Joke generator

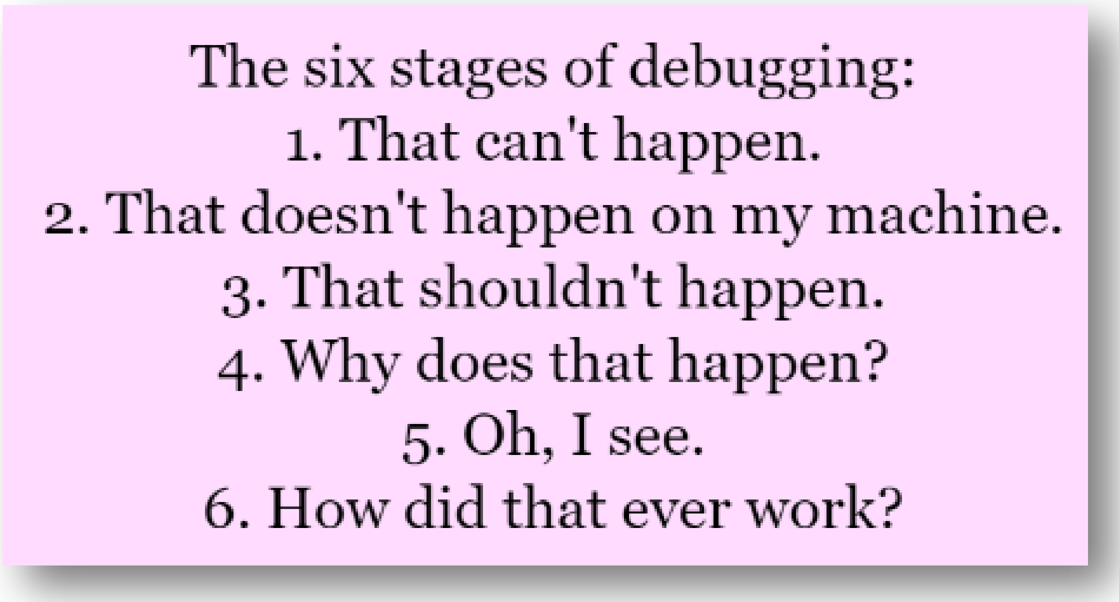
Roskilde University - Interactive Digital Systems F2024

By

Simona Kirilova Velichkova

An Erasmus student in 6th semester

stud-velichkova@ruc.dk



The six stages of debugging:
1. That can't happen.
2. That doesn't happen on my machine.
3. That shouldn't happen.
4. Why does that happen?
5. Oh, I see.
6. How did that ever work?

Characters with spaces	13 402
GitHub repository	https://github.com/pren0Sima/JokeGenerator

Table of Contents

Introduction

The inspiration for this project comes from a project I did 2 years ago on a high-level language called "Scratch"[i]. I found this course project to be the perfect opportunity to extend it. The overall idea is to make a programme which fetches jokes and displays them to the user. Since there was a big focus on ESP32, electronical components and their control over an Arduino IDE, I had to think of a way to mix the two concepts. Eventually, the formulation of the project ended up like this: A program that fetches jokes from an API and visualizes them in a browser when a physical button is pressed. There is also a LCD instructing the user what to do.

Hardware

Components

ESP32

ESP32 is a microcontroller unit which allows us to interact with electronical components as well as make connections using Wi-Fi and Bluetooth.

USB cable

It facilitates the connection between our computer and the microcontroller.

Breadboard

A breadboard is the base construction used to build electrical circuits.

Jumper wires

Jumper wires are types of electrical wires allowing connections between two points on the breadboard without soldering. We can think of them as the paths the electrical current flows through.

