RFID Spotify Jukebox

by d00mfish

In this day and age, everything is digital and app controllable. This brings a lot of comfort and simplicity with it. However, the elderly and young people in particular are being left behind.

For example, for a long time it was not possible for my little sister to turn on an audio book by herself without asking someone with a phone to put it on. This problem and the motivation to make music physical and tangible again moved me to this project. The interaction with music and other people through jukeboxes (at least as it is often portrayed in movies) intrigued me, and especially for house parties, it's a great feature to actively involve the guests in the party.

As a tech nerd, this makes the connection between technology and proven systems obvious to me. Also having the option of playing playlists containing brand new songs without manually updating locally stored music-archives is a great advantage and also the reason why Spotify came into the mix (pun not intended).

Using a Raspberry Pi with an RFID-reader, any album and playlist can be connected to a card. Also the desired playback device can be selected with a setup card.

The result is a digital jukebox with volume control, pause, skip and shuffle button that can control any device that supports Spotify Connect.

DISCLAIMER:

The python-code and installation process is not optimal. If you encounter some problems during installation it may be caused by new incompatible versions or another installation order. Feel free to look into the provided links to solve your issues. Also some python skills would be good to fix something if it doesn't work out for you.

Therefore, the project is not recommended for people who want a plug and play / one-script solution, and are not willing or capable of deal with technical issues. (It's not that advanced though so just try it ;))

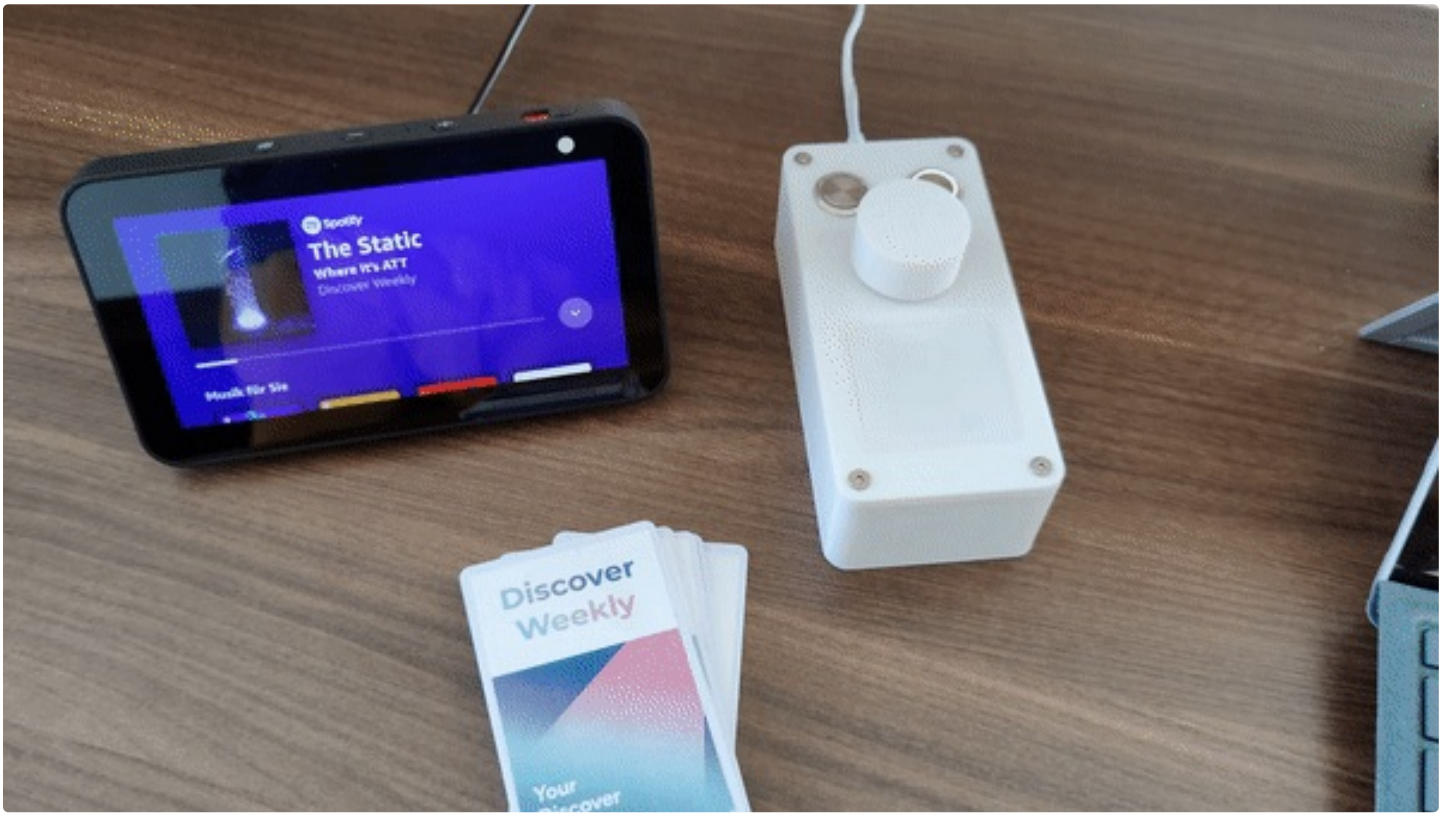
Supplies:

The required hardware is as follows:

- 1x Raspberry Pi Zero W(H)
- 1x PN532 RFID Reader / Writer - Rotary encoder
- 2x LED-Buttons 3V
- 1x Rotary Encoder
- As many Ntag2xx Cards as desired, the Github contains a deprecated branch for Mifare Classic 1k tough.
- Wires
- Screws for the case

Tools:

- 3D-Printer and Filament
- Printer for Card Lables
- Soldering equipment
- Standard tools (Screwdrivers, Pliers, etc.)



Step 1: Define Functions and Usecase

The device should be usable by everyone. This means that the operation must be simple and the functions not too extensive.

For a simple music control we use a:

- **Play/Pause** function
- **Skip** function
- **Volume up/down** function

Since I mainly want to play playlists, a **shuffle button** is also a handy feature.

So there are **two buttons** (skip and shuffle) and a **rotary encoder** (volume) with play/pause when pressed.

In addition, the buttons have LED lighting to provide feedback and indicate whether shuffle is enabled or not. When adjusting the volume, the brightness of the LEDs also shows the volume in percent.

Less frequently used functions can also be realized with NFC cards. This is also the case here:

To select the desired playback device, there will be a **"set device" card**. As soon as music is played on the device, the card will store the ID of the playback-device and use it later for the play command.

In case of a new playlist, old playback cards can be overwritten or new cards can be learned. For this, you play the desired playlist manually, then scan the **"learning card" - card** and the empty or old card afterwards. The device then writes the playlist ID to a new card.

Step 2: Printing the Case

I decided to use a 3D printed case for the prototype. Unfortunately, it is not really aesthetic but very flexible in creation and use.

However, in the future a wooden case is planned.

Files are attached, nothing special to consider here. The design is so simple, I printed it at 0.26mm to finish faster.

The knob was just pressed-fitted on the Rotary encoder.

https://www.instructables.com/ORIG/F5I/AW7B/KTH8B40H/F5IAW7BKTH8B40H.stl	View in 3D	Download
https://www.instructables.com/ORIG/FO6/AA5T/KTH8B40I/FO6AA5TKTH8B40I.stl	View in 3D	Download
https://www.instructables.com/ORIG/FEW/NZMJ/KTH8B40J/FEWNZMJKTH8B40J.stl	View in 3D	Download
https://www.instructables.com/ORIG/FOV/Q2JO/KTH8B40K/FOVQ2JOKTH8B40K.f3d	View in 3D	Download

Step 3: Making Card-lables

I made a simple Photoshop template to achieve a simple and consistent design.

After I printed the labels and glued them onto the cards. Using glossy paper or vinly stickers would probably look even better than normal paper.

