EMBEDDED SOFTWARE FOR THE IOT

# SPOTIFY DIRECT MANAGER

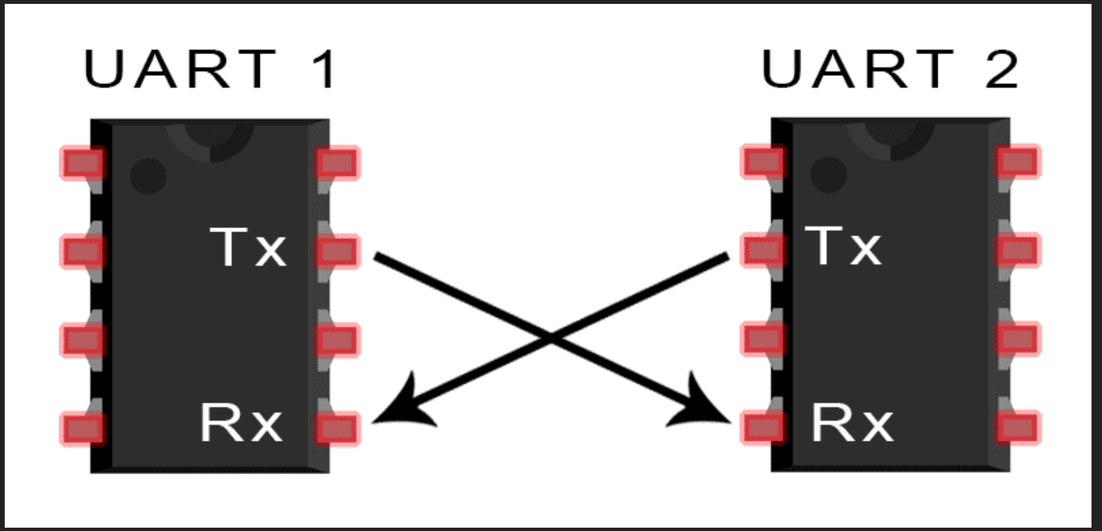
# INTRODUCTION

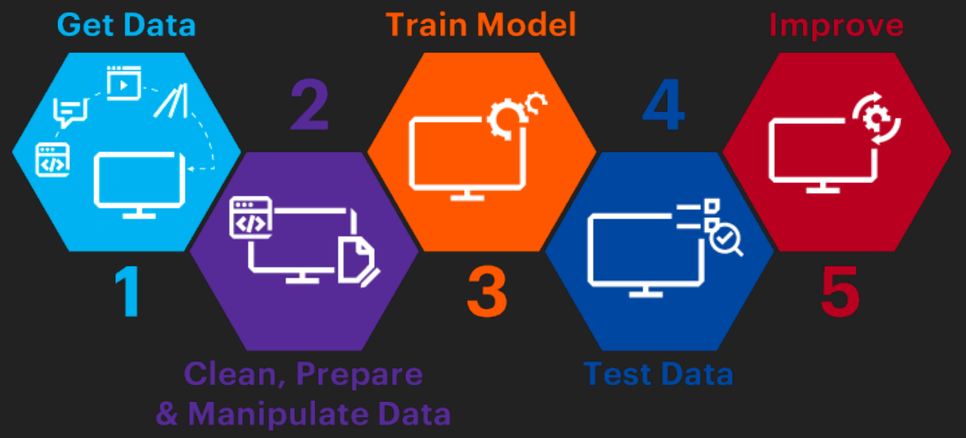
# PROBLEM STATEMENT

- Our main goal was to build a system that could manage our Spotify client remotely.
- Another key idea was to have some keyword spotting system to better manage our client

