Bachelor's thesis

Bachelor of Engineering

Information Technology

2014

Amrit Regmi

DESIGN AND IMPLEMENTATION OF A REMOTELY MANAGED DIGITAL SIGNAGE SOLUTION

TURUN AMMATTIKORKEAKOULU
TURKU UNIVERSITY OF APPLIED SCIENCES

BACHELOR´S THESIS | ABSTRACT
TURKU UNIVERSITY OF APPLIED SCIENCES

Information Technology | Networking

2014 | 48

Instructor: Patric Granholm

Amrit Regmi

DESIGN AND IMPLEMENTATION OF A REMOTELY MANAGED DIGITAL SIGNAGE SOLUTION

Industries and businesses are moving to digital signage from the traditional printed signage as a tool for advertisement and broadcasting information to clients, potential customers, and the public. The availability of affordable flat screen digital displays has enabled the businesses to replace printed signage with digital signage. As a result, digital signage is expected play an important role in delivering information to the public as it can be customized to display timely information to a targeted audience while reducing the financial and environmental cost related with traditional printed signage. The focus of this thesis is the design and implementation of a digital signage solution based on Raspberry Pi for digital displays located on one or more locations.

The digital signage system displays the information on the full screen web browser running on Raspberry Pi. Each display is connected to a central server and is remotely accessible by the authorized administrator via the Internet or the local network. Contents can be dynamically added or removed from one or more displays based on their location on the content management system.

This thesis also discusses how various technologies are used together to develop a digital signage system that is remotely accessible and manageable.

KEYWORDS:

PHP, Yii, WebSocket, Raspberry Pi, digital signage.

CONTENTS

FIGURES

TABLES

LIST OF ABBREVIATIONS (OR) SYMBOLS

LAN	Local Area Network
NAT	Network Address Translation
MVC	MVC, which stands for Model-View-Controller, is the software architectural pattern for implementing user interfaces.
HTTP	Hyper Text Transfer Protocol
HTTPS	HTTP Secure
HTTPd	A server program based on Hyper Text Transfer Protocol
NCSA	National Center for Super Computing Application
CRUD	Create Remove Update Delete, CRUD refers to all of the major functions that are implemented on relational database based application
JSON	JavaScript Object Notation
SOC	System on Chip
GPU	Graphical processing unit
API	Application Programming Interface

1 INTRODUCTION

Digital signage is display of text, images or multimedia contents shown in digital formats over the internet or on television (BusinessDictionary.com, 2014). It is based on various methods of using the display devices in ways that are efficient to provide advertisements and information to people in public spaces. The popular approach to digital signage today is that the logical or playback device stores all the content in the storage device and feed it to the displays connected to it (Rouse, 2014; Anon, 2014). The contents are manually uploaded to the storage device or downloaded by the logical device via LAN or internet and saved on the storage device. However, this approach lacks flexibility, as the user has to be physically present at the display locations if any changes are to be made on the logical device. Additionally, if the logical device is behind a NAT router, the contents on the display cannot be managed from a remote location. The implementation of such system is quite difficult if the displays are scattered in a distributed network across different locations. This project aims to solve these problems by creating a digital signage solution based on Raspberry Pi that is accessible and manageable via internet.

1.1 Project Scope

Digital signage technologies can be deployed in many business sectors. However, each business sector has specific needs for the management system of the service. For example from the prospect of management system requirement, the solution used in restaurants to display the menus will be entirely different from the system used in airports to display transit information.

The final output of project will be a digital signage solution for the public transport industry that can be used to display advertisements and other information inside the public vehicles and is remotely manageable from the web

interface. The content providers (hereinafter referred as subscribers) should subscribe on the system based on amount of content and the number of displays before using the system. The administrator can upload, manage and distribute the contents based on the subscription via the web interface. The management system will be capable of distributing the contents to single or group of displays across different locations based on the data provided by the subscribers or any other designated user. The system administrator has full access over the system and can view status, add, update and delete contents from display/displays. The system administrator will also be able to remotely shutdown, reboot and remotely login to the displays and make configuration changes whenever necessary.

2 DEVELOPMENT TOOLS

This chapter introduces and describes different tools and technology used to develop and implement a remotely manageable digital signage solution. Only open source tools and technology are used to reduce the overall cost of the system.

2.1 Debian

Debian is an open source operating system maintained by a group of developers under the Debain project. Most of the work done in debain is under the GNU public license. The minimal version of Debian is used as the operating system for this project server environment. Stable release of Apache (Httpd.apache.org, 2014) with MySQL and PHP is installed on this operating system so that it performs optimally as a web server.

2.2 PHP

PHP is an open source server side scripting language intended for web development. It can be used with numerous web frameworks. It can also be used as general-purpose programming language (Php.net, 2014). PHP version 5.5.9 which was the latest stable version at that time of writing this statement, with the Yii framework is used in this project and is run on Apache web server.

2.2.1 Yii

Yii (acronym for "Yes It Is!") is an open source, object-oriented, component-based MVC PHP application framework. It comes with the automatic code generation tool called gii for CRUD applications (Yiiframework.com, 2014). Yii version 1.1 is used in this project.

2.3 MySQL

MySQL is an open source relational database management system that runs on web servers. It is the second most widely used database system (Db-

engines.com, 2014). MySQL is the ideal choice for applications using PHP and Apache web server as it can run stable on these systems. MySQL version 5.5.38 is used in this project.

2.4 Apache HTTP server

The Apache HTTP server, which is often referred as the Apache server, is an open-source web server application originally based on the NCSA HTTPd server. An open community of developers is involved in developing and maintaining the application under the auspice of the Apache Server Foundation (Wikipedia, 2014). Apache version 2.2.22 has been used in the project.

2.5 Raspbian

Raspbian is an operating system based on the Linux distro Debian. It is a free operating system optimized to run on Raspberry Pi hardware. It is distributed with a set of basic programs and utilities that are required for the Raspberry Pi to run efficiently. The usage of Raspbian operating system has many advantages over its predecessors. It is much faster and stable, is under active development and is considered the preferred operating system for optimal usage of Raspberry Pi by majority of developers system (Raspbian.org, 2014).

2.6 JavaScript

JavaScript is a dynamic programming language, which allows the client side scripts to interact with users. It is commonly used as part of web browsers. All the modern browsers support JavaScript. JavaScript alongside the jQuery framework is used in this system to add interactivity between clients, displays and the management system.

2.6.1 JQuery

JQuery is an open source cross-platform JavaScript library. It was designed to simplify the client side scripting of HTML. Most modern web-browsers support

jQuery. In this project, jQuery has been used alongside the JavaScript and Yii-booster inherently makes extensive use of this library.

2.7 Yii-booster

Yii-booster is an extension developed for the Yii framework based on Twitter-bootstrap framework and JQuery. Twitter-bootstrap is a development tool published by Twitter, which provides templates combining HTML and CSS for the development of the user interface of its web application. Yii-booster is used in this project in order to reduce the time to design the user interface of the system.

2.8 NetBeans

NetBeans is an integrated development environment for developing applications based on JAVA, PHP, C++ and HTML5. NetBeans is written in the JAVA programming language and it supports multiple platforms. In this project, NetBeans is installed on the Windows 8 operating system that is used as a development environment.

3 RASPBERRY PI

Raspberry Pi is a single-board credit card size microcomputer developed by Raspberry Pi foundation, UK. It was developed as an effort to encourage the young people to learn about programming and computer science and use it as educational tool for teaching schoolchildren on these topics (Raspberrypi.org, 2014). The second version of Raspberry Pi also known as Raspberry Pi model B has been used in this project. It is the latest model that was easily available at the time of writing this statement, although model B+ has been released it was not easily available at that time due to the high demand for this model. Model B was chosen as the logical device over model B+ because model B+ did not have significant improvements that would enhance the performance of our system and model B was the only easily available model at that time.

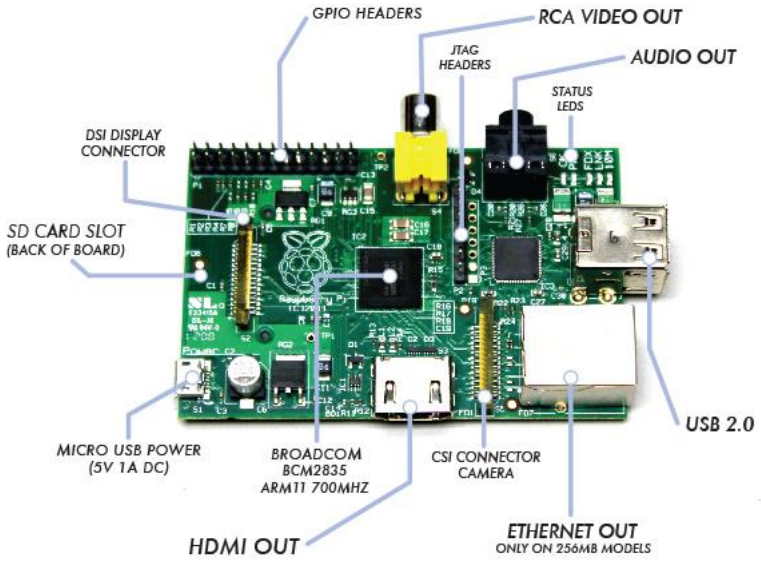


Figure 1. Raspberry Pi model B (Raspberry Pi Model B revision 2.0 Board - 512MB RAM, 2014)

3.1 Hardware and technical specification

The SoC of the Raspberry Pi Model B is Broadcom BCM2835. The SoC contains an ARM11 processor that uses ARMv6 architecture core with floating

point running at 700 MHz and Videocore 4 GPU along with 512 MB SDRAM. Raspberry Pi model B has a composite and HDMI outport on the board, so it can be connected to an old analogue TV through the composite port or to any media that supports HDMI connections (Verry, 2014). It has a 10/100 Mbps RJ45 ethernet port alongside two USB 2.0 ports which can also be used for network connectivity by connecting Wi-Fi dongle or USB internet stick to this port. It has a SD card slot for data storage and operating system. An external SD card with operating system configured on FAT32 partition is required that is plugged in on Raspberry Pi as it does not have onboard storage facility like on modern computers. It is powered through micro-usb port or GPIO pins with 5V dc, 700-1500 mA power supply.

