

SMART MUSIC PLAYER

User Manual

**Team Member:**

Hien Tran

Minh Nguyen

Duc Nguyen

Hoang Nguyen

Linh Nguyen

Huy Nguyen

**Returned: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Checked: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Approved: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

# Introduction

## Target audience

This document targets general users trying to replicate the final result of the Smart Music Player project.

## General description of the system and interfaces

This project is developed targeting people’s need of entertainment, specifically listening to music. The product aims to eliminate the paradox of choice - the inconvenience of browsing and choosing a suitable track among too many music tracks.

The following figure shows a draft of the system interacting with its environment.

A picture containing screenshot

Description automatically generated

Figure 1. The system and its environment

The interfaces of the system are:

* End users
  + Input: Control from the push buttons
  + Output: Audio from the 3.5 mm output jack
* 3rd party location server
  + Input: GPS data
  + Output: Weather of the location

## Hardware and software requirements

The following table shows the requirements of the system

Table 1. System requirements

|  |  |  |  |
| --- | --- | --- | --- |
|  | Type | Detail | Version |
| **Node** | Machine | Raspberry Pi | 3B+ |
| Operating system | Raspbian Buster | September 2019 |
| Other hardware | Deltaco USB sound card | 7.1 |
| Dependencies | libssl-dev | 1.1.1c-1 |
| libjsoncpp-dev | 1.7.4-3 |
| libcurl4-openssl-dev | 7.64.0-4 |
| libmpg123-dev | 1.25.10-2 |
| **Backend** | Machine | AMD64 |  |
| Operating system | Debian | 10 (Buster) |
| Dependencies | NodeJS | 10.16.2 |
| npm | 6.13.1 |
| OpenSSL | 1.1.1d |
| MongoDB | 4.2.1 |

There is also a custom button board for the project. The design and layout of it will be specify later in this document.

## Abbreviations

The following table explains the abbreviations used in the document

Table 2. Abbreviations

|  |  |
| --- | --- |
| Abbreviations | Details |
| GPS | Global Positioning System |
| SMP | Smart Music Player |
| SSL | Secure Socket Layer |
| SFTP | Secure File Transfer Protocol |
| HTTPS | Hypertext Transfer Protocol Secure |
| REPL | Read-Eval-Print Loop |
| RSA | Rivest – Shamir – Adleman public-key cryptosystem |
| CURL | Client Uniform Resource Locator |
| IP | Internet Protocol |
| API | Application Programming Interface |
| PCB | Printed Circuit Board |

## Contact information

The following table contains the contact detail of the project group

Table . Contact detail

|  |  |  |
| --- | --- | --- |
| Member | Student number | Email |
| Nguyen Minh | e1601125 | e1601125@edu.vamk.fi |
| Tran Hien | e1601116 | e1601116@edu.vamk.fi |
| Nguyen Linh | e1500972 | e1500972@edu.vamk.fi |
| Nguyen Duc | e1601145 | e1601145@edu.vamk.fi |
| Nguyen Hoang | e1601111 | e1601111@edu.vamk.fi |
| Nguyen Huy | e1601119 | e1601119@edu.vamk.fi |

Any faults of the program can be reported to Nguyen Minh by email or a by submitting a pull request on the project’s Github folder.

# Commissioning of the system

## Backend

### Acquire the software

Before installation, make sure you have the operating system and all dependencies up and running. The software can be acquired from the link from the reference [1]. You can download the zip file or clone the git repos to your server machine.

### Acquire the DarkSky API key

Acquire the DarkSky API key from the link from the reference [2]. Replace the api\_key constant in the /src/api.ts file with this key.

### Generate RSA key

Run the script keygen.sh in the project root folder to generate the RSA keys. The keys will be stored in the SSL folder.

### Configure the database

Make sure you have the MongoDB database up and running and acquire the connection string beforehand. Open the file src/index.ts with your preferred text-editor and replace the string in the connect() function with your connection string.

### Run the server

In the project root folder, run npm install to install the dependencies.

Run npm run start to start the server.

### Setup SFTP

Follow this guide from the reference [3] to setup SFTP.

Replace the line ForceCommand internal-sftp with ForceCommand internal-sftp -R in the file /etc/ssh/sshd\_config to set the sftp server to be read-only.

### Maintain the database

Put the path to the folder containing the songs into the song\_path constant declared in maintain/index.ts. Run npm run build-maintain to build the maintain script. Run npm run maintain to start the maintain REPL.

In the maintain REPL

* Run update() to update the database
* The update() function will add songs that have files in the song folder to the database if no song with the same file name is found, as well as removing songs from the database automatically if the files for the songs cannot be not found. This function work on wav and mp3 files only.
* It will also create a file named songs.json in the maintain folder. Change the song name and tag in this file and then run the update() function once more to update new song to the database
* If the song file name is written in snake\_case, the song name will be deducted automatically. For example, a song file "lioness\_-\_dayfox.wav" will have the name "Lioness - Dayfox".
* Run .exit to exit the REPL

## Music player node

### Acquire the software

Before installation, make sure you have the operating system and all dependencies up and running. The software can be acquired from the link from the reference [4]. You can download the zip file or clone the git repos to your node.