# CHALLENGES

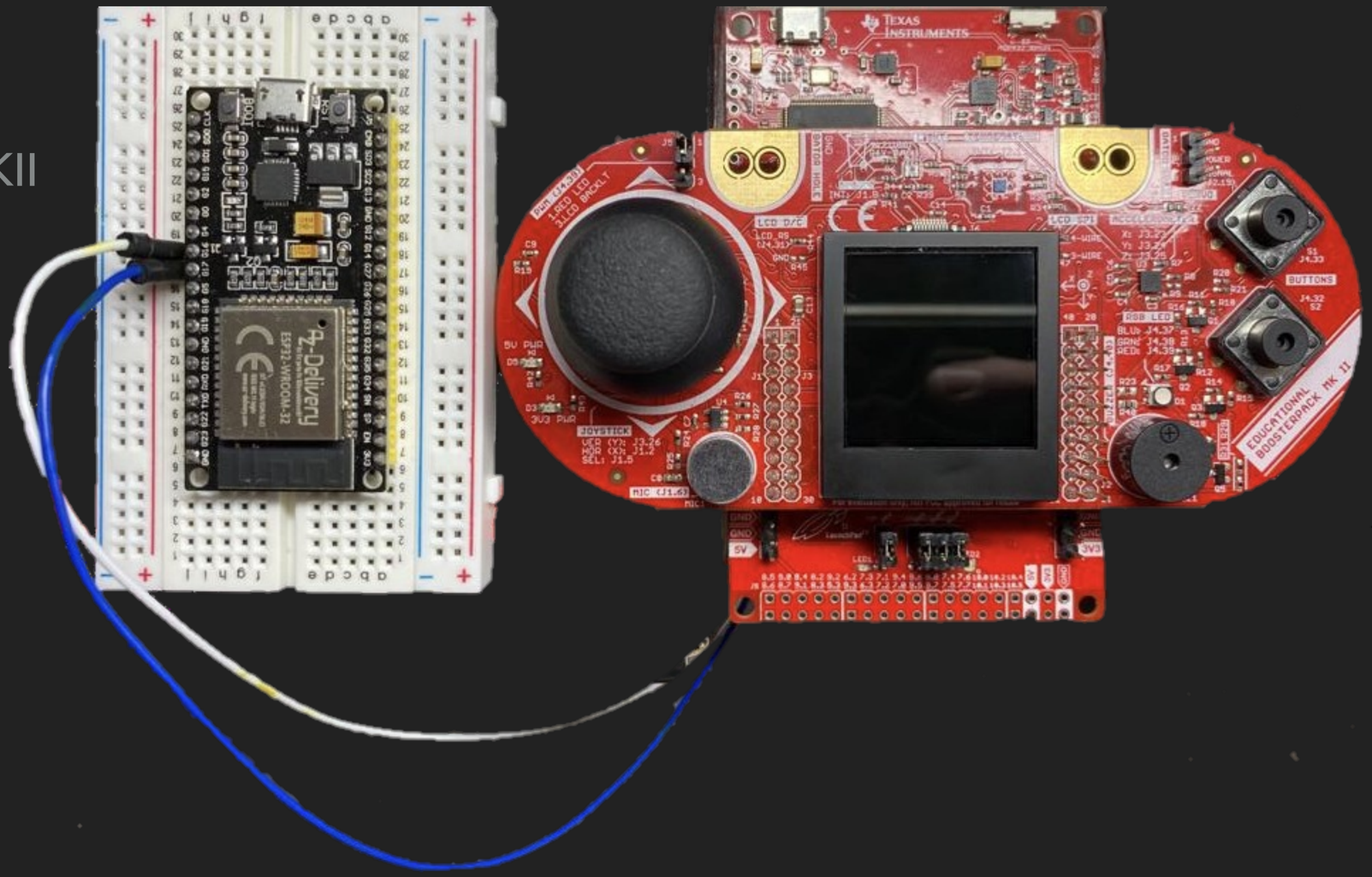
- Making the Spotify API usable with C
- Enable internet access (first through the CC3100, then with ESP32)
- Establish a communication protocol between the MSP and the ESP32
- Training a model to recognize sounds
- Integrating our ML model in our project



