Summer Fellowship Report

On

Arduino On Cloud

Submitted by

Navonil Das
Meet Shah

Under the guidance of

Prof. Kannan Moudgalya
Chemical Engineering Department
IIT Bombay

May 2020

Acknowledgement

We, the summer interns of the **FOSSEE - Arduino On Cloud** are overwhelmed in all humbleness and gratefulness to acknowledge our deep gratitude to all those who have helped us put our ideas to perfection and have assigned tasks well above the level of simplicity and into something concrete and unique. We wholeheartedly thanks **Prof. Kannan M. Moudgalya** for having faith in us, selecting us to be a part of his valuable project and for constantly motivating us to do better. We thanks **Mr. Nagesh Karmali** and **Ms. Firuza Aibara** for providing us the opportunity to work on this project. We are also very thankful to our mentors for their valuable suggestions. They were and are always there to show us the right track when needed help. With help of their brilliant guidance and encouragement, we all were able to complete our tasks properly and were up to the mark in all the tasks assigned. During the process, we got a chance to see the stronger side of our technical and nontechnical aspects and also strengthen our concepts. Last but not the least, we sincerely thank all our other colleagues working in different projects under **Prof. Kannan M. Moudgalya** for helping us evolve better with their critical advice.

Declaration

We declare that this written submission represents our ideas in our own words and whenever others' ideas or words have been included, We adequately cited and referenced the original sources. We declare that We have properly and accurately acknowledged all sources used in the production of this thesis.

We also declare that We have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will be a cause for disciplinary action by the Institute and can also evoke penal action from the sources which have not been properly cited or from whom proper permission has not been taken when needed.

Navonil Das
Meet Shah

Contents

List of Figures

Chapter 1

Introduction

Currently, the most of products available are not in open source which limits people from using it. This project is about to develop an open source Web Application for building **Arduino Projects**.

Arduino On Cloud is a Web Application which simulates 8 bit microcontroller mostly the Arduino Family.It is an alternative for the software like tinkercad and proteus as both of them are private software and proteus is paid software. It will be easy to prototype for example user may perform wrong connection which can damage the component. This can be overcomed through our application.

This Application allows the user to choose components from the lists and drag-and-drop on the main workspace for building circuitry with component name and value.User can connect components using virtual wires which will stick to the pins of the components legs, parts of Arduino etc.Whole circuit can be stored in a file format at the backend by just a button for uploading.

Secondly, it also allows to compose the code and provides simulation of the circuit by retreiving Output in console(terminal).

1.1 Project Requirements

We are using the following tools during development

- Angular 7 (v7.0) - used for front-end development.
- Material Design UI Angular.
- Bootstrap (v4.5).
- Rahphaeljs (v2.2.7) - Javascript library for vector graphics on web
- AVR8js(v0.9.0) - Open-source Arduino simulator based on JavaScript.
- Arduino CLI - For Compiling Arduino Source Code
- Canvg(v3.0.6) - JavaScript SVG parser and renderer on Canvas.
- Django (v1.11) - used for developing the back-end of the platform.
- Celery - open source asynchronous task queue or job queue
- MySQL/ postgresql(default v9.6) - the database used for storing the data.

Chapter 2

Front-end

2.1 Front UI

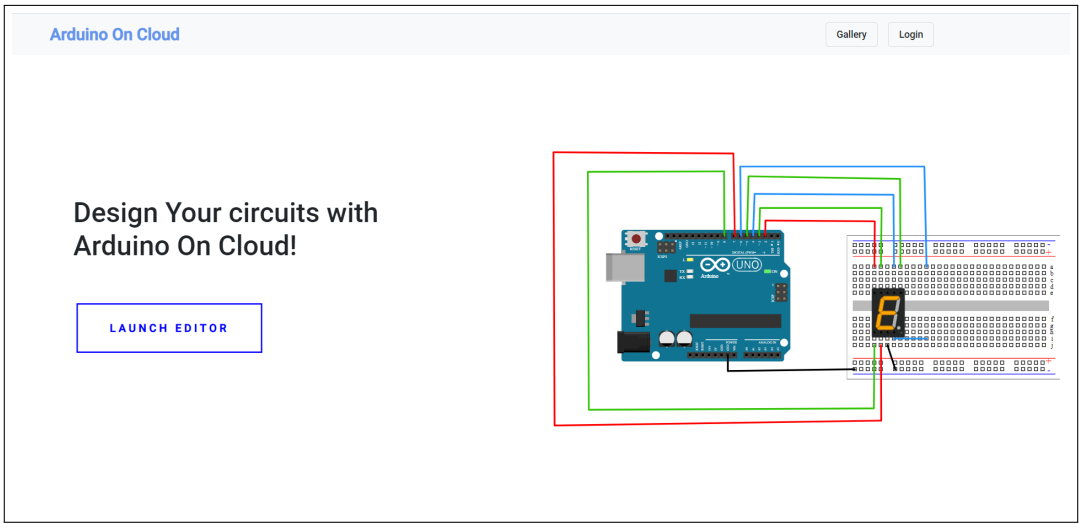

Figure 2.1: Front Page

It is the landing page for a particular user using our Web Application. User can either direct to **Simulation Editor** through it or can view the Gallery. **Gallery** contains the public projects which are contributed by users.

2.2 Gallery UI

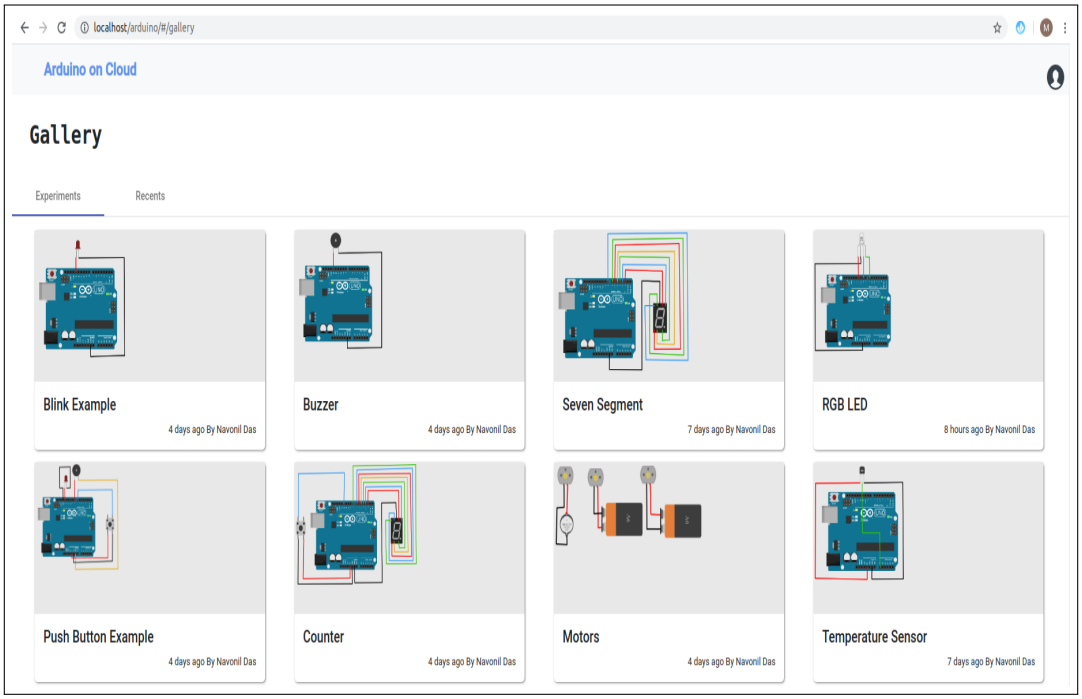

Figure 2.2: Gallery

The Gallery contains public circuits which are contributed by users.This is an addon material helpful for novice users to know what others have created.

2.3 Dashboard UI

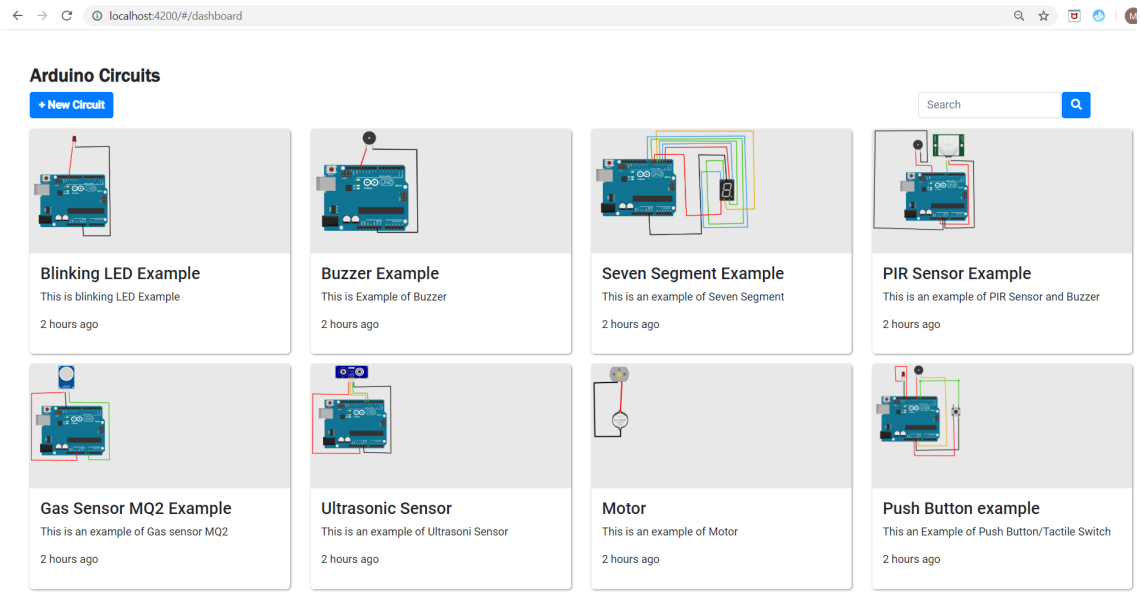

Figure 2.3: Dashboard

The Dashboard UI is used for displaying user-created projects. After clicking to a particular project the project properties can be viewed.It allows the users the view and search theirs circuits whether he has saved On Cloud i.e building and saving the circuit after login into system or Temporary(building and saving the circuit without login).

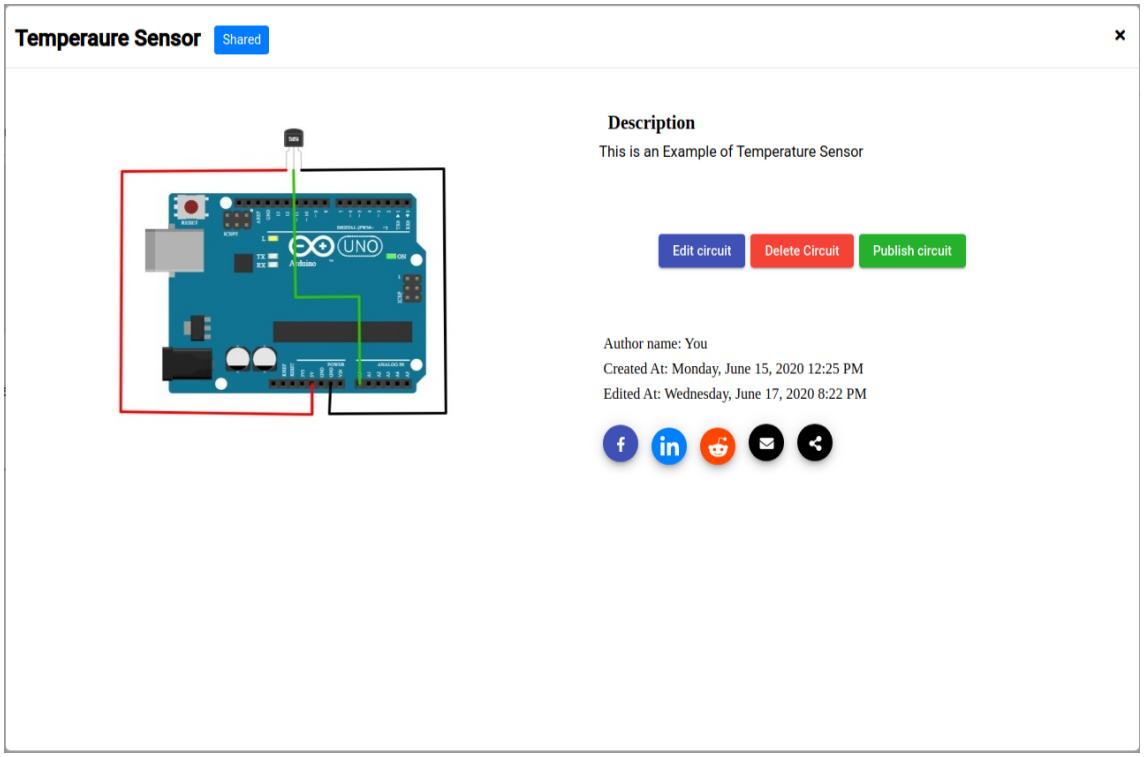

Figure 2.4: Project Properties

The User will able to edit and delete the circuit, reap description for that circuit, and other details like when he created, edited it.User can share the circuit through social media options, by mail and through Sharing URL option.