LCD

"A liquid-crystal display (LCD) is a flat-panel display or other electronically modulated optical device that uses the light-modulating properties of liquid crystals combined with polarizers."[ii]

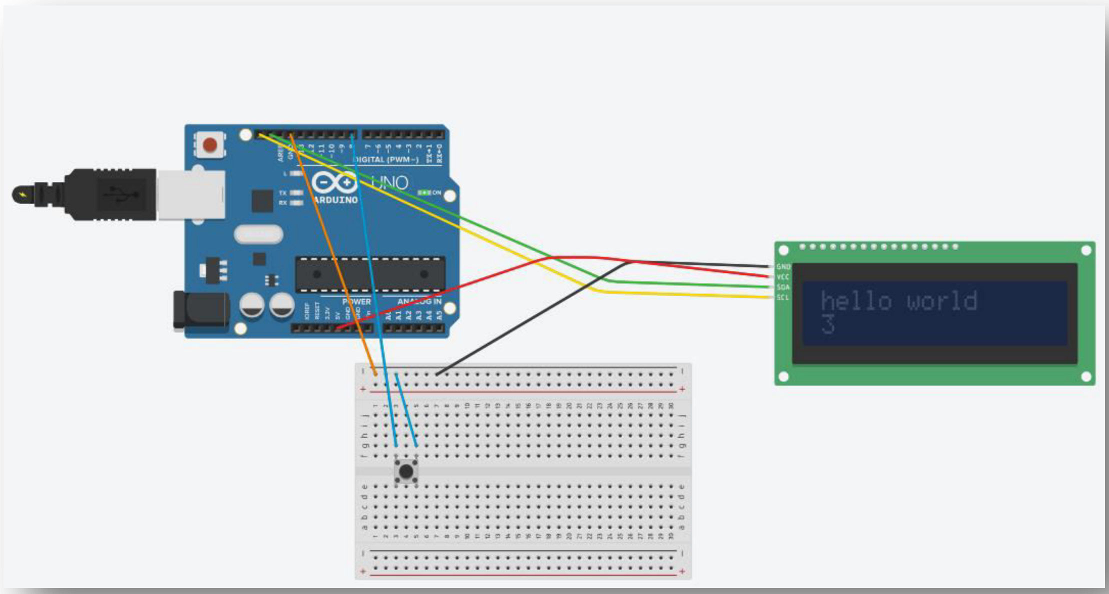
Button

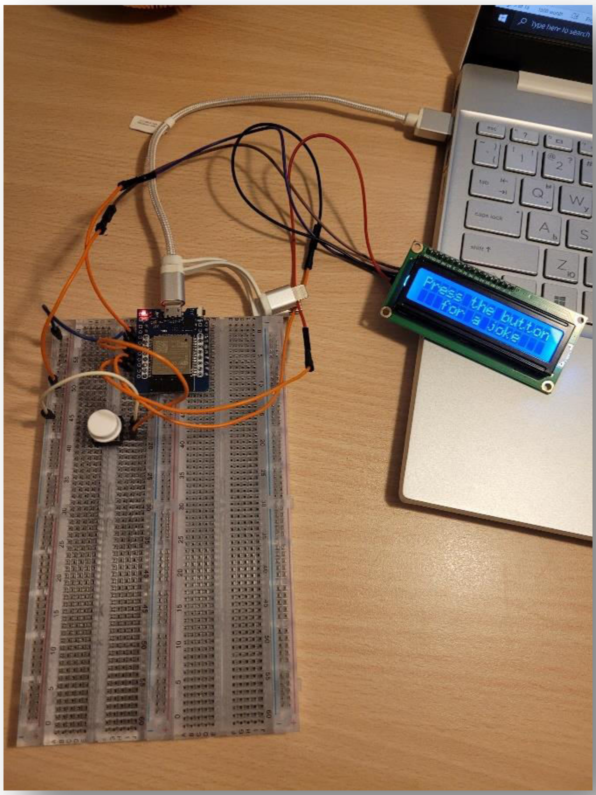
It is an electronical switch – it has two states – low and high. Depending on it, electrons flow through the circuit or do not.