Daily Drive

Release Radar

Device Card


Learn
Card


Red Hot
Chili
Peppers

Download

https://www.instructables.com/ORIG/FWD/7ZVG/KU8DP16P/FWD7ZVGKU8DP16P.psd

Step 4: Wiring

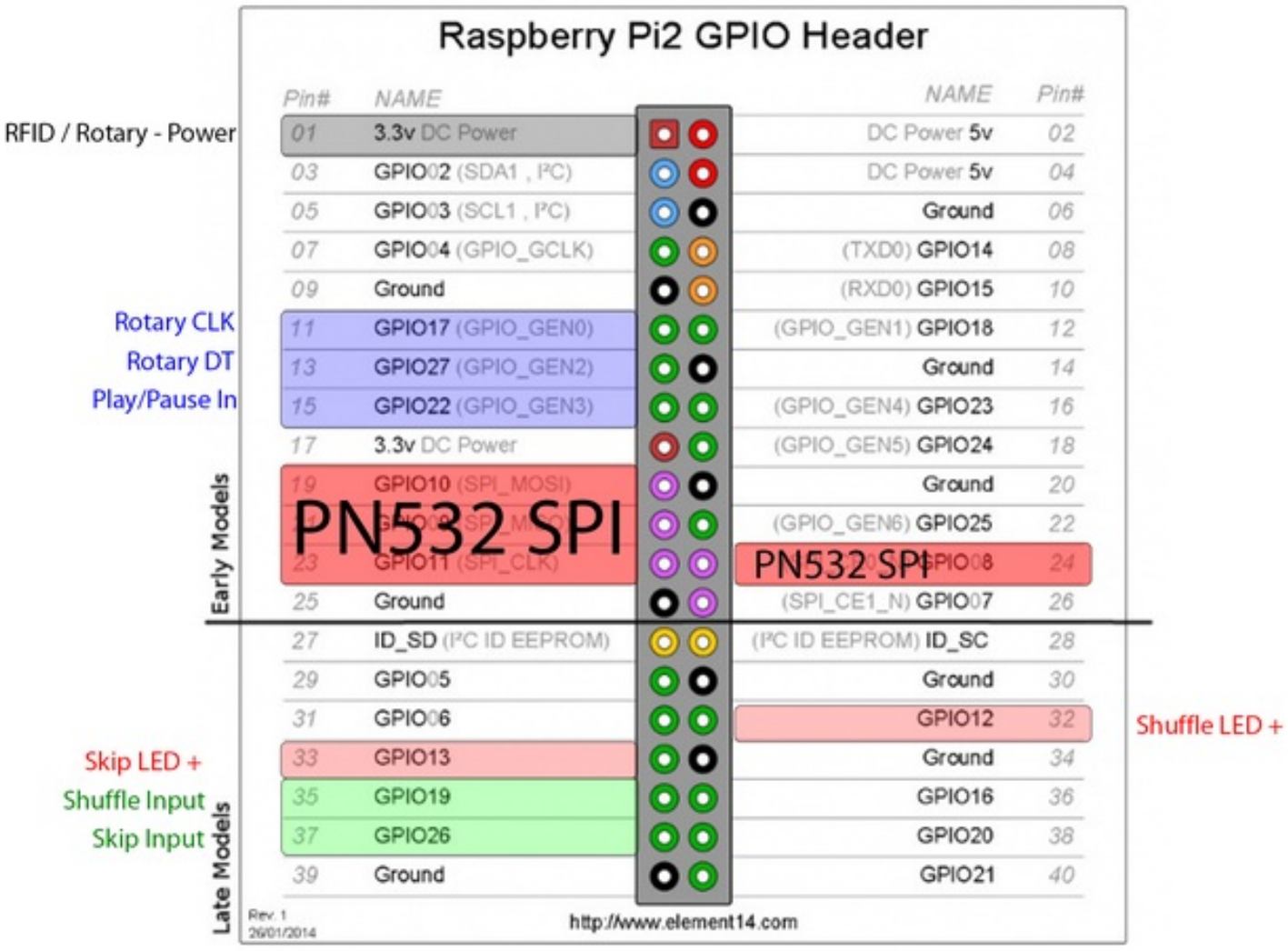
As you can see in the picture I connected all the components to the Raspberry Pi.

