Semester-Long Internship Report

On

Playing songs on Arduino, Inline Assembly Language support & LTI and Auto Grading support

Submitted by

Deepam Priyadarshi
Vellore Institute of Technology, Chennai

Under the guidance of

Prof. Kannan Moudgalya
Chemical Engineering Department
IIT Bombay

Mentors

Mr. Nagesh Karmali
Ms. Firuza Aibara

August 2022

Acknowledgement

I **Deepam Priyadarshi**, the summer intern of the **FOSSEE - Arduino On Cloud, 2022** is overwhelmed in all humbleness and gratefulness to acknowledge my deep gratitude to all those who have helped me put our ideas to perfection and have assigned tasks well above the level of simplicity and into something concrete and unique. I wholeheartedly thank **Prof. Kannan M. Moudgalya** for having faith in me, selecting me to be a part of his valuable project and for constantly motivating me to do better. I thank **Mr. Nagesh Karmali** and **Ms. Firuza Aibara** for providing me the opportunity to work on this project. I am also very thankful to our mentors for their valuable suggestions. They were and are always there to show me the right track when needed help. With help of their brilliant guidance and encouragement, I was able to complete my tasks properly and was up to the mark in all the tasks assigned. During the process, I got a chance to see the stronger side of my technical and nontechnical aspects and also strengthen my concepts. Last but not the least, I sincerely thank all our other colleagues working in different projects under **Prof. Kannan M. Moudgalya** for helping me evolve better with their critical advice.

Declaration

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

I also declare that I 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. I 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.

Deepam Priyadarshi

Contents

List of Figures

Chapter 1

Playing songs on Arduino

1.1 Introduction

This task deals in making the Arduino play tones of various frequencies through a piezo buzzer on the *Arduino On Cloud* platform. Initially, the buzzer present on the aforementioned platform was able to generate sound only of a single frequency. Analogous to real world scenario, an Arduino is able to generate various tones on piezo buzzer by generating square waves of 50% duty cycle of the specified frequency. The default *Arduino.h* library provides us with function named tone(pin, frequency, delay), written by *Brett Hagman*, which encapsulates all the functionalities required to generate the 50% duty cycle square wave of given frequency. This task requires the `tone()` function to be compatible with the virtual piezo buzzer present on *Arduino On Cloud* platform and vice versa which enables us to write codes for playing various well known songs having notes of already known frequency. A sample of such codes, playing songs using Arduino can be found on the following GitHub repository - arduino-songs.

1.2 Literature Review

As per Arduino's microprocessor, *ATmega328p*, datasheet, Arduino has 3 timers namely Timer0, Timer1 and Timer2. Timer0 and Timer2 are of 8 bits each while Timer1 is of 16 bits. Each of the timers have the follwing registers associated to manipulate the behavior of the respective time ('x' at the end corresponds to respective timer number) :-

1. **TCCRx** - Timer/Counter Control Register. Can be used to modify timer prescaler value or the timer mode.
2. **TCNTx** - Timer/Counter Register. Stores the value of the counter.
3. **OCRx** - Output Compare Register. Useful in CTC mode of the timer.

According to *Brett Hagman's* code, the `Tone.cpp` code utilises the CTC mode of Atmega328p's Timer2. In Clear Timer on Compare match (CTC) mode, the timer counts until the value stored in the TCNTx register matches a pre-specified value

TOP value, after which an interrupt is generated which can be used to toggle any selected GPIO pins. For Timer2, this TOP value is stored in OCR2A register. The timer mode specification present/set using the Waveform Generation Mode pins (WGM02, WGM01 & WGM00) present of the TCCR2A(WGM01 & WGM00) and TCCR2B(WGM002) for Timer2. The CTC mode can be used to manipulate the frequency of an output signal by assigning the prescaler bits in the last 3 bit of TCCR2B register. The available prescaler values in Arduino for a 8-bit timer are 1, 8, 64, 256 and 1024. All this functionalities are performed together by `tone(int pin, int frequency)` alone to generate a square wave of 50% duty cycle of the specified frequency and on the specified pin.

1.3 Problem Solving Approach

The *Arduino On Cloud* platform uses *AVR8js* library which provides an Object Oriented model of Arduino's 8-bit micro-controller. The CPU class has the necessary methods along with their appropriate access modifiers to manipulate various data members of the class representing different registers for the CPU. In a real world scenario, a piezo buzzer's piezo element has a natural phenomena to vibrate with the applied signal's frequency. To simulate it on a computer we need to know the applied frequency of the signal on the pin connected to the piezo buzzer. There are two approach to the solution :-

1. Taking the inverse of time interval between two different toggle state at the given pin. This method cannot provide accurate figures for most of the frequency values an Arduino can generate.

2. The second approach makes use of the formula given below, provided in the datasheet

$$F_w = \frac{F_{cpu}}{2N(C+1)} \tag{1.1}$$

$$\begin{aligned} F_w &= \text{Output Frequency,} \\ F_{cpu} &= \text{Clock Frequency of Arduino (16MHz),} \\ N &= \text{Prescalar value,} \\ C &= \text{TOP count present in OCR2A register} \end{aligned}$$

from the equation 1.1 we calculate the output frequency F_w by reading the N and C values from the last 3 bits of TCCR2B register and OCR2A register respectively and substituting it in the equation 1.1. This frequency can then be fed to the sound generating module of the buzzer. The N and C values are repeatedly read every millisecond using `setInterval()` method provided by JavaScript.