Setup

Circuit view

The scheme below uses an Arduino One instead of an ESP32 but the connection is analogous. It is here for better visibility. An actual picture of the setup is included below it. It was created using Autodesk tinkercad.[iii]





The actual setup

The display pins are connected as follows:

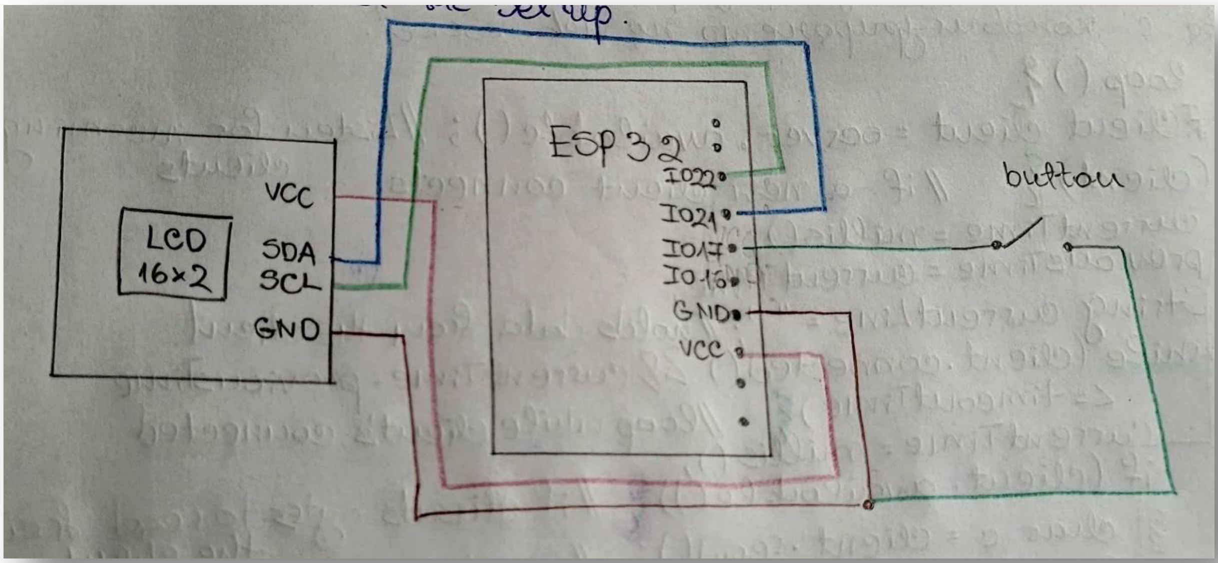
I2C	ESP32
GND	GND
VCC	VCC
SDA	IO 21
SCL	IO 22

Button	ESP32
LOW	GND
HIGH	IO 17

The button connections are as follows:

Schematic view

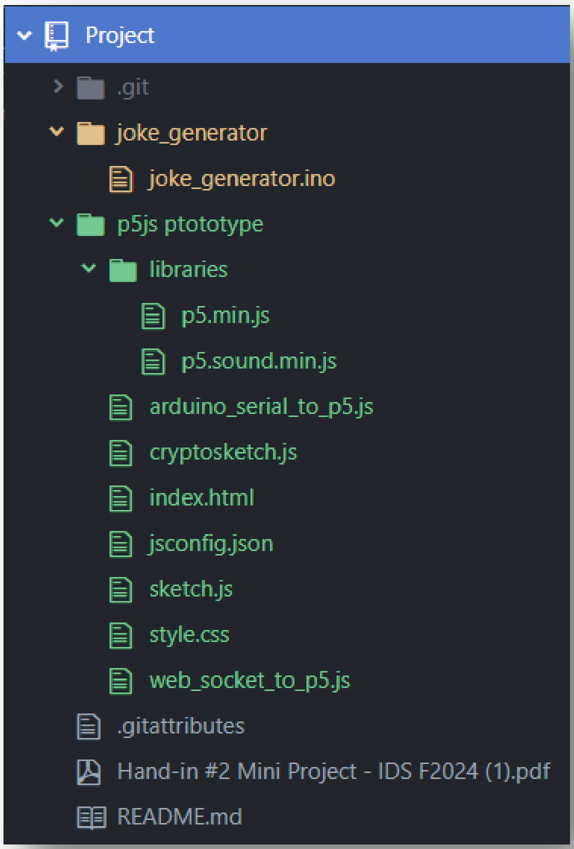
This is the schematic view of the circuit.



Code

Almost every function was inspired by code found in the study materials from the lectures.