Table 1. Hardware comparison between Model B+ B and A

	Model B+	Model B	Model A
Chip	Broadcom BCM2835 SoC full HD multimedia applications processor	Broadcom BCM2835 SoC full HD multimedia applications processor	Broadcom BCM2835 SoC full HD multimedia applications processor
RAM	512 MB SDRAM @ 400 MHz	512 MB SDRAM @ 400 MHz	256 MB SDRAM @ 400 MHz
Storage	MicroSD	SDCard	SDCard
USB 2.0	4x USB Ports	2x USB Ports	1x USB Port
Power Draw / voltage	600mA up to 1.8A @ 5V	750mA up to 1.2A @ 5V	600mA up to 1.2A @ 5V
GPIO	40	26	26

3.2 Performance

The GPU provides Open GL ES 2.0, hardware-accelerated OpenVG and 1080p30 H.264 high profile decoding, is capable of 1.0 Gpixel/s, 1.5 Gtesrl/s or 24 GFLOPS of general-purpose computing, and features texture filtering and DMA infrastructure (Verry, 2014). With a class 10 SD card and Raspbian operating system, the boot time of the Raspberry Pi is 10-15 seconds. Web performance is outstanding considering the hardware specifications and power consumption data. In a real world scenario, the performance of Raspberry Pi could be compared to 300 MHz Pentium 2 computer. Table 2 shows the web

page loading performance comparison of Raspberry Pi model B with other computer systems.

Table 2. Raspberry Pi web page loading performance comparison (Raspberry model B and other computer systems)

	Raspberry Pi	**1.2GHz Marvell Kirkwood 6281**	**1GHz Allwinner A10**	**1.6GHz Intel Atom 330**	**2.6GHz. G620 Pentium processor**
Pages/Sec	17	25	12	39	174
Mb/Sec	1.1	1.64	0.77	2.5	11.2
Power (W)	3	13	3-4	4	35
Pages/Sec/W	5	2	3	10	5

4 SYSTEM DESIGN

The centralized server provides control over multiple display systems from the web interface. The server comprises of the web server for HTTP connections and a WebSocket server for handling WebSocket connections. A database is used to store the system data and is accessible to both the web server and the WebSocket server. The display system comprises of the Raspberry Pi and a display. Raspberry Pi is the logical device of the display system. The display is connected to the HDMI port of Raspberry Pi. The Raspberry Pi system is divided into system services and front end. System services comprises of the WebSocket client and other dependent systems. The frontend and backend of the system run on the web browser and are responsible for managing and displaying contents on the display. The JavaScript WebSocket client on both frontend and backend runs interactively with web browser. The end user can remotely access the display system via central server. The end–to–end digital signage system integration structure is given in Figure 2.

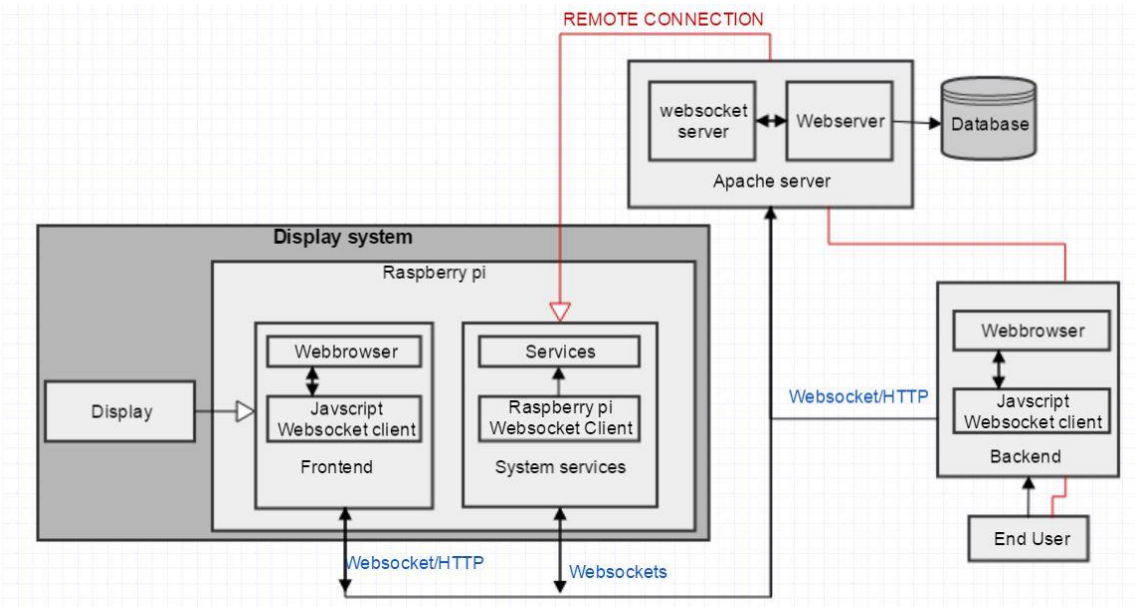


Figure 2. Digital signage system integration

5 DEVELOPMENT OF DIGITAL SIGNAGE SYSTEM

The backend and frontend of the system are developed using the Yii framework, Yii booster, JavaScript and jQuery. System services on Raspberry Pi are developed using the Python programming language. WebSocket is used to enable the real-time monitoring and control of Raspberry Pi and real-time communication between all the modules.

5.1 Raspberry Pi development

5.1.1 Raspberry Pi as display device

Since most of our system heavily depends upon web, the chosen display hardware should be at least capable of the following tasks:

- Connect to internet via a 3G-Dongle connected on USB port
- Run a browser that supports CSS3, HTML 5 and jQuery library
- Render the contents on web browser without significant lag.

Additionally, for optimal performance the chosen hardware should also fulfill the requirements listed below:

- Consume low power and run continuously for more than a day without requiring any human interference.
- Run programs written in the Python programming language.

Although there are many other alternatives that fulfill the requirements, Raspberry Pi is chosen as the suitable hardware for this project as it is the cheapest hardware available on the market and meets the minimum system requirements (Atwell, 2014).

Raspberry Pi feeds the contents to the display connected to it via an HDMI out port. The Raspbian operating system also known as Raspbian Wheezy is installed on the Raspberry Pi. Raspberry Pi will be connected to the internet

using USB 3G Dongle. It uses the Chromium web browser to connect to the content management system running on remote Apache server and load the content on browser window in full screen mode. Interactive content could be displayed on the screens using the HTML5 and JavaScript library.

5.1.2 System services development

The information and contents uploaded by the subscriber are displayed using the web browser running on Raspberry Pi on full screen mode. As a security feature, web browsers are designed to run on Sandbox so that no system level commands could be executed from within web browser. In order to have remote access over Raspberry Pi, the system level services for Raspbian operating system are developed. The Python programming language is used to develop these services.

The system services are mainly responsible for bidirectional communication with the Apache server, receive and execute commands sent by user and send information about the status of Raspberry Pi. Most of the services start when the Raspberry Pi boots up and runs continuously unless manually stopped.