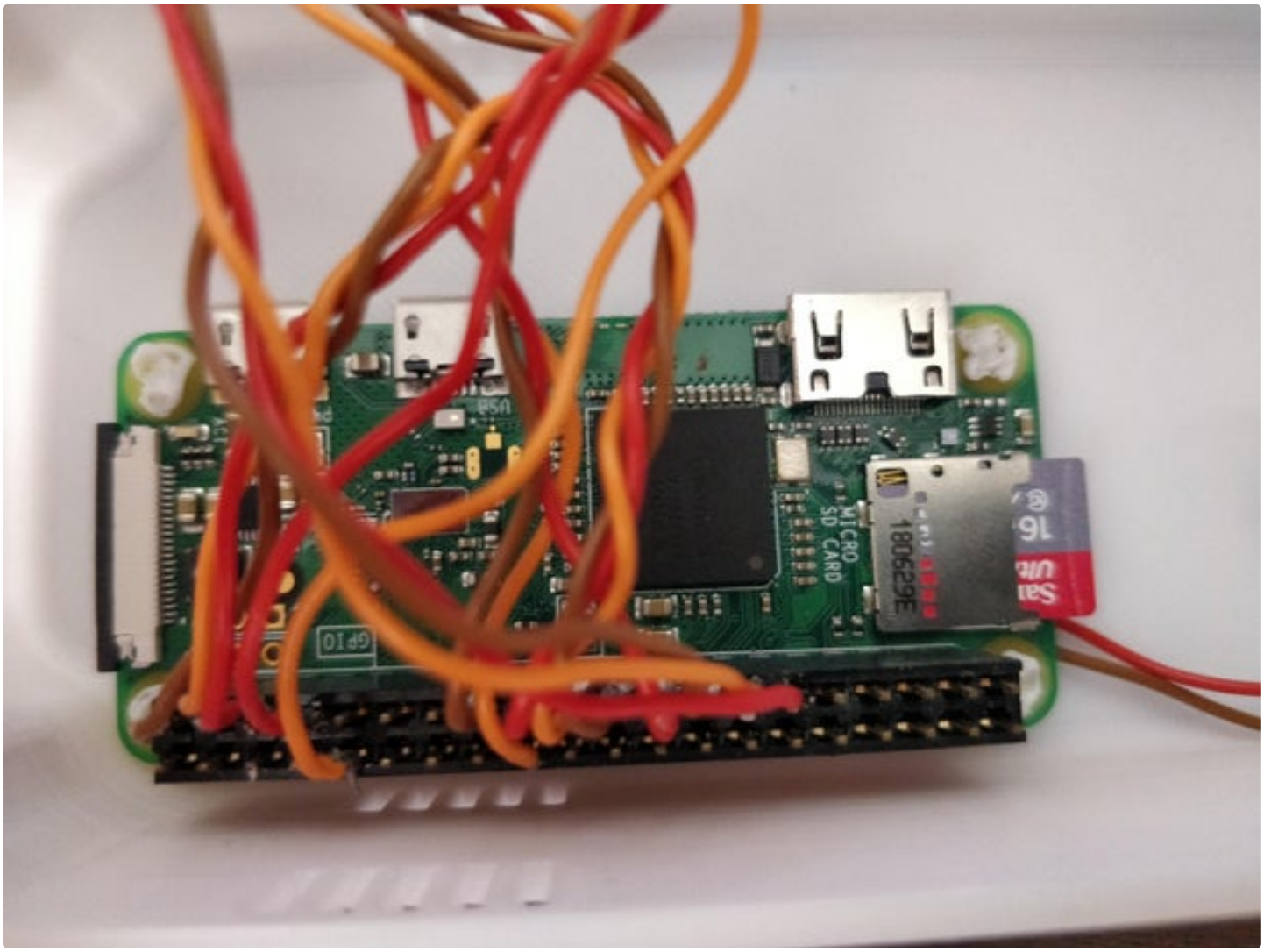
The RFID reader was connected with SPI, but the connection via I2C worked exactly the same.

For wiring, just write down what pins you use for what and put those information into the config.cfg.

Make sure that the Button-Led + Pins support **PWM** and are **not occupied by SPI or I2C**

It looks more complicated than it is.





Step 5: Setting Up the Raspberry

Since we are using the PN532 chipset, the setup is a bit tricky as we need Circuit Python libraries. This requires the installation of the Adafruit-Blinka Compatibility layer.

Also we need LibNFC to handle the communication with the RFID-reader.

Adafruit Blinka:

Follow this installation **Tutorial** HERE

After executing the Blinka-setup.py, just type yes to every question.

Libraries:

After this we can install all the libs:

```
pip3 install adafruit-circuitpython-pn532 spotipy pyky040 evdev
```

THIS is another helpful link, showing **wiring** and **installation**.

Install pigpio:

```
sudo apt-get install pigpio python3-pigpio
```

and enable the deamon:

```
sudo systemctl enable pigpiod
```

Now we need to setup the rotary encoder using the device overlay. This makes it more reliable than using the GPIO variant which is implemented though and can be switched to in the "hw_com.py" file See more information about this HERE.