A key word in the code that I wrote is Modularization. I tried my best to break down all tasks in separate functions for the sake of readability and order. All functionality is contained in a single file called ***joke_generator.ino***.



```
Project
  .git
  joke_generator
    joke_generator.ino
  p5js ptototype
    libraries
      p5.min.js
      p5.sound.min.js
    arduino_serial_to_p5.js
    cryptosketch.js
    index.html
    jsconfig.json
    sketch.js
    style.css
    web_socket_to_p5.js
  .gitattributes
  Hand-in #2 Mini Project - IDS F2024 (1).pdf
  README.md
```

The file structure of the project looks like this:

The ***joke_generator*** folder is where the ***joke_generator.ino*** file is.

The folder ***p5js prototype*** is the p5js implementation of the project.

.gitattributes contains the history of the project.

Hand-in #2 Mini Project – IDS F2024 (1).pdf is the assignment.

README.md is the file that explains the project in a nutshell.

Functions

setUpButton

It sets up the button on the breadboard. The default value of the pin is 17.

Reference material: https://learn.hobye.dk/kits/ids-kit#h.7ifbt8jfu0gd

Implementation

```
void setUpButton(int button_value = 17) {
  BUTTON_PIN = button_value;
  pinMode(BUTTON_PIN, INPUT_PULLUP);
}
```

setUpDisplayMessage

It sets up the text displayed on the LCD. The message is hardcoded inside and it is "`Press the button`" + "`   for a joke`".

Reference material: https://randomnerdtutorials.com/esp32-esp8266-i2c-lcd-arduino-ide/

Implementation

```
void setUpDisplayMessage() {
  int lcdColumns = 16;
  int lcdRows = 2;
  LiquidCrystal_I2C lcd(0x27, lcdColumns, lcdRows);
  lcd.init();
  lcd.backlight();
  lcd.setCursor(0, 0);
  lcd.print("Press the button");
  lcd.setCursor(0, 1);
  lcd.print("   for a joke");
}
```

connectToWifi

This function connects the ESP32 to a WiFi network. The network name and password are passed as arguments. The function displays messages to the serial monitor in order to keep track of the process of connecting, as well as the IP address assigned to the device. After establishing a connection, a server is started, so as to allow the ESP32 to serve web pages and handle HTTP requests.

Reference material: https://learn.hobye.dk/kits/iot-tutorial#h.4xts4s7hg55m

Implementation

```
void connectToWifi(char *name, char *pass) {
  Serial.print("Connecting to ");
  Serial.println(name);
  WiFi.begin(name, pass);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print("Connecting to Wi-Fi.");
  }
  Serial.println("\nConnected to Wi-Fi");
  // Print local IP address and start web server
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
  server.begin();
}
```

getDataFromAPI

It sends a GET request to an API and if the transaction is successful, or in other words – if a joke in our case is fetched – it is returned by the function. Otherwise, an error message is displayed.

Reference material: https://randomnerdtutorials.com/esp32-https-requests/#esp32-https-requests-wificlientsecure

Implementation

```
String getDataFromAPI(char *jokeAPI) {
  HTTPClient http;
  Serial.print("Sending HTTP GET request to API: ");
  Serial.println(jokeAPI);

  if (http.begin(jokeAPI)) {
    int httpCode = http.GET();
    if (httpCode > 0) {
      if (httpCode == HTTP_CODE_OK) {
        String payload = http.getString();
        return payload;
      }
    } else {
      Serial.printf("HTTP GET failed, error: %s\n",
http.errorToString(httpCode).c_str());
      return "";
    }
    http.end();
  } else {
    Serial.println("Unable to connect to server");
    return "";
  }
}
```

extractJoke

It is meant to extract just the joke from a json file. Since currently there are two types of jokes: single-liners and two-part ones, there are two cases covered in the function. Of course, if the joke does not fit into either category, the whole json file is returned.