5.1.2.1 Systeminfo.py

Systeminfo.py is a Python class file responsible for generating the system information. It has methods for getting bandwidth usage, temperature, CPU-load, memory and storage information, MAC address and the serial number of Raspberry Pi. This class file is used by other files to get the relevant information about the system. The ‘getData()’ method which returns all the data generated by Systeminfo class is shown in Figure 3.

```
@staticmethod
def getData():
    data = {'uptime': systeminfo.getUptime(),
        'ram':  systeminfo.getRam(),
        'cpu':  systeminfo.getCpuLoad(),
        'cputemp':  systeminfo.getCputemp(),
        'storage':  systeminfo.getStorage(),
        'bandwidth':  systeminfo.getBandwidthUsage(),
        }
    return  data
```

Figure 3. Excerpt of method from systeminfo.py that returns the system data

5.1.2.2 SSH.py

SSH.py is a class file responsible for handling SSH connection commands. It has methods for closing and initiating the SSH tunnel. This class is initiated by the WebSocketClient class whenever the relevant command is sent by the administrator. The excerpt from the class is given in Figure 4.

```
from subprocess import *
import os
class ssh():
    @staticmethod
    def reverseSShinit(port):
        os.system("ssh -N -R 42:localhost:22 -p "+port+" root@149.255.96.181 -f")

    @staticmethod
    def reverseSShend():
        #pid = os.system("pidof ssh")

        pipe = Popen("pidof ssh", shell=True, stdout=PIPE).stdout
        output = pipe.read()

        os.system("kill "+str(output))
```

Figure 4. Excerpt of code from SSH.py

5.1.2.3 WebSocketClient.py

WebSocketclient.py is a Python class file that is responsible for initiating WebSocket connections and other services. It is run as the daemon process when the Raspberry Pi boots up. Daemon (acronym for Disk and Execution Monitor) is a computer program that runs as the background process rather than being under the direct control of interactive users and answers the request for services (Kb.iu.edu, 2014). This class makes extensive use of the Python

WebSocket library (Pypi.Python.org, 2014). It is discussed in detail in the section WebSockets. An excerpt of the code that handles the messages sent from server is shown in Figure 5.

```
def connect(url):
    def on_message(ws, message):

        logger('Message Received '+ url +'\n'+message)
        message = json.loads(message)


        if message['type'] == 'cmd':
            if message['cmd'] == 'shutdown':
                logger('Shutdown on request')
                data  ={'type':'togglestatus','status':'0'}
                ws.send(json.dumps(data))
                os.system("sudo shutdown -h now")

            elif message['cmd'] ==  'reboot':
                logger('Reboot on request')
                data  ={'type':'togglestatus','status':'3'}
                ws.send(json.dumps(data))
                os.system("sudo reboot")

            elif  message['cmd'] ==  'openTunnel':
                ssh.reverseSShinit(message['params'])
                logger('Tunnel opened on port '+message['params'])
                mac = systeminfo.getMac()
                data  ={'type':'ack','ack':'2','info':message['params']}
                ws.send(json.dumps(data))
                #print 'opencon'
        elif message['type']== 'request':
            data = {'type':'response','response':{}}
            data['response'] = systeminfo.getData()
            #print json.dumps(data)
            ws.send(json.dumps(data))
```

Figure 5. Excerpt of method from WebSocketClient.py that handles the massage received from server.

5.1.3 Start-up configuration

As the displays will be displayed in remote locations, there is no possibility for Raspberry Pi to get user input while it boots up. Thus, it is configured to initiate all the necessary services automatically, start a web-browser and load the specified webpage in full screen mode. The specified web page is the front end of our system where the contents are displayed interactively.

5.1.3.1 X Window system

X window system commonly referred as X is an open source client-server system for managing windowed graphical user interface used on UNIX like system such as Linux. Versions of X have also been developed for other operating systems (Siever, 2014). An X session is started by entering the command "startx". “startx” typically runs without command line arguments, but the command line arguments will override its normal behavior. In this project, it gets client arguments from the ‘.xinitrc’ file located on the home directory of root user. ‘.xinitrc’ is a Shell Script file read by xinit and by its frontend startx. The xinit program starts the X server and works as first client program on systems that are not using a display manager.

5.1.3.2 Rc.local

The rc.local file is common to major Linux distributions. It is located on the directory ‘/etc’. It contains the shell commands that should be executed after all the normal system services are started, at the end of the process of switching to a multiuser runlevel. A runlevel is a state of the system, indicating whether it is in the process of booting or rebooting or shutting down, or in single-user mode, or running normally. Shell scripts will be added to this file to ensure that the scripts will run during the startup process.

5.1.3.3 Automatic resolution detection

Raspberry Pi does not detect the correct screen resolution for some displays. Therefore, for such displays the screen resolution has to be set manually. Since the digital signage system will be used with different screens with varying resolutions, some changes have to made on the config.txt file and boot process so that the screen resolution is correctly detected for all the displays without user input. Config.txt is the file where the various system configuration parameters are stored and is read by GPU before the ARM core is initialized. In the config.txt file, the internal framebuffer is set to the largest possible value.

Then monitor capabilities are detected by the system and adjusted accordingly. The corresponding changes made to config.txt file is given in Figure 6.

```
disable_overscan=1
framebuffer_width=1900
framebuffer_height=1200
framebuffer_depth=16
framebuffer_ignore_alpha=1
hdmi_pixel_encoding=1
hdmi_group=2
```

Figure 6. Additional settings added to config.txt file

After making the changes on the config.txt file, the shell script is appended to the rc.local file. The script waits for the display to be attached to the HDMI port, probes for its preferred mode and finally resets the frame buffer ready for X to take over. An excerpt of the script is given in Figure 7.

```
# Wait for the TV-screen to be turned on...
while ! $( tvservice --dumpedid /tmp/edid | fgrep -qv
'Nothing written!' ); do
    bHadToWaitForScreen=true;
    printf "===> Screen is not connected, off or in an
    unknown mode, waiting for it to become
    available...\n"
    sleep 10;
done;

printf "===> Screen is on, extracting preferred
mode...\n"
_DEPTH=16;
eval $( edidparser /tmp/edid | fgrep 'preferred mode' |
tail -1 | sed -Ene 's/^.+(DMT|CEA) \(([0-9]+)\)
([0-9]+)x([0-9]+)[pi]?
@.+/_GROUP=\1;_MODE=\2;_XRES=\3;_YRES=\4;/p' );

printf "===> Resetting screen to preferred mode: %s-%d
(%dx%dx%d)...\n" $_GROUP $_MODE $_XRES $_YRES $_DEPTH
tvservice --explicit="$_GROUP $_MODE"
sleep 1;

printf "===> Resetting frame-buffer to %dx%dx%d...\n"
$_XRES $_YRES $_DEPTH
fbset --all --geometry $_XRES $_YRES $_XRES $_YRES
$_DEPTH -left 0 -right 0 -upper 0 -lower 0;
sleep 1;
```

Figure 7. BASH script added to rc.local file

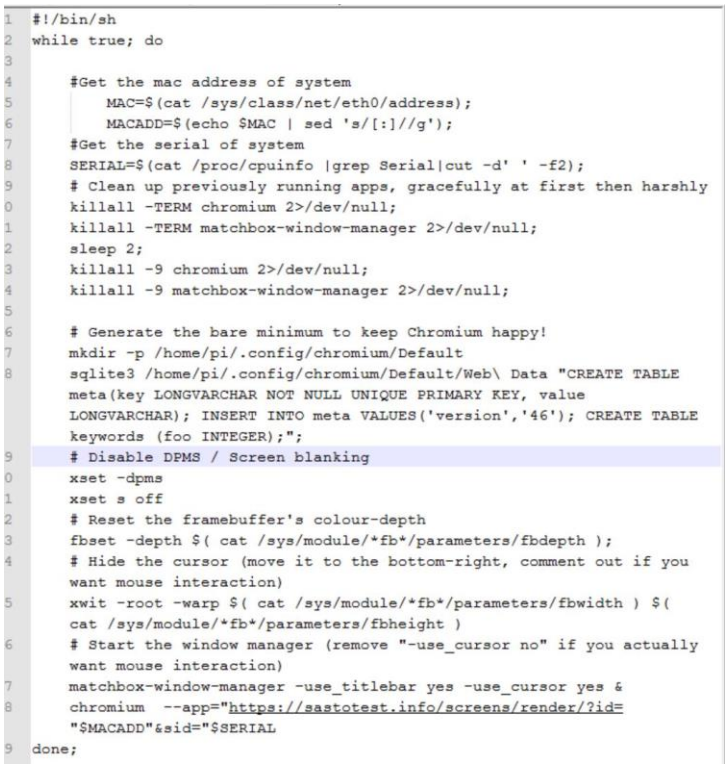
5.1.3.4 Boot to web browser

The X Window system is used in order to enable Raspberry Pi to boot to web-browser. Although there are multiple web browsers available for Raspberry Pi, the chromium web browser is used on our system due to its high level of stability compared to other browsers. In order to start a browser while Raspberry Pi boots, the shell script is added to the rc.local file. The script will start the X server using the tailored '.xintric' file. The '.xinitric' contains the scripts that will start the chromium web browser in full screen mode with url pointing to frontend of our system along with additional parameters for authentication and identification. The script to start the X server is given in Figure 8.

```
if [ -f /boot/xinitrc ]; then
    ln -fs /boot/xinitrc /home/pi/.xinitrc;
    su - pi -c 'startx' &
fi
```

Figure 8. Shell script to start X server.