Change the configuration

### Generate key pair for SFTP communication

Generate a private/public key pair for the SFTP communication

sudo ssh-keygen -m PEM -t rsa -N "" -f ./files/config-files/ssh-key

Upload the public key to the server so that it will trust us for accessing the specified user's account

sudo ssh-copy-id -i ./files/config-files/ssh-key.pub <server\_username>@<server\_IP>

Save the server as known host

ssh-keyscan <server\_IP> > ./files/config-files/known-hosts

If any path change, save the configurations to the configuration file

nano ./files/config.txt

### Configure sound card

Load the sound driver and check if it is loaded:

sudo modprobe snd-bcm2835

sudo lsmod | grep 2835

Select the output device for sound (0 = default/auo, 1 = jack 3.5, 2 = HDMI)

sudo amixer cset numid=3 1

Open terminal and test if ALSA is working (in either the built-in 3.5mm jack or sound card's jack)

aplay /usr/share/sounds/alsa/Front\_Center.wav

It is required to have the file asound.conf to set up output hardware/sound card properly.

sudo nano /etc/asound.conf

Copy and paste the following code and save it:

pcm.!default {

type plug

slave {

pcm "hw:1,0"

}

}

ctl.!default {

type hw

card 1

}

### Start the program after boot (optional)

In the file ./files/musicplayer.service, fix the directory configuration if needed.

nano ./files/musicplayer.service

Copy the service file to /etc/systemd/system

sudo cp ./files/musicplayer.service /etc/systemd/system/

Start the service automatically on reboot

sudo systemctl enable musicplayer.service

The service log can be checked using:

journalctl -u musicplayer.service

## Button board

### Theory

The button control circuit was originally designed with only pull-down resistor system. As an input, the microcontroller pin can take one of these states: high, low and floating or high impedance. The purpose of using pull down resistor is to ensure the values of the input is always in a known state. In this case, a pull-down resistor pulls the signal to low state unless it is driven high.