1.4 Implementation

All the code modification related to this task was done to the `ArduinoFrontend/src/app/Libs/outputs/Buzzer.ts` file's `initSimulation()` method as shown in the code snippet below :-

```
import { CircuitElement } from '../CircuitElement';
import { Point } from '../Point';
import { ArduinoUno } from '../outputs/Arduino';
        .
        .
        .
        .
        .
        .

  /**
   * Logic for beeping sound
   * @param val The Value on the positive pin
   */
  logic(val: number) {
    // TODO: Handle PWM
    if (this.nodes[0].connectedTo && this.nodes[1].connectedTo) {
      if (val === 5) {
        if (this.oscillator && !this.sound) {
          this.oscillator.connect(this.audioCtx.destination);
          this.sound = true;
        }
      } else {
        if (this.oscillator && this.sound) {
          this.oscillator.disconnect(this.audioCtx.destination);
          this.sound = false;
        }
      }
      this.nodes[1].setValue(val, null);
    } else {
      // TODO: Show Toast
      window.showToast('Buzzer is not Connected properly');
    }
  }
  /**
   * returns properties object
   * @param keyName Unique Class name
   * @param id Component id
   * @param body body of property box
   * @param title Component title
   */
```

```
properties() {
  const body = document.createElement('div');
  return {
    title: 'Buzzer',
    keyName: this.keyName,
    id: this.id,
    body
  };
}
/**
 * Initialize Variable and callback when start simulation is pressed
 */
initSimulation() {
  const trig = (this.pinNamedMap['POSITIVE'] as Point);
  if (trig.connectedTo) {
    if (trig.connectedTo.start.parent instanceof ArduinoUno) {
      this.arduino = trig.connectedTo.start.parent;
    } else if (trig.connectedTo.end.parent instanceof ArduinoUno) {
      this.arduino = trig.connectedTo.end.parent;
    }
  }
  const AudioContext = window.AudioContext || window.webkitAudioContext;
  this.audioCtx = new AudioContext();
  this.oscillator = this.audioCtx.createOscillator();
  this.oscillator.type = 'square';
  this.oscillator.frequency.value = 2300;
  const prescaler = [8, 32, 64, 128, 256, 1024];
  this.setIntervId = setInterval(() => {
    try {
      const tccr2b = this.arduino.runner.timer2.TCCRB;
      const ocr2a = this.arduino.runner.timer2.ocrA;
      if (ocr2a !== 0) {
        this.oscillator.frequency.value = Math.round(16000000 / (2 *
         ↪  prescaler[tccr2b - 2] * (ocr2a + 1)));
      }
    } catch (error) {
    }
  }, 10);
  this.oscillator.start();
}

.
.
.
.
.
```

```

}
```

The calculated frequency is constantly assigned to the frequency parameter of the `AudionContext()` object, a web audio api provided by WebKit.

1.5 Result

The code testing and the necessary simulations has been shown in the demo video viewable by clicking on the figure 1.1. The song played in the demo video is the *Harry Potter* theme song whose code and many other songs code can be found on the GitHub link provide below -
Arduino Songs

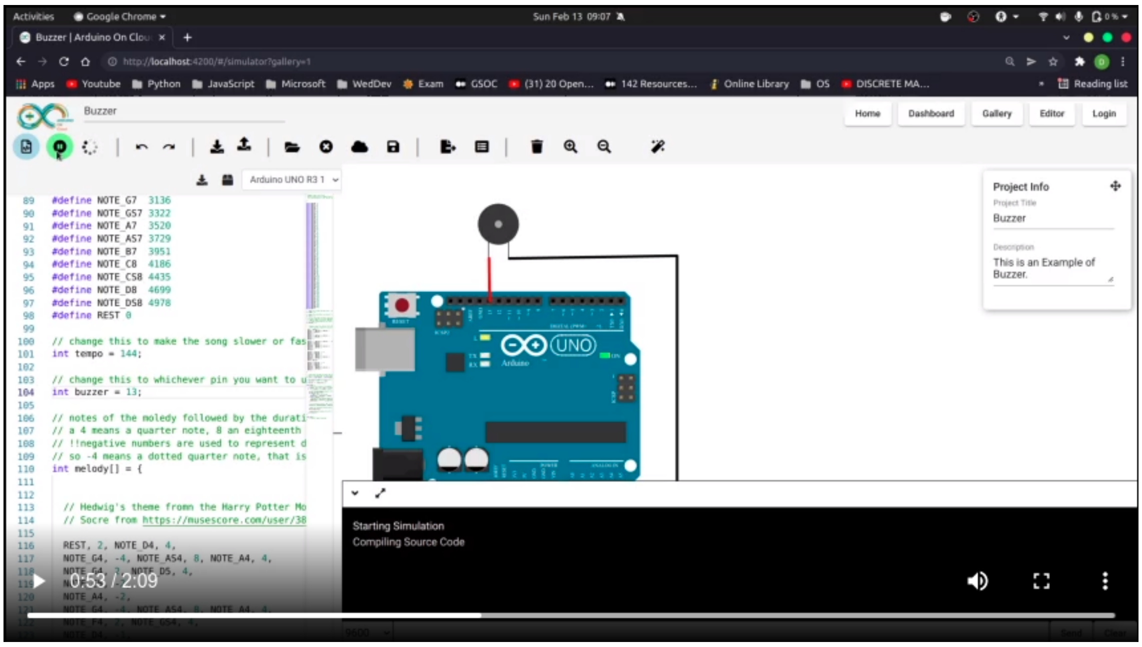

```
#define NOTE_G7  3136
#define NOTE_GS7 3322
#define NOTE_A7  3520
#define NOTE_AS7 3729
#define NOTE_B7  3951
#define NOTE_C8  4186
#define NOTE_CS8 4435
#define NOTE_D8  4699
#define NOTE_DS8 4978
#define REST 0

// change this to make the song slower or fas
int tempo = 144;

// change this to whichever pin you want to u
int buzzer = 13;

// notes of the moledy followed by the durati
// a 4 means a quarter note, 8 an eighteenth
// !!negative numbers are used to represent d
// so -4 means a dotted quarter note, that is
int melody[] = {


  // Hedwig's theme fromn the Harry Potter Mo
  // Socre from https://musescore.com/user/38

  REST, 2, NOTE_D4, 4,
  NOTE_G4, -4, NOTE_AS4, 8, NOTE_A4, 4,
  NOTE_G4, 2, NOTE_D5, 4,
  NOTE_C5, -2,
  NOTE_A4, -2,
  NOTE_G4, -4, NOTE_AS4, 8, NOTE_A4, 4,
  NOTE_F4, 2, NOTE_GS4, 4,
  NOTE_D4, -1,
```

Figure 1.1: Playing songs on Arduino.

