



Design & Analysis of IoT Software Systems - SOFE4610U

Assignment 3

CRN: 44432

Group Number 8

Group Member 1

Name: Alden O'Cain

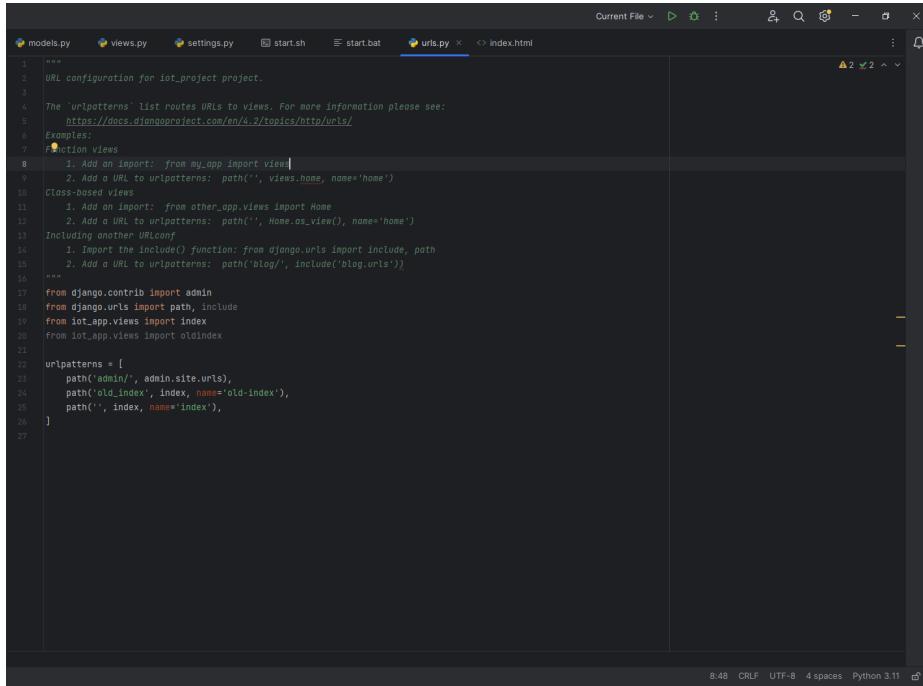
Student ID: 100558599

Group Member 2

Name: Liam Rea

Student ID: 100743012

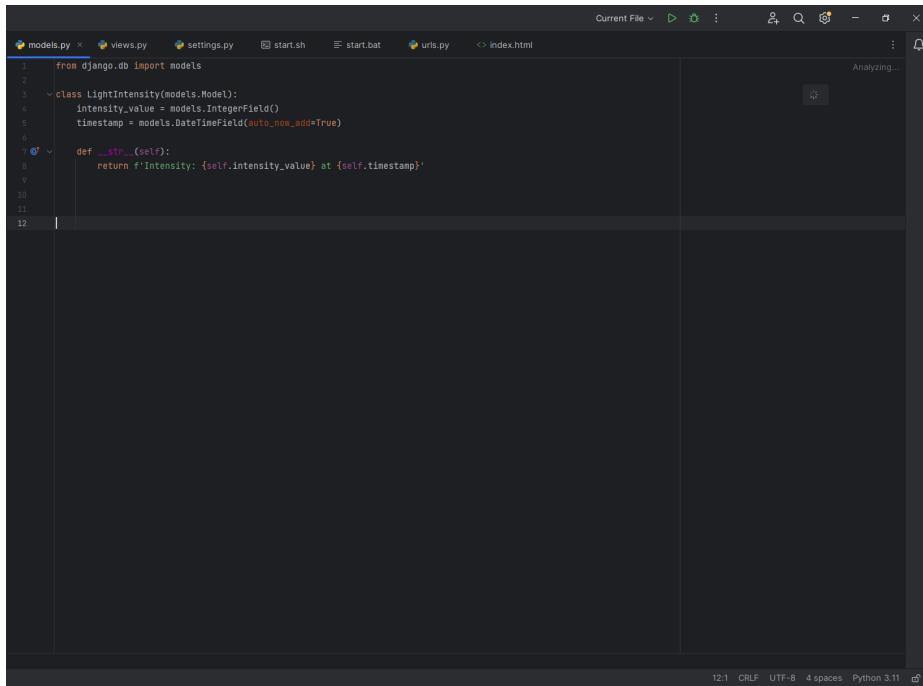
Project Repository



A screenshot of a code editor showing the `urls.py` file for a Django project. The file contains code for defining URL routes. The code includes imports for `views`, `path`, and `include`, and defines a list of `urlpatterns` that map URLs to views.

```
1 /**
2  * URL configuration for iot_project project.
3  *
4  * The 'urlpatterns' list routes URLs to views. For more information please see:
5  *     https://docs.djangoproject.com/en/4.2/topics/http/urls/
6  *
7  * Examples:
8  *     1. Add an import: from my_app import views
9  *    2. Add a URL to urlpatterns: path('', views.home, name='home')
10 * Class-based views
11 *     1. Add an import: from other_app.views import Home
12 *     2. Add a URL to urlpatterns: path('', Home.as_view(), name='home')
13 * Including another URLconf
14 *     1. Import the include() function: from django.urls import include, path
15 *     2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))
16 *
17 * from django.contrib import admin
18 * from django.urls import path, include
19 * from iot_app.views import index
20 * from iot_app.views import oldindex
21 *
22 * urlpatterns = [
23 *     path('admin/', admin.site.urls),
24 *     path('old_index', index, name='old-index'),
25 *     path('', index, name='index'),
26 * ]
```

Defining URL routes in the Django app



A screenshot of a code editor showing the `models.py` file for a Django model named `LightIntensity`. The code defines a new model with two fields: `intensity_value` (an integer) and `timestamp` (a date/time field). It also includes a `__str__` method to return a string representation of the object.