The Bash scripts are saved on /boot/xinitric so that it is easier to modify on non-Linux machines. The contents of the xinitric file, displayed in Figure 9, will be copied at startup to the .xinitric file.



```
#!/bin/sh
while true; do

    #Get the mac address of system
        MAC=$(cat /sys/class/net/eth0/address);
        MACADD=$(echo $MAC | sed 's/[:]//g');
    #Get the serial of system
    SERIAL=$(cat /proc/cpuinfo |grep Serial|cut -d' ' -f2);
    # Clean up previously running apps, gracefully at first then harshly
    killall -TERM chromium 2>/dev/null;
    killall -TERM matchbox-window-manager 2>/dev/null;
    sleep 2;
    killall -9 chromium 2>/dev/null;
    killall -9 matchbox-window-manager 2>/dev/null;

    # Generate the bare minimum to keep Chromium happy!
    mkdir -p /home/pi/.config/chromium/Default
    sqlite3 /home/pi/.config/chromium/Default/Web\ Data "CREATE TABLE
    meta(key LONGVARCHAR NOT NULL UNIQUE PRIMARY KEY, value
    LONGVARCHAR); INSERT INTO meta VALUES('version','46'); CREATE TABLE
    keywords (foo INTEGER);";
    # Disable DPMS / Screen blanking
    xset -dpms
    xset s off
    # Reset the framebuffer's colour-depth
    fbset -depth $( cat /sys/module/*fb*/parameters/fbdepth );
    # Hide the cursor (move it to the bottom-right, comment out if you
    want mouse interaction)
    xwit -root -warp $( cat /sys/module/*fb*/parameters/fbwidth ) $(
    cat /sys/module/*fb*/parameters/fbheight )
    # Start the window manager (remove "-use_cursor no" if you actually
    want mouse interaction)
    matchbox-window-manager -use_titlebar yes -use_cursor yes &
    chromium  --app="https://sastotest.info/screens/render/?id=
    "$MACADD"&sid="$SERIAL
done;
```

Figure 9. Xinitric file

5.2 WebSocket

WebSocket is a bidirectional communication protocol between client and server making use of persistent TCP connection. The WebSocket protocol was standardized by the IETF as RFC 6455 in 2011, and the WebSocket API in Web IDL is being standardized by the W3C. WebSocket was originally designed to be implemented in web browsers and web servers but it can be used with any client and server that use TCP connection to communicate (WebSocket.org, 2014).

The HTML5 WebSocket specification (W3.org, 2014) defines API that enables web pages to use WebSocket to enable communication with the remote host. The API accounts for proxies and firewalls on the connection path, this makes bidirectional communication possible over any connection (WebSocket.org, 2014).

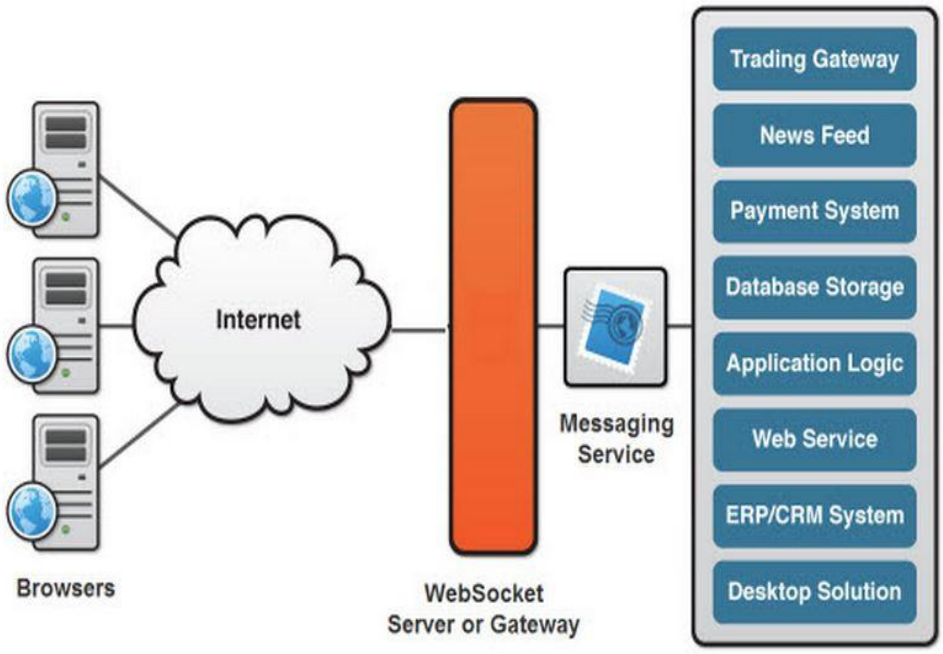


Figure 10. WebSocket Implementation architecture

The WebSocket protocol was designed to be implemented with existing web technologies. As a part of this design principle, the connection is initiated as HTTP connection, which later switches to WebSocket protocol. This guarantees the backward compatibility with the applications that do not use WebSockets. The switch from HTTP to the WebSocket protocol is referred as WebSocket handshake. The client application sends the request through the HTTP upgrade header indicating that the connection should be switched to WebSocket.

```
GET ws://echo.websocket.org/?encoding=text HTTP/1.1
Origin: http://websocket.org
Cookie: __utma=99as
Connection: Upgrade
Host: echo.websocket.org
Sec-WebSocket-Key: uRovscZjNol/umbTt5uKmw==
Upgrade: websocket
Sec-WebSocket-Version: 13
```

Figure 11. Upgrade header sent by client

To connect to an WebSocket server, a new WebSocket instance is created providing the new object with a URL that represents the WebSocket server to

be connected, ws:// and wss:// prefix are used to indicate a websockocket and a secure WebSocket connection, respectively(WebSocket.org, 2014).

```
var myWebSocket = new WebSocket("ws://www.websockets.org");
```

Figure 12. Creating WebSocket instance in JavaScript

If the server supports the WebSocket protocol, it agrees to the request and switches to the WebSocket through the upgrade header. At this point, the HTTP connection is replaced by the WebSocket over the same TCP/IP connection. Once the connection is established, the client and server can send the data frames in full duplex mode. The WebSocket connection uses the same port as HTTP by default.

```
HTTP/1.1 101 WebSocket Protocol Handshake
Date: Fri, 10 Feb 2012 17:38:18 GMT
Connection: Upgrade
Server: Kaazing Gateway
Upgrade: WebSocket
Access-Control-Allow-Origin: http://websocket.org
Access-Control-Allow-Credentials: true
Sec-WebSocket-Accept: rLHCkw/SKsO9GAH/ZSFhBATDKrU=
Access-Control-Allow-Headers: content-type
```

Figure 13. Upgrade header sent by server

The applications based on WebSocket implements the event driven functions for connection open, close, message and error on both client and server side. The corresponding syntax for these events differs according to the programming language and library used.

In order to enable communication between Raspberry Pi and the server at system level, a WebSocket client is developed for Raspberry Pi. The HTML 5 WebSocket API is utilized across the system using JavaScript to add

interactivity between user and display system. The WebSocket server is developed using PHP that runs on the Apache server alongside the content management system. The server listens to port 8080 for incoming WebSocket connections. The WebSocket server runs independent of Yii framework, so the WebSocket client that runs on the server side is developed for integrating the Yii framework with WebSocket server.

5.2.1 Raspberry Pi Client

The WebSocket client referred as Raspberry Pi client is developed for Raspberry Pi making use of the Python WebSocket library. It is implemented on the class WebSocketClient.py. It is responsible for handling the request, commands sent by the server and replying with requested information or acknowledgement of the message received. It is started as a daemon service at startup by modifying rc.local file. When the connection request is initiated additional header parameter specifying the mac-address and serial number of Raspberry Pi is added which is used for security verification.

```
ws = websocket.WebSocketApp(url,
                            on_message = on_message,
                                        on_error = on_error,
                                        on_close = on_close,
                            header = ["macaddress:"+mac,"serial:"+serial])
ws.on_open=on_open
ws.run_forever(sslopt={"check_hostname": False,"cert_reqs": ssl.CERT_NONE})
```

Figure 14. Excerpt from WebSocketClient.py demonstrating WebSocket connection initiation

On the Python WebSocket library ‘on_open’, ‘on_close’, ‘on_message’ and ‘on_error’ corresponds to open, close, message and error event functions, which should be overridden as per the application requirement. A connection attempt is automatically made after waiting 30 seconds if there is error in connection or the connection is closed by the server. This ensures that the Raspberry Pi is always accessible even after the connection is reset or closed due to connection errors.

```
def on_error(ws, error):
    logger('Socket Closed with error '+ url,error)

    #if the connection closes with error thaen wait 30 seconds  nad try connecting again
    time.sleep(30)
    connect(url)

def on_close(ws):
    logger('Socket Closed '+ url +'\n')
    time.sleep(30)
    connect(url)
    #if the connection is closed  then wait 30 seconds and try again
```

Figure 15. Excerpt from WebSocketClient.py demonstrating actions on error and close events.