1.6 References

1. https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf

2. https://www.arduino.cc/reference/en/language/functions/advanced-io/tone/

3. https://github.com/arduino/ArduinoCore-avr/blob/master/cores/arduino/Tone.cpp

4. https://github.com/robsoncouto/arduino-songs

5. https://github.com/wokwi/avr8js/blob/master/src/cpu/cpu.ts

Chapter 2

AVR-GCC support for Arduino

2.1 Introduction

This task deals in making the *Arduino On Cloud* platform compatible in compiling high-level C code to program the given Arduino. This mode of Arduino programming is also sometimes known as *Bare Metal programming* in Arduino. At present the *Arduino On Cloud* only support the compilation of .ino files. Providing the support for inline assembly programming helps in better understanding the hardware architecture of ATmega328p and can also be used for writing more optimized Arduino programs.

2.2 Literature Review

The support for the compilation of C code for converting it into AVR assembly language code can be achieved using AVR-GCC toolchain that comprises of compiler, assembler, linker and standard C and math libraries for AVR micro-controllers. The source code is now saved with '.c' extension. The AVR-GCC tool chain for Linux provides a compiler that converts the source file to object file. The object file is then converted to an executable file. This executable file is finally converted to HEX file which then needs to be uploaded on the Arduino.

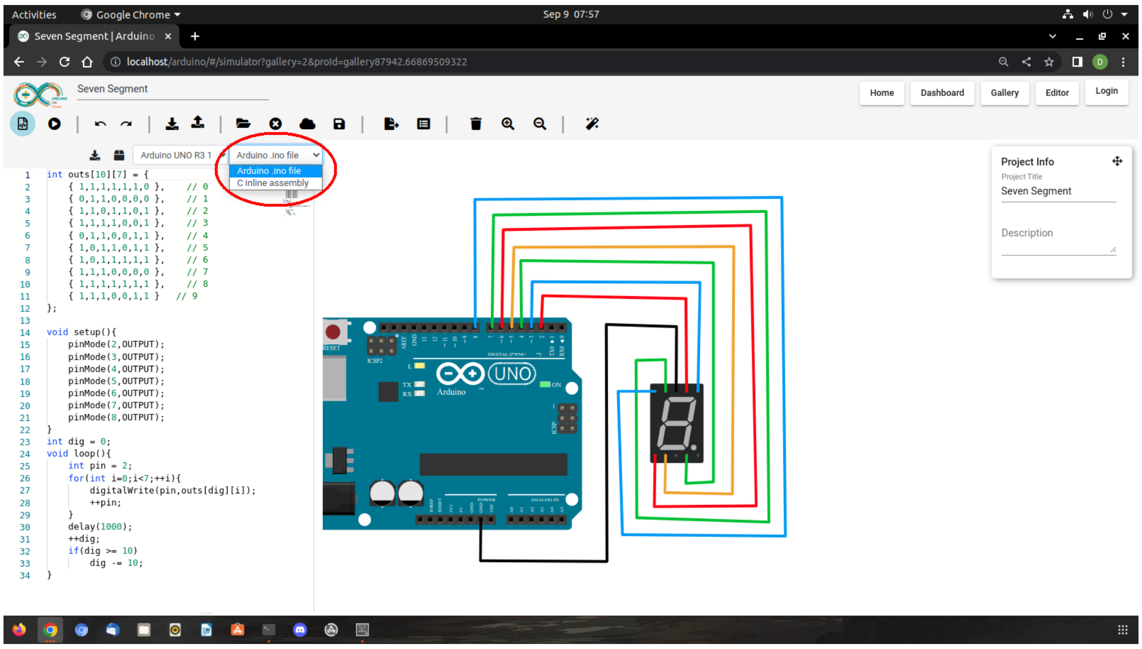
In a real world scenario, typically an Arduino is programmed using the *Arduino IDE* which provides an abstraction for all the above mentioned procedure in a single click of 'Upload Sketch' button. At the backend, the IDE also uses the same AVR-GCC toolchain, compatible to the given OS. Apart from this, the IDE also provides some additional headers (like *Arduino.h*) and macros which gets added to the source file prior to its compilation. This enables writing user-friendly code for the Arduino by providing encapsulation of various complicated low level functionality into a single function.

2.3 Problem solving approach

The user is provided with a selection menu in the code editor component to select the type of file user wants to create, `.ino` or `.c`. If the user selects the former, then the source file is compiled using `arduino-cli` toolchain or else it gets compiled using `AVR-GCC` toolchain. Both of these process return an equivalent HEX code of the source code, which get uploaded onto the Arduino and is then parsed by the Arduino Runner class.

2.4 Implementation

- The first change is the addition of a drop down selection menu to select the type of source code file. From figure 2.1 we can see that the platform provides user two modes to write code for Arduino.



```
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;
}
```

Figure 2.1: Drop down menu for file type selection.

- Upon selecting one of the two modes of programming, the respective endpoints also get mapped to 'Start Simulation' button which gets called to initiate the simulation. The two API endpoints are -

  (i) **POST api/arduino/compileINO** :-
  Saves the file with `.ino` extension and then uses `arduino-cli` toolchain for generating equivalent HEX code.

  (ii) **POST api/arduino/compileInlineAssembly** :-
  Saves the file with `.ino` extension and then uses `AVR-GCC` toolchain for generating equivalent HEX code.

- If 'Inline C Assembly' option is chosen then the `saveFiles()` function present on `eSim-Cloud/esim-cloud-backend/arduinoAPI/tasks.py` save the code as a `.c` on the container and returns the file name to the calling function.

