# NFC Spotify Player using Arduino

## Pre-build

Before beginning assembling the Spotify Player, I am testing the two displays I am deliberating in using, a 2.4” TFT LCD and a 0.96” LCD display. This is mostly to get comfortable with using a breadboard, wires, Arduinos and external modules.

For the project (maybe only for the pre-build, depending on how it goes) I will be using only ChatGPT as my assistant and guide to test its usefulness in such a setting.

### 2.4” TFT LCD

After some wiring, I have managed to connect the display to my Arduino, however I am still struggling to display things that aren’t just the standard white background. Chat GPT has been helpful, mainly in identifying the resistor I needed to use for the LED pin, but it has also been inconsistent in what pins to connect and to where, meaning I have had to rewire some connections.

The following day, I tried wire the 2.4” display again, using 3k3 and 2k2 resistors on the data pins to lower the voltage level to work at the 3.3v logic that the display used. However, after a lot of different configurations, I could still not get it to display anything other than a blank white screen.

Since I was not planning to use this display in the final project anyway, I have decided to move onto the 0.96” display.

I have also abandoned the notion of only using Chat GPT as it was not very useful and often contradicted itself. I did find that using other resources such as Google or Youtube initially and then using Chat GPT to refine any points I was unsure about was quite effective.

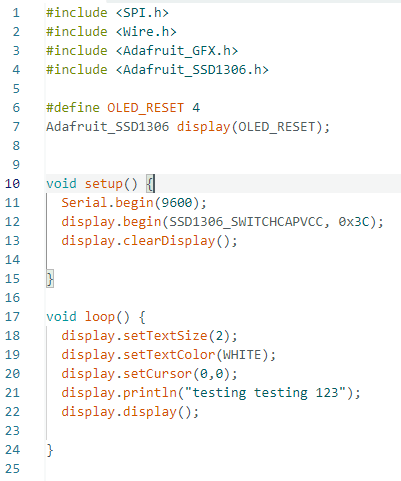
### 0.96” OLED

This display uses the I2C protocol, meaning that it only has 4 connections and should be a lot easier to work with, and it should leave plenty of digital pins for the other components in the project. However, this also means that its display capabilities are limited to just one colour.

I wired it up with the following configuration. The colour of the background is the colour of wire used.

|  |  |
| --- | --- |
| Arduino | OLED Display |
| 5V | VCC |
| GND | GND |
| A4 | SDA |
| A5 | SCL |

I managed to display some test text on the screen with the following code.



PIC OF TESTING DISPLAY HERE

After getting the display to work, I also decided to carry on testing some of the other components, first of which is the KC11B04 keypad.

### 4 Button Keypad

The keypad is very simple and only has 3 connections.

|  |  |
| --- | --- |
| Arduino | KC11B04 |
| A0 | AD |
| 5V | VCC |
| GND | GND |

The 4 buttons are detected by the output of the AD pin which is a value between 0-1024.

Button Values:

K1: 400

K2: 600

K3: 800

K4: 1024

The labelled keys printed their respective values in the Serial, and using if statements to check if the values are in a range allowed me to detect which buttons are pressed.

I put the following line in the header:

const int AD\_PIN = A0; /\*Keypad pin\*/

And the following lines elsewhere:

void loop() {

  display.setTextSize(2);

  display.setTextColor(WHITE);

  display.setCursor(0,0);

  display.println("testing testing 123");

  display.display();

  int adValue = analogRead(AD\_PIN);

  Serial.println(buttonFromValue(adValue));

}

byte buttonFromValue(int adValue) {

  if (adValue > 300 && adValue < 500){

    return 1;

  }

  if (adValue > 500 && adValue < 700){

    return 2;

  }

  if (adValue > 700 && adValue < 900){

    return 3;

  }

  if (adValue > 900){

    return 4;

  }

  return 0;

}

I noticed that when you touched the back of the keypad, your finger caused interference with the values returned, but the threshold for each keypad button is big enough to not be affected by this when a key is pressed.

