VALISORI -

IOT PROJECT

Project realised by the students:
Valentin NISTORAN
Sorina POPA

Timişoara
June, 2023

Summary

1. Introduction

Project Idea

The main idea of this project is implementing an interactive system which consists of voice controlled lights that can change colours based on the mood of the user.

Idea Description

Activated by a voice command or a button press, the system can analyse the sentiment of a spoken phrase which is used as an input for an algorithm that computes a score. This score is then associated with a colour that the Raspberry Pi will turn the LED into.

Main Features

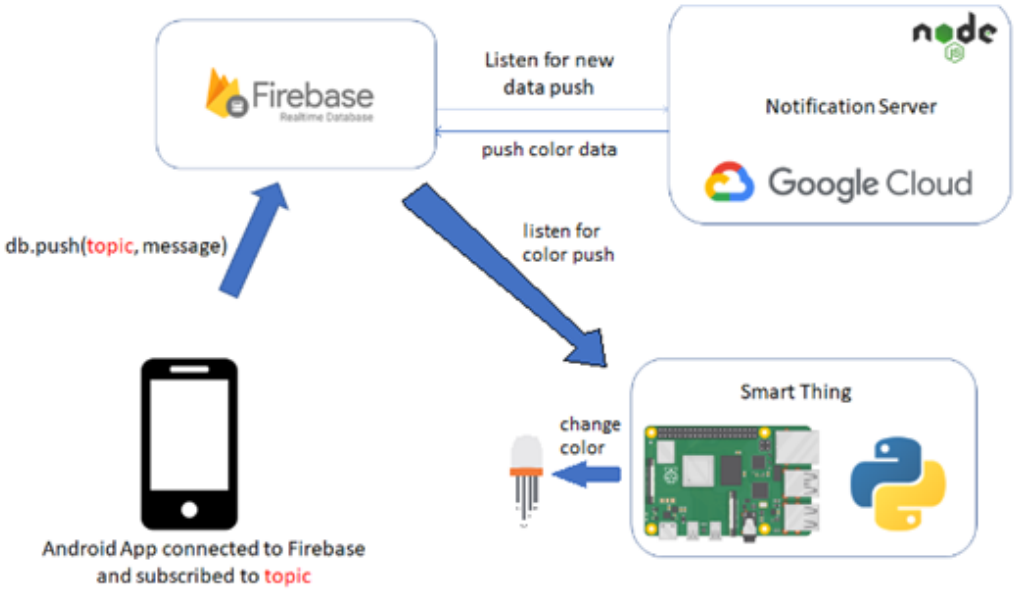
1. *Mobile App* – accessible for all Android users and having an user friendly interface, the mobile application is easy to use for everyone
2. *Raspberry Pi* – with high performance and having an internet connection, it can control the output colour of the light
3. *RGB LED* – providing a wide range of colours, the user can enjoy having a pleasant experience

Usage

Providing a new way of illuminating their personal spaces, customers will be able to enjoy a more dynamic environment in their own homes. Even so, with a wide usage area, the system will be perfect for event planners and party venues, bringing to life the festive experience.

2. System and Subsystems Architecture

System Architecture Diagram



System architecture

System Architecture Overview

Using the diagram provided above, the architecture of the system will be broken apart.

The process starts in the mobile application, which is used to send the input text to the Firebase database. At this step, the Google Cloud Service is actively listening for a new data push that, when it senses a new one, it converts the detected text into a resulting colour. The cloud service completes its part by sending the colour back into the database, from where it will be read by the Raspberry Pi (also connected to the firebase). Here, the colour is retrieved and the RGB LED colour is altered accordingly.

3. Technologies Used

I. Android Studio

The mobile application that controls the system has been developed using the Android Studio IDE using Kotlin as a programming language. Compatible with Android devices, the app uses the smartphone's microphone to listen for new data that, through a direct link, is sent to Firebase.

II. Firebase

The data provided by the user and the analysed colour value are stored in the realtime database provided by Firebase.

III. Google Cloud Provider

The principle behind the input analysis is the use of the *natural* JavaScript library, crafted for natural language processing. Three components that have been used from this library are the *WordTokenizer*, the *PorterStemmer*, and the *SentimentAnalyzer* for splitting the input text into individual tokens, reducing the words to their base form, and analysing the sentiment of the input.