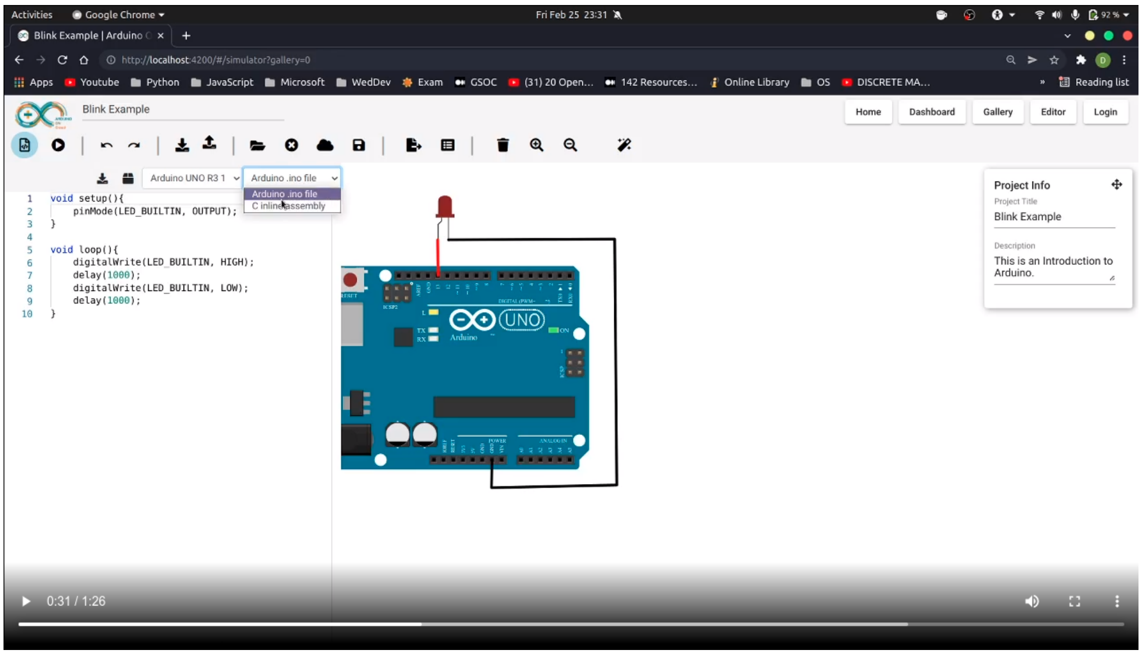
```python
def saveFiles(data, langIndex):
    # try:
    filenames = []
    if not os.path.exists(settings.MEDIA_ROOT):
        Path(settings.MEDIA_ROOT).mkdir(parents=True, exist_ok=True)
    for k in data:
        foldername = str(uuid.uuid4()) + '_' + str(k)
        work_dir = settings.MEDIA_ROOT+'/'+str(foldername)
        Path(work_dir).mkdir(parents=True, exist_ok=True)
        if langIndex == 0:
            filename =
            ↪   settings.MEDIA_ROOT+'/'+str(foldername)+'/sketch.ino'
        elif langIndex == 1:
            filename = settings.MEDIA_ROOT+'/'+str(foldername)+'/sketch.c'
        fout = open(filename, 'w', encoding='utf8')
        matches = re.finditer(PATTERN, data.get(k, ''), re.MULTILINE)
        for _, match in enumerate(matches, start=1):
            func_name = match.group().replace('{', '')
            func_name = func_name.strip() + ';'
            fout.writelines('#line 1 "{}"\n'.format(filename))
            fout.writelines('{}\n'.format(func_name))
        fout.writelines(data.get(k, ''))
        fout.close()
        filenames.append(foldername)
        logger.info('Creating')
        logger.info(filename)
    return filenames
```

- After saving the .c file, `CompileInlineAssembly()` function generate the object file out of it using the command `avr-gcc -Os -DF_CPU=16000000UL -mmcu=atmega328p -c -o sketch.o sketch.c`. The object file `sketch.o` is again converted to executable code using the command `avr-gcc -mmcu=atmega 328p sketch.o -o sketch`. This executable file is finally converted to equivalent HEX code which is the returned as shown in the code below.

```python
def CompileInlineAssembly(filenames):
    ret = {}
    try:
        for filename in filenames:
            c_name = settings.MEDIA_ROOT+'/'+str(filename)+'/sketch.c'
            obj_name = settings.MEDIA_ROOT+'/'+str(filename)+'/sketch.o'
            bin_name = settings.MEDIA_ROOT+'/'+str(filename)+'/sketch'
            out_name = settings.MEDIA_ROOT+'/'+str(filename)+'/out.hex'
            logger.info('Compiling')
            logger.info(c_name)

            ps = subprocess.Popen(
                "avr-gcc -Os -DF_CPU=16000000UL -mmcu=atmega328p -c -o " + obj_name + " " + c_name + " && avr-gcc -mmcu=atmega328p " + obj_name + " -o " + bin_name + " && avr-objcopy -O ihex -R .eeprom " + bin_name + " " + out_name,
                stdout=subprocess.PIPE,
                stderr=subprocess.PIPE,
                shell=True
            )
            output, err = ps.communicate()
            data = ''
            if err == '' and ps.returncode != 0:
                err = b'Code Cannot be Compiled: Unknown Reason'

            if os.path.isfile(out_name):
                data = open(out_name, 'r').read()
            pos = filename.find('_')
            if pos != -1:
                pos += 1
                key = filename[pos:]
            else:
                key = filename

            ret[key] = {
                'output': "Compiled Successfully",
                'error': re.sub(
                    rf'{settings.MEDIA_ROOT}/{filename}/',
                    '',
                    err.decode('utf-8')
                ),
                'data': data
            }

    except Exception:
        print(traceback.format_exc())
        return False
    finally:
        for filename in filenames:
            parent = settings.MEDIA_ROOT+'/'+str(filename)
            shutil.rmtree(parent, True)
            logger.info('Removing')
            logger.info(parent)
    return ret
```

2.5 Output

The working demo of the above implementation can be viewed by clinking on the image-link of figure 2.2. The code used in the video blinks the LED with a delay of 1000ms.



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

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

Figure 2.2: Arduino inline assembly coding.

2.6 References

1. https://spoken-tutorial.org/watch/Arduino/AVR-GCC+programming+through+Arduino/English/

2. https://linux.die.net/man/1/avr-gcc

3. https://create.arduino.cc/projecthub/milanistef/introduction-to-bare-metal-programming-in-arduino-uno-f3e2b4

4. https://github.com/arduino/arduino-ide

Chapter 3

LTI and Auto-Grading Support

3.1 Introduction

Learning Tool Interoperability(LTI), is a security standard developed by *1EdTech* (previously IMS Global Learning Consortium) for secure information exchange between a Learning Management Systems (LMS) and external learning tool. It is enabled via *OAuth2* through which LMS platforms like moodle, edX etc. can host an external learning tool's interface to work with, without having to log into the external system. The information about the learner and their activities are shared by LMS to the external system. Custom restriction on features of the hosted LTI interface can be added for free and premium user or even for creating assignment for student. Through LTI support on *Arduino on Cloud* platform, custom assignments related to Arduino circuits and its coding can created on moodle which can then be graded through auto grading support as per student simulation results.