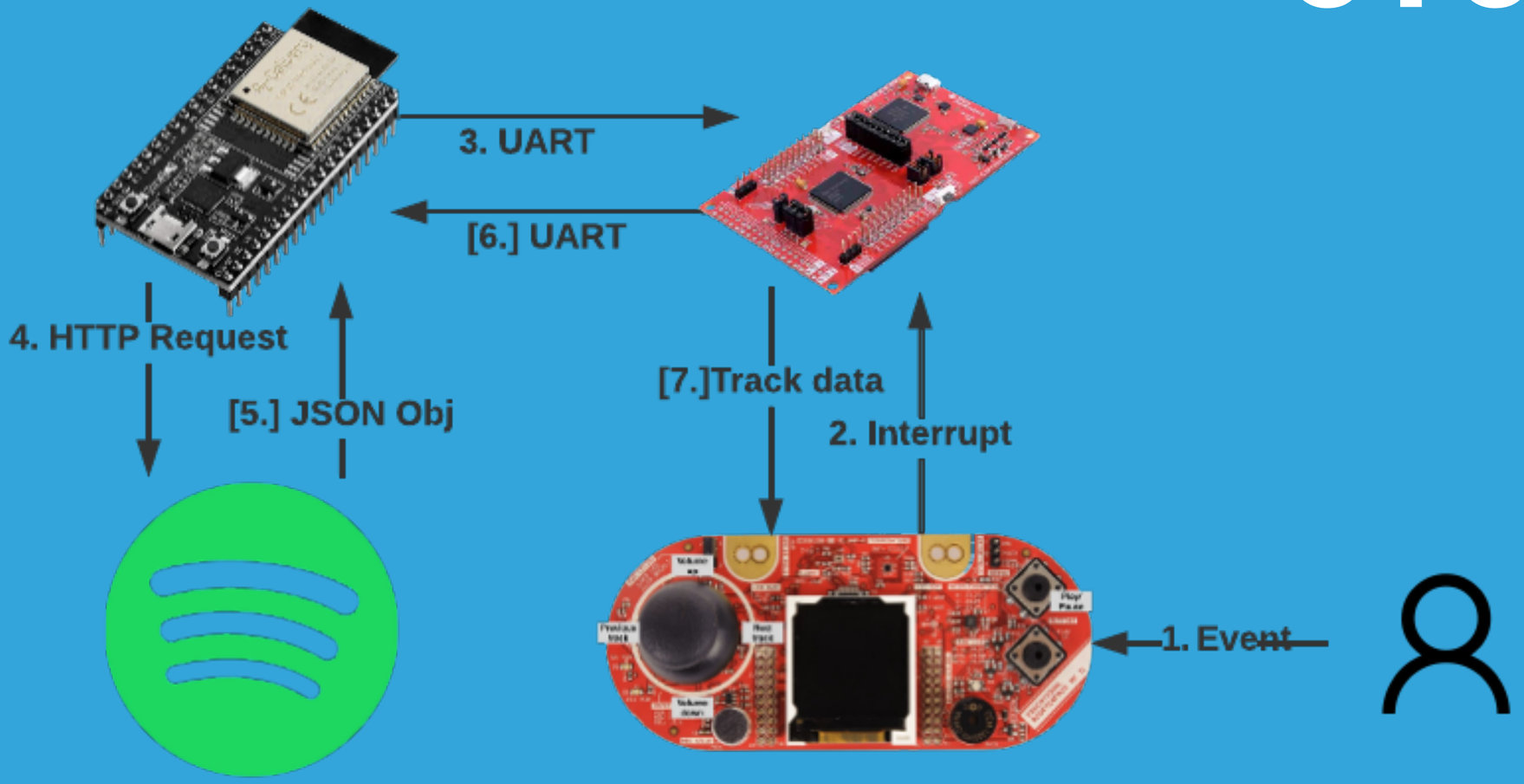



# HARDWARE

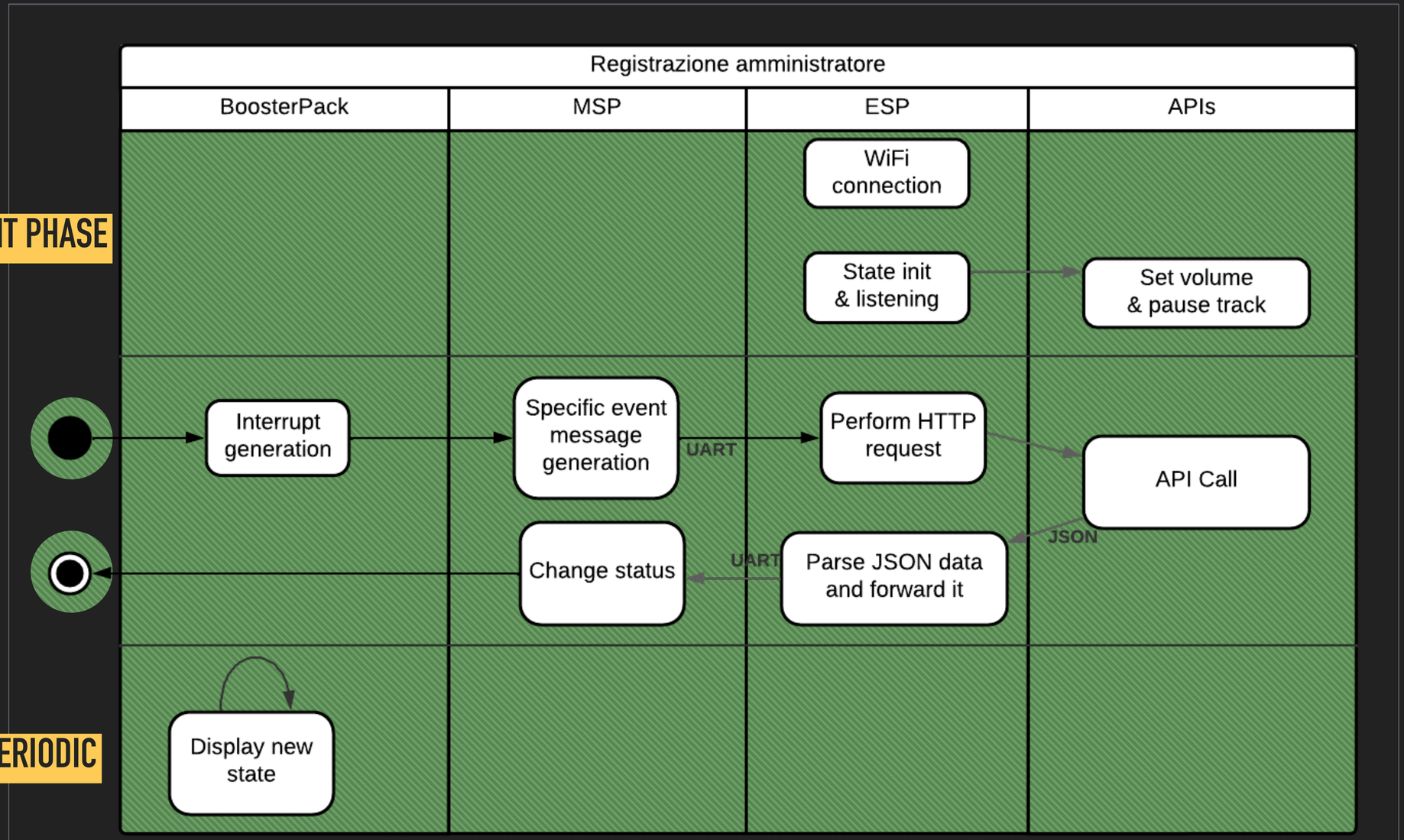
- MSP432P401R Launchpad
- MSP432P401R BoosterPackMKII
- ESP32
- USB cable
- Jumpers