Next, I will try to display the number of the button pressed on the OLED display as it is clicked.

void loop() {

  display.setTextSize(3);

  display.setTextColor(WHITE);

  display.setCursor(0,0);

  int adValue = analogRead(AD\_PIN);

  delay(100);

  int button = buttonFromValue(adValue);

  Serial.println(button);

  if (button != lastButton && button != 0){

    lastButton = button;

    display.clearDisplay();

    display.println(lastButton);

    display.display();

  }

}

This is the code I used to implement this. If a new button is pressed, it changes the display to said button. The delay of 100ms is necessary since sometimes the value of the AD pin goes down as you let go of a button, and drops into the undesired threshold. For instance, after pressing 3 and letting go, it sometimes drops into 2 and displays that instead.

PIC OF NUMBERED DISPLAY HERE

## Prototype 1

Now that I am comfortable with the display and keypad, I am going to create my first prototype that connects to my Spotify API and displays the name of the song playing, the artist, and lets me play and skip songs.

**Prototype 1 Components:**

* KC11B04 Keypad
* 0.96” OLED I2C Display
* Arduino Uno
* Jumper wires

**Prototype 1 Aims:**

* Connect to Spotify Account
* Display song name
* Display artist
* Play/pause using keypad
* Go back/skip songs using keypad

First, I tried to create a ‘thing’ on the Arduino IoT cloud but ran into my first problem – the Arduino Uno R3 cannot connect to the internet, so I will have to connect the ESP01 Wifi Module to it before I can proceed.

|  |  |
| --- | --- |
| Arduino | ESP01 |
| 3.3v | 3.3v |
| 3.3v | EN |
| GND | GND |
| D2 | TX |
| D3 | RX |

**NEXT FIND IP OF ESP01 AND THEN USE THAT TO CONNECT TO SPOTIFY API**

Its not working. Ditched Blynk, will send requests myself

Still not working. After a week of trying everything.

I can’t proceed. Spotify servers can only be communicated with via an SSL connection, and the Arduino does not have enough processing power to handle encryption. To get around this, I tried to handle the encryption on the ESP-01 itself, but trying to connect and program it as a slave to the Arduino and not directly proposed a whole heap of new problems. It just would simply not connect, regardless of what pins I connected, what version of drivers I had installed, which port I used, and which tutorial I followed.

## Prototype 2

I am going to try a new approach. I will host a web server and send simple strings to it using the Arduino. The web server will then handle these strings and do the appropriate action. This also means that I can interface with the Spotify API via Javascript or even Python, which I am much more comfortable with.

Having coded a simple express server with JS, I am now trying to send a POST request to it via my Arduino. However, I am having trouble getting a connection established to the internet.  
  
I have now rewired my ESP-01 so that its RX and TX pins connect to the Arduino’s TX and RX pins (1, 0) respectively. This has let me send AT commands and receive responses when writing the commands directly in the serial, but upon further research I found that the logic levels that the two components operate at differ. The Arduino operates at a 5v logic level, while the ESP-01 is at 3.3v, meaning I have to make use of my voltage dividers (LD1117V33) to make sure that the ESP-01 does not become damaged.

I then ran into another issue – the code wouldn’t upload to my Arduino unless the RST pin was connected on my ESP-01. I found that the reason for this was that the active Wifi module was interfering with the upload process, and resetting it would cause it to temporarily switch off to allow for the upload to go through. To fix this, I changed to pin 2 and 3 on the Arduino instead of RX and TX, which is used during the upload process.:

**Previous:**

|  |  |
| --- | --- |
| **Arduino** | **ESP01** |
| **RX** | **TX** |
| **TX** | **RX** |

**Now:**

|  |  |
| --- | --- |
| **Arduino** | **ESP01** |
| **D2** | **TX** |
| **D3** | **RX** |

Now that a lot of the technical issues I ran into are over with, and the ESP-01 is responding to my AT commands and is successfully connecting to wifi, it should (fingers crossed) be smooth sailing to send POST requests to the server I set up.

I’ve ran into another issue, which unfortunately is fatal to this prototype. The ESP-01 draws a lot of current, especially when transmitting over wifi. The Arduino’s 3.3v power supply cannot supply the current it needs.