3.2 Literature Review

LTI was created to standardize the creation of content among LMS platforms. Through LTI, functionalities of learning technologies can be delivered by these LMS platform through a secure plug and play environment. Students and instructors only have to keep track of one login. LTI1.3 has become the minimum requirement for tools that exchange sensitive and personally identifiable data. Tracking grades become more standardized across different LMS platforms since the grading specifications are handled by the learning tool itself and the final grade is scaled down to a uniform scale specified by LTI1.3. This enables to create assignments using more than learning tool provider, in a course, still maintaining the uniform scoring standards among them.

3.3 Problem Solving Approach

The user requirements for this particular component are:-

- Assignments that can be created should be of three type, either circuit oriented where the student is given an incomplete circuit and is asked to complete it as per the correct code given to him or code oriented where the complete circuit is given to student and is asked to write appropriate code for it or the student needs to create the circuit as well as write a code for it according to the specification provided in a question statement.

- The student is graded on the simulation results of his circuit and if the simulation results matches with instructors simulation data he/she is awarded with full marks. Partial marking is also awarded based on the extent of similarity in simulation results achieved by the student.
- The instructor has the privilege to decide the weightage of the circuit drawn and code.
- The instructor also has the privilege to select one of available valid test cases of the actual correct circuit for evaluation.
- While creating an assignment, the instructor can also choose from list of incorrect variations of the actual circuit to assign it as the initial circuit to work on, for the student.

Auto-Grading

Grading is done based on the data generated upon simulation of the circuit. The information contained in this data are the various pins of Arduino that were used in wiring connections. These are then matched with the pins used in instructor's simulation circuit and the resultant value contributes to connection weightage of the overall grade. The simulation data also contains the HEX values that are fed into the PORTB(pins 0-7) and PORTD(pins 8-13) registers by the CPU during the runtime. The binary equivalent of these HEX values decides which GPIO pin of Arduino will toggle to logic HIGH based on the position of the 1's in the 8-bit binary value. The HEX values received by both the registers follow a particular sequence during the runtime. If two similar such sequence exits then both the circuits are said to perform same simulation/operation. This contributes to the code weightage of the overall grade.

3.4 Implementation

3.4.1 Creating a LTI assignment

To create an LTI assignment the user needs to save the original correct circuit. Any variations of this circuit can also be saved along with it. Run the simulation for the original/correct circuit and save the necessary data points generated to choose one them as the test case for evaluation. This process has been clearly depicted clearly depicted in the figure 3.1.

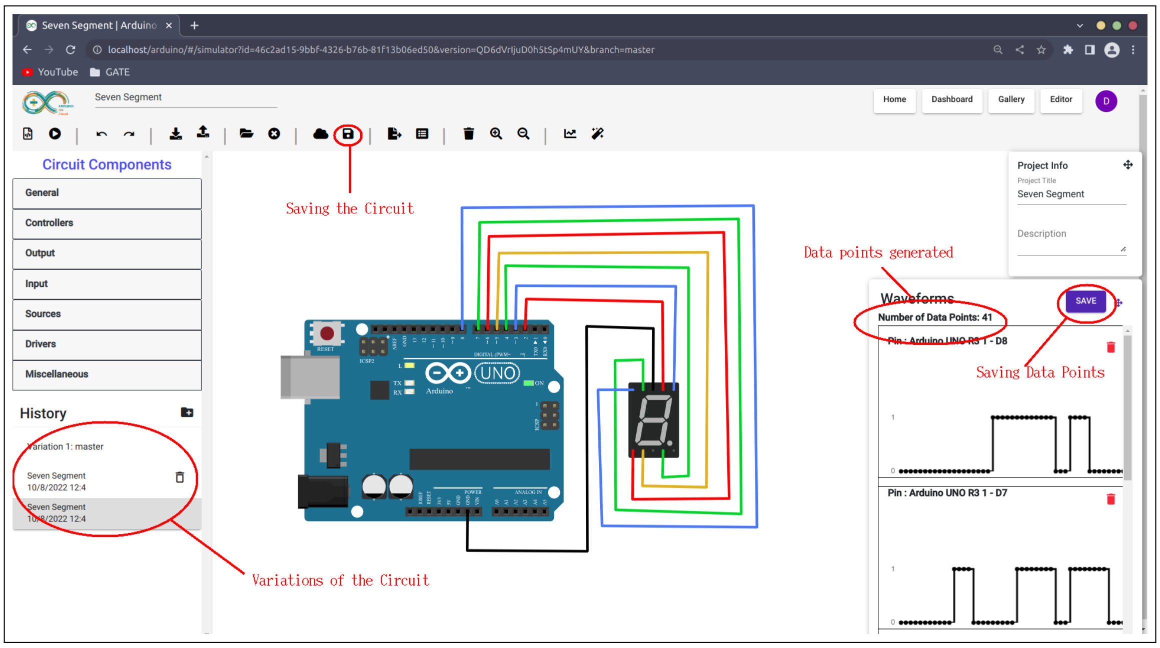
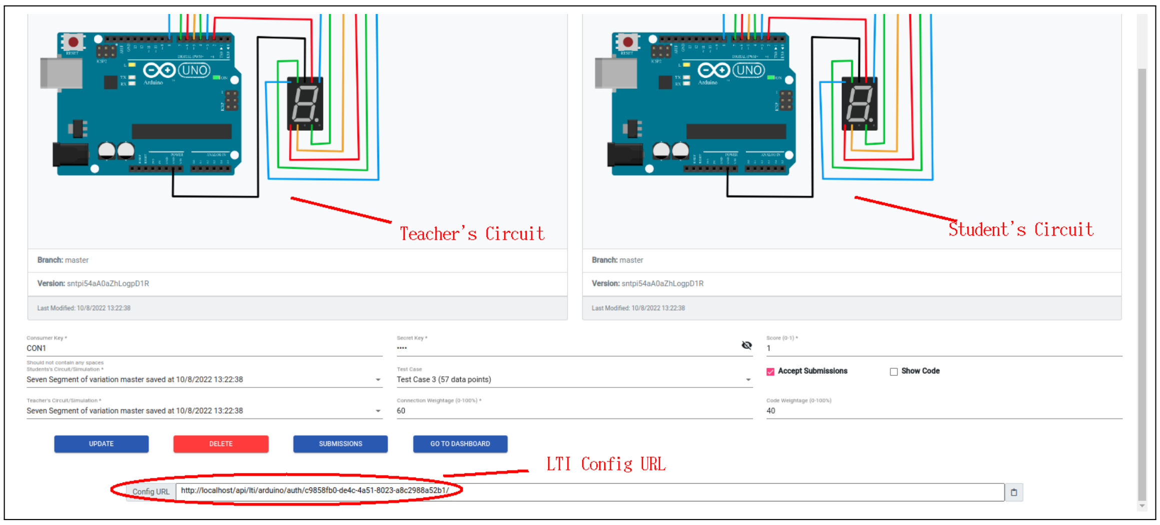