5.2.2 Php Client

Php client is the WebSocket client implementation with PHP using the Php WebSocket library. The Yii framework does not have implementation of WebSocket on its API, so the Php client is developed as an intermediate application between the WebSocket server and the Yii framework. On the events where the application based on the Yii framework needs to pass arguments to the WebSocket server running on the same machine, the Php client initiates the connection and sends the arguments as message to the WebSocket server. After the connection is initiated, it runs continuously which forces the remaining script to wait until the client is closed. Therefore, to ensure the remaining script continues to run after the event, the Php client immediately closes the connection after the message has been sent.

```
public static function pushToscreen($screenId,$params){
    $WebSocketClient = new WebsocketClient('localhost', 8080);
    $data = json_encode(array('type'=>'push','push'=>$params,'Screens'=>$screenId));
    echo $WebSocketClient->sendData($data);
    unset($WebSocketClient);

}
```

Figure 16. Php function using Php WebSocket client

5.2.3 JavaScript Client

The JavaScript Client is a JavaScript implementation of the HTML5 WebSocket. It is used on the front end of the digital signage system to update the contents on real time and on backend to get the real-time information of the Raspberry

Pi. JavaScript clients on both frontend and backend communicate with the server that does the task of relaying the information between frontend and backend. Similar to the Raspberry Pi client, reconnection attempts are made at certain intervals in the event of error or connection closed by either side, this ensures the persistent connection between Websocket server and JavaScript client.

```
function createWebSocket(){
    var ws = new WebSocket("wss://sastotest.info:9000/");

     ws.onopen = function()
     {
         console.log('connected sucessfully')
          attempts = 3;
          setInterval(function(){
                ws.send('1');
            }, 3000);

    };

    ws.onmessage = function(e) {
         var msg = JSON.parse(e.data);
         if(msg.type==='add'){
            var div =$('<div id ='+ msg.add.Id +'><img src="'+msg.add.source +'"/> </div>');
            $('#slideshow').append(div);
         }
         else if(msg.type==='remove'){
              var id = msg.remove.Id;
             $( '#'+id).remove();

         }

    };
    ws.onclose = function () {
    var time = generateInteval(attempts);
    setTimeout(function () {
        // We've tried to reconnect so increment the attempts by 1
        attempts++;

        // Connection has closed so try to reconnect every 10 seconds.
        createWebSocket();

    }, time);
```

Figure 17. Implementation of JavaScript WebSocket client on frontend

5.2.4 Php WebSocket server

The Php WebSocket server is a standalone WebSocket server developed using ratchet. Ratchet is a PHP library which provides tools to create a bidirectional communication between client and server using WebSocket and is compatible with all the modern browsers that support WebSocket (Boden, 2014). The WebSocket server is implemented using a shell script that starts the WebSocket server listening to a specified port and a PHP class that handles all the application logic. The shell script is stored on the file wsserver.Php. The shell

script is launched from the command line which starts the WebSocket server listening to the given port and implements the application logic class.

```php
<?php
use Ratchet\Server\IoServer;
use Wshandler\SocketController;

    require dirname(__DIR__) . '/vendor/autoload.php';

    $server = IoServer::factory(
        new SocketController(),
        8080
    );

    $server->run();
?>
```

Figure 18. Shell script that starts WebSocket server listening to port 8080 and implements application logic class SocketController.

```
root@181:~# php /var/www/a/protected/wsserver.php
```

Figure 19. Starting WebSocket server from command line.

5.2.4.1 SocketController.Php

SocketController.Php is the application logic class file that is implemented by the WebSocket server. It is responsible for all the application logic related to WebSocket. It implements the MessageComponentInterface class from the ratchet library and overrides ‘onOpen’, ‘onMessge’, ‘onError’ and ‘onClose’ event-driven functions as per the application logic. It tracks all the incoming connection and stores it in an array variable in order to send data on those connections whenever required. For each incoming connection, it verifies the source client and stores the connection for Raspberry Pi client, frontend JavaScript client and backend JavaScript client on three separate array variables. The Raspberry Pi client sends the mac-address value as header during connection initiation process. Each connection has a unique resource Id. The resource id is mapped to mac-address of the Raspberry Pi for Raspberry Pi

Client and to screenId for the frontend JavaScript client. The ScreenId is a unique id stored in the database generated while the Raspberry Pi is registered in the system. Therefore, each connection can be uniquely identified and the data is sent to the related device only instead of broadcasting to all the devices. However, the backend JavaScript client is not uniquely identified, thus the message is broadcasted to all the backend JavaScript clients if any event occurs on Raspberry Pi.

5.2.5 Communication

For the efficient handling of data between server and multiple clients on different platforms, a common data type and structure has to be adopted. The digital signage system uses the JSON data format tailored into specific structure for this purpose. Each client encodes the data into JSON format before sending it to the server. The server receives and decodes the data and adds additional parameters if required depending upon the data type. It encodes the data to JSON format and forwards it to the appropriate client. Each client decodes and analyses the data and performs the specific set of actions depending upon data type. The common structure for data is shown in Figure 20.

```
Generic data format for client-server or server - FrontEnd JavascriptClient,Raspberry pi Client communication
{ "type": "DataType", "Data": "Data--Data could be string or JSON depending upon DataType" }
Data format for BackEnd JavascriptClient - server for DataType Request and cmd
{ "type": "Request", "cmd": "Command to be exexuted to get requested data", "MacAdd":"MacAddress of raspberryPi" }
{ "type": "cmd", "cmd": "Command to be executed","MacAdd":"MacAddress of raspberryPi" }
Data format for Server to communicate with backend Javscript Client
{ "type": "DataType","Data": "Data","MacAdd":"MacAddress of raspberryPi" }
Acknowledgment data format
{ "type": "ack", "ack":"data", "MacAdd": "MacAddress of raspberry pi", "params": "additional parameters if any" }
```

Figure 20. JSON data structure for digital signage system

When the Raspberry Pi client is connected to server, the server broadcasts the message to backend JavaScript Clients with ‘type’ attribute set to ‘togglestatus’ and ‘data’ attribute set to ‘2’.Similarly, when the Raspberry Pi client is

disconnected from the server, it broadcasts the message to all backend JavaScript Clients with data set to ‘0’ for connection closed due to shut down, ‘1’ for network error and ‘3’ for restart. The ‘MacAdd’ attribute is always set to mac address of the Raspberry Pi from which the message was received. When the backend JavaScript client receives the message, it notifies the user about the change in status accordingly depending upon the received data.

The backend JavaScript client requests the information when the user navigates to the management panel of the Raspberry Pi or sends the control commands to that device. The data is sent to server with ‘type’ set to ‘Request’ or ‘cmd’. If ‘type’ attribute is set to ‘cmd’, the ‘cmd’ attribute is set to ‘shutdown’, ‘reboot’ or ‘opentunnel’. if the ‘type’ attribute is set to ‘Request’, the ‘cmd’ attribute is set to ‘getsysteminfo’. The ‘MacAdd’ attribute is set to mac-address of the target Raspberry Pi in both cases. When the server receives the data, it forwards the data to the specified Raspberry Pi by resolving the mac-address specified in the ‘MacAdd’ attribute with the connected mac-addresses. The server then sends the acknowledgment data to the client. The structure of acknowledgement data is shown in Figure 20. If the mac-address is found and the command is sent, the ‘ack’ attribute is set to ‘1’, otherwise the ‘ack’ attribute is set to ‘0’. If the ‘type’ attribute is ‘Request’, the corresponding Raspberry Pi responds with ‘type’ attribute set to ‘Response’ and the ‘data’ attribute is set to JSON data, otherwise if the ‘type’ attribute was set to ‘cmd’, the corresponding Raspberry Pi responds with acknowledgement data with the ‘ack’ attribute set to respond to the command received. The server will set the ‘MacAdd’ attribute to the mac-address of that device and broadcast it to the backend JavaScript Client. The backend client will receive the data and notify the user of the received information.

5.3 Remote management

5.3.1 Secure Shell and reverse tunneling

Secure shell (SSH) is a cryptographic network protocol that allows data to be exchanged between two networked devices over a secure channel (Wiki.archlinux.org, 2014). SSH is commonly used for remote command-line login, remote command execution and other secure network services. SSH uses public key cryptography to encrypt and decrypt the transmitted information. An SSH server listens on the standard TCP port 22 by default. An SSH client application is used for establishing connections to an SSHd daemon accepting remote connections.

While using the 3G service to connect to a network, in most cases the connection is behind the NAT router and provides the private IP address. As a result, it is impossible to establish a SSH connection to that device unless the NAT router is configured to allow such connection attempt. Reverse SSH tunneling is a technique to connect to a remote machine behind a firewall or a NAT router via SSH . In a normal SSH connection, a SSH client connects to a SSH server through the server's open port, but in the case of a reverse connection, the client opens the port that the server connects to (Chamith, 2012). The network structure and the reverse tunnel implementation for the digital signage system is given in Figure 21.

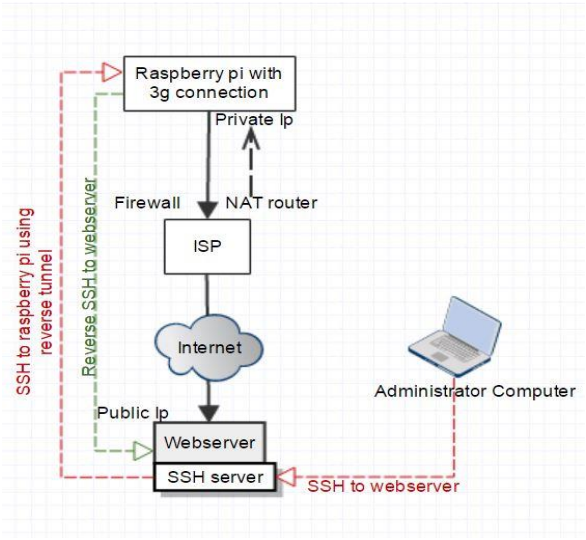


Figure 21. Network structure and reverse tunnel implementation.