To try solve the problem, I connected the 5v power from the Arduino to a 3.3v voltage regulator, in the hopes that it will increase the current but retain the 3.3v to prevent the module from being damaged but alas it does not work, and I have no clue why. The next, and possibly only step, would be to buy an external 3.3v power supply but it would not arrive in time (29th June at the moment and needs to be done by 7th July (running very late (I thought it would take like 5 days and I’ve been working on just the wifi for a good 2 weeks))) and I can’t guarantee that that would work if the 5v solution didn’t either.

So onto prototype 3 before prototype 2 ever got off the ground…

## Prototype 3

I’m ditching the ESP-01 and wifi, which means that it will have to be connected to a computer to be used, but its better than nothing and it means I will have something to show and test, and I can always add wifi capability to it later by getting a newer Uno or using my Raspberry Pi that I can’t get into the case of.

So prototype 3 will have the RFID sensor, the display and the keypads, meaning it will function exactly like intended, only it won’t be as portable. I’m also going to write the code in javascript instead of AVR assembly because I don’t want my hairs to turn grey quite just yet.

const { read } = require('johnny-five/lib/pin');

const { SerialPort } = require('serialport');

const sp = new SerialPort({

    path: 'COM3',

    baudRate: 9600,

    autoOpen: false

  });

  sp.open((err) => {

    if (err) {

      return console.log('Error opening port:', err.message);

    }

    console.log('Port has opened');

  });

  sp.on('data', async (data) => {

    const enc = new TextDecoder();

    const arr = new Uint8Array(data);

    const ready = enc.decode(arr);

    console.log('Data received:', ready);

    if (ready == 1) { // Assuming ready is a string '1'

      try {

        const playing = await spotifyCheckPlayback();

        console.log('Is Spotify playing?', playing);

        if (playing) {

          await spotifyPause();

        } else {

          await spotifyPlay();

        }

      } catch (err) {

        console.error('Error handling playback:', err);

      }

    }

  });

function spotifyPlay(){

  // Example: Resume playback

  spotifyApi.play({

    device\_id: deviceId,

  })

  .then(() => {

    console.log('Playback resumed!');

  })

  .catch((err) => {

    console.error('Error resuming playback:', err);

  });

}

function spotifyPause(){

    // Example: Pause playback

  spotifyApi.pause({

    device\_id: deviceId,

  })

  .then(() => {

    console.log('Playback paused!');

  })

  .catch((err) => {

    console.error('Error pausing playback:', err);

  });

}

async function spotifyCheckPlayback() {

  try {

    const data = await spotifyApi.getMyCurrentPlaybackState();

    if (data.body && data.body.is\_playing) {

      return true; // Spotify is playing

    } else {

      return false; // Spotify is not playing

    }

  } catch (err) {

    console.error('Error checking playback state:', err);

    return false; // Default to false in case of error

  }

}

**PROGRESS MADE!!** Above is javascript for handling the Spotify API, with a button on the keypad wired up to the Arduino playing and pausing songs.

Wiring intermission

|  |  |
| --- | --- |
| **Arduino** | **KC11B04 (Keypad)** |
| A0 | AD |
| GND | GND |
| 5V | VCC |

Finally, I can actually see something tangible and functional, even if its not much. I just need to code the other 3 buttons to skip, go back a song, and setting shuffle on/off, which shouldn’t take more than 10 minutes.

I’ve also put my biggest playlist on shuffle and I’m not going to use of the aforementioned features until I can do them using the keypad which is a fun little game. No songs I’ve wanted to skip so far thankfully.