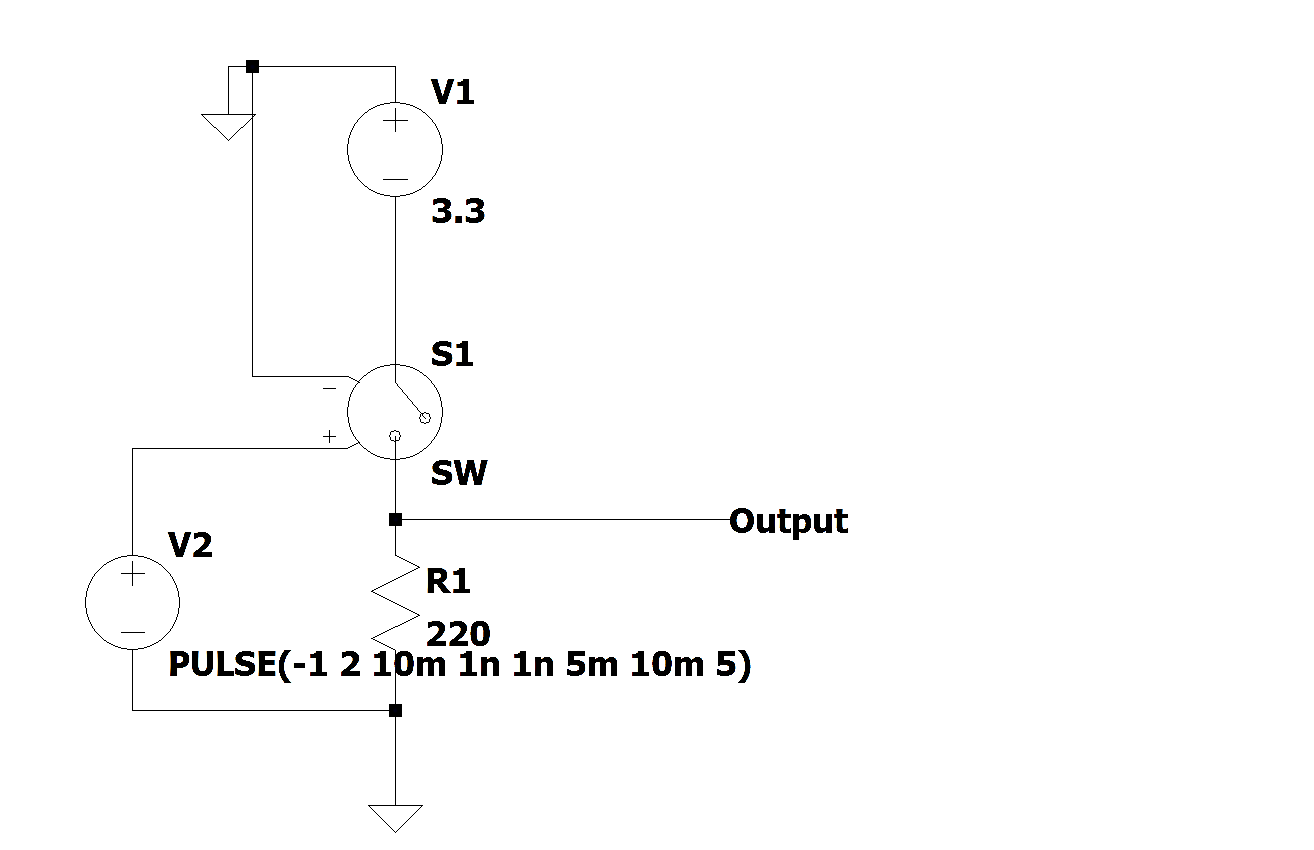


Figure . Pull-down resistor button circuit

However, push buttons generate spurious open/close transitions when pressed, due to mechanical and physical issues. Contact bounce is a common problem. Switch and relay contacts are usually made of springy metals so when a switch is pressed, its essentially two metal parts coming together and even though the connection may seem already made to the user, it may not happen immediately, as a matter of fact, it may make contact on one side, then both, and then the other side, technically bouncing between in-contact and not-in-contact until it finally settles down. As a result, these transitions may be read as multiple presses in a very short time fooling the program.

An RC circuit with inverting Schmitt trigger was used to solve this problem eliminating the noises of the buttons.

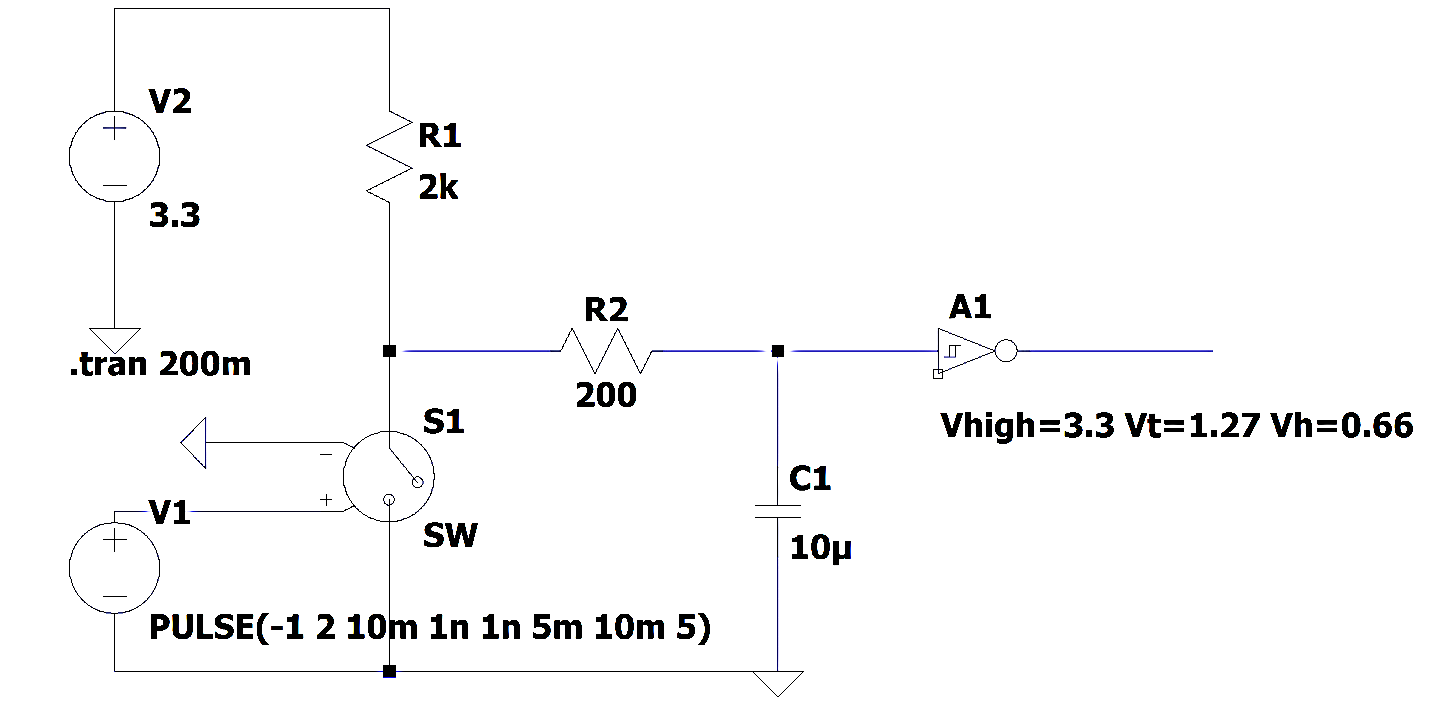


Figure . RC and inverting Schmitt trigger button circuit

The RC circuit acts as a low pass filter. The bouncing signal on created by the buttons in a short period could be considered as high frequency signal. Since the capacitor will need certain amount of time to charge and discharge, high frequency signal will be filtered out. Therefore, the logic state will only change when the voltage on the capacitor pass through the logic state threshold. However, it does not guarantee that unlucky bounces will not happen as the voltage on the capacitor reaching the threshold voltage.