2.4 Simualtion UI

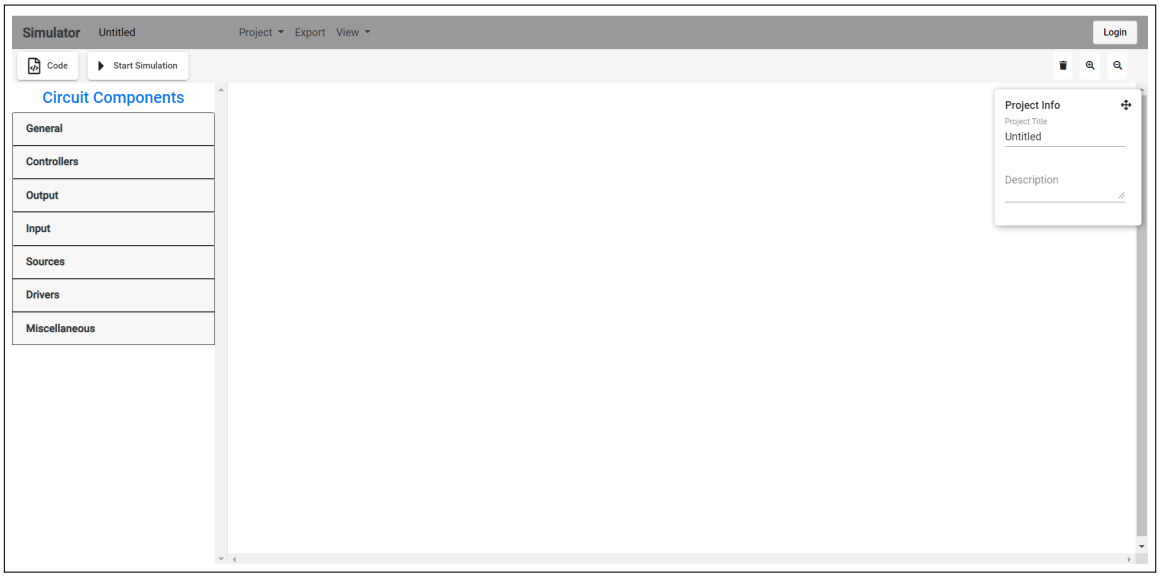

Figure 2.5: Simulator UI

After clicking launch editor we are directed to Simulator UI.It ia an editor where user can build and edit his circuit, put down his code and simulate it.

Following are the sections of this page :

- Components Section.
- Code Editor.
- Workspace.

2.4.1 Component Section

It is categorized into six divisions:

General section contains mostly/generally used components like resistors, breadboard etc.
Controller section contains all programmable devices.
Output section consists of output components that will generate output like Buzzer, led, RGBled and many more..
Input section consists of switches and sensors.
Sources part consist of components which will supply energy source to our circuit.
Miscellaneous part consist of labelling of wires.
All the Components used in the building circuit are created in **Raphael** [1] and **Inkscape** [2].

2.4.2 Code Editor

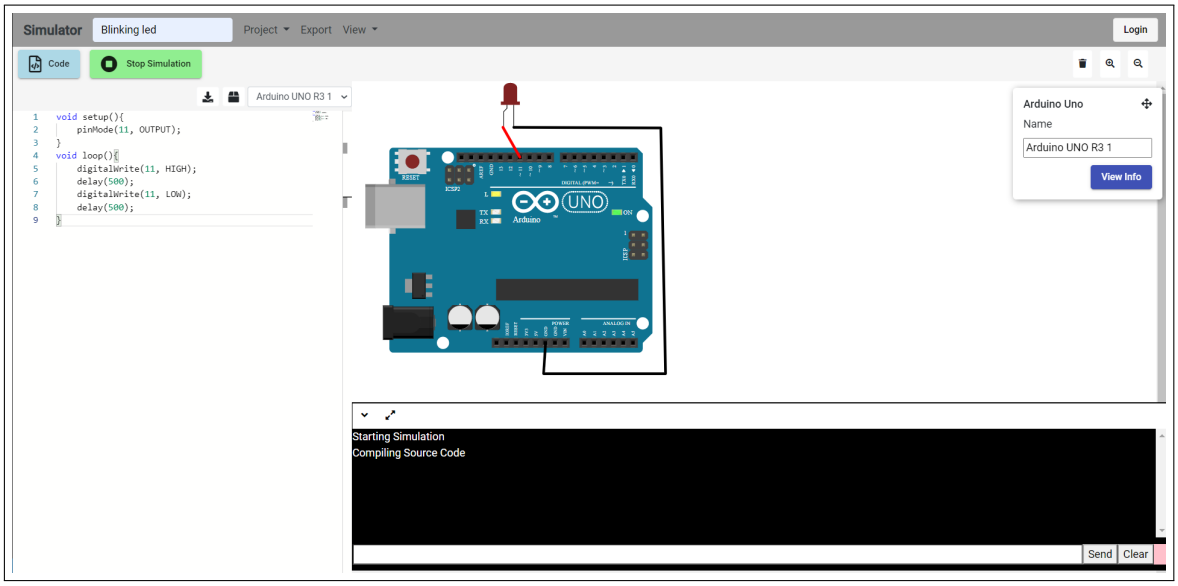

Figure 2.6: Code Editor

After building circuit the user can write code by clicking to **Code** button which opens to Code Editor. The code Editor is a **Monaco code-editor**[3] for composing code(logic) for programmable devices.

It allows user to jot down functions from suggestions and the suggestion will auto complete syntax for it to.User can also download the code(ino file) and compile it in Arduino IDE.There is also a facility to include the supported header files like EEPROM, LiquidCrystal, Servo, SoftwareSerial, Wire, and SPI.

Code editor opens only if Programmable Components(Arduino or any other controller) is present in the workspace else shows a message as "No Programmable Component present in the Circuit".

2.4.3 Workspace

Workspace is a area where user can build his circuit by dragging and dropping the components and connect components using virtual wires which will stick to the pins of the component legs,parts of Arduino etc.Components can be copied, paste, and deleted by right click on context menu or can be done through keyboard options.Circuit can also be zoom in and zoom out. Labels to wires can be given while dealing with complex circuits.

2.4.4 Other Features

Export

The Export feature will allow user to export his circuit in different file format(mainly SVG, PNG, JPEG).

View Component List

View option shows the Component List (the components present on Workspace) which will specify the name, quantity, and info of the components.The Component List can also be downloaded.

View Info Box

A View Info Box appears after the user drag-n-drop the component on the workspace, where he can view the specification, description, and datasheets of a particular component.

2.5 View Project

The user can share his circuit with his friends or colleagues by various sharing options present in project properties. He can share his circuit through social media like Facebook, LinkedIn, and Reddit and also through Mail.

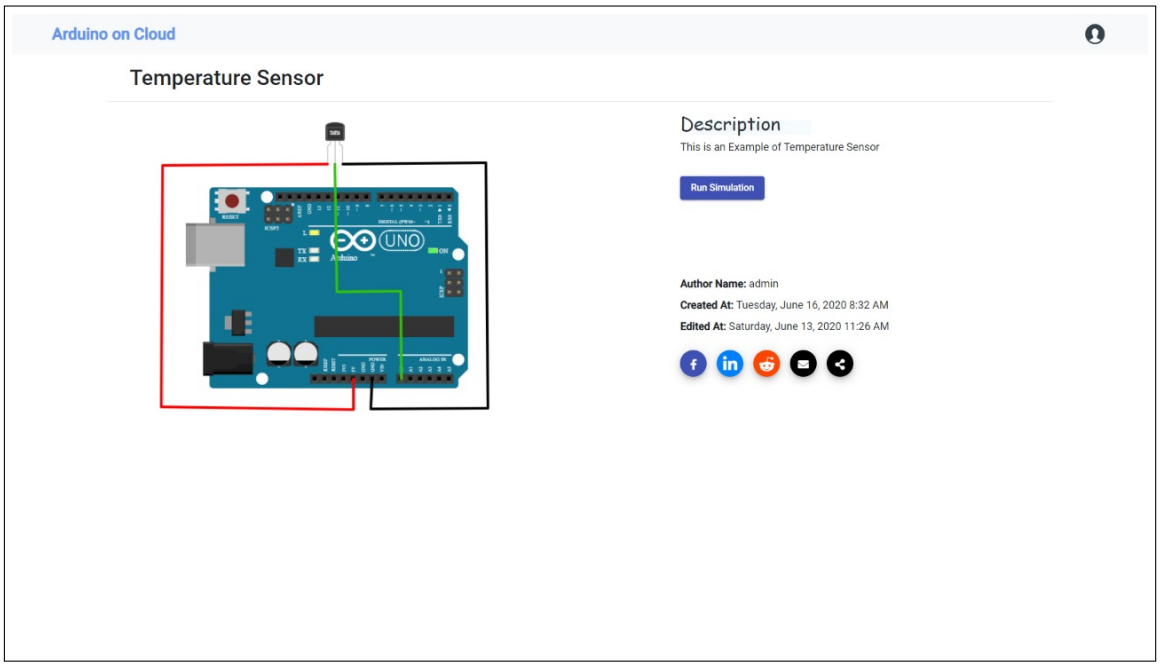

Figure 2.7: View Project

Clicking on sharing option through URL and launching the URL will redirect the user to **View Project UI** where user can reap description, get the details when we created it, view and simulate his circuit but cannot edit.

Chapter 3

Backend

The Backend is written on the Django and uses Django rest framework for API. We use API for compilation and saving the circuit on the cloud.

3.1 Compilation

For Compilation, we use **Arduino Cli**[4]. The Arduino CLI takes sketch files as input and returns compiled hex file. For Alpine Linux which we were using in docker, the Arduino CLI was not compiling so we have to switch our compiler to avr gcc.

3.2 Celery

We are using Celery [5] for Task Queue, As we need to compile the source code, for large source code the compilation could take up resource we don't want the system to stop for compilation so we use the task queue. Each compilation process is a task. The initial state for a task is **PROGRESS**. if our task is done or failed the status is updated accordingly. If the task is completed we send the response back to the user. if there is some compilation error we send those to the user specifying the error inside the code.

3.3 Database

We are using SQL[6] and MongoDB[7] as our databases. We Store Circuits inside Mongo and for publishing the circuit we are using SQL.

3.4 API Endpoints

3.4.1 POST /arduino/compile

This API is used to append the compilation task into the task queue. It Returns a compilation id which can be used to get compilations status.

Input

The number inside the json is the id of arduino.

```
{
   "1":"void setup(){delay(100);}void loop(){",
   "2":"#include <IRremote.h>\nvoid setup(){}void loop(){}",
   "3":"void setup(){}void loop(){}",
   "6":"void setup(){pinMode(13, OUTPUT);}void loop(){
      digitalWrite(13, HIGH); delay(1000);digitalWrite(13, LOW);
      delay(1000);\n}",
   "14":"#include <Servo.h>\nvoid setup(){}void loop(){}",
   "17":"void setup(){}void loop(){}"
}
```

Output

It returns the current state and the uuid which is the compilation id for codes.

```
{
    "state": "PENDING",
    "uuid": "18890ff8-82a6-4c92-94a7-611555a2b68f"
}
```

3.4.2 GET /arduino/compile/status

(arduino/compile/status?task_id=18890ff8-82a6-4c92-94a7-611555a2b68f) This API returns the compilation status of the respective compilation id.