# SYSTEM

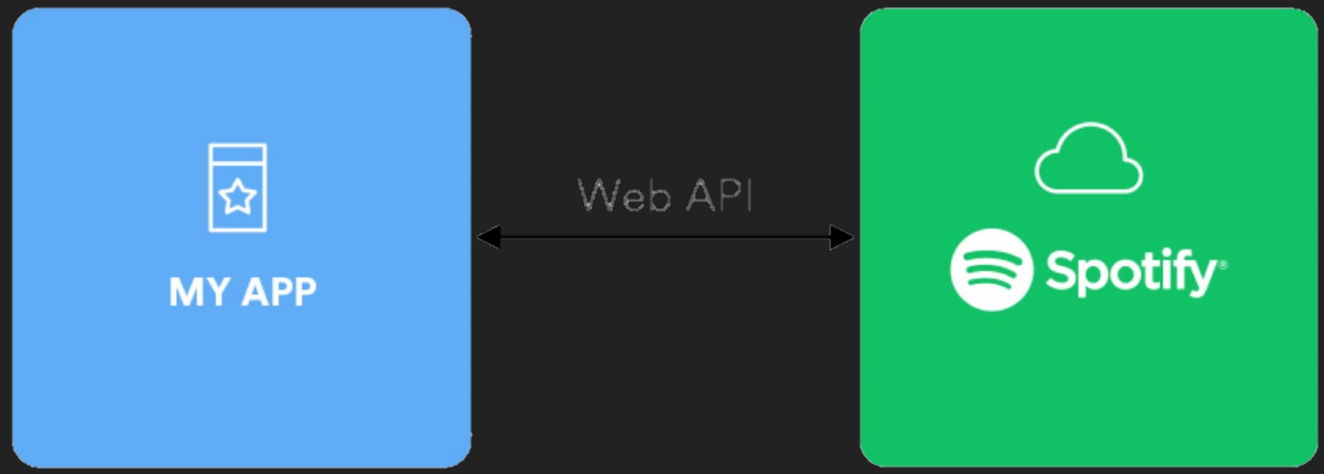

3. UART
[6.] UART
4. HTTP Request
[5.] JSON Obj
[7.]Track data
2. Interrupt
1. Event

# INTERRUPT HANDLING



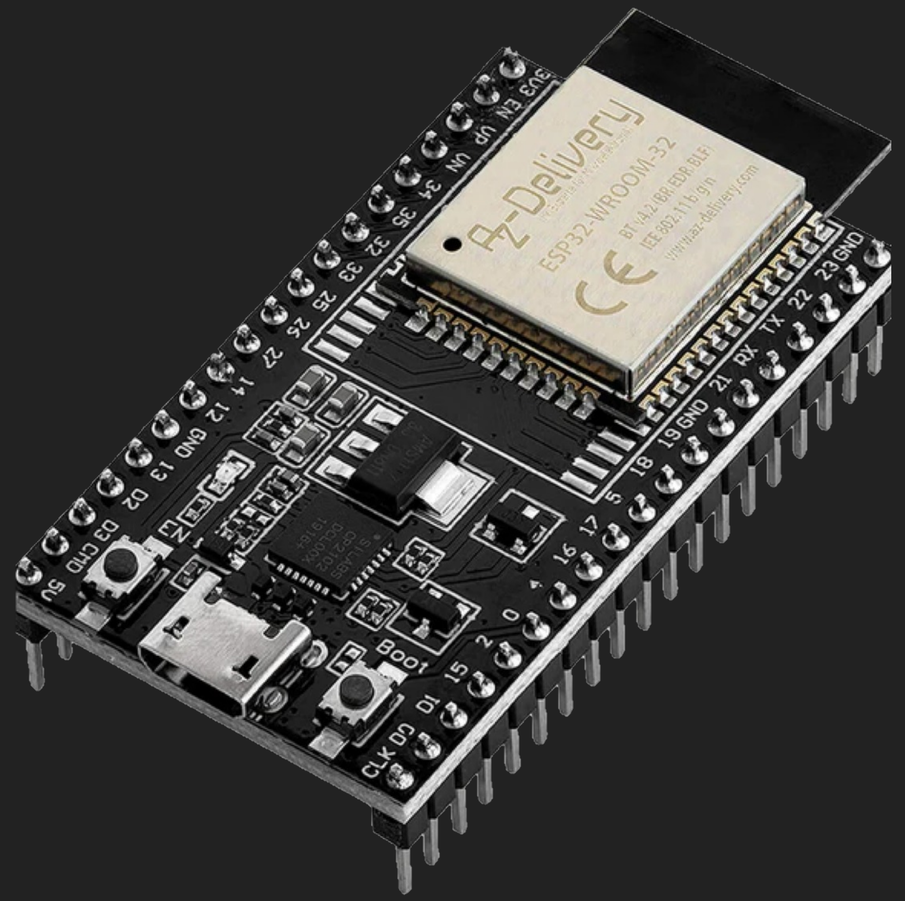
# SPOTIFY APIS SIDE

- Spotify Dev
- Getting session token
- HTTP to use the APIs
- Parsing JSON responses