Reference material: https://www.arduino.cc/reference/en/language/variables/data-types/stringobject/

Implementation

```
String extractJoke(String json) {
  int start;
  int end;
  String extractedJoke;
  if ((start = json.indexOf("\"type\": \"twopart\"")) != -1) {
    start = json.indexOf("\"setup\": \"") + 10;
    end = json.indexOf("\"flags\": {") - 7;
    extractedJoke = json.substring(start, end);
    extractedJoke.replace("    \"delivery\": \"", "");
    extractedJoke.replace("\",", "");
    return extractedJoke;
  }
  else if (json.indexOf("\"type\": \"single\"") != -1) {
    start = json.indexOf("\"joke\": \"") + 9;
    end = json.indexOf("\"flags\"") - 7;
    return json.substring(start, end);
  } else return json;
}
```

formHeader

It forms the necessary HTTP header that includes the response code (200) and content type sent to the client.

Reference material: https://learn.hobye.dk/kits/iot-tutorial#h.4xts4s7hg55m

Implementation

```
void formHeader(WiFiClient client) {
  client.println("HTTP/1.1 200 OK");
  client.println("Content-type:text/html");
  client.println("Connection: close");
  client.println();
}
```

loadPage
This is the HTML and css code used for the web page. It displays the joke in the browser.

Reference material: https://learn.hobye.dk/kits/iot-tutorial#h.4xts4s7hg55m

Implementation

```
void loadPage(WiFiClient client, String joke) {
  client.println("<!DOCTYPE html><html data-bs-theme=\"light\" lang=\"en\">");
  client.println("<head><meta charset=\"utf-8\">");
  client.println("<meta name=\"viewport\" content=\"width=device-width, 
initial-scale=1.0, shrink-to-fit=no\">");
  client.println("<title>Joke generator!</title></head>");
  client.println("<body style=\"background: #bbdaff;\"><p style=\"font-size: 
25px; font-family: 'Courier New', monospace; font-weight: bold; color: 
#660033; text-align: center;\">");
  client.println(joke + "</p></body></html>");
  client.println();
}
```

setup
It handles the set up for the serial monitor, the LCD, the button and the WiFi, as well as providing some helpful messages in the serial monitor.

Implementation

```
void setup() {
  Serial.begin(9600);
  delay(1000);
  Serial.println("Before setting up the display.");
  setUpDisplayMessage();
  Serial.println("Display set up.");
  Serial.println("Setting up the button.");
  setUpButton(17);
  connectToWifi("Simona's Galaxy A51", "nnal8860");
}
```

loop
The control over the whole program is here. When the button is pressed, jokes are displayed both on the serial monitor and on the browser.

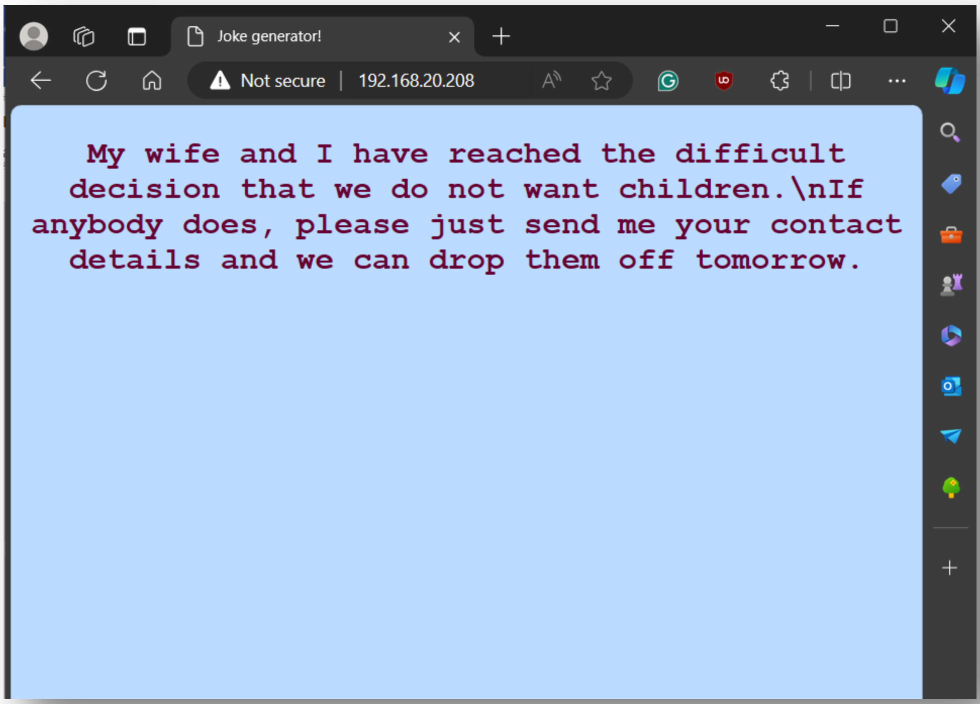
Implementation

```
void loop() {
  int buttonState = HIGH;
  int lastButtonState = HIGH;
  buttonState = digitalRead(BUTTON_PIN);
  if (buttonState == LOW && lastButtonState == HIGH) {
    WiFiClient client = server.available();
    if (client) {
      Serial.println("New Client.");
      if (client.available()) {
        char c = client.read();
        Serial.write(c);
        formHeader(client);
        Serial.println("Button pressed and now fetching joke...");
        delay(250);
        String joke = extractJoke(getDataFromAPI(jokeAPI));
        Serial.println(joke);
        loadPage(client, joke);
      }
      client.stop();
      Serial.println("Client disconnected.");
      Serial.println("");
    }
  }
}
```