IV. Raspberry Pi

- *Hardware components*: the system uses as a control board a Raspberry Pi 3 B+ which has a simple RGB module connected to it. Taking advantage of the smartphone's components, the input is controlled by the device's microphone and displayed on its screen.

- *Software components*: the code that changes the colours of the module is written in Python. Thanks to the Raspberry Pi's direct connection to the internet, the link to Firebase provides the colour values and then sends the equivalent signal through the module's pins, lighting up the LED.

4. Code Examples

➔ Mobile Application Code - valisori

As an example from the mobile application source code, the function *sendDataToFirebaseObserver()* listens for data input from the user and sends it to Firebase, using the direct link provided by the Android Studio IDE.

As presented in the picture below, creating a new reference in the database called *data*, the code has an observer for the button *Send* that gets activated when it is pressed. Using the moment of the press timestamp, the input from the user is pushed to Firebase, keeping the information in chronological order. The user will be able to see if the push has been done successfully through a *Toast* that appears at the bottom of the screen.

```
private fun sendDataToFirebaseObserver() {
    //initialise Firebase database and reference
    val database = FirebaseDatabase.getInstance()
    val timestamp = System.currentTimeMillis().toString()
    val myRef = database.getReference( path: "data")

    mainViewModel.onInputButtonClicked.observe( owner: this) { value ->
        if (value) {
            val data = userInput.editText?.text.toString().trim()
            if (data.isNotEmpty()) {
                val timestamp = System.currentTimeMillis().toString()
                val childRef = myRef.child(timestamp)
                childRef.setValue(data)
                Toast.makeText(
                    context: this,
                    text: "Data pushed to server with timestamp $timestamp",
                    Toast.LENGTH_LONG
                ).show()
            } else {
                Toast.makeText( context: this,  text: "Please enter data", Toast.LENGTH_LONG).show()
            }
        }
    }
}
```

This is only a small part of the mobile app. For more details, the code is available in the git repository.

➔ Google Cloud Provider Code - analyzer.py

The program updated to the cloud has three focus points: listening for the data push, associating a colour to the input, and the colours array.

The two pictures below present the prerequisites that are necessary for the program flow.

Reading from top to bottom, the code makes the administrative access to Firebase services possible, and it takes advantage of the *natural* module's language processing capabilities. Then it initialises a tokenizer (to facilitate further input analysis), an analyser, and a stemmer. The last two will both be part of the construction of the final sentiment analyser, together with the AFINN-111 wordlist. These lines of code lay the groundwork for sentiment analysis, enabling the assessment of textual sentiment in a Firebase-admin environment.

```
const admin = require('firebase-admin');
const natural = require('natural');
const tokenizer = new natural.WordTokenizer();
const Analyzer = natural.SentimentAnalyzer;
const stemmer = natural.PorterStemmer;
const analyzer = new Analyzer("English", stemmer, "afinn");
```

The next part reveals the colours dictionary, which is predefined, and matches a score to a specific colour. As the score computed by the analyser is variable and can have plenty of values, five fixed milestones – with corresponding colour values – seemed like the ideal implementation.

```
const colorsDictionary = {
    5: [255, 0, 127],
    2.5: [28, 232, 21],
    0: [255, 233, 0],
    '-2.5': [199, 36, 177],
    '-5': [3, 37, 126]
  };
```

The following picture illustrates how the cloud service listens for new pushes under the *data* node, and each time it detects a new input it builds a new *colors* node. It requests the analyser to retrieve a score based on the text input from the recent push, it calls the associateColor*()* method with the computed score as the argument. Then it attaches the resulting colour to the *colors* node, and finally pushes it back into the database.

```
function listenForDataPush() {
    const data = ref.child('data');
    const messageRef = ref.child('colors');
    let tokens;
    let score;
    data.on('child_added', function(requestSnapshot) {
        var text = requestSnapshot.val();
        const { spawn } = require('child_process');
        tokens = tokenizer.tokenize(text);
        score = analyzer.getSentiment(tokens);
        messageRef.set({
            value: associateColor(score)
        }).then(() => {
          console.log('Message pushed to Firebase');
        }).catch((error) => {
          console.error('Error pushing message to Firebase:', error);
        });
    }, function(error) {
      console.error(error);
    });

  }
```