Reverse SSH tunneling is used with the WebSocket service in order to establish the SSH connection on demand to Raspberry Pi. The backend JavaScript client sends the data to Php WebSocket server with the ‘type’ attribute set to ‘cmd’ and the ‘cmd’ attribute set to ‘opentunnel’. The Php server, after decoding the received data, finds the open port on Apache and appends the ‘params’ attribute set to the open port before encoding and forwarding it to the Raspberry Pi. If the data is successfully sent, the Php server sends the acknowledgement data to the backend JavaScript client with the ‘ack’ attribute set to ‘1’, the ‘MacAdd’ attribute set to the mac-address of Raspberry Pi and the ‘params’ attribute set to the open port. When the Raspberry Pi receives the data, the SSH.py class is initiated which establishes the reverse tunnel on the port specified by ‘params’ attribute. After initiating the reverse SSH connection, the Raspberry Pi sends the acknowledgement data to the server, which is then forwarded to the backend JavaScript client. The SSH.py class is given in Figure 5 and the corresponding code for the process is given in Fgure 22. The flowchart for the process is given in Figure 23.

```
elif  message['cmd'] ==  'openTunnel':
        ssh.reverseSShinit(message['params'])
        logger('Tunnel opened on port '+message['params'])
        mac = systeminfo.getMac()
        data  ={'type':'ack','ack':'2','info':message['params']}
        ws.send(json.dumps(data))
```

Figure 22. Excerpt from WebSocketClient.py demonstrating SSH request handling

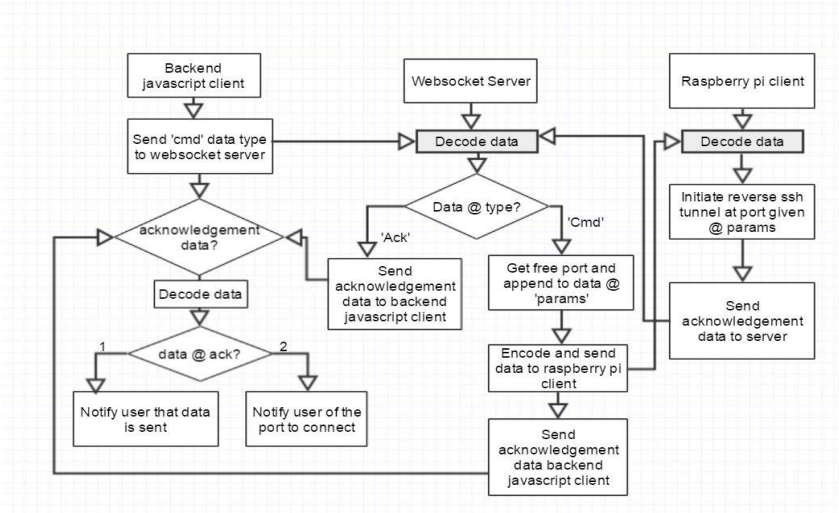


Figure 23. Flowchart for establishing reverse tunnel

The backend JavaScript client notifies the user of the port to be used to connect to the Raspberry Pi. Then the authenticated user can SSH to the Apache server and further SSH to the Raspberry Pi at the given port to access the terminal of Raspberry Pi.

5.3.2 Managing contents

When the content needs to be changed, the Php client sends data to server with the ‘type’ attribute set to ‘Push’. The ‘Screens’ is set to JSON data with the ‘Add’ and ‘Remove’ attributes with values set to list of screenId. The ‘data’ attribute is set to the url of the content. The server decodes the received data and sends the data to the corresponding frontend JavaScript client by resolving

the connection with the list of screenIds set on the received data. If the screenId is the value inside the ‘Add’ attribute, the ‘type’ attribute is set to ‘Add’, else if the screenId id is thevalue inside the ‘Remove’ attribute, the type is set to ‘Remove’. The ‘data’ attribute is set as it is without any changes in both cases. Based on the ‘type’ attribute of the received data, the frontend JavaScript client adds or removes the content from the display. The flowchart for the content management is given in Figure 24. Figure 25 shows the excerpt of the code from WebSocket server, responsible for content management.

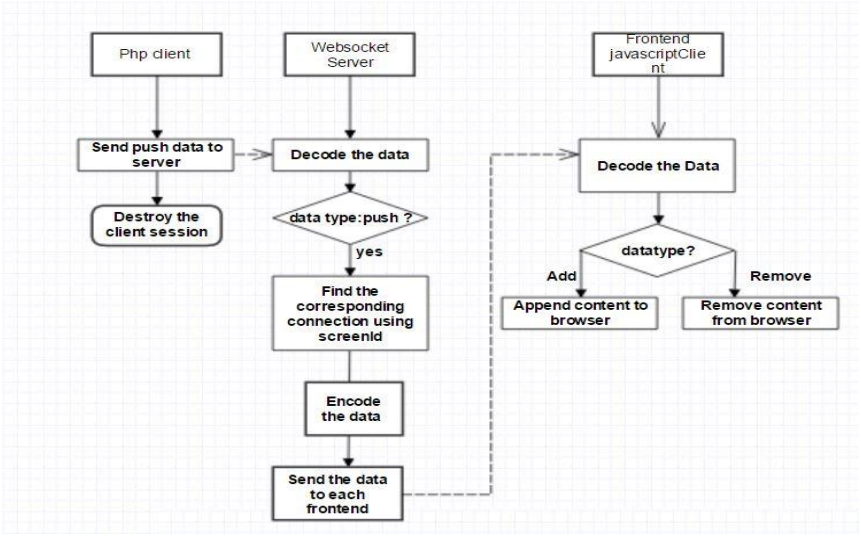


Figure 24. Flowchart for managing content

```
elseif($msg['type']=='push'){
    $from->close();
    if(array_key_exists('remove',$msg['Screens'])){
        $removefromScreens = $msg['Screens']['remove'];
        $data= array('type'=>'remove','remove'=>$msg['push']['data']);
            foreach($removefromScreens as $key=>$value){

                if(isset($this->ripi_browser[$value])){
                   $this->ripi_browser[$value]->send(json_encode($data));
                   echo 'Sent remove to Screenid '.$value.PHP_EOL;
                }
            }

    }
    if(array_key_exists('add',$msg['Screens'])){
        $addtoScreens = $msg['Screens']['add'];
        $data= array('type'=>'add','add'=>$msg['push']['data']);
            foreach($addtoScreens as $key=>$value){
                if(isset($this->ripi_browser[$value])){
                   $this->ripi_browser[$value]->send(json_encode($data));
                   echo 'Sent add to Screenid '.$value.PHP_EOL;
                }
            }

    }
```

Figure 25. Excerpt from SocketController.Php

5.4 Backend development

The backend of the digital signage system comprises of the interrelated systems for management of the displays, subscribers and contents.

5.4.1 Display management

For the ease of management of displays in different vehicles, each vehicle is registered on the system based on the routes they operate. Each display is assigned to a route and content can be added or modified to all the displays belonging to that route with a single action. The information about the route can be viewed and managed from the dashboard. The dashboard for the management of route is displayed in Figure 26.

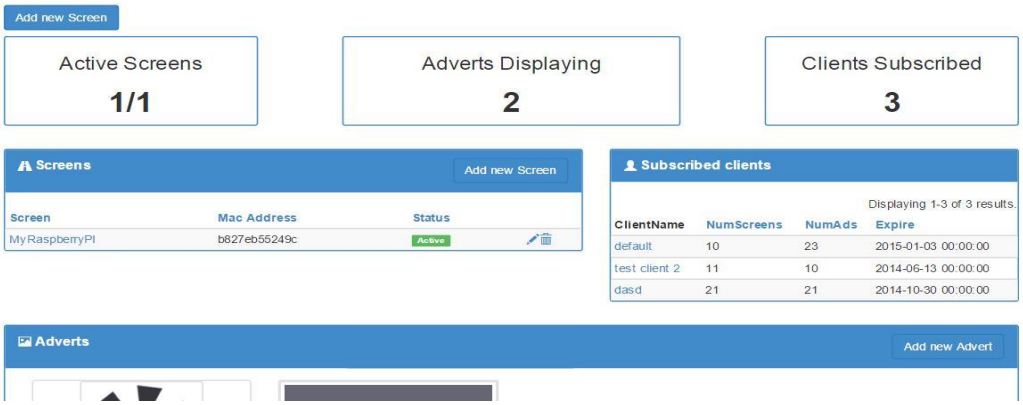


Figure 26. Dashboard for route management

Apart from the route, each display can also be managed through the display dashboard as shown in Figure 27. If the display is connected to the server, the real-time information of the Raspberry Pi is fetched when the display dashboard is loaded. The Raspberry Pi can be shut down, restart or enabled to accept the SSH connection with control buttons built into the dashboard. The content being displayed on the display can also be modified from the same interface.