This is relatively straighforward:

```
sudo nano /boot/config.txt
# (replacing {CLK_PIN} and {DT_PIN} by their real values)
dtoverlay=rotary-encoder,pin_a={CLK_PIN},pin_b={DT_PIN},relative_axis=1,steps-per-period=2
```

After closing and saving, **reboot** the Pi.

LibNFC:

Depending on wether you use SPI or I2C, there are different setup methods.

HERE is another Tutorial to setup using I2C, maybe this helps if problems occur.

```
sudo apt install libnfc5 libnfc-bin libnfc-examples
```

Then go to:

```
sudo nano /etc/nfc/libnfc.conf
```

Paste to the end:

for I2C:

```
device.name = "PN532 over I2C"
device.connstring = "pn532_i2c:/dev/i2c-1"
```

for SPI:

```
device.name = "PN532 over SPI"
device.connstring = "pn532_spi:/dev/spidev0.0:500000"
```

If you encounter problems, look up the baud rate for your specific module and replace the 500000 at the end.

```
nfc-list
```

should list your reader now.

Helpful links for troubleshooting and UART setup:

1. wiki.sunfounder.cc (Just don't use the download link, it's not working. Use apt install as shown.)
2. blog.stigok.com
3. learn.adafruit.com

Code:

You can find the code HERE on GitHub.

Clone it to your Pi and rename it for convenience using:

```
wget https://github.com/d00mfish/NFC-Spotify-Player
mv NFC-Spotify-Player NSP
cd NSP
```

```
nano config.cfg
```

Then open the "config.cfg" where you need to input all **Pins**, the **UID of your setup and learn cards** and most importantly your **Spotify client ID and secret**.

To get the UID of your card, you can scan it with a smartphone app or use the "nfc-poll" command and hold it to the reader. The UID will be printed out.

The Device section can be left as is.

Step 6: Spotify Authentification

Creating a Spotify Dev Account

Got to developer.spotify.com, click on Dashboard, log in, create an app, and give it some name and description.

After that, you find the information we need right under the Name you just chose for your app.

Then paste the Spotify client-ID and -secret into the **config.cfg** from the repository.

Assuming you did everything right, now it comes down to granting your app permission to change your live-playback.

I made sure to just include the permissions really needed. Those only affect setting and getting current playback, aswell as reading your private playlists in order to be able to write them on the cards. They are all listed in the config.cfg under scope, but leave it as is, otherwise there will be problems.

For the first run, execute the "spotify_api.py" to start the Authentification process:

```
python3 spotify_api.py
```

Now you should be prompted to open a link where you login with your Premium-Account and grant permissions:

```
pi@raspberrypi:~/NSP $ Go to the following URL: https://accounts.spotify.com/authorize?client_id=.....
```

Follow the instructions and you should get redirected to google.com. **Copy the URL** and paste it back into the console.

- If you now get **Error 404: reason: NO_ACTIVE_DEVICE** you have done **everything right!**
- If you get Error 400: Bad Request something went wrong, try again and make sure to copy and paste the whole URL.

From now on, the library should handle all the token renewal and stuff so we won't have to care about that in the future.

Even after half a year of not using the prototyping-pi, I didn't need to reauthenticate for it to work.

Step 7: First Run

To get a gist on whats going on and how the Button LEDs react, first run it in console to see the output.

First you should set your Playback device:

Set playback device:

1. Play music on Device
2. Scan device card to set it as default (device card UID must be set in config.cfg)

Then setup your first card:

Learn a card:

1. Play a Playlist or Album you want to assign to the card.
2. If you want to listen to everything from a specific artist, I recommend using the "This is _____" playlists.
3. Scan learn-card.
4. Scan empty card (Scan learn-card again to abort).

Step 8: Configure Auto-run at Boot

Unfortunately the RFID-reader was not recognized as long as it was not listed by libnfc using the nfc-list command.

(If anybody fixed this, I would be happy to know.)

Therefore I wrote a script that executes all commands for the boot:

```
cd
nano boot.sh
```

Paste this into the shell script:

```
#!/bin/sh
nfc-list
cd /home/pi/NSP
python3 main.py
```

Save and exit.

Then we execute the script from crontab:

```
crontab -e
```

and add:

```
@reboot bash /home/pi/boot.sh
```

This will auto-start the program at boot.

It's done. Everything should work now.

If you have questions, let me know.

Otherwise: **Happy listening!**