```
1 from django.db import models
2
3 class LightIntensity(models.Model):
4     intensity_value = models.IntegerField()
5     timestamp = models.DateTimeField(auto_now_add=True)
6
7     def __str__(self):
8         return f'Intensity: {self.intensity_value} at {self.timestamp}'
```

Creating a model for light intensity data

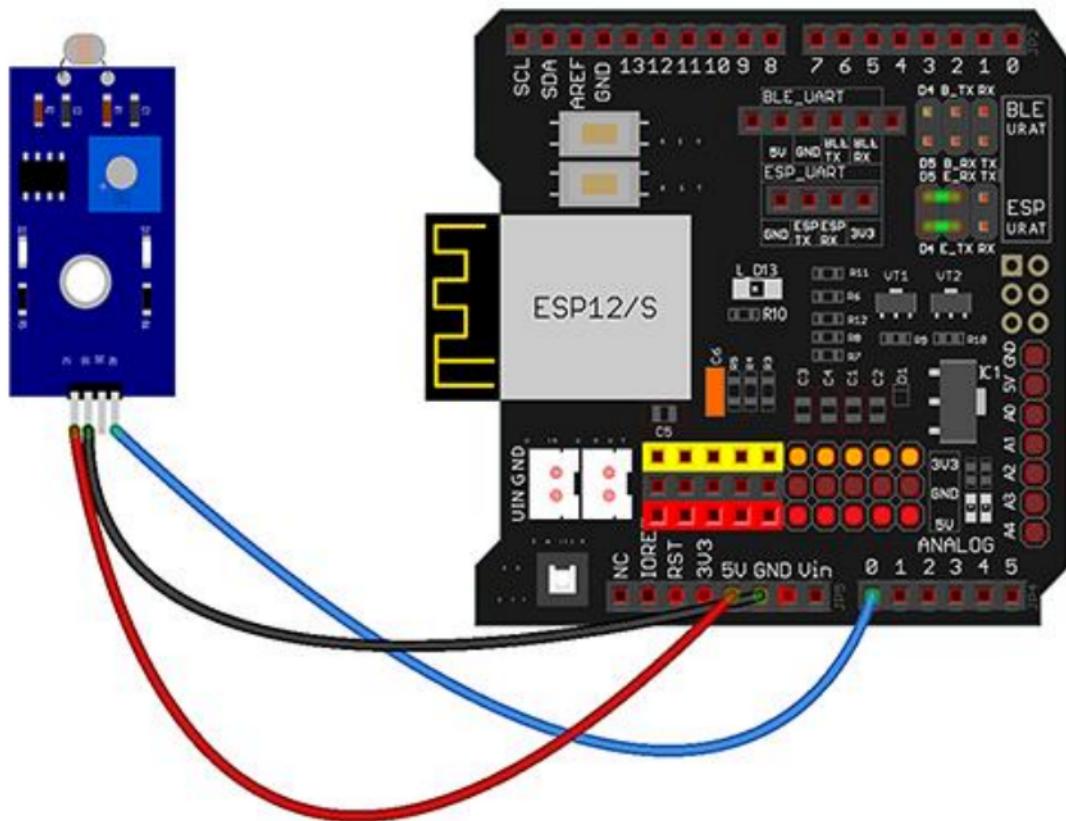
```
models.py  views.py  settings.py  start.sh  urls.py  index.html
7
8     intensity_value = 42
9     light_intensity = LightIntensity.objects.create(intensity_value=intensity_value)
10    light_intensity.save()
11
12
13    from django.http import HttpResponseRedirect
14    from django.views.decorators.csrf import csrf_exempt
15    from .models import LightIntensity # Import your LightIntensity model
16
17
18    6 usages
19    def index(request):
20        ser = serial.Serial()
21        ser.port = "/dev/rfcomm0"
22        ser.baudrate = 115200
23        ser.timeout = 1
24        ser.setDTR(False)
25        ser.setRTS(False)
26        ser.open()
27        new_data = 0
28        ser.flushInput()
29        ser.flush()
30        ser.flushOutput()
31        time.sleep(1)
32        raw_data = ser.readline()
33        try:
34            ser.flushInput()
35            new_data = int(raw_data)
36            ser.flush()
37        except ValueError:
38            pass
39        latest_intensity = raw_data
40        # latest_intensity = lightintensity.objects.last()
41        return render(request, template_name='index.html', context={'latest_intensity': latest_intensity if latest_intensity else 'N/A'})
42
43
44    2 usages
45    def oldIndex(request):
46        latest_intensity = LightIntensity.objects.last()
47        return render(request, template_name='index.html', context={'latest_intensity': latest_intensity.intensity_value if latest_intensity else 'N/A'})
```

