.3: Output as Image

The above screenshot shows the image that will be displayed on the board when user uploads an Image file.

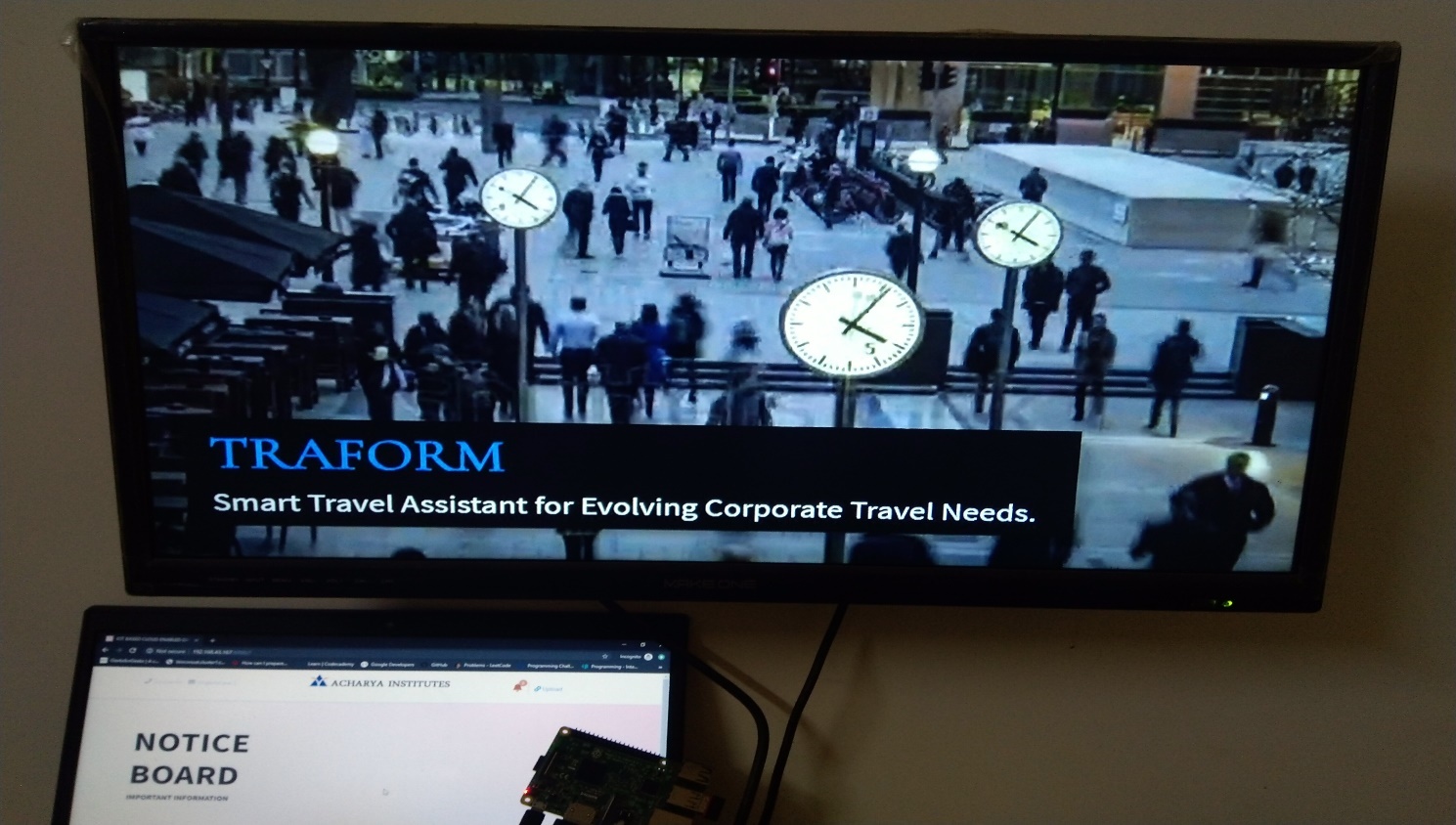


Fig 8.4: Output as Video

The above screenshot shows the Video that will be displayed on the board when user uploads a Video file.

**CONCLUSION & FUTURE ENHANCEMENT**

Now our world is moving towards digitalization, so if we want to do some changes in the previously used system, we have to use the new techniques. Wireless technology provides fast transmission of data over long range. It saves time, cost of cables, and size of the system.

Username and password type authentication system is provided for adding securities. Previously the notice board using GSM was used with limited range, but in our system, Internet is used as communication medium. So, there is no problem with range.

In future, IOT based wireless smart board can be enhanced by making an android and iOS application for sending the data to the display board. In order to increase the speed of data transmission, we can make use of more advanced technologies such as Li-Fi.

With the advancement of technology, new security algorithms will be introduced which can be implemented in the project. After process usage testing the project can be deployed for public use.

**REFERENCES**

1. Large Screen Wireless Notice Display System by Yash Teckchandani, G. Siva Perumal, Radhika Mujumdar, Sridhar Lokanathan [June 2015].
2. GSM Based Wireless Electronic Notice Board Display through ARM7 and LED by P. Sampath Kumar, V. Priyanka, Lakshmi Surekha, Y. Harish Reddy [May 2016].
3. Wireless Electronic Notice Board Using Raspberry Pi 3 by Er. G. Jalalu, Er. Polepogu Rajesh [June 2017].
4. An IOT Based Web Page Controlled Digital Notice Board by P. Bhaskara Chary, Dr. T. Srinivasulu [August 2017].
5. IoT based web-controlled notice board by Divyashree M, Harinag Prasad S, Sandeep G T, Bhavya S N, Poornima S [April 2018].
6. www.python.org for details on python programming.
7. www.raspberrypi.org for details on Raspberry Pi.
8. www.raspbian.org for details on Raspbian OS.
9. www.w3schools for building the front end.
10. www.flask.pocoo.org for learning the python micro-framework.