Output

```
{
    "state": "SUCCESS",
    "details": {
        "1": {
            "output": "Compilation success output.",
            "error": "If Compilation failed this contains the error.",
            "data": "This will contain the hex."
        },

        ...
    }
}
```

3.4.3 POST /save

Input

```
{
    "data_dump":"JSON Data for Circuit",
    "is_arduino":true,
    "description":"Project Description",
    "name":"Project Name",
    "base64_image":"Project Thumbnail in base 64"
}
```

Output

returns the save id for the respective circuit.

3.4.4 GET /save/arduino/list

This API returns a list of Arduino circuits created by a particular user.

3.4.5 GET /save/search

This API returns a list of arduino circuits created by a particular user which contains some search parameters. The Search parameters are query parameters and case insensitive.

Query Parameters

name__icontains: Checks that the string is inside the Project Name
description__icontains: Check that the string is inside the Project Description
save_time__icontains: Check if the time is inside the save time.
create_time__icontains: Check if the time is inside the create time.
is_arduino: It is Boolean true if we want to search Arduino Circuits false if we want to search EDA circuits.

Example

```
save/search?name__icontains=seven&is_arduino=true
```

3.4.6 GET /save/{SaveID}

This API is used to read a respective circuit using the save id.

Input

SaveID It is a UUID which is received after saving.

Output


```
{
   "save_time":"2020-06-16T04:01:32.707350Z",
   "save_id":"b67207bf-db37-414b-8b96-748d48a75400",
   "data_dump":"",
   "owner":"User name",
   "shared":false,
   "base64_image":"Saved Thumbnail Url",
   "create_time":"2020-06-13T05:56:05.481143Z",
   "is_arduino":true
}
```


3.4.7 POST /save/{SaveID}

This API is used to update the existing Saved Circuit. The input is same as the /save api but it doesn't returns any saved it, It returns a Boolean value that tells true if update wass success,

3.4.8 DELETE /save/{SaveID}

This API is used to Delete an existing circuit.

3.4.9 POST /save/{SaveID}/sharing/on

This API is used to Enable the sharing for a respective circuit.

3.4.10 POST /save/{SaveID}/sharing/on

This API is used to Disable the sharing for a respective circuit.

Chapter 4

Working

4.1 Simulation

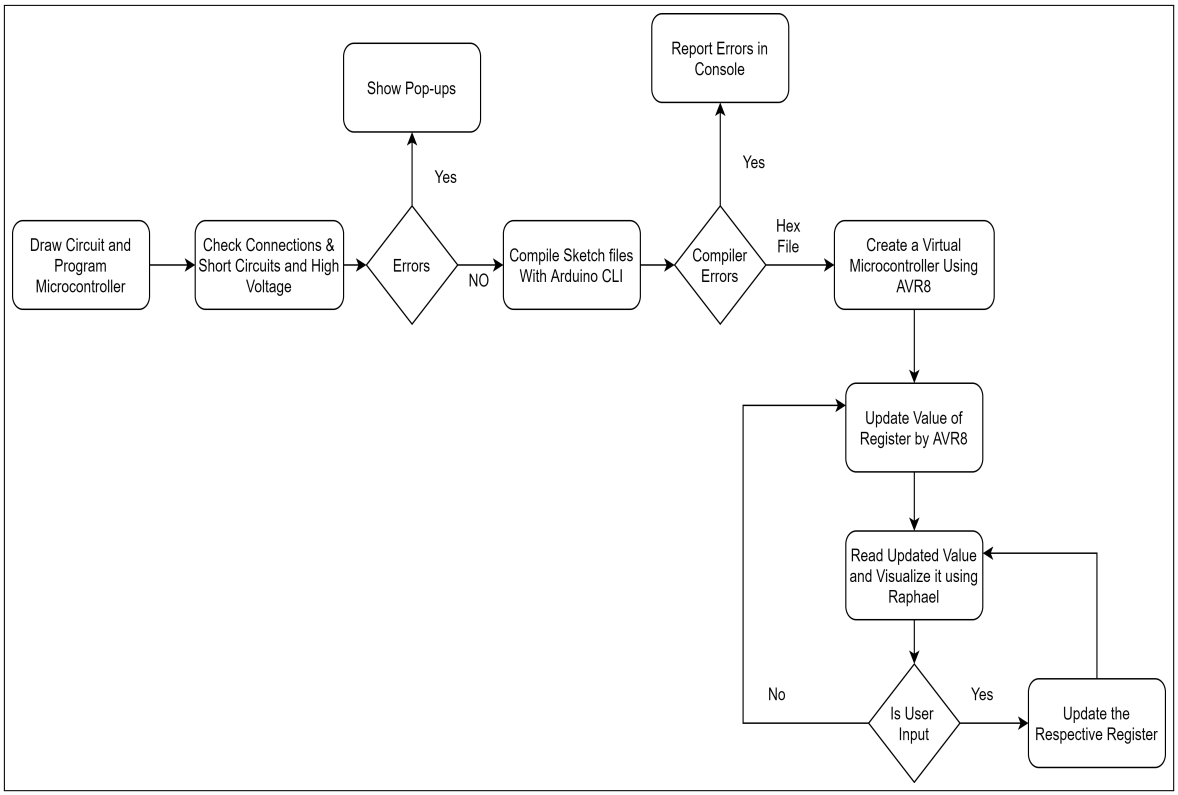

Figure 4.1: Simulation Flow

The User First needs to draw his/her respective circuit. If the circuit contains a programmable device then the user needs to provide the respective code. The basic template of the code is already provided the user just needs to modify the code or can write his own. As the user presses the Start Simulation button. checking is done and the code is sent to the server for compilation. The Server queues the compilation task using celery. After the compilation is done the user gets back the compiled hex file. The hex is parsed by avr8js[8], for each microcontroller we need

to specify some configuration to avr8js. The avr8js changes the value of the register depending on the instruction and calls a hook (function). Inside the hook, we read the value of the register depending on which we visualize it. The visualization is done by Raphael js[1]. Now to simulate a component we have to write the simulation logic. The Simulation Logic is in frontend in their respective classes the reason being we need to simulate everything with precision. As the user presses stop simulation we stop the simulation loop and clear all variable or animation applied.

4.2 Saving

As the user presses the save button we take the SVG of the circuit and try to generate a thumbnail which is a png file. In simple terms, we need to render the SVG to PNG. This can be done using the python library however the library does not support nested image tag which mostly used in our circuit. To solve this we used a different approach we render SVG to HTML5 Canvas using the library **Canvg**[9]. The rendered canvas is then transformed into a base64 string. The base64 string is sent to the server saves it into an image file and save its path on the database.

The circuit is saved as a JSON inside the database, fields like name, description, creation date, modification date are stored in a separate column. To redraw circuit we parse the saved JSON and create components with respect to it.

Chapter 5

Creating an Custom Component

As our frontend needs to work as close as realtime our component logic and simulation logic are written in frontend. Each component in the workspace is a child of a Circuit Element Class

5.1 Steps

- The first thing we need to understand how our system renders component. If our component is static during the whole simulation we use an image. If some animation is to be applied to the component we use shapes. To be noted inside the component if only one portion is to be simulated you should shape to make the system more efficient.
- We need to create a data JSON file which will store the information regarding one component.

```
{
  "name": <Component Name>,
  "className": <Class Name For the Component>,
  "pointHalf": <Half Size of the Circuit Node(Red Rectangle While
     connecting)>,
  "pins": [
    {
      "x": <X offset>,
      "y": <Y Offset>,
      "name": <Label For the Circuit Node>
    },
    .......
  ],
  "draw": [
    {
      "type": "path",
      "value": "M5,35L5,70",
      "stroke": "#000000"
```

```
    },
    {
      "type": "circle",
      "radius": 30,
      "x": 0,
      "y": 5
    }
   {
      "type": "rectangle",
      "width": 60,
      "height": 60,
      "x": 0,
      "y": 0,
      "radius": 10,
      "stroke": "#969696",
      "fill": "#bfbfbf"
    },
    {
      "type": "circle",
      "radius": 5,
      "x": 10,
      "y": 10,
      "fill": "#383838",
      "stroke": "#383838"
    },
   {
      "type": "image",
      "url": "assets/images/smokingEffect.svg",
      "width": 327,
      "height": 290,
      "x": -163.5,
      "y": -300
    }
  ],
  "data": {
   <Key Value Pairs of data like colors of LED etc.>
  },
  "info": {
    "name": <Complete Name>,
    "image": <URL of the Image>,
    "description": <Description of Component>,
    "properties": {
      <Key Value Pairs to be represented in a Tabular Form>
    }
  }
}
```

As we can see the draw array contains shapes and images to be drawn. The last element of the array will be at the top while drawing. The x,y values inside the drawing object are the X offset and Y Offset i.e if our component is to be placed at (0,0) how far it is from the origin. We get these values from Inkscape or Illustrator.

The info object will contain information which needs to be displayed inside the View Info Dialog. Remember to put JSON inside **assets/json** Folder.

- Now we have the data file we have to create a class file. You can place the class file anywhere inside the angular project however we recommend to use the **Libs** Folder. Class Needs to inherit the Circuit Element Class. After inheriting we may get a few errors as we have not implemented the abstract functions and called the constructor properly.

```
export class MyClass extends CircuitElement {
    /** Declare Variable **/
    constructor(public canvas: any, public x: number, y: number) {
        super('MyClass', x, y, 'data_filename.json', canvas);
    }

    /* This function is called after rendering */
    init() {
    }

    /**
   * Function provides component details
   * @param keyName Unique Class name
   * @param id Component id
   * @param body body of property box
   * @param title Component title
   */
    properties(): { keyName: string; id: number; body: HTMLElement;
        title: string; } {
        const body = document.createElement('div');
        return {
            keyName: this.keyName,
            id: this.id,
            title: 'Propeties box title',
            body
        };
    }

    initSimulation(): void {
        // Called When Start Simulation is pressed
      // Write Simulation Logic Here
    }

    closeSimulation(): void {
```

```
        // Called When Stop simulation is pressed
    }
    simulate(): void {
        // No Use For Now
    }

    SaveData() {
        return {
            // key Value data that needs to be stored in database for
                recreation
        };
    }
    LoadData(data: any) {
        // This is called after retriving of data
        // data.data will contain the actual data
    }
}
```

- Now our class file is completed we are almost done. Now Typescript has a layer of abstraction we know that in javascript we can create an object of a class by only using class Name as a string. We cannot do that in our system. This needs to be solved for dynamically object creation while dragging and drop. We solve this issue by creating a map for a string class name to the Class. This is done inside the **utils** file.

Inside **/src/app/Libs/Utils.ts** we have to create the mapping.

```
  static components = {
      MyClass: {
        name: 'Name to be displayed inside the compoent bar',
        image: '<thumbnail image path>',
        className: <Link to the Class>
      }
   };
```

Note: Don't delete existing components variable the object MyClass is just need to be implemented inside the components. To show the component inside the list the user has to modify the **"componentBox"** variable according to the category just append the class name.

The whole procedure is complex and requires good knowledge of AVR8js, Raphaeljs and our existing system. As we are doing modification to the Frontend we have to build the app for production mode.

5.2 Future Improvements

- Creating an API to load components List.
- As the structure of the data file is almost same we can have models and admin/contributor pages where one can write these data files easily with validation.
- The whole component is a part of Frontend which makes the building procedure complex. As the number of the component increases the Frontend could become heavy. One can use some method to lazy load Javascript files for the respective component.
- To write component one must be proficient in typescript. This can cause a problem for beginners who are new and wants to add a component. This can be solved by using some approach as **NGSPICE** does for creating a model.