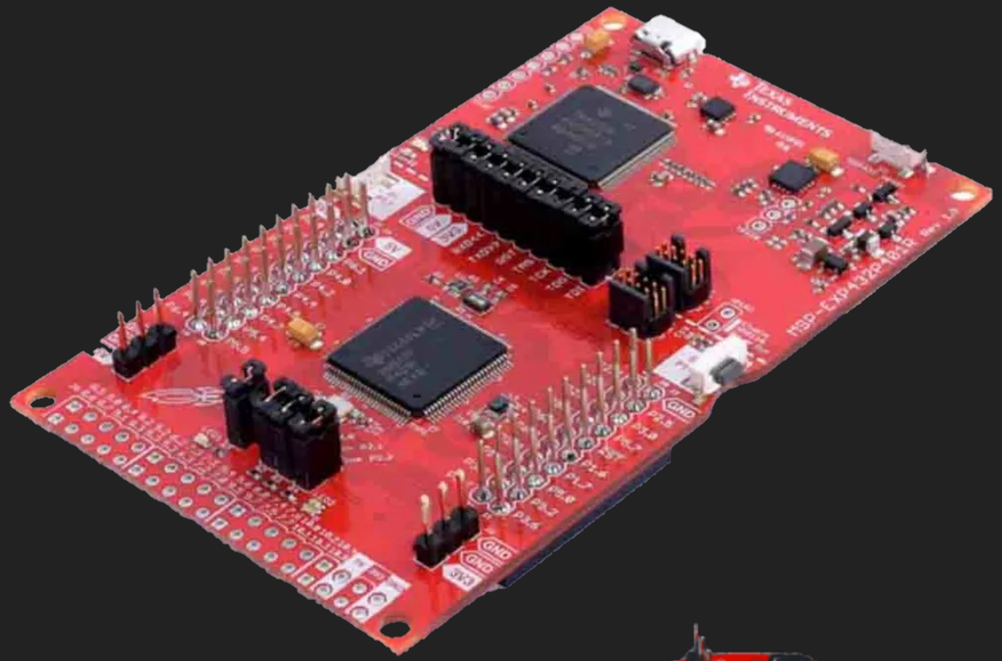
# ESP DEVELOPMENT

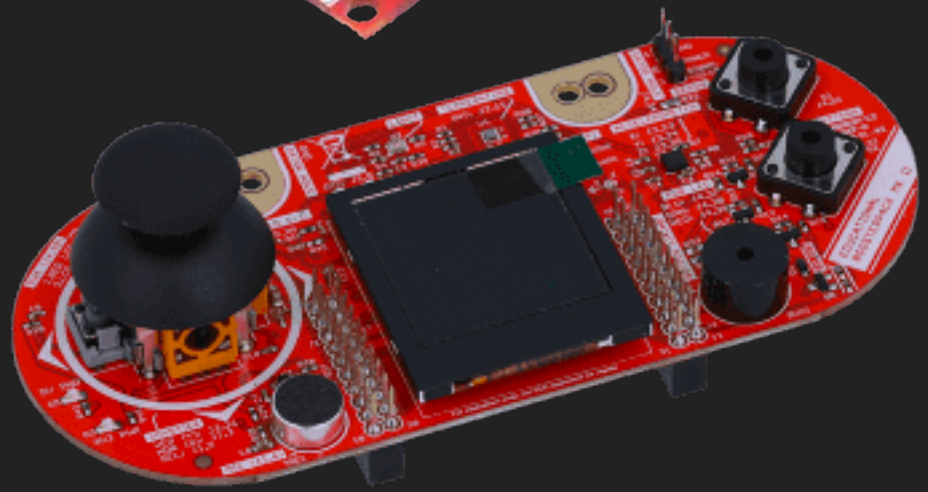
- Connecting to WiFi
- Making HTTP requests to use Spotify APIs
- Parse and filter JSON responses
- Sending data through UART to MSP



# MSP DEVELOPMENT

- Setting up interrupts with ADC
  - 2-axis joystick
  - User push buttons
  - 3-axis accelerometer (only Z)
  - Color TFT LCD
- Getting data parsed by ESP
- Designing "UI" on LCD screen
- Getting APIs response
- Finished!





# OR SO WE THOUGHT...

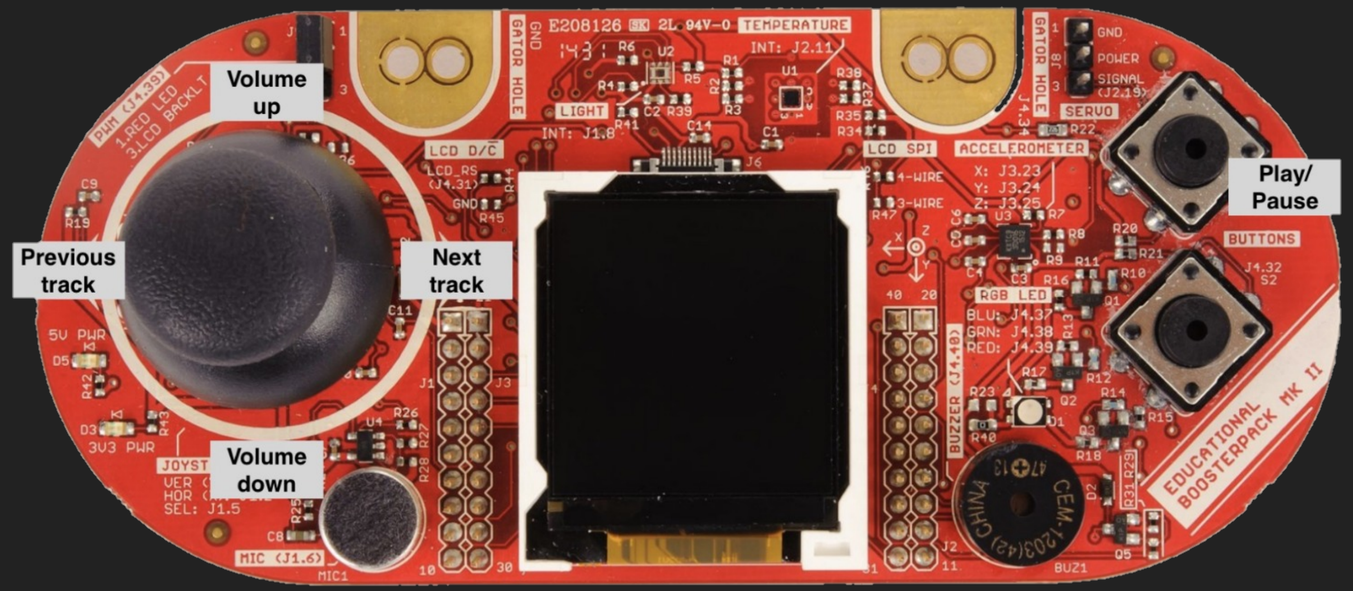
- Defining an actual UART protocol
- LSB vs MSB dilemma
- Making UART work with LCD screen