Lastly, the *associateColor()* function is pretty simple. It takes the score retrieved by the analyser as the argument then, using the *reduce* method, by computing an absolute difference it finds the value from the *colorsDictionary* that is closest to the score. Finally, it returns the RGB tuple, associated with the final score.

```
function associateColor(score) {
    const roundedScore = Object.keys(colorsDictionary).reduce((a, b) => {
        return Math.abs(b - score) < Math.abs(a - score) ? b : a;
      });
      const colorTuple = colorsDictionary[roundedScore];
      return colorTuple;
}
```

➔ Raspberry Pi Code - changer.py

The following snippet is the first part of the code uploaded on the Raspberry Pi. Through it, the connection to the Firebase database is established and the setup for the GPIO pins is established. Each LED colour is linked to a pin number, which then is set to output.

```
import pyrebase
import RPi.GPIO as GPIO

RED_PIN = 17
GREEN_PIN = 27
BLUE_PIN = 22

config = {
    "apiKey": "AIzaSyCiyx86AFTnFwPkjNnI9CIhbG_iov6HuR8",
    "authDomain": "valisori-72068.firebaseapp.com",
    "databaseURL": "https://valisori-72068-default-rtdb.firebaseio.com",
    "storageBucket": "valisori-72068.appspot.com"
}

firebase = pyrebase.initialize_app(config)
db = firebase.database()

GPIO.setmode(GPIO.BCM)
GPIO.setup(RED_PIN, GPIO.OUT)
GPIO.setup(GREEN_PIN, GPIO.OUT)
GPIO.setup(BLUE_PIN, GPIO.OUT)

red_pwm = GPIO.PWM(RED_PIN, 100)
green_pwm = GPIO.PWM(GREEN_PIN, 100)
blue_pwm = GPIO.PWM(BLUE_PIN, 100)
```

The next part of the code, presented in the picture below, is the one responsible for making the RGB LED change its colour. The code actively listens for data pushes and each time one is detected, the *stream_handler()* function is called. There, the resulting colours that have been computed by the cloud are extracted in the *(r, g, b*) tuple that is used as an argument for the call to the *set_color()* function that follows. There, after some simple computations, the duty cycle for each PWM signal is modified, the LED being lit up with the desired value from the colour dictionary.

```
def set_color(r,g,b):
    red_pwm.start(r*100/255)
    green_pwm.start(g*100/255)
    blue_pwm.start(b*100/255)

def stream_handler(message):
    if message["event"] == "put":
        if message["path"] == "/":
            r,g,b = message["data"]["value"]
            print(r,g,b)
            set_color(r, g, b)

my_stream = db.child("colors").stream(stream_handler)

try:
    my_stream = db.child("colors").stream(stream_handler)

    while True:
        pass

except KeyboardInterrupt:
    my_stream.close()
    GPIO.cleanup()
```

5. Future Work

Vision

As any other product that exists, there is always room for improvement. The system will have the possibility to go through upgrades, from software to hardware ones. But most importantly, feedback from the customers will be gladly appreciated.

Expanding to Other Platforms

When it comes to software, there is always the desire to expand the product towards as many platforms as possible. Thus, the first step will be coming up with an IOS version of the application. Afterwards, there will be implementations compatible with smart cars, watches, etc. and smart homes computers.

Additionally, as the current system uses only five working colours, the range of possible colours will be expanded so the customers will have the most pleasant experience with our product.

Providing a Larger Variety of Lighting

Related to the hardware part, it would be a good idea for the colour producing element to be customisable. Currently, the project only uses a simple RGB module, which is the most practical implementation for now. In the future, the system will be compatible with other components, like RGB led strips, light rings or smart light bulbs.

This upgrade will be suitable not only for a customer having a better ambiental environment, but also for more massive situations such as concert light shows and night parties.

6. References

Bibliography

1. “Android Developers,” Android. https://developer.android.com/docs

2. “Firebase Documentation,” Firebase. https://firebase.google.com/docs?hl=en&authuser=0&_gl=1*1etmebr*_ga*MTk1MzY2ODYxOC4xNjgzOTg4ODE5*_ga_CW55HF8NVT*MTY4NjA4NjcwNi41LjEuMTY4NjA4Njc0MC4wLjAuMA

3. “Raspberry Pi Documentation,” Raspberry Pi. https://www.raspberrypi.com/documentation/

4. “Natural Language Toolkit,” NTLK. https://www.nltk.org/

5. Materials provided at the IIOTCA laboratory

Git Repository:

https://github.com/SorinaPopa/valisori