Testing

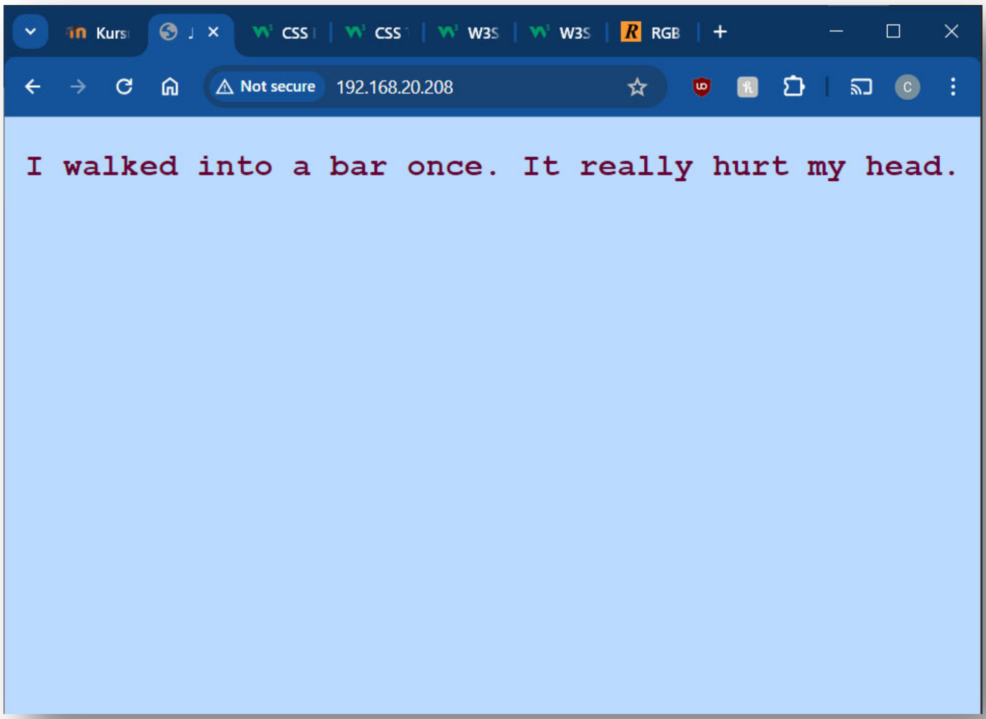
The program was tested on two browsers.

Microsoft edge



My wife and I have reached the difficult decision that we do not want children.\nIf anybody does, please just send me your contact details and we can drop them off tomorrow.

Google chrome



I walked into a bar once. It really hurt my head.

Reflection

Joke formatting

I made an effort to find a C++ library that allows json data manipulation similar to the one JavaScript has. Possible ones are ***nlohmann/josn.hpp***, ***folly/dynamic.h*** and ***folly/json.h***. I tried to implement them, but at some point, I figured out that it would take me way more time than if I were to write the functionality myself. It would have been great to add them for scalability, they would allow flexibility with other APIs, as well as better formatting.