A Schmitt trigger with 2 threshold values will transform any sine wave into square wave signal as what we need for the input signal for the microcontroller. In this project, we are using an inverting Schmitt Trigger. The logic level change from high to low when the feeding signal reaches the upper threshold and vice versa as illustrated in Figure 4.

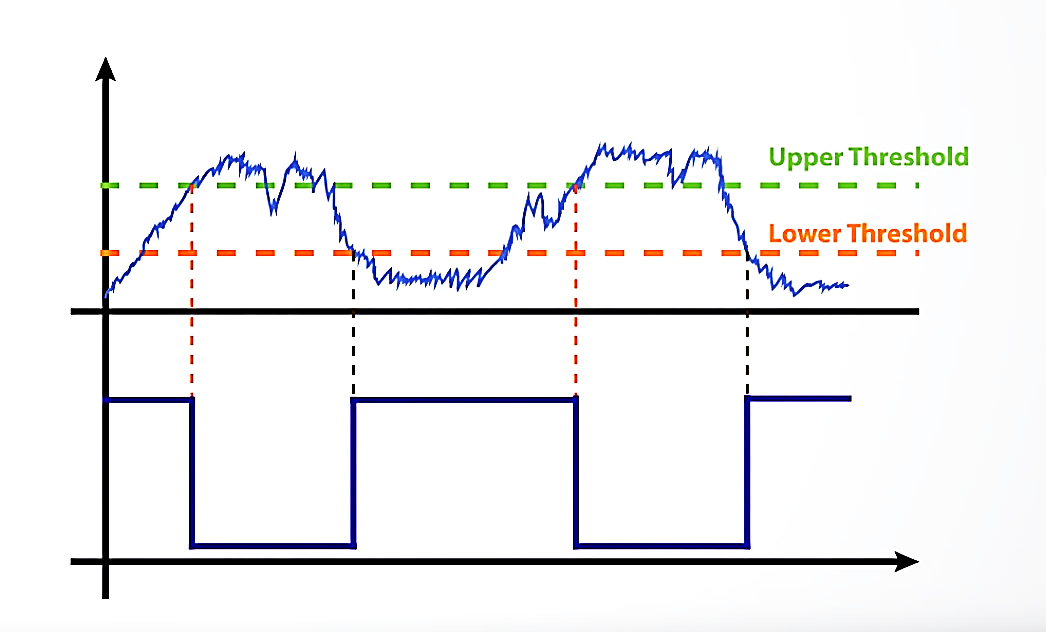


Figure . Inverting Schmitt trigger

Figure 5 explains the behavior of the input and output signals of the button control circuit. Blue line illustrates the bouncing when the button is pressed. The red line which is the out voltage of RC circuit follows filtering the high frequency signal. The signal is then fed to the inverting Schmitt trigger. As a result, we get very clean square wave (cyan line) as the output.