Figure 3.1: Saving the Circuit.

Open the saved the circuit from the dashboard. Click on "Create LTI App" option from the dialog box that appears. This brings you to the LTI setup page. Here you can fill in the LTI specifications related to the assignment. For creating the LTI app you need to specify the 'Consumer Key', 'Secret Key', maximum possible score of the assignment (0.0 to 1.0), select the initial circuit for student to work on - 'Student's Circuit', the original correct circuit - 'Teacher's Circuit', Test case data, enabling auto-grading by accepting submissions, showing the code to the student or not and specifying the code and circuit weightage. After filling all the compulsory details the 'Save' button will get activated which will save the details onto the database and will also generate the LTI config url to be used at the LMS platform as shown in figure 3.2.



Branch: master

Version: sntpi54aA0aZhLogpD1R

Last Modified: 10/8/2022 13:22:38

Branch: master

Version: sntpi54aA0aZhLogpD1R

Last Modified: 10/8/2022 13:22:38

Consumer Key *
CON1
Should not contain any spaces

Secret Key *
••••

Score (0-1) *
1

Students's Circuit/Simulation *
Seven Segment of variation master saved at 10/8/2022 13:22:38

Test Case
Test Case 3 (57 data points)

[✓] Accept Submissions
[ ] Show Code

Teacher's Circuit/Simulation *
Seven Segment of variation master saved at 10/8/2022 13:22:38

Connection Weightage (0-100%) *
60

Code Weightage (0-100%)
40

Config URL http://localhost/api/lti/arduino/auth/c9858fb0-de4c-4a51-8023-a8c2988a52b1/

Figure 3.2: LTI Created.

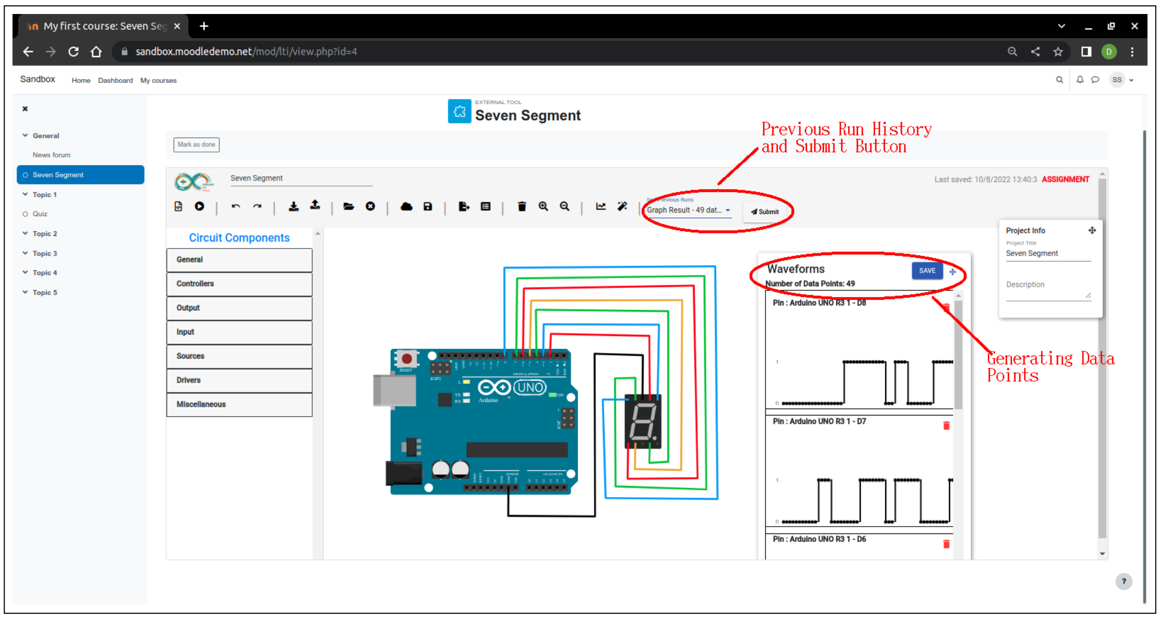


Figure 3.3: Moodle View.

3.4.2 Submission and Auto-Grading

On LMS after completing the external tool setup by using the earlier provided consumer key, secret key and LTI config URL, student gets to sees an *Arduino On Cloud* platform's interface embedded into the LMS assignment section of a course as shown in the figure 3.3. The student needs to modify the circuit or code or even both as per the questions requirement. Then the simulations needs to be run for generating the data points for evaluation. A student can run simulation any number of time. All his simulation records will be listed in the 'See Previous Runs' drop down menu. From there anyone of them can be chosen for evaluation by clicking on the submit button. The auto-grading process evaluate the circuit based on the GPIO pins used and HEX sequence generated in the PORT registers.