Coded all of the buttons on the keypad:

* Key 1 => Play/Pause
* Key 2 => Next song
* Key 3 => Previous song
* Key 4 => Toggle Shuffle

Here are the rest of the functions. Note handling the keypad has been moved to a dedicated function too.

async function handleKeypad(key){

  if (key == 1) {

    await spotifyTogglePlayback();

  };

  if (key == 2) {

    await spotifyNext();

  };

  if (key == 3) {

    await spotifyBack();

  };

  if (key == 4){

    await spotifyToggleShuffle();

  };

};

async function spotifyNext(){

  spotifyApi.skipToNext({

    device\_id: deviceId,

  })

  .then(() => {

    console.log('Skipped to next track!');

  })

  .catch((err) => {

    console.error('Error skipping to next track:', err);

  });

}

async function spotifyBack(){

  spotifyApi.skipToPrevious({

    device\_id: deviceId,

  })

  .then(() => {

    console.log('Skipped to previous track!');

  })

  .catch((err) => {

    console.error('Error skipping to previous track:', err);

  });

};

async function checkShuffleState() {

  try {

    const data = await spotifyApi.getMyCurrentPlaybackState({ device\_id: deviceId });

    if (data.body && data.body.shuffle\_state) {

      return true; // Shuffle mode is enabled

    } else {

      return false; // Shuffle mode is disabled

    }

  } catch (err) {

    console.error('Error checking shuffle state:', err);

    return false; // Return false in case of error

  }

}

async function spotifyToggleShuffle(deviceId) {

  try {

    const shuffleState = await checkShuffleState();

    // Await the setShuffle call directly instead of using .then() and .catch()

    await spotifyApi.setShuffle(!shuffleState, { device\_id: deviceId });

    console.log('Shuffle mode toggled!');

  } catch (error) {

    console.error('Error toggling shuffle state:', error);

  }

}

I’ve soldered the pins onto the RC552 RFID Module and am about to test it. Below is its wiring.

|  |  |
| --- | --- |
| **Arduino** | **RC522** |
| 3.3v | 3.3v |
| GND | GND |
| D9 | RST |
| D10 | SDA |
| D11 | MOSI |
| D12 | MISO |
| D13 | SCK |

Code to handle RFID Module:

void handleRFID(){

  if (mfrc522.PICC\_IsNewCardPresent()){

    if (!mfrc522.PICC\_ReadCardSerial()) return;

    for (byte i = 0; i < mfrc522.uid.size; i++){

      Serial.print(mfrc522.uid.uidByte[i], HEX);

      mfrc522.PICC\_HaltA();

    }

    Serial.print("#");

  }

}

async function spotifyPlayPlaylist(playlistId) {

  try {

    await spotifyApi.play({

      context\_uri: `spotify:playlist:${playlistId}`,

      device\_id: deviceId,

    });

    console.log('Playlist playback started!');

  } catch (error) {

    console.error('Error playing playlist:', error);

  }

}

I also coded the current song playing to be checked every 10 seconds, with the display being updated accordingly.

async function writeToDisplay(msg){

  sp.write(msg, (err) => {

    if (err) {

      return console.log('Error on write: ', err.message);

    }

    console.log('message written');

  });

}

async function spotifyCurrentSong(){

  const data = await spotifyApi.getMyCurrentPlayingTrack();

    if (data.body && data.body.is\_playing) {

      const track = data.body.item;

      const song = track.name;

      const artists = track.artists.map(artist => artist.name).join(', ');

      console.log(`Now Playing: ${song} by ${artists}`);

      writeToDisplay(song + "#" + artists);

    } else {

      console.log('No song currently playing.');

    }

}

cron.schedule('\*/10 \* \* \* \* \*', spotifyCurrentSong);

Prototype 3 is a success! Unless I decide to use a Raspberry Pi instead of the Arduino, the final version will be very similar to it. I can measure and send off the dimensions to get its case 3D printed.

**Prototype 3 Overview**

**Wiring:**

|  |  |
| --- | --- |
| Arduino | OLED Display |
| 5V | VCC |
| GND | GND |
| A4 | SDA |
| A5 | SCL |

|  |  |
| --- | --- |
| Arduino | KC11B04 |
| A0 | AD |
| 5V | VCC |
| GND | GND |

|  |  |
| --- | --- |
| **Arduino** | **RC522** |
| 3.3v | 3.3v |
| GND | GND |
| D9 | RST |
| D10 | SDA |
| D11 | MOSI |
| D12 | MISO |
| D13 | SCK |

**AVR Code uploaded to Arduino:**

#include <Wire.h>

#include <SPI.h>

#include <Adafruit\_GFX.h>

#include <Adafruit\_SSD1306.h>

#include <MFRC522.h>

//Display

#define OLED\_RESET 4

Adafruit\_SSD1306 display(OLED\_RESET);