Chapter 6

Examples

6.1 Blink Example

For a beginner, the Blink example is an introduction to Arduino. In this example, we change the state of the Digital pin 13 From to HIGH to LOW with a delay of 1 second in a repetition.

6.1.1 Code

```
void setup(){
   pinMode(LED_BUILTIN, OUTPUT);
}

void loop(){
   digitalWrite(LED_BUILTIN, HIGH);
    delay(1000);
   digitalWrite(LED_BUILTIN, LOW);
    delay(1000);
}
```

6.1.2 Connection

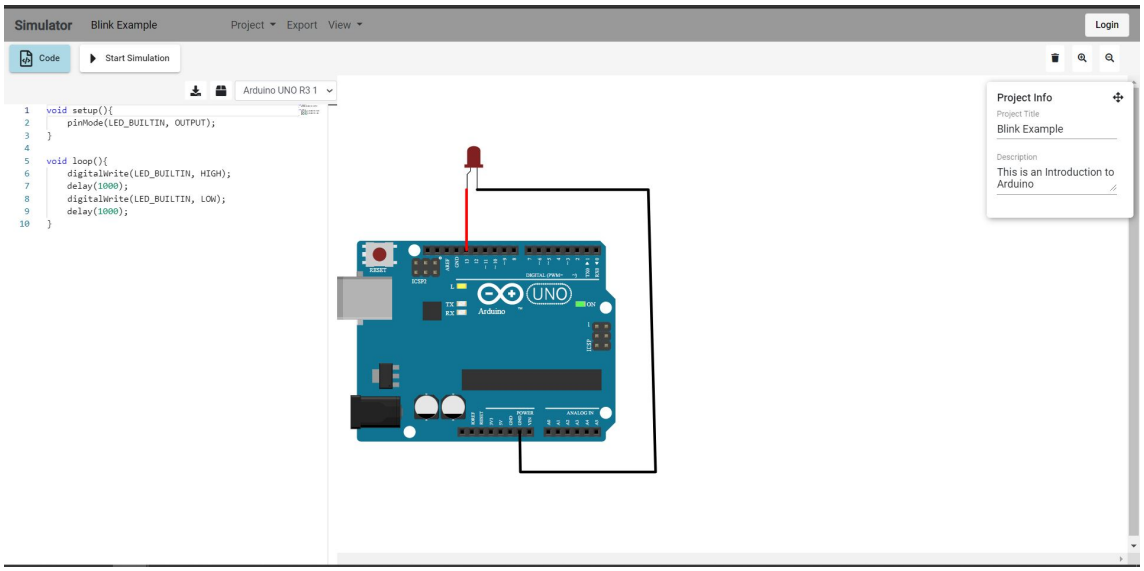

Figure 6.1: Blink Example Connection

6.1.3 Output

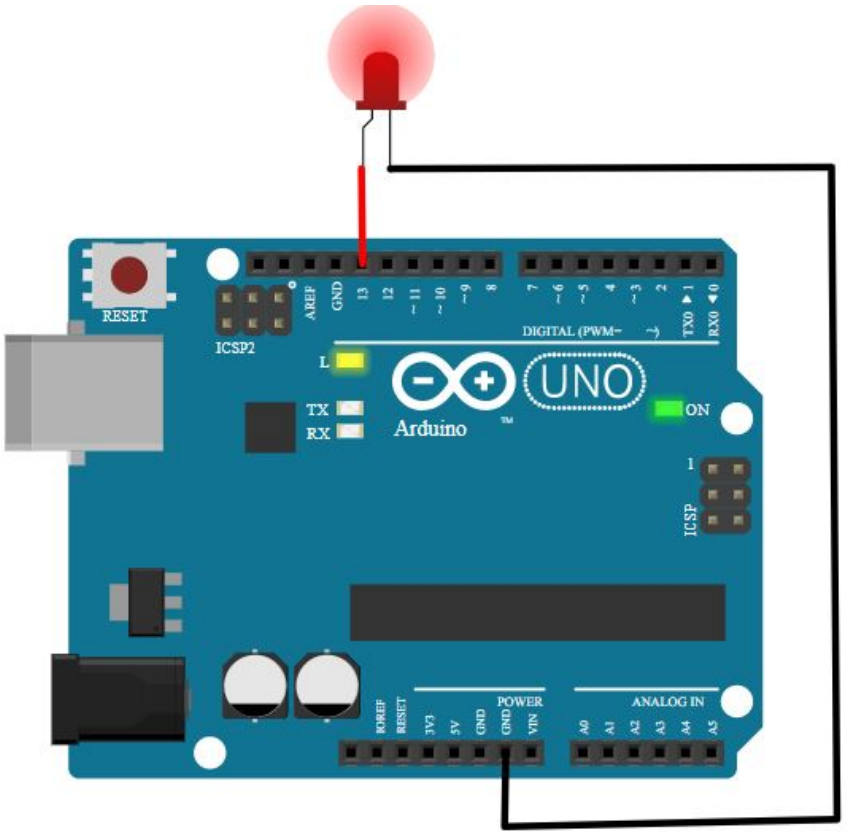

Figure 6.2: Blink Example Output

6.2 Seven Segment

In this example we will interface a seven segment display with the Arduino.The Pins a to g are connected to the Arduino Digital pins from 2 to 8 respectively.

6.2.1 Code

```
int outs[10][7] = {
   { 1,1,1,1,1,1,0 },  // 0
    { 0,1,1,0,0,0,0 },  // 1
    { 1,1,0,1,1,0,1 },  // 2
    { 1,1,1,1,0,0,1 },  // 3
    { 0,1,1,0,0,1,1 },  // 4
    { 1,0,1,1,0,1,1 },  // 5
    { 1,0,1,1,1,1,1 },  // 6
    { 1,1,1,0,0,0,0 },  // 7
    { 1,1,1,1,1,1,1 },  // 8
    { 1,1,1,0,0,1,1 } // 9
};

void setup(){
   pinMode(2,OUTPUT);
   pinMode(3,OUTPUT);
   pinMode(4,OUTPUT);
   pinMode(5,OUTPUT);
   pinMode(6,OUTPUT);
   pinMode(7,OUTPUT);
   pinMode(8,OUTPUT);
}
int dig = 0;
void loop(){
   int pin = 2;
   for(int i=0;i<7;++i){
      digitalWrite(pin,outs[dig][i]);
      ++pin;
   }
    delay(1000);
    ++dig;
    if(dig >= 10)
        dig -= 10;
}
```

6.2.2 Connections

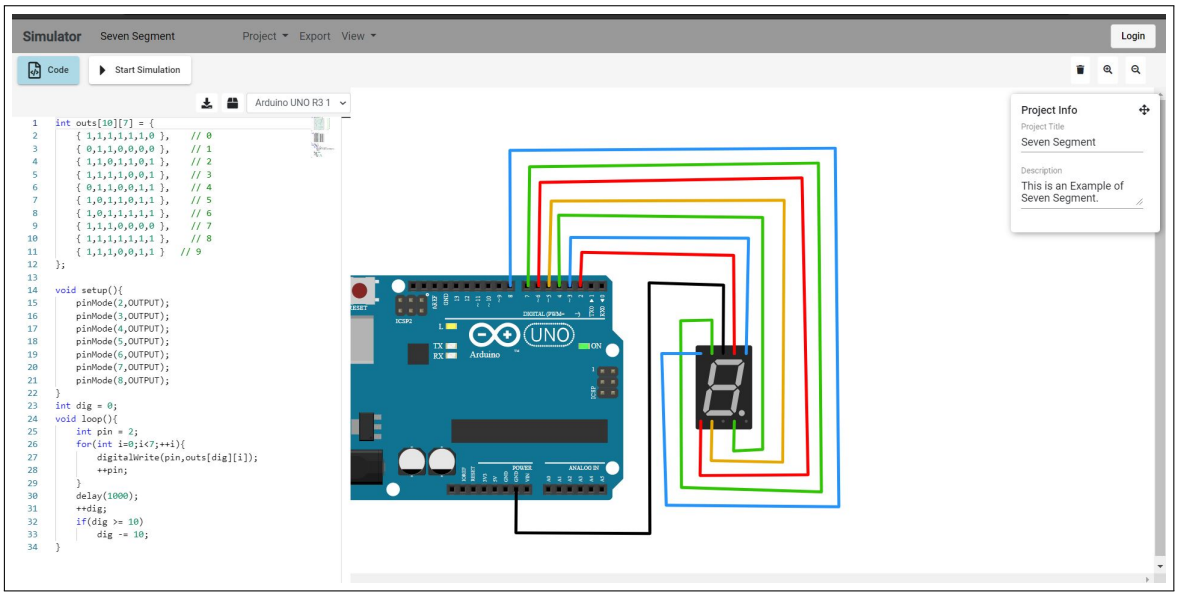

Figure 6.3: Seven Segment Connection

6.2.3 Output

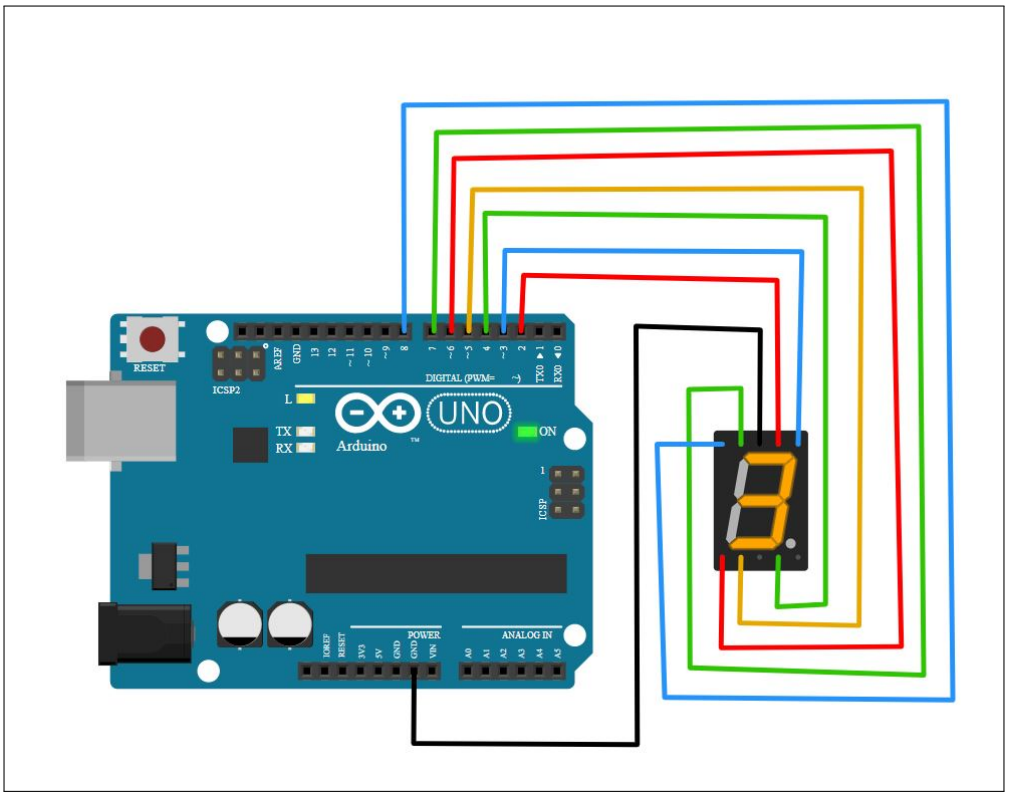

Figure 6.4: Seven Segment Output

6.3 Counter

In this example we will use a Push button with a Seven Segment Display.The User can Provide Input to the push button and the Seven Segment will show the count.