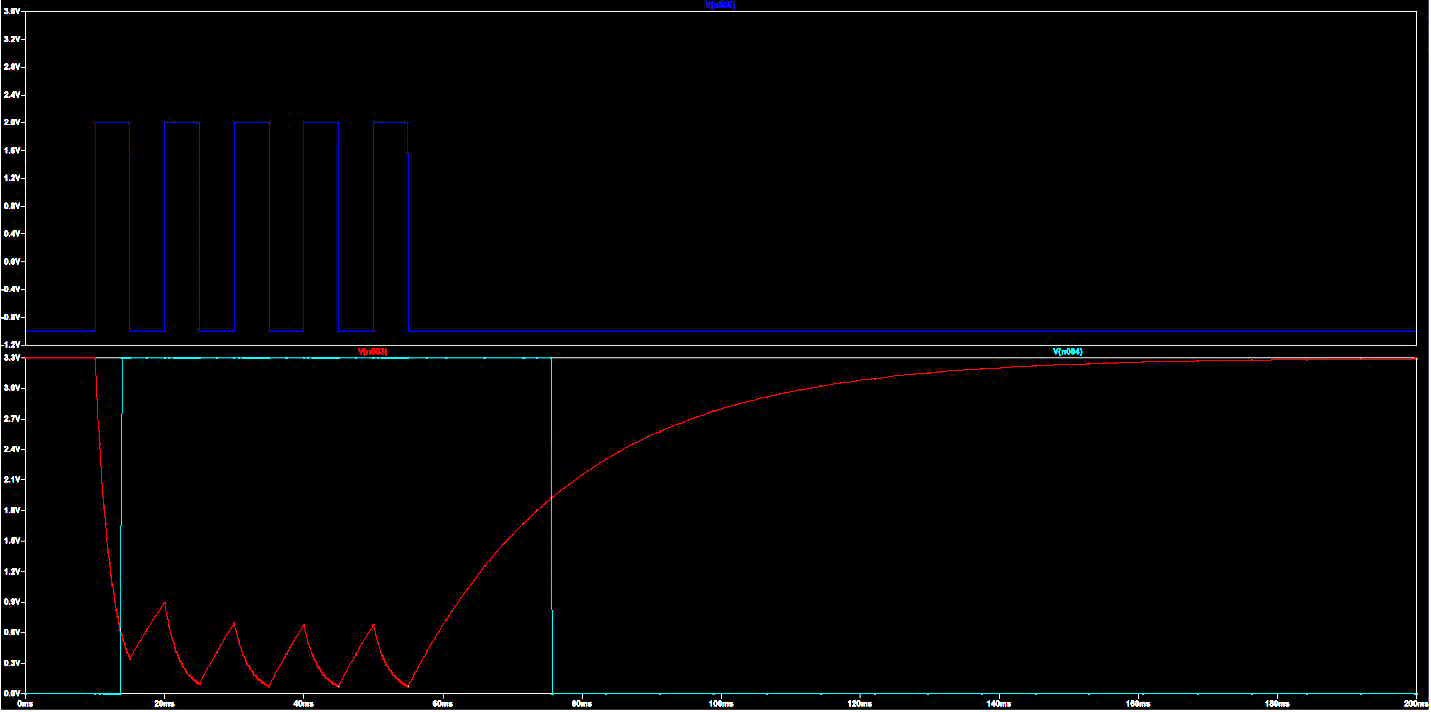


Figure . RC and inverting Schmitt trigger button circuit voltage level

### Layout

Figure 6 and Figure 7 contain the logical connection and layout of the printed circuit board.