3.5 Output

After evaluation, the marks are displayed on the submission confirmation dialog box as shown in figure 3.4. The marks shown in the figure are scaled down to LMS uniform grading standard for LTI apps. In figure 3.4 since simulation data and the circuit matches with the valid reference circuit of the instructor, hence the student has been awarded 1.0 marks. Figure 3.5 shows that the student has been awarded 0.91 marks. The marks was deducted due to the submission of partially incorrect circuit. Though the student got full 60% marks for his submitted code but was only able to get 31% marks out 40% marks, for his submitted circuit.

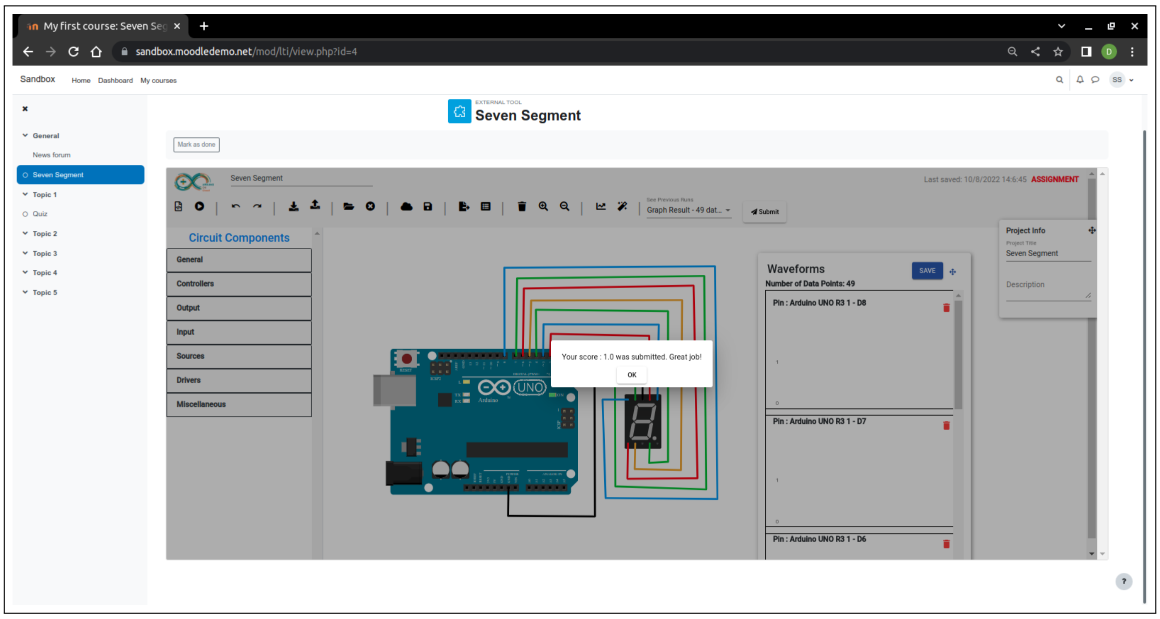

Your score : 1.0 was submitted. Great job!

Figure 3.4: Submission of correct answer.

Another possible scenario is shown in figure 3.6 where the circuit submitted is completely correct but the code submitted is only partially correct. Since the code has a higher weightage than circuit component, the deducted marks are also higher in this case. Hence the student is only awarded 0.87 marks.

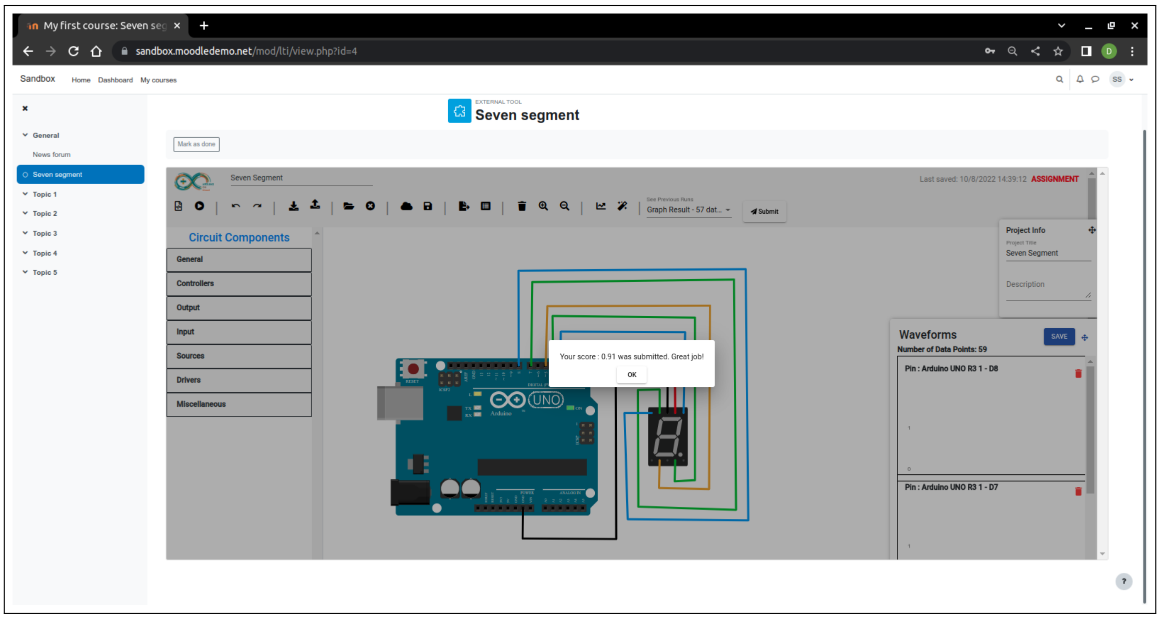
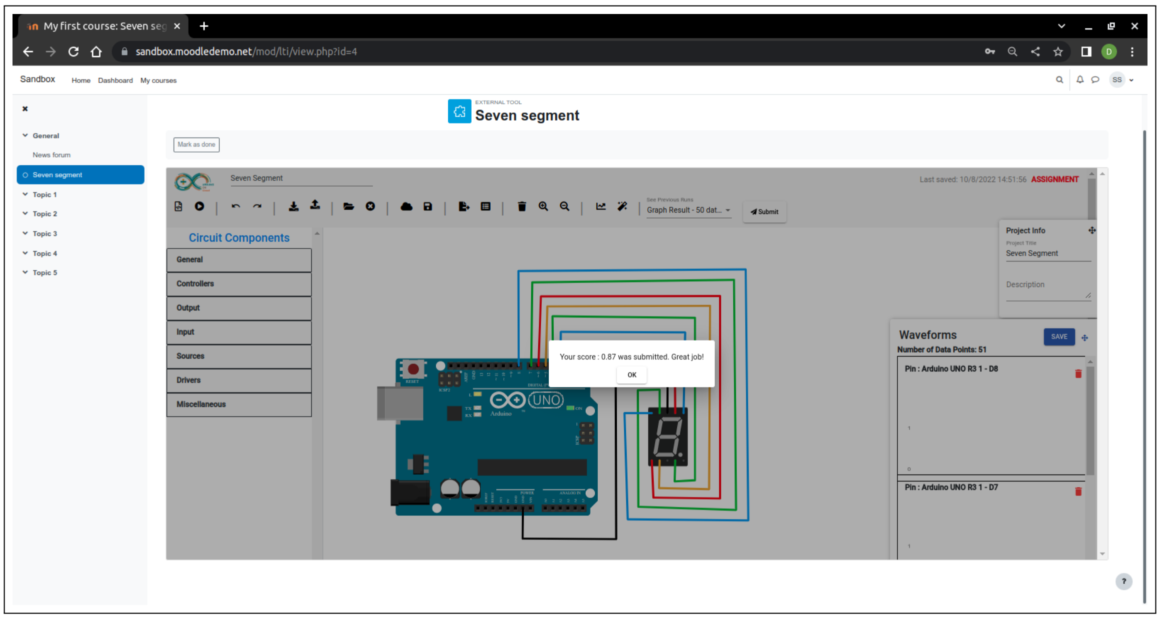