Creating a view for the index page (oldIndex displays a static value)

Light Intensity Display

Latest Light Intensity: 42

Old index displays a static value to test the HTML template



Wiring the photoresistor sensor to the Arduino WiFi shield

Testing reading the sensor and printing to the Serial Monitor

IOT_lab2.ino

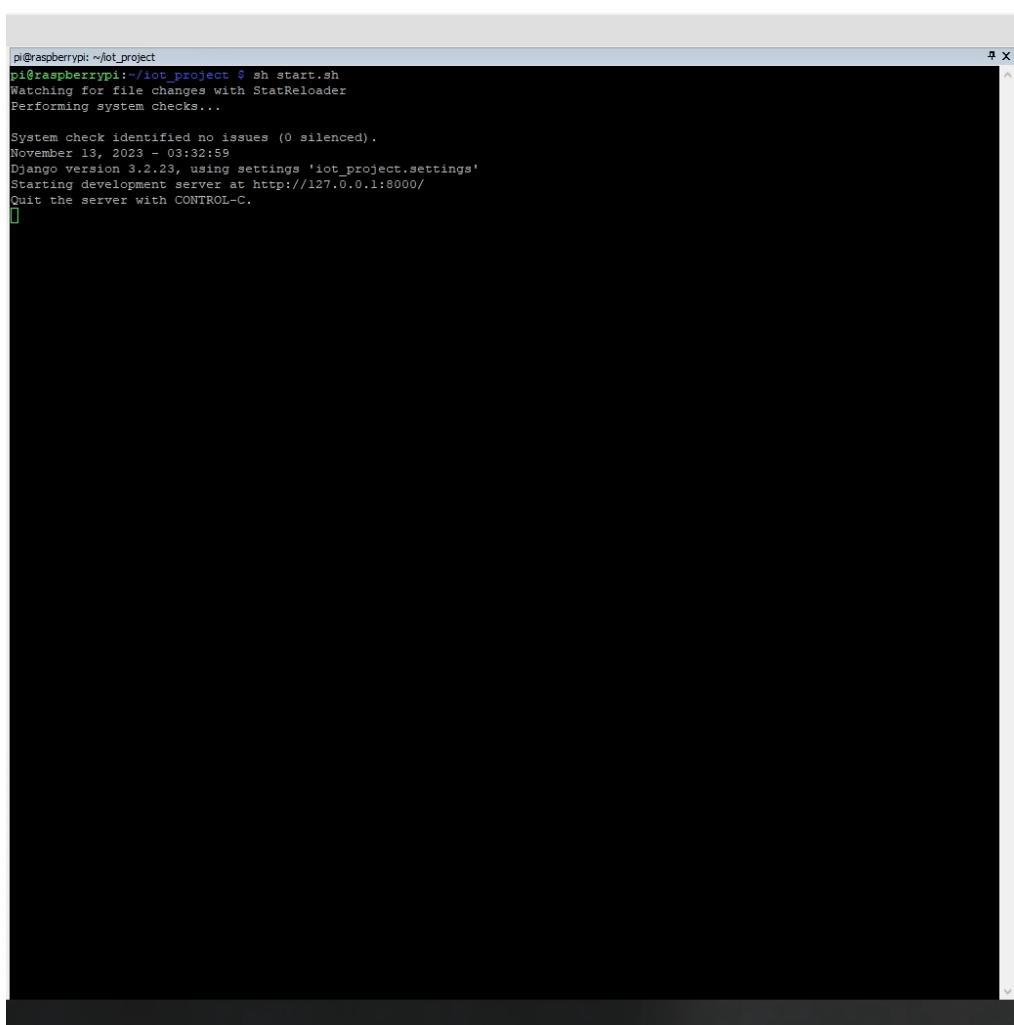
```
1 #include <SoftwareSerial.h>
2
3 #define DHTPIN A0
4
5 SoftwareSerial btSerial(4, 5); // TX, RX
6
7 void setup() {
8     btSerial.begin(9600);
9     Serial.begin(9600);
10    btSerial.flush();
11    Serial.flush();
12 }
13
14 void loop() {
15     int sensorValue = analogRead(A0);
16     btSerial.println(sensorValue);
17     delay(1000); // Send data every 10 seconds
18 }
```