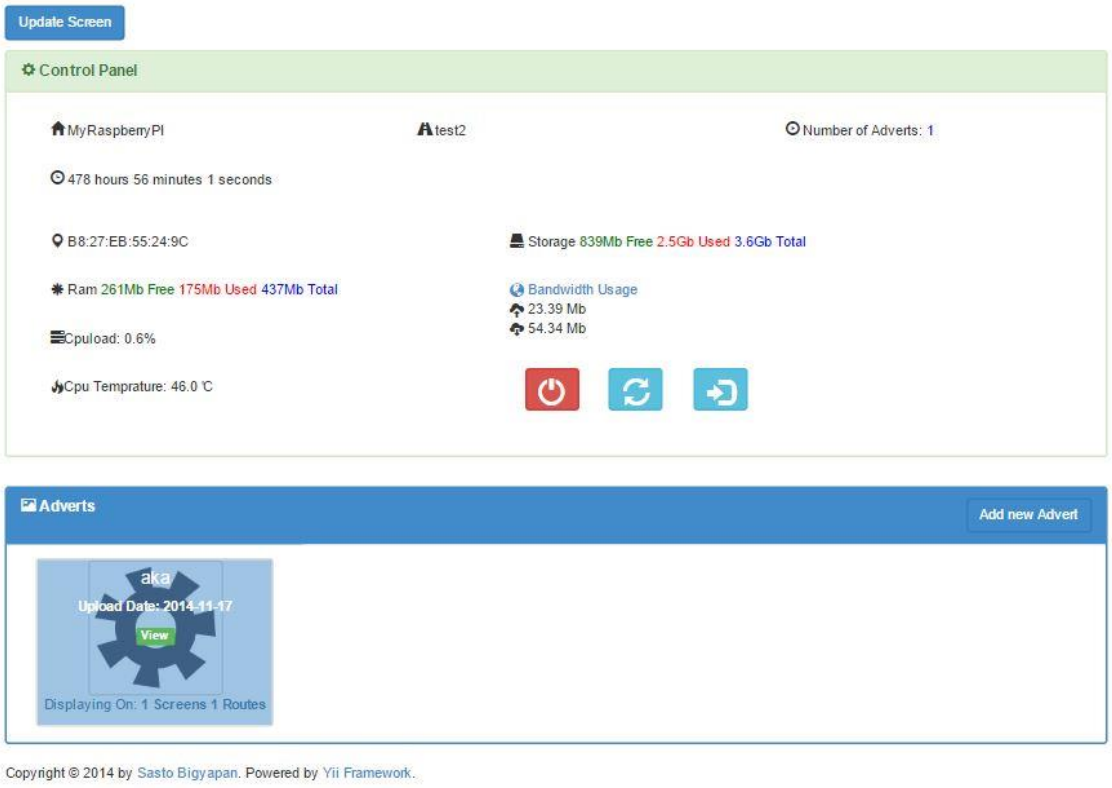


Figure 27. Dashboard for managing individual display

5.4.2 Subscriber management

The content providers for the digital signage system have to subscribe to the system based on number of displays and content. The information about each subscriber can be managed through the subscriber dashboard as shown in Figure 28. The subscription data is used by other modules to restrict or allow subscribers access to add contents on displays.

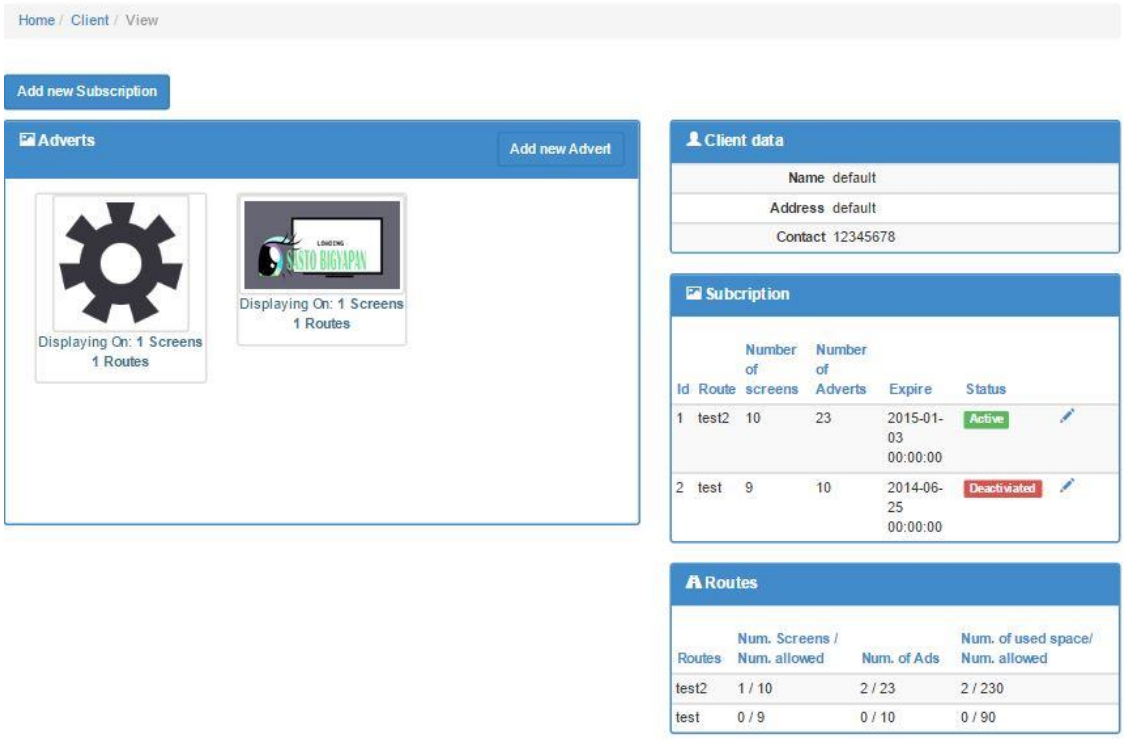


Figure 28. Dashboard for managing subscriber

5.5 Frontend development

The front end of the digital signage system is responsible for presentation of the subscriber's uploaded contents on the display. The uploaded contents are displayed in loop. The web browser on the Raspberry Pi is pointed to the url of the front end. The JavaScript WebSocket client is implemented in frontend in order to enable the remote management of the contents. Apart from the contents uploaded by the subscribers, the frontend also displays the news and weather information on real time.

The news service is loaded on the bottom of the display and is updated every 15 minutes via an ajax call. This is implemented using the RSS service from the news provider. The weather service is loaded on the left of the display. The data required for the weather service is fetched using the API provided by

openweathermap.org. The weather data is updated automatically at one-hour interval. The screen-shot of the frontend service is given in figure 29.

Figure 29. Screen-shot of the frontend.

6 SECURITY

6.1 Communication security

It is crucial for the digital signage system that the communication between client and server is secure and reliable. HTTPS is implemented on the system so that the communication between frontend, backend and server is always encrypted. The WebSocket communication is secured by using wss:// prefix on the URL referring to WebSocket server. The use of wss:// ensures that the communication between client and server is encrypted.

The Apache server listens to port 443 while using the HTTPS connection. Connection attempts to any other port will be automatically rejected while using HTTPS. The WebSocket will be running on the different port than 443, so Apache is configured to allow WebSocket to connect to another port over the same HTTPS connection after connection upgrade. The Stunnel library is used to configure the Apache server for secure WebSocket communication. The stunnel is a program that is designed to work as an SSL encryption wrapper between remote client and local or remote server. Stunnel uses the OpenSSL library for cryptography, so it supports the cryptographic algorithms that are compiled into the library (Stunnel.org, 2014). Stunnel can be installed by using the command “apt-get install stunnel4 -y”. Stunnel configures itself using the file named stunnel.conf, which is by default located on “etc/stunnel” directory. The configuration for stunnel is given in Figure 30.

```
cert = /etc/apache2/ssl/ssl.crt
key = /etc/apache2/ssl/private.key
pid = /stunnel4.pid
[websockets]
accept = 9000
connect = 8080
```

Figure 30. . Excerpt from stunnel.conf file

As shown in Figure 30, the cert parameter is the path to the SSL certificate file and the key parameter is the path to the SSL private key file. The accept parameter is the port to which the WebSocket clients should connect to. The connection is then forwarded to the port specified on the connect parameter.

6.2 Authentication

Any client trying to access the system or WebSocket server is authenticated before giving access to the system. Different authentication algorithms are implemented for different types of clients.

6.2.1 Administrator authentication

The administrator is responsible for managing the content and displays on the system. When the administrator account is activated, the administrator can choose a unique username and password. The username and password could be used to authenticate the administrator. If the authentication of the administrator succeeds, the session id of the session is stored in the database. The session id is used for the JavaScript WebSocket client authentication.

6.2.2 Raspberry Pi client authentication (WebSocket)

The serial number and mac-address of the Raspberry Pi is added to the database along with the corresponding screenId while the displays are registered on the system. The Raspberry Pi client is configured to add the serial number and mac address of the Raspberry Pi on HTTP upgrade header while the connection is initiated. The serial number of Raspberry Pi is unique to each Raspberry Pi. Additionally, the url pointing to the frontend of the system should also contain the mac-address and serial number of Raspberry Pi. The WebSocket server checks if the serial number and mac-address pair data exist on the record in MySQL database. If the record with the serial number and mac-address pair data exists on the database, the Raspberry Pi client is given access to the system else the connection is dropped. This ensures only the display registered on the system can connect to the server.

6.2.3 JavaScript client authentication (WebSocket)

When the Javascript client sends the connection request, by default, the cookie information of the session is also transmitted. The cookie information consists of the session id that was stored in database while the administrator logged in. When the WebSocket server receives the request, it checks if the transmitted session id on the cookie exists on the record in MySQL database. If the record with the session id exists in the database, the JavaScript client is given access to the WebSocket server else, the connection is terminated. The flowchart for authentication is given in Figure 31.