Currently, the jokes from this API are extracted successfully, but sometimes there are some formatting literals included inside the text. Example:

```
\"Knock, knock.\"\n\"Who's there?\"\n\n[very long pause]\n\n\"Java.\"
```

P5js

Another possible solution to the previous point would be to handle the http requests through JavaScript, since it is way more convenient.

Additionally, the website design would be easier to manipulate with p5js. Currently it is generated with HTML syntax, which is longer and more prone to mistakes.

Security

The current version of the project is insecure, which means it is more vulnerable to hacker attacks. However, it is beyond the scope of the project idea to add security measures such as a TLS/SSL certificate. No sensitive information (no passwords, usernames and personal details) is found in it, so encrypting data is redundant.

User experience

For the time being the flow of the program is not smooth. The button sometimes needs to be pressed a couple of times in order to fetch the jokes and for them to be displayed on the site. It is still under investigation what the exact reason is.

Hard-coding

I am fully aware this is a bad practice, but there are some places in which values are hard-coded.

setUpDisplayMessage

The massage in this function is currently hard-coded, but it would be better if it was passed as an argument. Since there are two rows, perhaps the data could be passed as an array of type string(`string arr`) or as a two-dimensional char array (`char** arr`).

Another hard-coded part in this function is the number of lcd columns and rows, as well as the address where the display can be accessed at. To solve this problem a I2C scanner function that determines the starting address could be added. When it comes to the number of lights, I still have not found a suitable solution.

extractJoke

There is a fair amount of hard-coding here as well. Because of it, it probably won't be possible to be used for different joke APIs, since the format may differ.

Modularization

More abstractions could be moved into separate functions. For example, the joke cases `extractJoke` could be extracted into two separate ones, which could be added in it. It would look something like this:

```
String extractJoke(String json) {
  int start;
  String extractedJoke;
  if ((start = json.indexOf("\"type\": \"twopart\"")) != -1) {
    return extractTwoPartJoke((json.indexOf("\"setup\": \"") + 10),
(json.indexOf("\"flags\": {") - 7));
  }
  else if (json.indexOf("\"type\": \"single\"") != -1) {
    return extractSingleJoke((json.indexOf("\"joke\": \"") + 9),
(json.indexOf("\"flags\"") - 7))
  } else return json;
}
```

It seems a little bit cleaner to me.

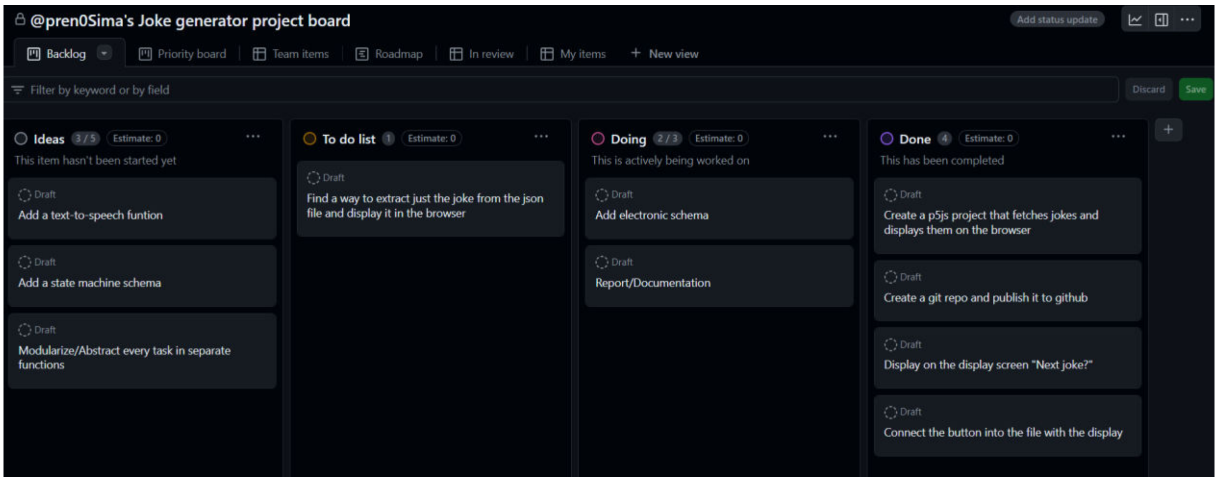
Appendices

GitHub

The project is available on my GitHub page.[iv]

Kanban board

In order to keep track of the process and organize better I used the built-in kanban board in GitHub.