String songTitle = "Song Title";

String artists = "Artists";

//Keypad

const int AD\_PIN = A0;

int lastButton = 0;

//RFID

#define RST\_PIN 9

#define SS\_PIN 10

MFRC522 mfrc522(SS\_PIN, RST\_PIN);

void setup() {

  Serial.begin(9600);

  SPI.begin();      // Init SPI bus

  mfrc522.PCD\_Init();   // Init MFRC522

  if (!display.begin(SSD1306\_SWITCHCAPVCC, 0x3C)) {

    Serial.println("SSD1306 allocation failed");

    for (;;);

  }

  display.clearDisplay();

}

void loop() {

  // reply only when you receive data:

  if (Serial.available() > 0) {

    String data = Serial.readString(); // Read until newline or your chosen delimiter

    int delimiterIndex = data.indexOf('#'); // Replace ',' with your delimiter

    songTitle = data.substring(0, delimiterIndex);

    artists = data.substring(delimiterIndex + 1);

  }

  handleKeypad();

  handleDisplay();

  handleRFID();

}

byte buttonFromValue(int adValue) {

  if (adValue > 300 && adValue < 500){

    return 1;

  }

  if (adValue > 500 && adValue < 700){

    return 2;

  }

  if (adValue > 700 && adValue < 900){

    return 3;

  }

  if (adValue > 900){

    return 4;

  }

  return 0;

}

void handleKeypad(){

  int adValue = analogRead(AD\_PIN);

  int button = buttonFromValue(adValue);

  if (button != lastButton && button != 0) {

    lastButton = button;

    Serial.print(lastButton);

    delay(10);

    Serial.println("#");

  }

}

void handleDisplay() {

  display.clearDisplay();

  display.setTextSize(1);

  display.setTextColor(WHITE);

  display.setCursor(0, 0);

  display.println(songTitle);

  display.setCursor(0, 16);

  display.println(artists);

  display.display();

  delay(100);  // Optional delay for stability

}

void handleRFID(){

  if (mfrc522.PICC\_IsNewCardPresent()){

    if (!mfrc522.PICC\_ReadCardSerial()) return;

    for (byte i = 0; i < mfrc522.uid.size; i++){

      Serial.print(mfrc522.uid.uidByte[i], HEX);

      mfrc522.PICC\_HaltA();

    }

    Serial.print("#");

  }

}

**Javascript Code (sensitive info ommitted):**

//==============================================SPOTIFY SETUP

const SpotifyWebApi = require('spotify-web-api-node');

const access\_token = '';

const refresh\_token = '';

const deviceId = '';

const cron = require('node-cron');

const spotifyApi = new SpotifyWebApi({

  clientId: '',

  clientSecret: '',

});

spotifyApi.setAccessToken(access\_token);

//============================================Port handling

const { read, write } = require('johnny-five/lib/pin');

const { SerialPort } = require('serialport');

var receivedData = '';

const sp = new SerialPort({

  path: 'COM3',

  baudRate: 9600,

  autoOpen: false

});

sp.open((err) => {

  if (err) {

    return console.log('Error opening port:', err.message);

  }

  console.log('Port has opened');

});

sp.on('data', async (data) => {

  var enc = new TextDecoder();

  var arr = new Uint8Array(data);

  var chunk = String(enc.decode(arr));

  chunk = chunk.trim();  // Remove leading/trailing whitespace

  //# is delimiter character

  if (chunk == "#"){

    handleKeypad(receivedData);

    handleRFID(receivedData);

    receivedData = '';

    chunk = "";

  }

  else {

    receivedData += chunk;  // Append received chunk to accumulated data

    console.log("Received data:", receivedData)

  }

});

async function writeToDisplay(msg){

  sp.write(msg, (err) => {

    if (err) {

      return console.log('Error on write: ', err.message);

    }

    console.log('message written');

  });

}

//================================== SPOTIFY FUNCTIONS

function spotifyPlay(){

  spotifyApi.play({

    device\_id: deviceId,

  })