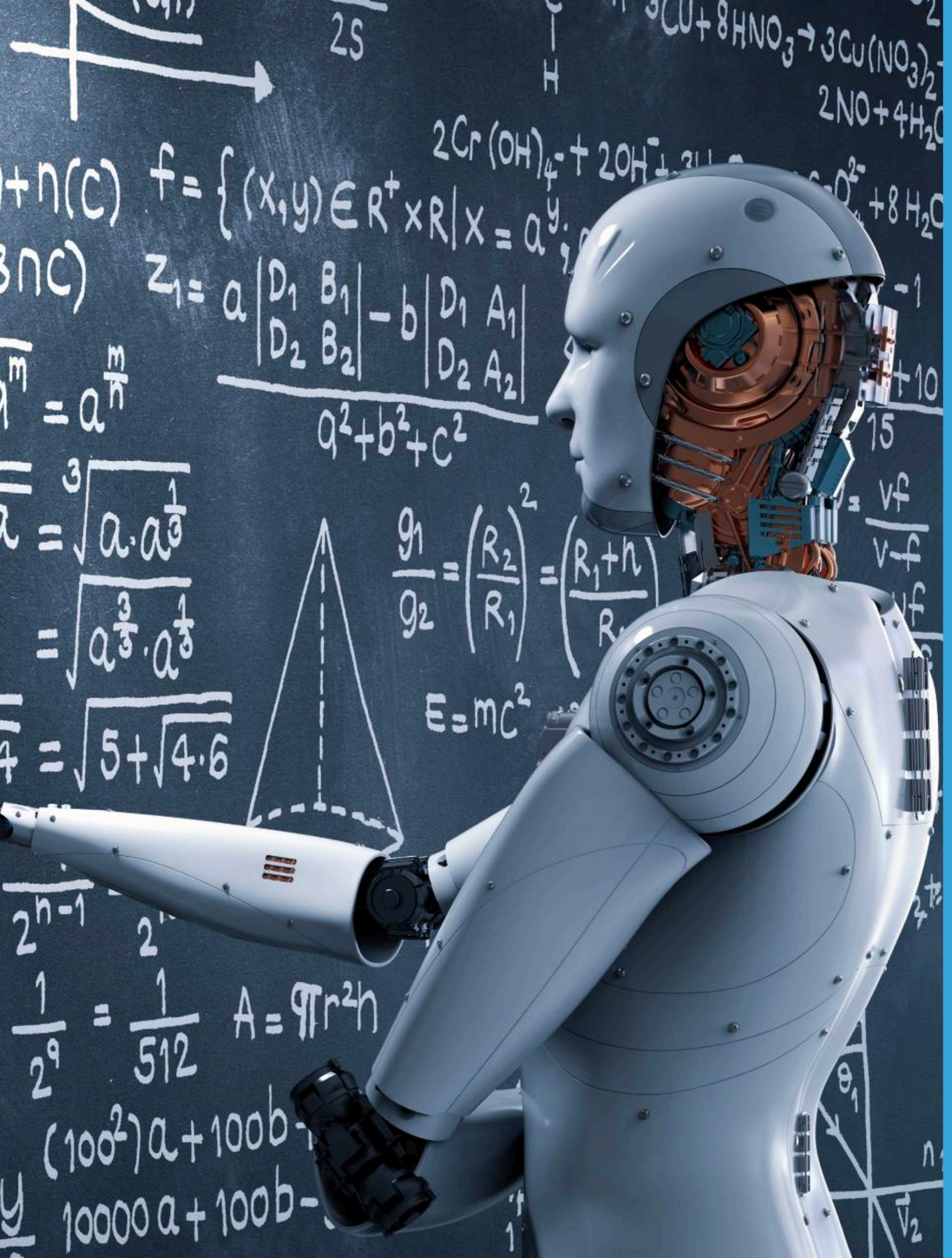
# ACTIONS

- 5 actions can be performed :
  - Volume Up/Down with joystick Y axis and accelerometer Z axis
  - Next/Previous song with joystick X axis
  - Play/Pause with Button

# DISPLAY

- Currently playing artist and song name
- Dynamic volume bar visibility
- Spinning logo!