6.3.1 Code

```
int outs[10][7] = {
   { 1,1,1,1,1,1,0 },  // 0
    { 0,1,1,0,0,0,0 },  // 1
    { 1,1,0,1,1,0,1 },  // 2
    { 1,1,1,1,0,0,1 },  // 3
    { 0,1,1,0,0,1,1 },  // 4
    { 1,0,1,1,0,1,1 },  // 5
    { 1,0,1,1,1,1,1 },  // 6
    { 1,1,1,0,0,0,0 },  // 7
    { 1,1,1,1,1,1,1 },  // 8
    { 1,1,1,0,0,1,1 } // 9
};

void setup(){
   pinMode(2,OUTPUT);
   pinMode(3,OUTPUT);
   pinMode(4,OUTPUT);
   pinMode(5,OUTPUT);
   pinMode(6,OUTPUT);
   pinMode(7,OUTPUT);
   pinMode(8,OUTPUT);
   pinMode(12,INPUT);
}
int dig = 0;
void loop(){
   if(digitalRead(12) == HIGH){
      int pin = 2;
      for(int i=0;i<7;++i){
         digitalWrite(pin,outs[dig][i]);
         ++pin;
      }
      ++dig;
        delay(100);
   }
    if(dig >= 10)
       dig -= 10;
}
```

6.3.2 Connections

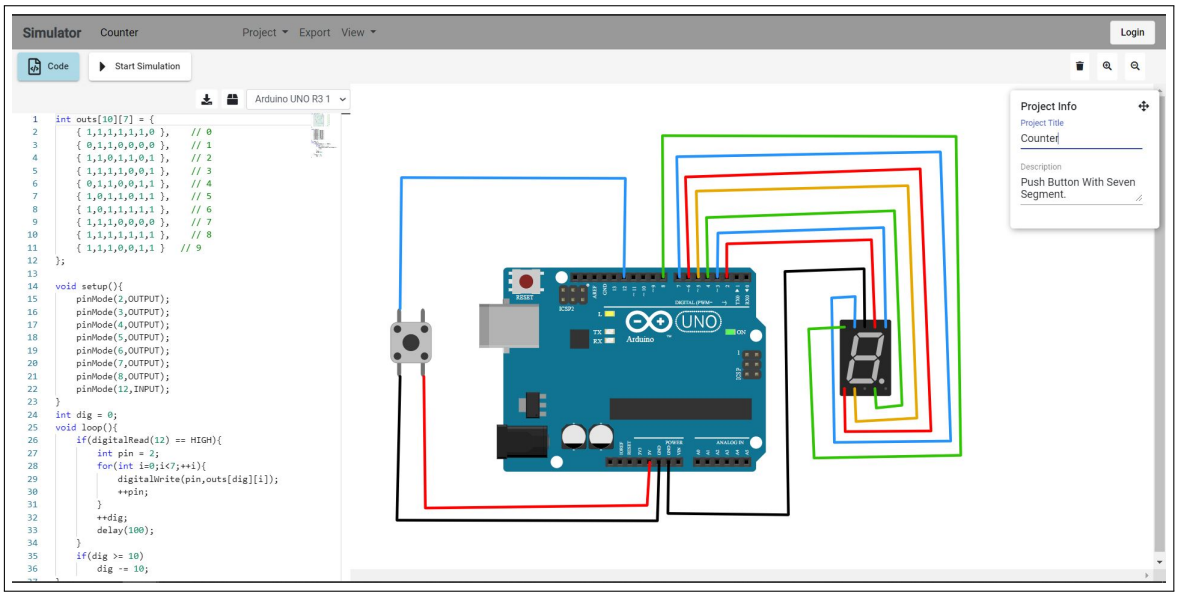

Figure 6.5: Counter Connection

6.3.3 Output

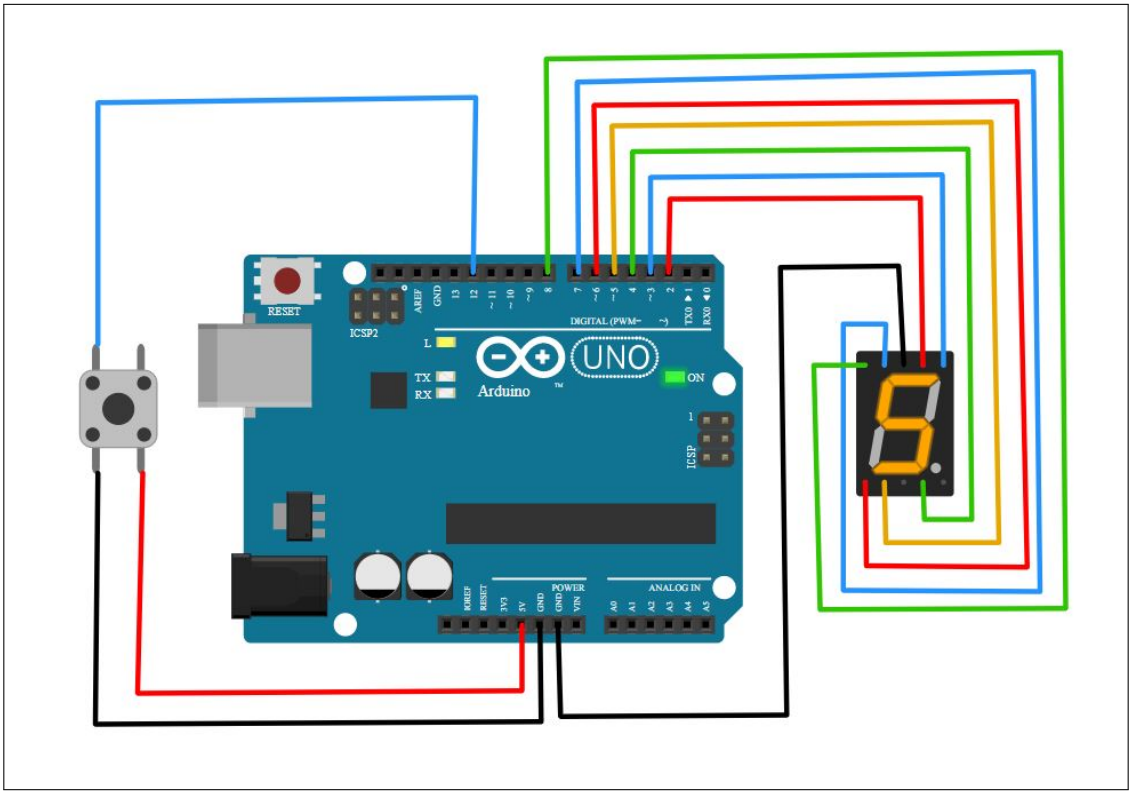

Figure 6.6: Counter Output

6.4 Buzzer With User Input

In this example we will read user input and beep the Buzzer according to it.

6.4.1 Code

```
void setup(){
   pinMode(LED_BUILTIN, OUTPUT);
   pinMode(8, OUTPUT);
   pinMode(7, INPUT);
}

void loop(){
    if(digitalRead(7) == HIGH){
      digitalWrite(LED_BUILTIN, HIGH);
      digitalWrite(8, HIGH);
    }else{
        digitalWrite(LED_BUILTIN, LOW);
        digitalWrite(8, LOW);
    }
}
```

6.4.2 Connections

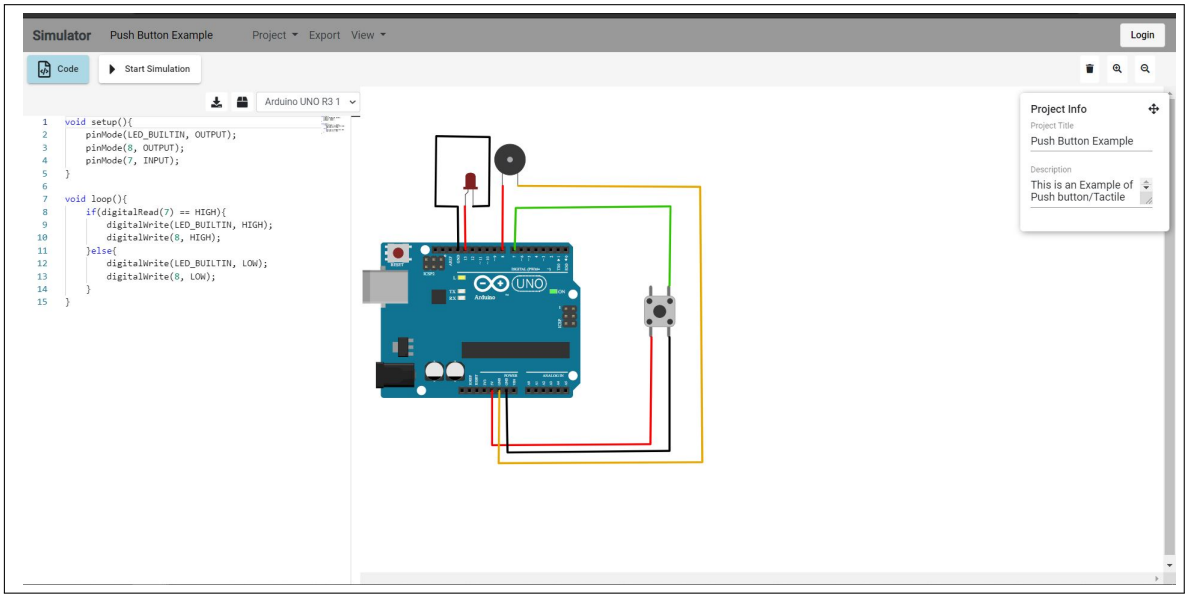

Figure 6.7: Buzzer Connections

6.4.3 Output

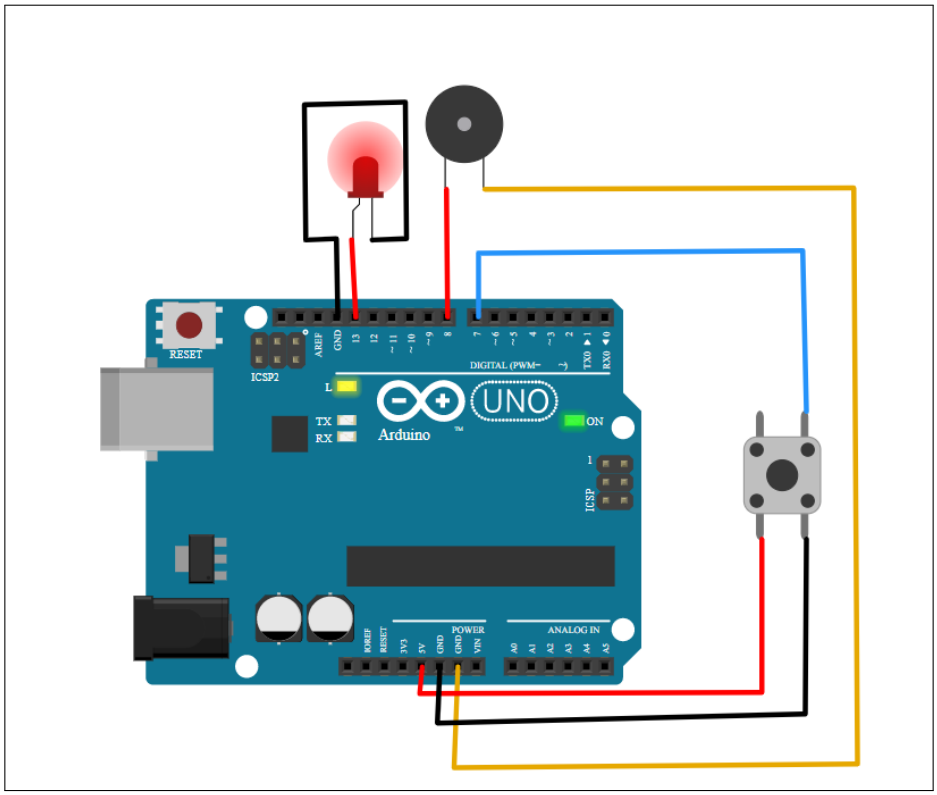

Figure 6.8: Buzzer Output

6.5 Motion Detection

In this example we will use PIR Sensor and Buzzer.