  .then(() => {

    console.log('Playback resumed!');

  })

  .catch((err) => {

    console.error('Error resuming playback:', err);

  });

}

function spotifyPause(){

  spotifyApi.pause({

    device\_id: deviceId,

  })

  .then(() => {

    console.log('Playback paused!');

  })

  .catch((err) => {

    console.error('Error pausing playback:', err);

  });

}

async function spotifyTogglePlayback() {

  try {

    const data = await spotifyApi.getMyCurrentPlaybackState();

    if (data.body && data.body.is\_playing) {

      spotifyPause();

    } else {

      spotifyPlay();

    }

  } catch (err) {

    console.error('Error checking playback state:', err);

    return false;

  }

}

function spotifyGetDevices(){

  spotifyApi.getMyDevices()

  .then((data) => {

    const devices = data.body.devices;

    if (devices.length > 0) {

      console.log('List of devices:');

      devices.forEach((device) => {

        console.log(`Name: ${device.name}, ID: ${device.id}`);

      });

    } else {

      console.log('No devices found.');

    }

  })

  .catch((err) => {

    console.error('Error fetching devices:', err);

  });

}

async function spotifyNext(){

  spotifyApi.skipToNext({

    device\_id: deviceId,

  })

  .then(() => {

    console.log('Skipped to next track!');

  })

  .catch((err) => {

    console.error('Error skipping to next track:', err);

  });

};

async function spotifyBack(){

  spotifyApi.skipToPrevious({

    device\_id: deviceId,

  })

  .then(() => {

    console.log('Skipped to previous track!');

  })

  .catch((err) => {

    console.error('Error skipping to previous track:', err);

  });

};

async function checkShuffleState() {

  try {

    const data = await spotifyApi.getMyCurrentPlaybackState({ device\_id: deviceId });

    if (data.body && data.body.shuffle\_state) {

      return true; // Shuffle mode is enabled

    } else {

      return false; // Shuffle mode is disabled

    }

  } catch (err) {

    console.error('Error checking shuffle state:', err);

    return false; // Return false in case of error

  }

}

async function spotifyToggleShuffle(deviceId) {

  try {

    const shuffleState = await checkShuffleState();

    await spotifyApi.setShuffle(!shuffleState, { device\_id: deviceId });

    console.log('Shuffle mode toggled!');

  } catch (error) {

    console.error('Error toggling shuffle state:', error);

  }

}

async function spotifyPlayPlaylist(playlistId) {

  try {

    await spotifyApi.play({

      context\_uri: `spotify:playlist:${playlistId}`,

      device\_id: deviceId,

    });

    console.log('Playlist playback started!');

  } catch (error) {

    console.error('Error playing playlist:', error);

  }

}

async function spotifyCurrentSong(){

  const data = await spotifyApi.getMyCurrentPlayingTrack();

    if (data.body && data.body.is\_playing) {

      const track = data.body.item;

      const song = track.name;

      const artists = track.artists.map(artist => artist.name).join(', ');

      console.log(`Now Playing: ${song} by ${artists}`);

      writeToDisplay(song + "#" + artists);

    } else {

      console.log('No song currently playing.');

    }

}

//================================== HANDLING INPUTS

async function handleKeypad(key){

  if (key == '1') {

    await spotifyTogglePlayback();

    await spotifyCurrentSong();

  };

  if (key == '2') {

    await spotifyNext();

  };

  if (key == '3') {

    await spotifyBack();

  };

  if (key == '4'){

    await spotifyToggleShuffle();

  };

};

async function handleRFID(uid){

  if (uid == "B3814C27"){

    spotifyPlayPlaylist('5wZAfBSotlMHA6tfuh6MgT?si=eb218af942ba443a');

  }

};

cron.schedule('\*/10 \* \* \* \* \*', spotifyCurrentSong);

**Dimensions:**

Breadboard + Arduino side by side: 115mm x 80mm

Height (of wiring): 9cm

Display slot: 25mm x 15mm

Keypad slot: 41mm x 12mm

RFID holder: 62mm x 42mm x 12mm (height)

Serial Port: 13mm x 13mm