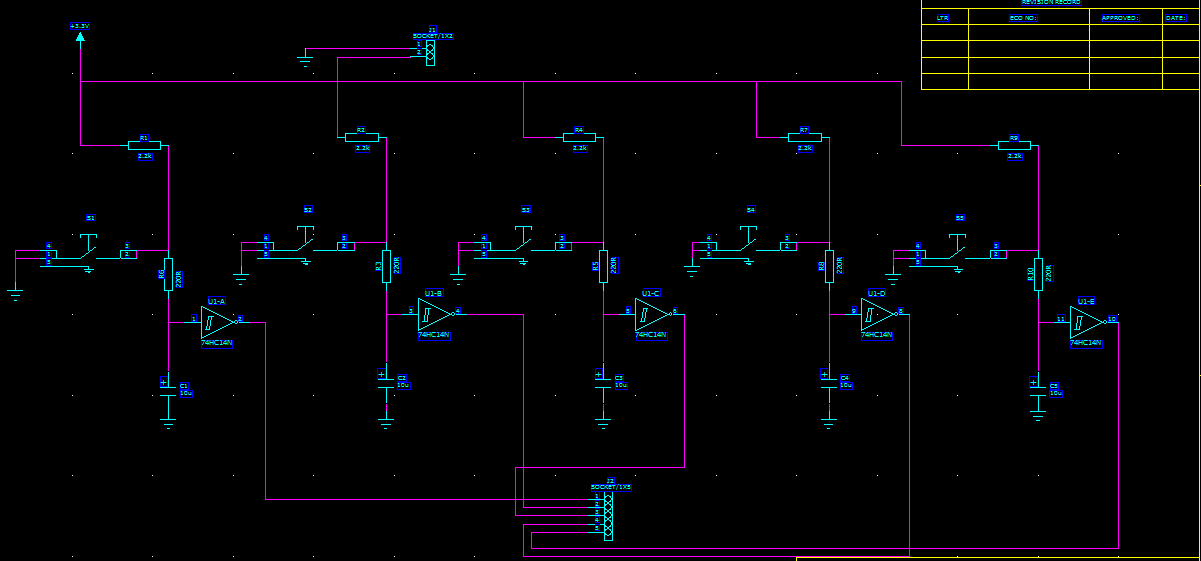


Figure . Final button control circuit

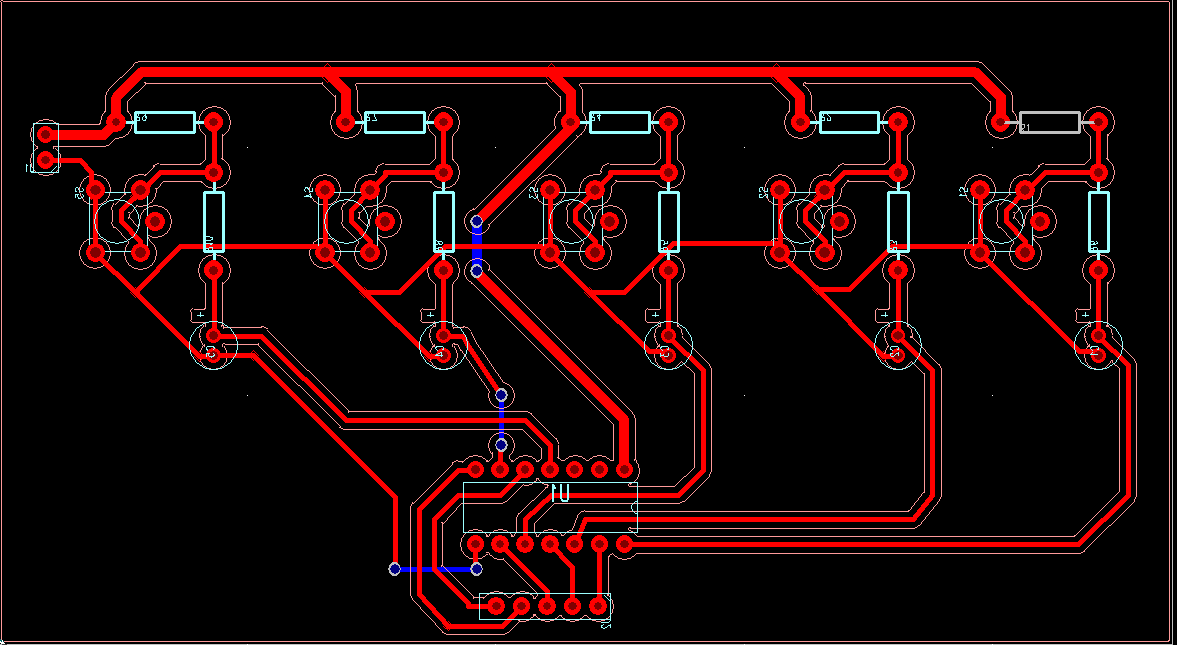


Figure . Final button control PCB layout

# Functions

## Backend

The general structure of the backend server is as follow:

A screenshot of a cell phone

Description automatically generated

Figure . Backend data flow

The backend software is responsible for:

* Receiving GPS data and return weather tag
* Receiving weather tag and return song metadata
* Serving song files

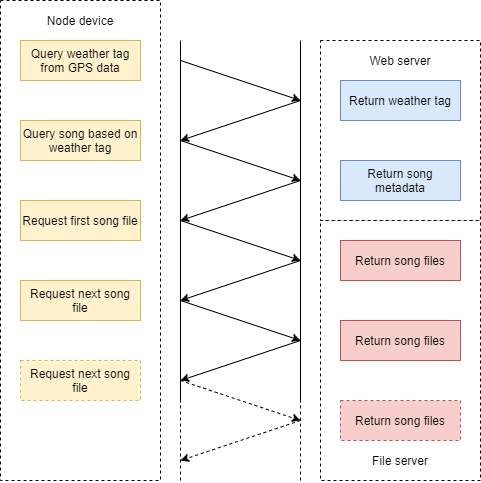


Figure . Backend communication flowchart

The server provides two API endpoint, accessible only via HTTPS:

* **Weather tag**
  + **URL:** /weather?longitude=<longitude>&latitude=<latitude>
  + **Method:** GET
  + **Arguments:**
    - <longitude>, <latitude>: the GPS position of the node
  + **Return value:** One of the weather tags, or an error message
* **Song metadata**
  + **URL:** /song?tag=<tag>
  + **Method:** GET
  + **Argument:**
    - <tag>: the weather tag return from previously
  + **Return value**: a list of song metadata corresponding to the tag.

The song metadata has the following fields:

* name (string): the name of the song
* filename (string): the name of the file for the song
* tag (enum): the corresponding weather tag. Available values are:
  + clear
  + rain
  + snow
  + wind
  + fog
  + cloudy
  + default

## Music player node

The functions’ documentation can be found in the link from the reference [5].

The following figure contains the packages of the music player application and their relationships:

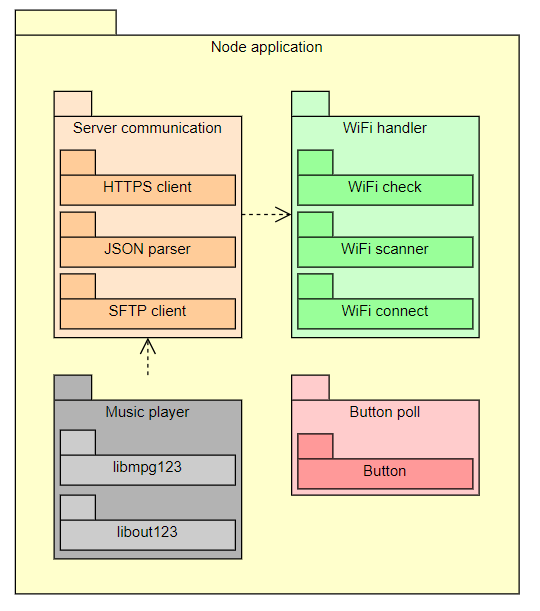


Figure . Node application package diagram

The following figure illustrate the main activities of the music player application:

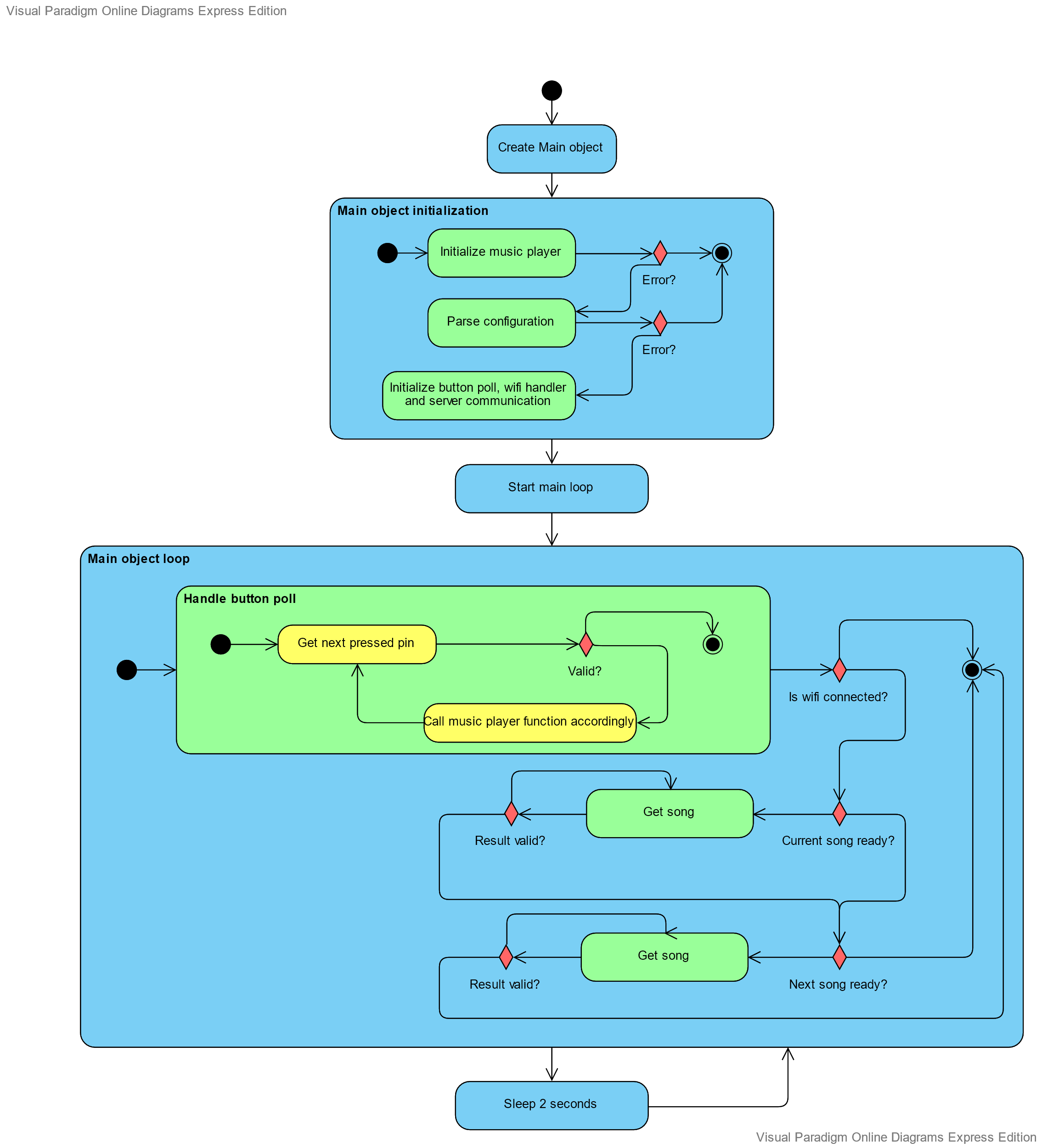


Figure . Node application activity diagram

# Problem situations

## Backend

### Invalid weather tag request

**Endpoint:** /weather

**Message:** Latitude and longitude must be number

**Source:** ./src/app.ts

**Possible reason:**

* The request missing the longitude or latitude field.
* The longitude or latitude field is wrongly formatted.

## Music player node

### Perform CURL fail

**Message**: SftpClient: Perform curl fail: <CURL error code>

**Source**: ./src/sftpclient.cpp

**Error code reference**: Link from reference [6]

**Possible reasons**:

* When generating the key pair for SFTP (part 2.2.2), the user generates a passphrase for the private key. Currently, libcurl is using libssh2, which is using libgcrypt, and libgcrypt does not support passphrase
* The server has changed (either IP or some configuration). In this case, the configuration file must be changed accordingly and step 3 in part 2.2.2 has to be redone.