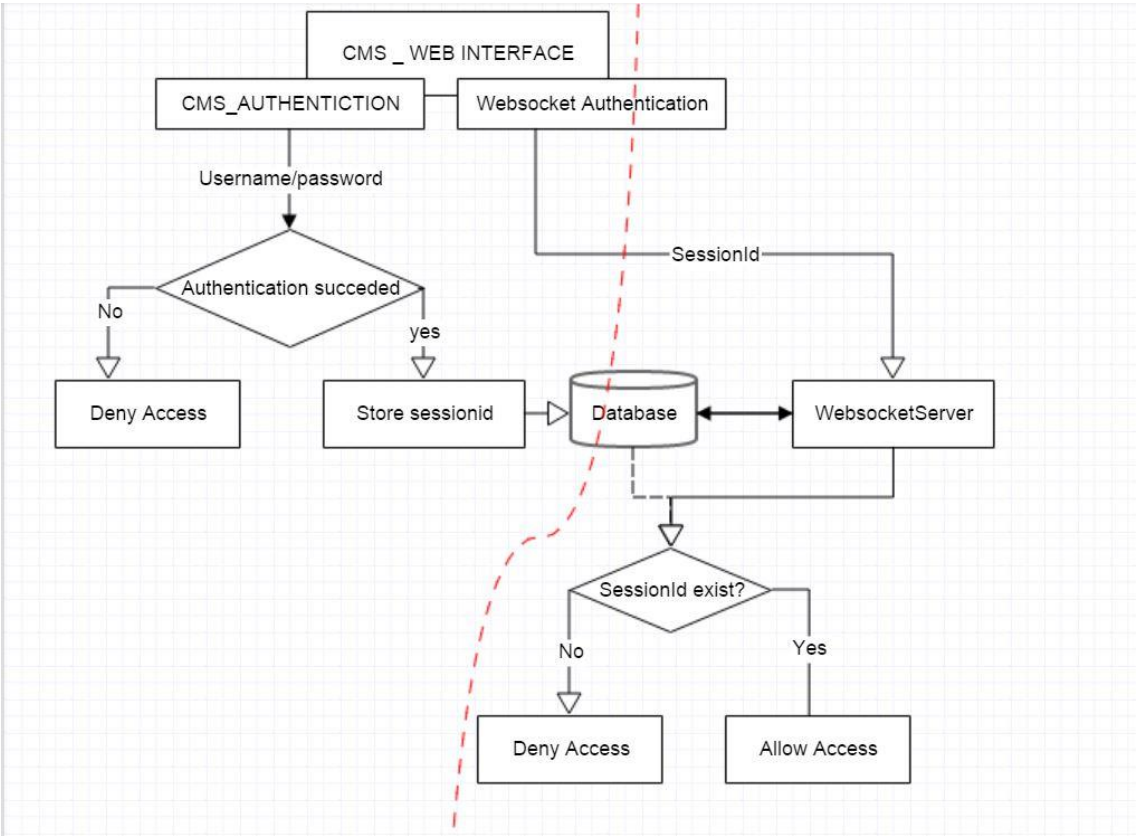


Figure 31. Flowchart for user authentication

7 CHALLENGES

7.1 Data corruption

Raspberry Pi uses an SD card as a storage device. If the raspberry is unplugged from the power source without a proper shutdown procedure or if the SD card is removed from the Raspberry Pi while the Raspberry Pi is still running, it is very likely that the data on the SD-card will be corrupted. The SD card is prone to data corruption if not removed safely from the system. The Raspberry Pi will be unable boot from the corrupted SD card. In order to reduce the chances of data corruption, Raspberry Pi should always be connected to the reliable power source with the backup source in case of power failure.

7.2 Physical security

As the digital signage system is designed to be implemented on remote locations, there lies the risk of vandalism, theft, and other physical security risk to Raspberry Pi and the display. The system should be installed on the places where the constant supervision is possible or should be enclosed inside the secured protective case to prevent the physical damage or theft of the system and components.

7.3 Reliability of internet connection

As the system retrieves all the contents from the central server via internet, the reliability of the internet service is extremely crucial for the operation of the system. While using the 3G service for internet connectivity, there is always a risk of losing the internet connectivity depending upon the location of the device on the 3G service coverage area and the quality of the service provided. The system cannot download the contents and the remote access of the system is not possible if there is a drop in the 3G connection.

8 CONCLUSION

The aim of this thesis was to design and implement a digital signage system that is remotely accessible and manageable. WebSocket technology with Python, Php and JavaScript has been implemented together on Raspberry Pi and webserver to achieve the desired result. The content management system of the digital signage is designed with focus on managing the content distribution on the public transport industry. Each display system can be remotely managed and controlled via web interface. The approach of controlling the display system via a web interface over the internet could be useful for implementation of other similar applications that require remote access and monitoring.

REFERENCES

Anon, (2014). [online] Available at: http://www.wirespring.com/pdf/intro_to_digital_signage.pdf [Accessed 6 Nov. 2014].

Atwell, C. (2014). News: Raspberry Pi vs. The World: compare and compare and contrast the competition | element14. [online] Element14.com. Available at: http://www.element14.com/community/community/news/blog/2013/07/26/raspberry-pi-vs-the-world-compare-and-contrast-the-competition [Accessed 7 Nov. 2014].

Boden, C. (2014). Ratchet - What is a WebSocket?. [online] Socketo.me. Available at: http://socketo.me/docs/ [Accessed 21 Nov. 2014].

BusinessDictionary.com, (2014). What is digital signage? definition and meaning. [online] Available at: http://www.businessdictionary.com/definition/digital-signage.html [Accessed 17 Nov. 2014].

Chamith, B. (2012). SSH Tunneling Explained. [online] Source Open. Available at: http://chamibuddhika.wordpress.com/2012/03/21/SSH-tunnelling-explained/ [Accessed 7 Oct. 2014].

Db-engines.com, (2014). DB-Engines Ranking - popularity ranking of database management systems. [online] Available at: http://db-engines.com/en/ranking [Accessed 17 Nov. 2014].

Hoeven, R. (2014). Raspberry Pi performance. [online] Freedomboxblog.nl. Available at: http://freedomboxblog.nl/raspberry-pi-performance [Accessed 17 Nov. 2014].

Httpd.apache.org, (2014). Apache HTTP Server Version 2.2Documentation - Apache HTTP Server Version 2.2. [online] Available at: http://httpd.apache.org/docs/2.2/ [Accessed 25 Oct. 2014].

Kb.iu.edu, (2014). In Unix, what is a daemon?. [online] Available at: https://kb.iu.edu/d/aiau [Accessed 17 Nov. 2014].

Mysql.com, (2014). MySQL :: The world's most popular open source database. [online] Available at: http://www.mysql.com/ [Accessed 17 Nov. 2014].

Php.net, (2014). PHP: What is PHP? - Manual. [online] Available at: http://Php.net/manual/en/intro-whatis.Php [Accessed 17 Nov. 2014].

Pypi.Python.org, (2014). WebSocket-client 0.21.0 : Python Package Index. [online] Available at: https://pypi.Python.org/pypi/WebSocket-client/ [Accessed 28 Aug. 2014].

Python.org, (2014). Welcome to Python.org. [online] Available at: https://www.Python.org/ [Accessed 16 Nov. 2014].

Raspberry Pi Model B revision 2.0 Board - 512MB RAM. (2014). [image] Available at: http://www.arduiner.com/en/home/2333-raspberry-pi-model-b-revision-20-board-512mb-ram.html [Accessed 7 Nov. 2014].

Raspberrypi.org, (2014). What is a Raspberry Pi? | Raspberry Pi. [online] Available at: http://www.raspberrypi.org/help/what-is-a-raspberry-pi/ [Accessed 17 Nov. 2014].

Raspbian.org, (2014). FrontPage - Raspbian. [online] Available at: http://www.Raspbian.org/ [Accessed 7 Nov. 2014].

Rouse, M. (2014). What is Digital Signage (dynamic signage)? - Definition from WhatIs.com. [online] whatis.techtarget.com. Available at: http://searchcrm.techtarget.com/definition/digital-signage [Accessed 6 Nov. 2014].

Siever, E. (2014). What Is the X Window System - O'Reilly Media. [online] Linuxdevcenter.com. Available at: http://www.linuxdevcenter.com/pub/a/linux/2005/08/25/whatisXwindow.html [Accessed 8 Nov. 2014].

Stunnel.org, (2014). stunnel: Home. [online] Available at: https://www.stunnel.org/index.html [Accessed 11 Nov. 2014].

Verry, T. (2014). What is the Raspberry Pi? | ExtremeTech. [online] ExtremeTech. Available at: http://www.extremetech.com/computing/124317-what-is-raspberry-pi-2 [Accessed 20 Oct. 2014].

W3.org, (2014). The WebSocket API. [online] Available at: http://www.w3.org/TR/websockets/ [Accessed 24 Oct. 2014].

WebSocket.org, (2014). WebSocket.org | About WebSocket. [online] Available at: https://www.WebSocket.org/aboutWebSocket.html [Accessed 12 Nov. 2014].

Wiki.archlinux.org, (2014). Secure Shell - ArchWiki. [online] Available at: https://wiki.archlinux.org/index.Php/Secure_Shell [Accessed 19 Nov. 2014].

Wikipedia, (2014). Apache HTTP Server. [online] Available at: http://en.wikipedia.org/wiki/Apache_HTTP_Server [Accessed 21 Nov. 2014].

Yiiframework.com, (2014). Yii PHP Framework: Best for Web 2.0 Development. [online] Available at: http://www.Yiiframework.com/ [Accessed 10 Nov. 2014].