6.5.1 Code

```
void setup(){
    pinMode(12, OUTPUT);
    pinMode(13, OUTPUT);
    pinMode(7, INPUT);
}

void loop(){
   if(digitalRead(7) == HIGH){
        digitalWrite(13, HIGH);
        digitalWrite(12, HIGH);
    }else{
        digitalWrite(13, LOW);
        digitalWrite(12, LOW);
    }
}
```

6.5.2 Connections

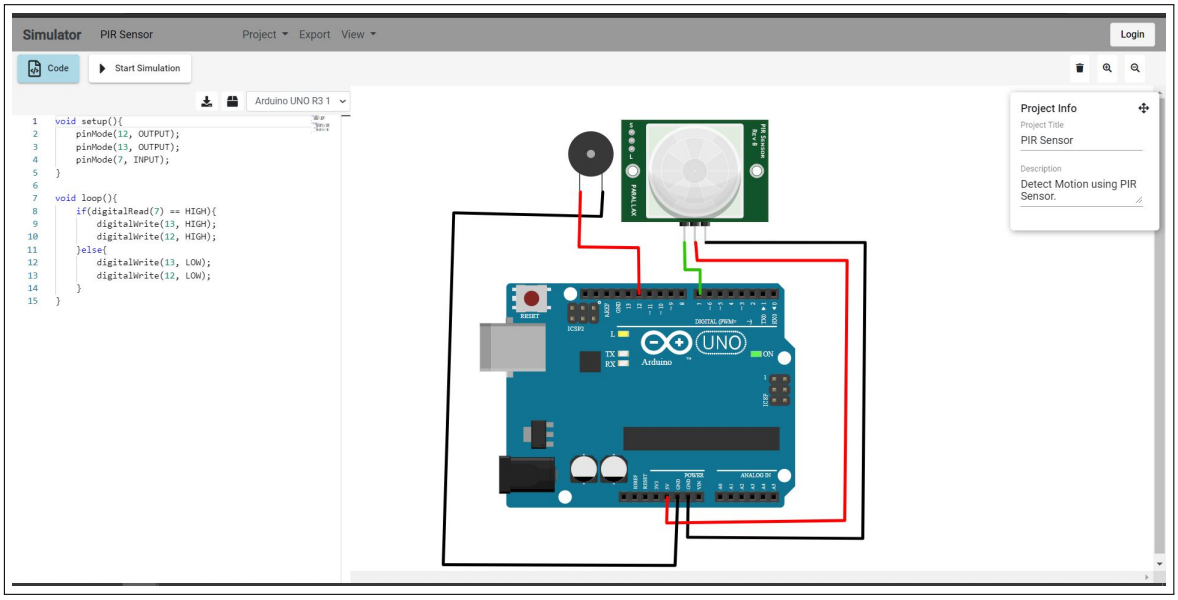

Figure 6.9: PIR Sensor Connections

6.5.3 Output

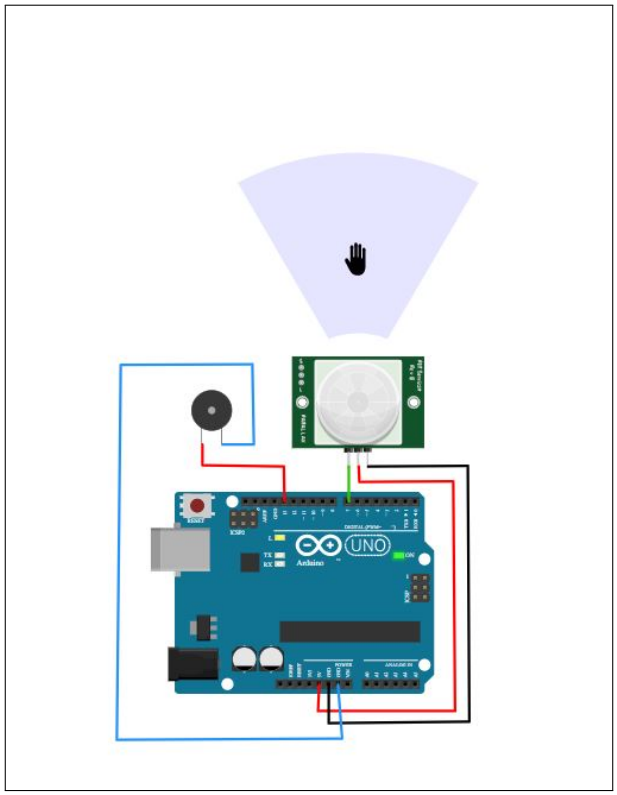

Figure 6.10: PIR Sensor Output

6.6 Temperature Sensor

In this example we will use temperature sensor.

6.6.1 Code

```
void setup()
{
  Serial.begin(9600);
}

void loop() {
 int reading = analogRead(A0);
 float voltage = reading * 5.0;
 voltage /= 1023.0;

 float temperatureC = (voltage - 0.5) * 100;
 Serial.print(temperatureC);
 Serial.println(" degrees C");
 delay(1000);
}
```

6.6.2 Connections

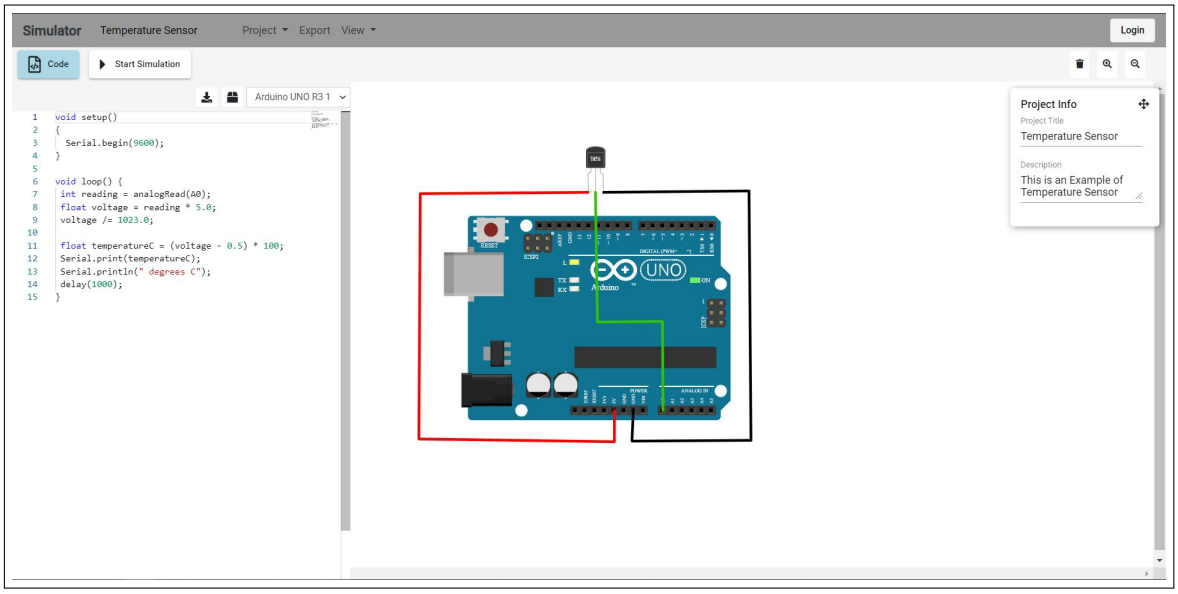

Figure 6.11: Temprature Sensor Connection

6.7 Output

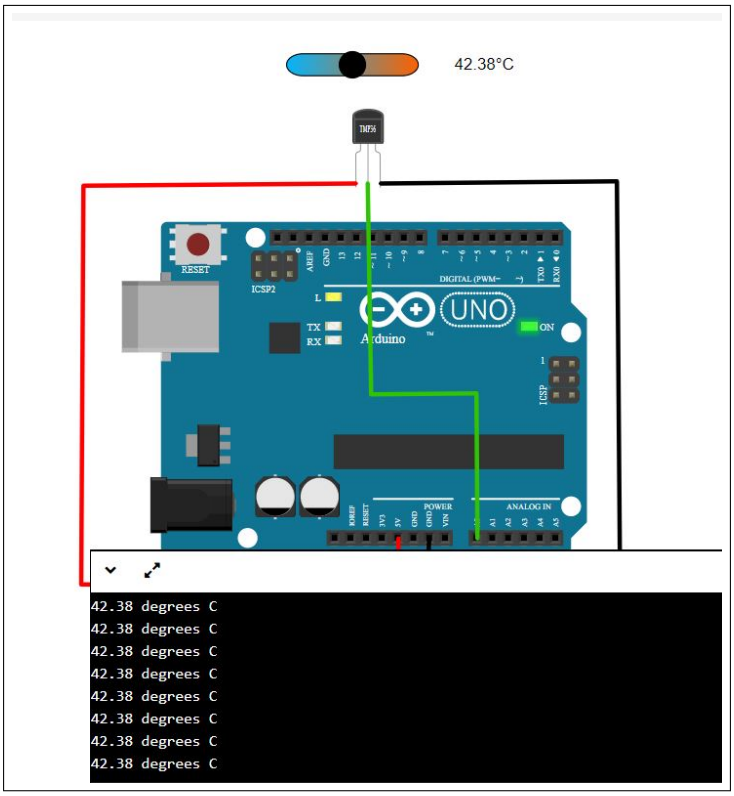

Figure 6.12: Temprature Sensor Output

6.8 Measuring Distance

In this example we will use ultrasonic sensor.

6.8.1 Code

```
const int pingPin = 7; // Trigger Pin of Ultrasonic Sensor
const int echoPin = 6; // Echo Pin of Ultrasonic Sensor

void setup() {
   pinMode(pingPin, OUTPUT);
   pinMode(echoPin, INPUT);
   Serial.begin(9600); // Starting Serial Terminal
}

void loop() {
   long duration, inches, cm;
   digitalWrite(pingPin, LOW);
   delayMicroseconds(2);
   digitalWrite(pingPin, HIGH);
   delayMicroseconds(10);
   digitalWrite(pingPin, LOW);
   duration = pulseIn(echoPin, HIGH);
   cm = duration / 29 / 2;
```

```
    Serial.print(cm);
    Serial.print("cm");
    Serial.println();
    delay(1000);
}
```

6.8.2 Connections

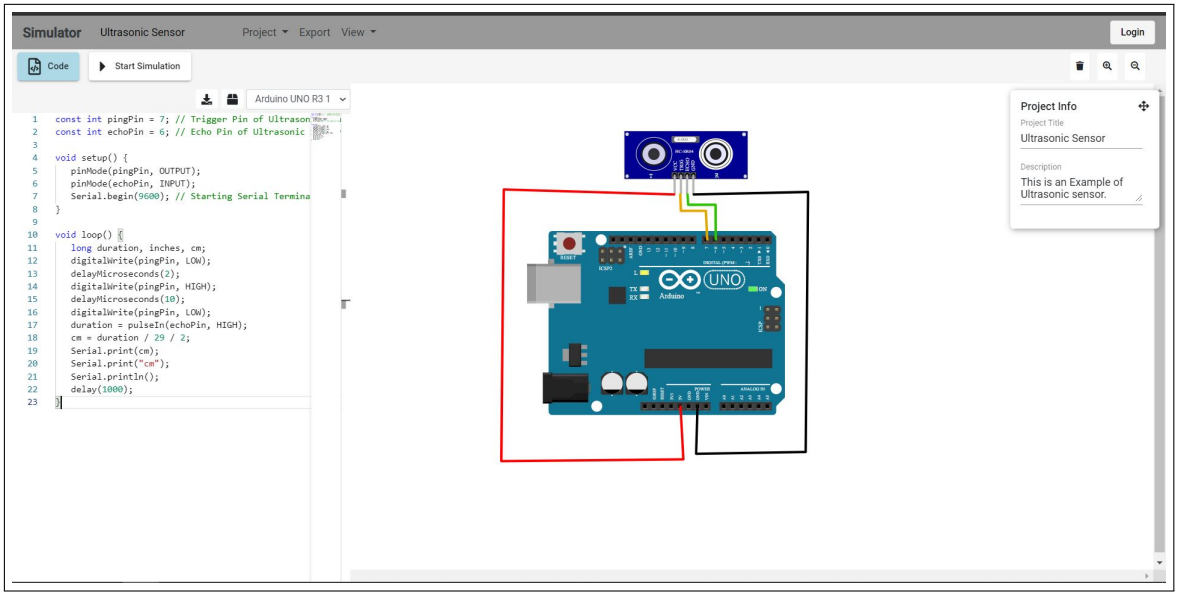

Figure 6.13: Ultrasonic Sensor Connection

6.8.3 Output

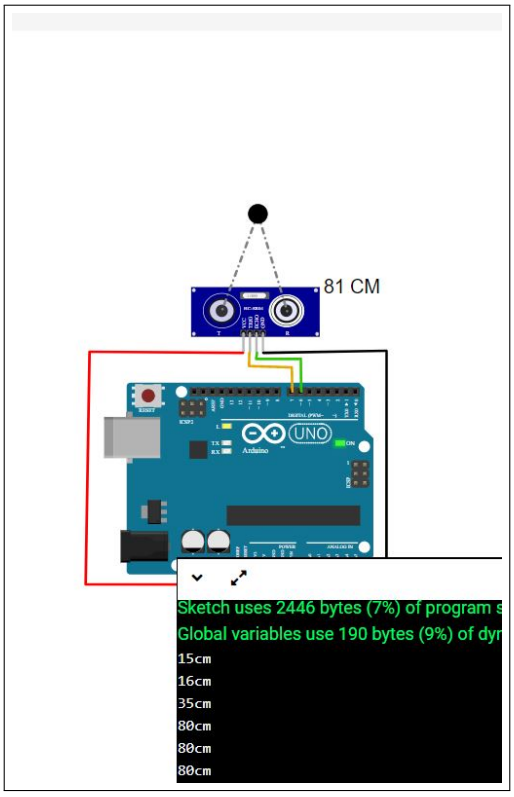

Figure 6.14: Ultrasonic Sensor Output

6.9 Detecting Smoke

In this example we will use the Gas Sensor(MQ2).