### Cannot get weather tag

**Message**: ServerComm: getWeatherTag() failed

**Source**: ./src/servercomm.cpp

**Possible reasons:**

* Server wrongly configured
* 3rd party weather API is jammed

# Maintenenace

## Backend

After updating the song in the songs’ folder, run the maintenance script as instructed in part 2.1.7 to update the database

## Music player node

### Possible development

* **GPS data processing**: The current implementation only has a dummy function for getting GPS location since the project team lacks a working GPS module. With a working GPS module, real data acquisition and processing can be done.
* **User interface to control Wi-Fi connection**: The current implementation already has API to check whether the device is online, scan available Wi-Fi access points, and connect to access points with WPA2 encryption, but lacks a user interface.

# References

|  |  |
| --- | --- |
| [1] | "SMP Github folder - Backend," [Online]. Available: https://github.com/VAMK-embedded-project-2019A/Backend. |
| [2] | "DarkSky API link," [Online]. Available: https://darksky.net/dev. |
| [3] | "Configure SFTP backend," [Online]. Available: https://www.linuxtechi.com/configure-sftp-chroot-debian10. |
| [4] | "SMP Github folder - Node," [Online]. Available: https://github.com/VAMK-embedded-project-2019A/Development-Board-Application. |
| [5] | "Music player node documentation," [Online]. Available: https://vamk-embedded-project-2019a.github.io/Development-Board-Application/. |
| [6] | "CURL error codes," [Online]. Available: https://curl.haxx.se/libcurl/c/libcurl-errors.html. |

# Index

Table of Contents

[1 Introduction 2](#_Toc31290639)

[1.1 Target audience 2](#_Toc31290640)

[1.2 General description of the system and interfaces 2](#_Toc31290641)

[1.3 Hardware and software requirements 3](#_Toc31290642)

[1.4 Abbreviations 4](#_Toc31290643)

[1.5 Contact information 4](#_Toc31290644)

[2 Commissioning of the system 5](#_Toc31290645)

[2.1 Backend 5](#_Toc31290646)

[2.1.1 Acquire the software 5](#_Toc31290647)

[2.1.2 Acquire the DarkSky API key 5](#_Toc31290648)

[2.1.3 Generate RSA key 5](#_Toc31290649)

[2.1.4 Configure the database 5](#_Toc31290650)

[2.1.5 Run the server 5](#_Toc31290651)

[2.1.6 Setup SFTP 5](#_Toc31290652)

[2.1.7 Maintain the database 5](#_Toc31290653)

[2.2 Music player node 6](#_Toc31290654)

[2.2.1 Acquire the software 6](#_Toc31290655)

[2.2.2 Generate key pair for SFTP communication 6](#_Toc31290656)

[2.2.3 Configure sound card 7](#_Toc31290657)

[2.2.4 Start the program after boot (optional) 7](#_Toc31290658)

[2.3 Button board 8](#_Toc31290659)

[2.3.1 Theory 8](#_Toc31290660)

[2.3.2 Layout 11](#_Toc31290661)

[3 Functions 12](#_Toc31290662)

[3.1 Backend 12](#_Toc31290663)

[3.2 Music player node 14](#_Toc31290664)

[4 Problem situations 16](#_Toc31290665)

[4.1 Backend 16](#_Toc31290666)

[4.1.1 Invalid weather tag request 16](#_Toc31290667)

[4.2 Music player node 16](#_Toc31290668)

[4.2.1 Perform CURL fail 16](#_Toc31290669)

[4.2.2 Cannot get weather tag 16](#_Toc31290670)

[5 Maintenenace 17](#_Toc31290671)

[5.1 Backend 17](#_Toc31290672)

[5.2 Music player node 17](#_Toc31290673)

[5.2.1 Possible development 17](#_Toc31290674)

[6 References 18](#_Toc31290675)

[7 Index 19](#_Toc31290676)

Table of Figures

[Figure 1. The system and its environment 2](#_Toc31290677)

[Figure 2. Pull-down resistor button circuit 8](#_Toc31290678)

[Figure 3. RC and inverting Schmitt trigger button circuit 9](#_Toc31290679)

[Figure 4. Inverting Schmitt trigger 10](#_Toc31290680)

[Figure 5. RC and inverting Schmitt trigger button circuit voltage level 10](#_Toc31290681)

[Figure 6. Final button control circuit 11](#_Toc31290682)

[Figure 7. Final button control PCB layout 11](#_Toc31290683)

[Figure 8. Backend data flow 12](#_Toc31290684)

[Figure 9. Backend communication flowchart 13](#_Toc31290685)

[Figure 10. Node application package diagram 14](#_Toc31290686)

[Figure 11. Node application activity diagram 15](#_Toc31290687)

Table of Tables

[Table 1. System requirements 3](#_Toc31290688)

[Table 2. Abbreviations 4](#_Toc31290689)

[Table 3. Contact detail 4](#_Toc31290690)