@pren0Sima's Joke generator project board

Backlog | Priority board | Team items | Roadmap | In review | My items | + New view

Filter by keyword or by field

Ideas 3 / 5 Estimate: 0
This item hasn't been started yet
- Draft — Add a text-to-speech funtion
- Draft — Add a state machine schema
- Draft — Modularize/Abstract every task in separate functions

To do list 1 Estimate: 0
- Draft — Find a way to extract just the joke from the json file and display it in the browser

Doing 2 / 3 Estimate: 0
This is actively being worked on
- Draft — Add electronic schema
- Draft — Report/Documentation

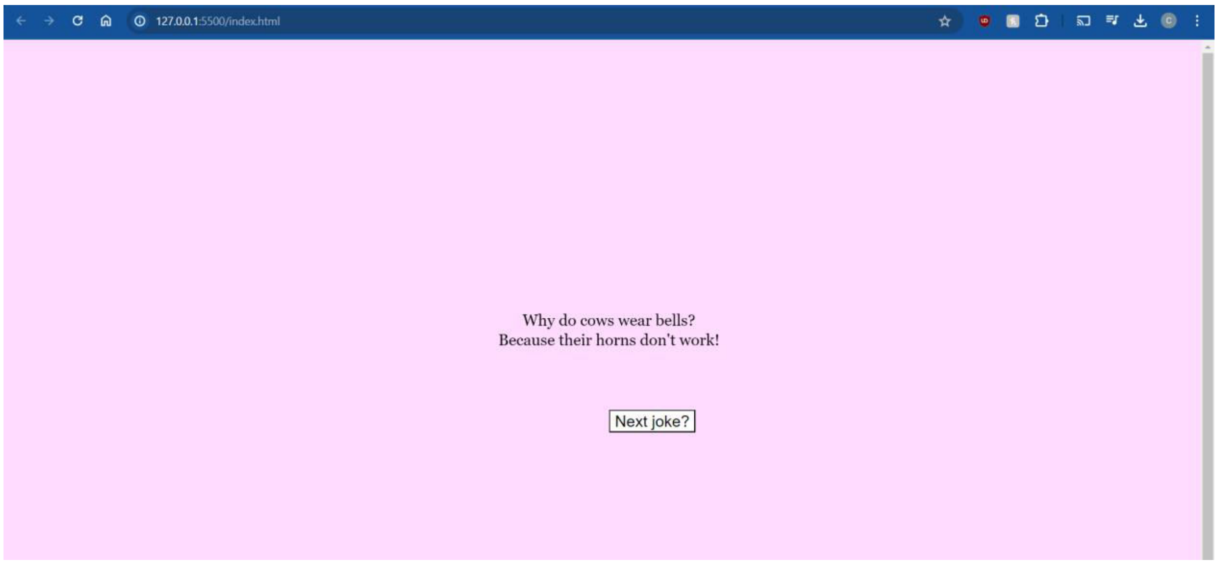
Done 4 Estimate: 0
This has been completed
- Draft — Create a p5js project that fetches jokes and displays them on the browser
- Draft — Create a git repo and publish it to github
- Draft — Display on the display screen "Next joke?"
- Draft — Connect the button into the file with the display

GitHub desktop

Git commands were executed using GitHub desktop. Thanks to it, a GitHub repository was created, all changes to the project could be tracked and all commits are visible in the GitHub project page.

P5Js prototype

The development process started with this prototype. I decided to keep it in the project folder in case I feel inspired to connect it to the ESP32 part of the project.



127.0.0.1:5500/index.html

Why do cows wear bells?
Because their horns don't work!

---

[i] https://scratch.mit.edu/projects/708151217/
[ii] https://en.wikipedia.org/wiki/Liquid-crystal_display
[iii] https://www.tinkercad.com/dashboard
[iv] https://github.com/pren0Sima/JokeGenerator