6.9.1 Code

```
void setup(){
   Serial.begin(9600);
}

void loop(){
   Serial.println(analogRead(A0));
    delay(1000);
}
```

6.9.2 Connection

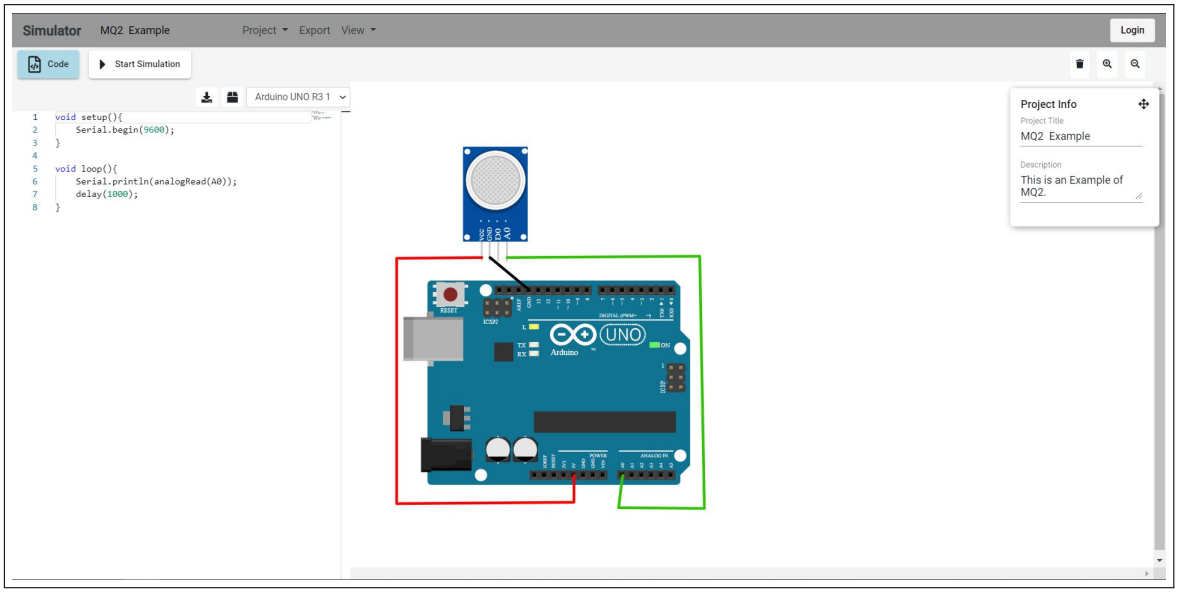

Figure 6.15: Gas Sensor Connection

6.9.3 Output

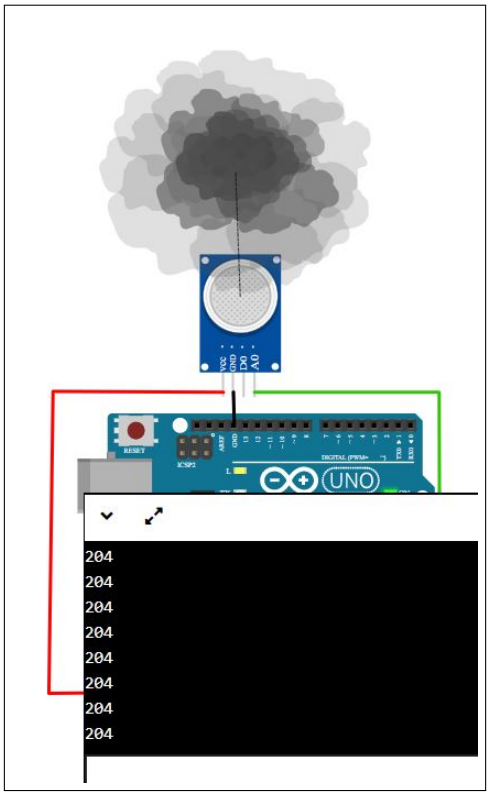

Figure 6.16: Gas Sensor Output

6.10 Relay Example

In this example we will use a relay module which can be used to switch on/off light bulb, fan or other appliance.

6.10.1 Code

```
void setup(){
   pinMode(LED_BUILTIN, OUTPUT);
}

void loop(){
   digitalWrite(LED_BUILTIN, HIGH);
    delay(1000);
   digitalWrite(LED_BUILTIN, LOW);
    delay(1000);
}
```

6.10.2 Connections

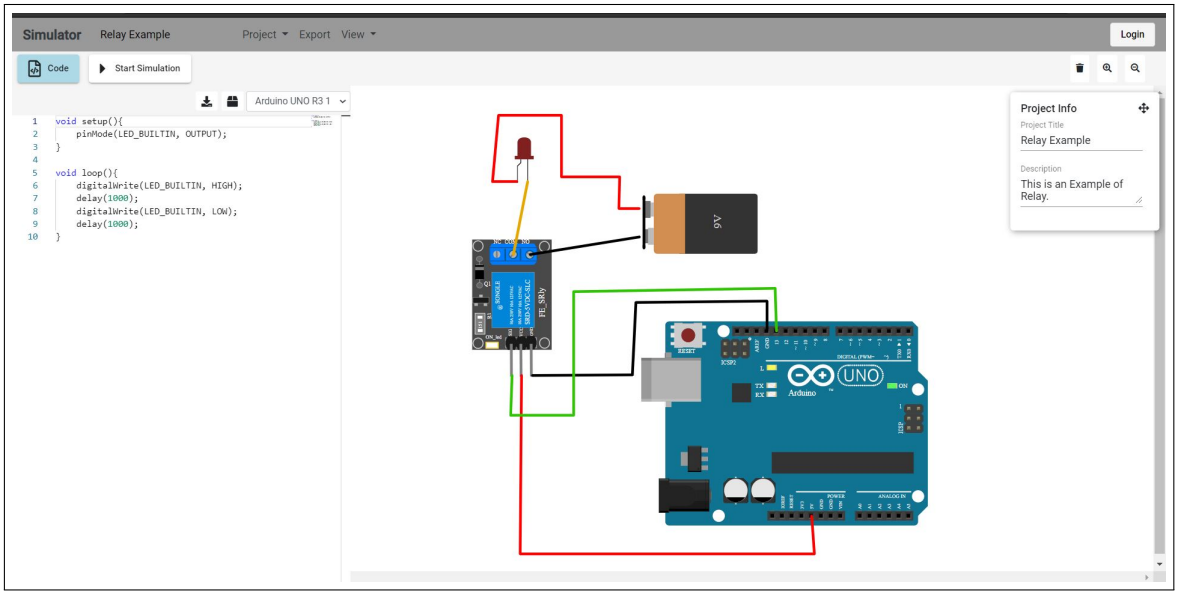

Figure 6.17: Relay Connection

6.10.3 Output

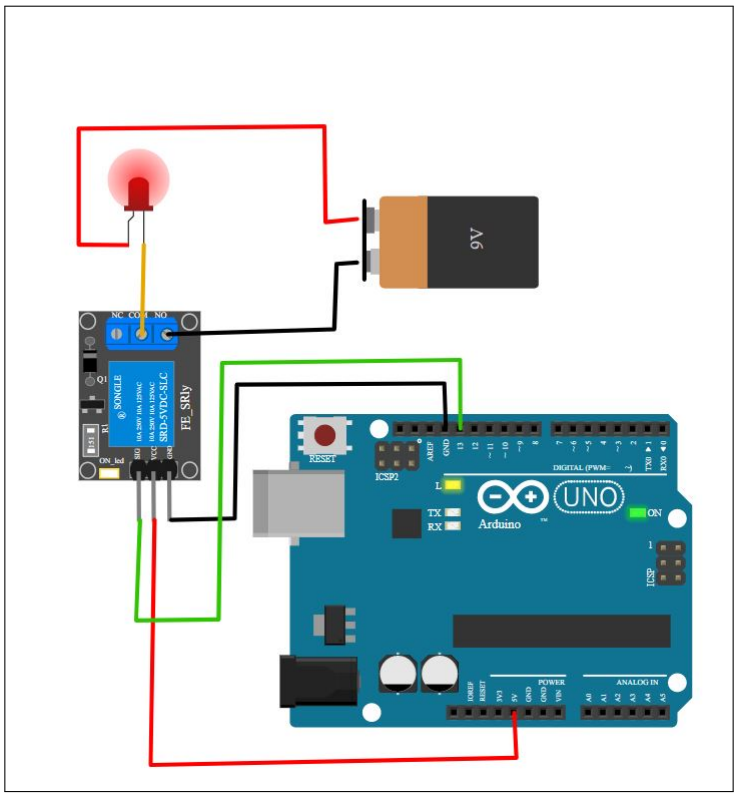

Figure 6.18: Relay Output

6.11 Motors

In this example we will simulate electric motors.