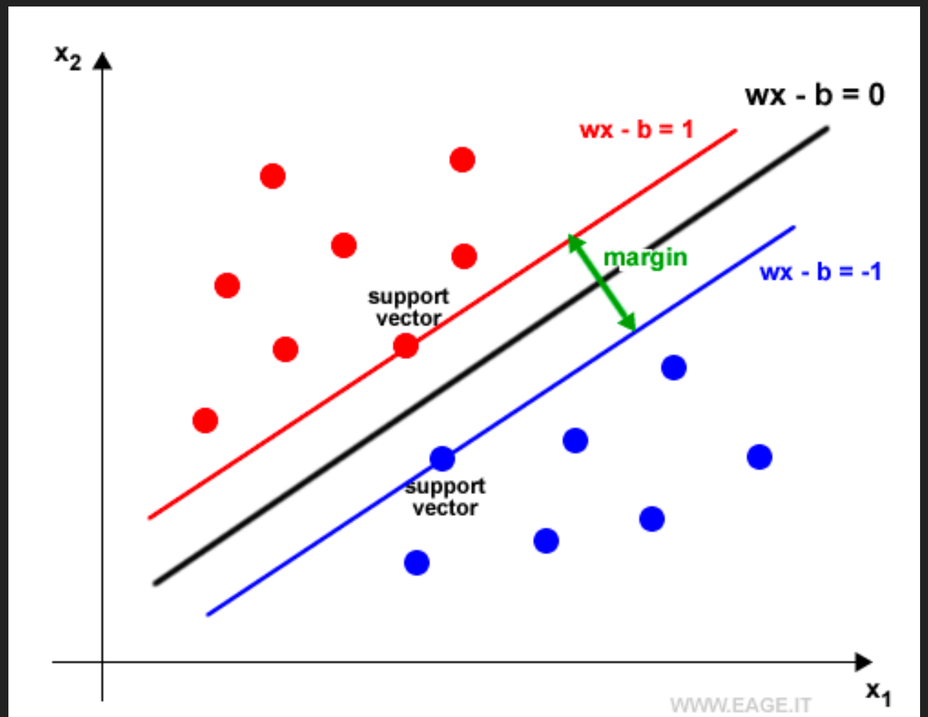


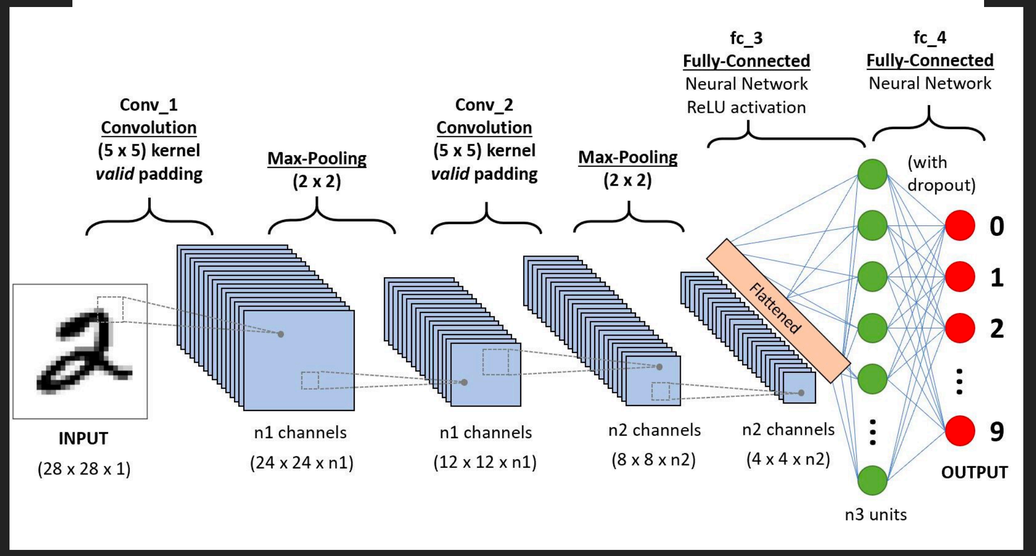


# THE ML FAILURE

# THE ML MODEL ROADMAP

- Getting the keyword dataset
- Implementing SVM
  - Low accuracy on SVM (17%)
- Development of CNN using EdgeImpulse
- Test and integration with project
  - Not implemented on our project...



# BUT CAN BE TESTED HERE!

## LOAD DATASET AND SPILT IT INTO TRAINING AND TEST DATA

```
train, test = loadDataset()
X_train = np.asarray(list(map(lambda x: x[0], train[:]["fft"])))
y_train = np.asarray(list(map(lambda x: x[0], train[:]["command"])))
X_test = np.asarray(list(map(lambda x: x[0], test[:]["fft"])))
y_test = np.asarray(list(map(lambda x: x[0], test[:]["command"])))
```

## PRINCIPAL COMPONENT ANALYSIS

```
## PCA
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

pca = PCA()
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
```

## WORK OUT MEAN AND STANDARD DEVIATION TO NORMALISE DATA

```
global means, stds, means1, stds1
means = np.zeros(num_features, dtype=np.float64)
stds = np.zeros(num_features, dtype=np.float64)
for i in range(num_features):
    means[i] = (np.mean(X_train[:,i]))
    stds[i] = (np.std(X_train[:,i]))
for i,j in range(num_data, num_features):
    X_train[i][j] = (X_train[i][j] - means[j]) / stds[j]
    X_test[i][j] = (X_test[i][j] - means[j]) / stds[j]
```

## CALL SVM CLASSIFIER

```
#================================================================================
# Init the SVM model and train it
start = time.time()
main = Main(X_train, y_train, clf, num_labels=num_features, num_data=num_data)
end = time.time()
trainingTime = end - start
# Test the model
start = time.time()
main.test_model(X_test, y_test)
end = time.time()
testTime = end - start

print(f"\033[0;32 Training time: {trainingTime}\tTest time: {testTime}")
```

# CONCLUSION