Your score : 0.91 was submitted. Great job!

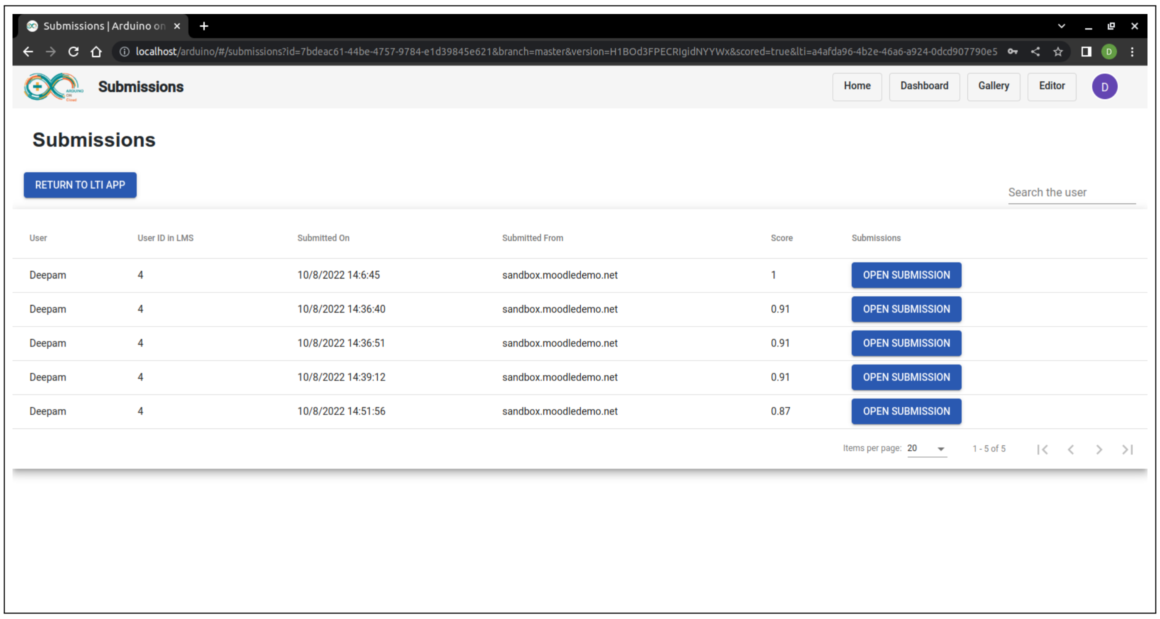
Figure 3.5: Submitting incorrect circuit but valid code.



Your score : 0.87 was submitted. Great job!

Figure 3.6: Submitting valid circuit but invalid code.

The circuit and the code submitted by the students can be viewed on the *Arduino On Cloud* platform by navigating to submissions page from the dashboard or from the ‘Edit LTI page’.


Submissions | Arduino on
localhost/arduino/#/submissions?id=7bdeac61-44be-4757-9784-e1d39845e621&branch=master&version=H1BOd3FPECRIgidNYYWx&scored=true&lti=a4afda96-4b2e-46a6-a924-0dcd907790e5
Submissions
Home
Dashboard
Gallery
Editor
RETURN TO LTI APP
Search the user

Submissions

User	User ID in LMS	Submitted On	Submitted From	Score	Submissions
Deepam	4	10/8/2022 14:6:45	sandbox.moodledemo.net	1	OPEN SUBMISSION
Deepam	4	10/8/2022 14:36:40	sandbox.moodledemo.net	0.91	OPEN SUBMISSION
Deepam	4	10/8/2022 14:36:51	sandbox.moodledemo.net	0.91	OPEN SUBMISSION
Deepam	4	10/8/2022 14:39:12	sandbox.moodledemo.net	0.91	OPEN SUBMISSION
Deepam	4	10/8/2022 14:51:56	sandbox.moodledemo.net	0.87	OPEN SUBMISSION

Items per page: 20 1 - 5 of 5

Figure 3.7: Submissions page.

3.6 References

1. https://spoken-tutorial.org/watch/Arduino/AVR-GCC+programming+through+Arduino/English/
2. https://moodle.com/news/what-is-lti-and-how-it-can-improve-your-learning-ecosystem/
3. https://www.imsglobal.org/activity/learning-tools-interoperability
4. https://github.com/rohitjose/django-lti-auth
5. https://github.com/Harvard-ATG/django-app-lti