6.11.1 Connections

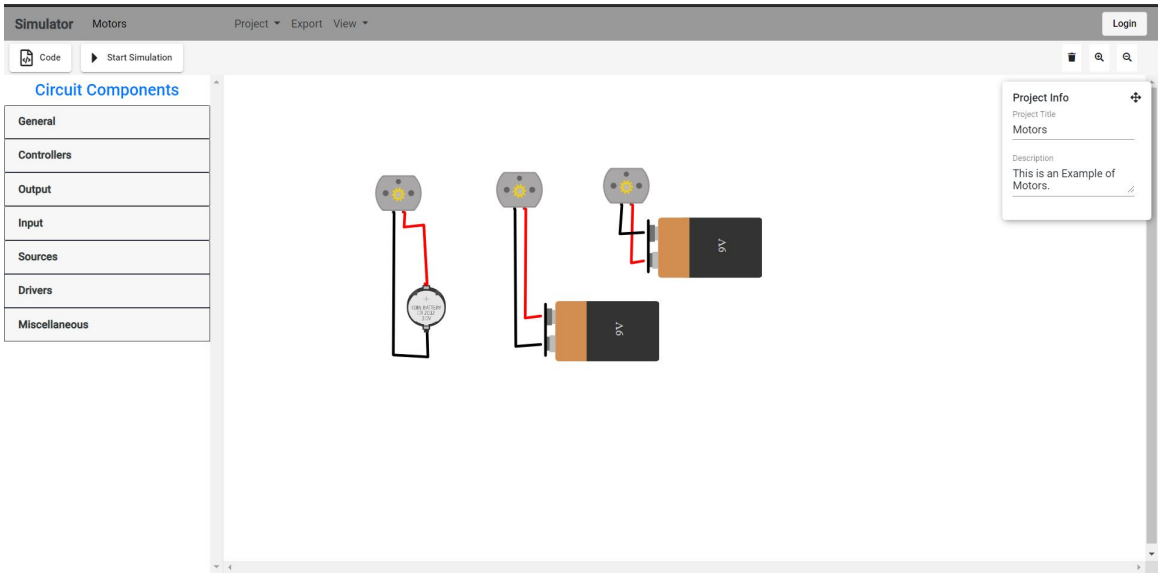

Figure 6.19: Motor Connection

6.11.2 Output

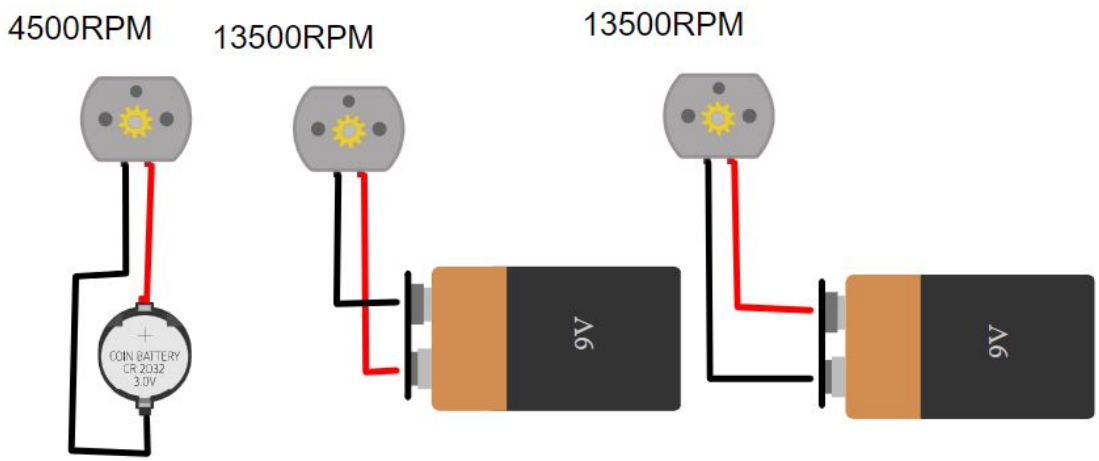

Figure 6.20: Motor Output

6.12 Servo Motor

In this example we will rotate Servo Motor from 0 to 180 degree back and forth.

6.12.1 Code

```
#include <Servo.h>
Servo myservo;
int pos = 0;

void setup() {
   myservo.attach(3);
}

void loop() {
    for (pos = 0; pos <= 180; pos += 1) {
      myservo.write(pos);
      delay(15);
    }
    for (pos = 180; pos >= 0; pos -= 1) {
        myservo.write(pos);
        delay(15);
    }
}
```

6.12.2 Connections

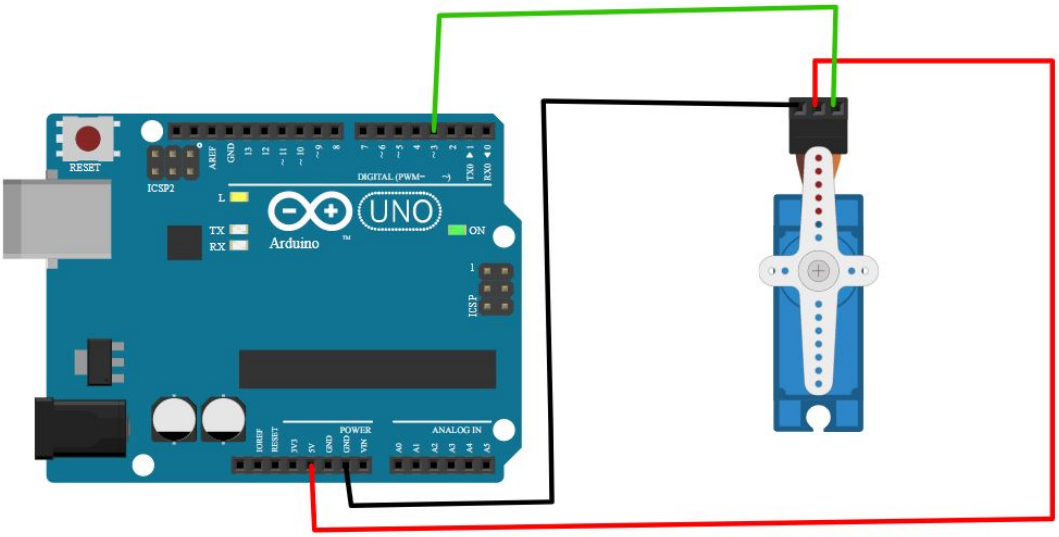

Figure 6.21: Servo Motor Connection

6.12.3 Output

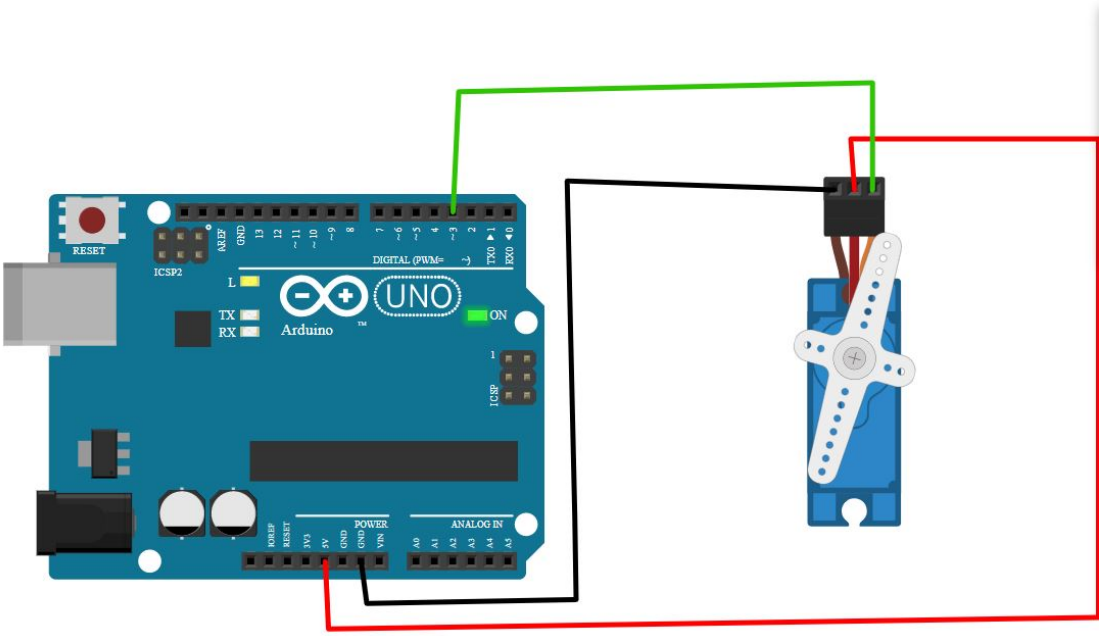

Figure 6.22: Servo Motor Output

6.13 RGB LED

Interfacing RGB LED with arduino.

6.13.1 Code

```
void setup(){
   pinMode(11, OUTPUT);
   pinMode(9, OUTPUT);
   pinMode(7, OUTPUT);
}

void loop(){
   for(int i=1;i<=7;++i){
        digitalWrite(11, i&1);
        digitalWrite(9, (i>>1)&1);
        digitalWrite(7, (i>>2)&1);
        delay(1000);
    }
}
```

6.13.2 Connections

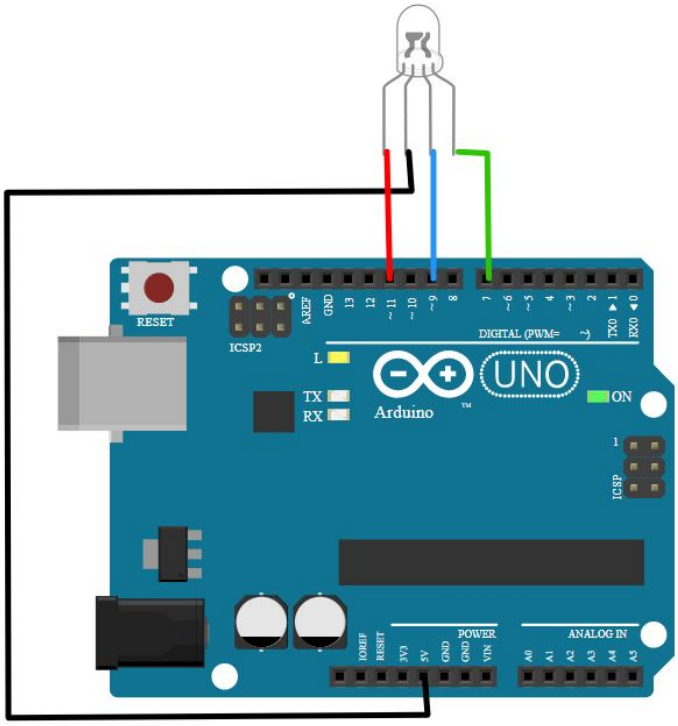

Figure 6.23: RGB LED Connection

6.13.3 Output

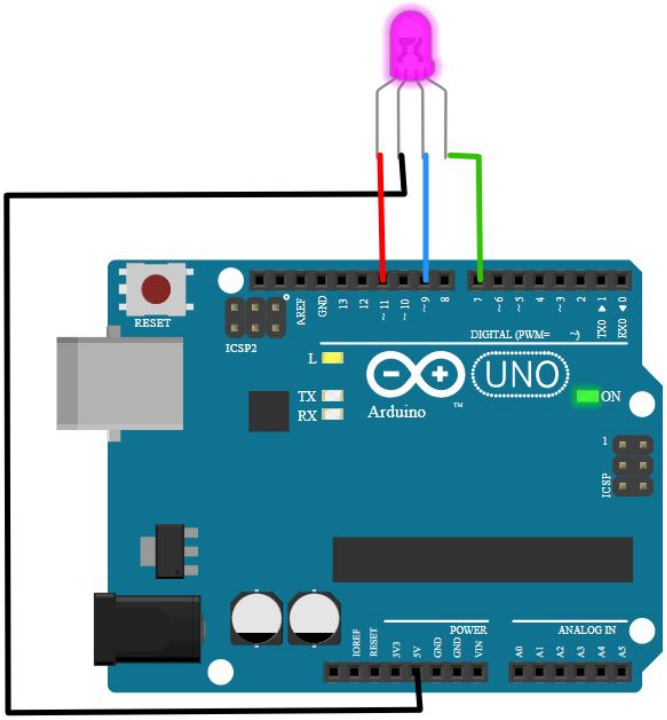

Figure 6.24: RGB LED Output

Chapter 7

Limitations And Future Scopes

7.1 Issues

Issues exisiting in our system is mentioned in Github Issues. You can also contribute to our project by solving few issues or creating issue into the following link
`https://github.com/frg-fossee/eSim-Cloud/issues?q=is%3Aopen+is%3Aissue+label%3AArduino+label%3Abug`

7.2 Limitations

- We cannot communicate between two microcontrollers.
- Simulating more than 10 microcontroller simultaneously can crash browser as it require more memory.
- For Simulating microcontroller we need an accuracy of microseconds however browser can only provide accuracy of millisecond.

7.3 Future Scope

- Adding Block Programming like Scratch.
- Embedded Circuit in different website.
- Adding Digital Circuit components like AND gate etc.
- Adding More Sensors and Drivers.
- Adding More programming language like LUA,Python etc.
- Adding More Microcontrollers like ESP8266,Arduino Nano etc.
- Desktop and Smartphone Compatibility.
- Multiple user working on same Circuit in Real Time. Like Google Docs
- Interactive Book Containing Examples and resource for Learning Arduino.

Bibliography

[1] https://dmitrybaranovskiy.github.io/raphael/.

[2] https://inkscape.org/.

[3] https://microsoft.github.io/monaco-editor/.

[4] https://github.com/arduino/arduino-cli.

[5] https://www.celeryproject.org/.

[6] https://www.mysql.com/.

[7] https://www.mongodb.com/.

[8] https://github.com/wokwi/avr8js.

[9] https://canvg.github.io/canvg/.