Output Serial Monitor x

Not connected. Select a board and a port to connect automatically.

Updating the Arduino sketch to send the sensor value through bluetooth

Testing capturing the photoresistor sensor values using a simple python script

A terminal window titled "Terminal" is displayed on a Raspberry Pi. The window shows the command "sh start.sh" being run in the directory "/iot_project". The output of the command is visible, indicating that the Django development server is starting at "http://127.0.0.1:8000/".

```
pi@raspberrypi:~/iot_project
pi@raspberrypi:~/iot_project $ sh start.sh
Watching for file changes with StatReloader
Performing system checks...
System check identified no issues (0 silenced).
November 13, 2023 - 03:32:59
Django version 3.2.23, using settings 'iot_project.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Running the updated Django server on the Raspberry Pi

MTPutTY (Multi-Tabbed PuTTY)

Server View Tools Help

pi@raspberrypi: ~/docker-example-python-web-server X

```
pi@raspberrypi:~/docker-example-python-web-server $ curl http://localhost:8000
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Light Intensity Display</title>
</head>
<body>
    <h1>Light Intensity Display</h1>
    <p>Latest Light Intensity: b'1023\r\n'</p>
</body>
</html>
pi@raspberrypi:~/docker-example-python-web-server $
```

Testing the output of the Django web server locally on the Raspberry pi using curl after incorporating reading the bluetooth data inside the Django view

Light Intensity Display

Latest Light Intensity: b'1023\r\n'

Django server displaying the correct values to another computer on the network after changing the Django settings to bind to the private IP address of the Raspberry Pi

Implementing Manual and Automatic Control of a Light

Light Intensity Display

Latest Light Intensity: 50

Control: AUTOMATIC

LED: OFF

[Manual](#)

Simulating (due to faulty photoresistor sensor) a low value (brightness) resulting in AUTO light OFF

Light Intensity Display

Latest Light Intensity: 50

Control: MANUAL

LED: OFF

[Automatic](#) [Toggle](#)

Simulating (due to faulty photoresistor sensor) a low value (brightness) resulting in MANUAL mode with the ability to toggle the light

Light Intensity Display

Latest Light Intensity: 11000

Control: AUTOMATIC

LED: ON

[Manual](#)

Simulating (due to faulty photoresistor sensor) a high value (darkness) resulting in AUTO light ON

Light Intensity Display

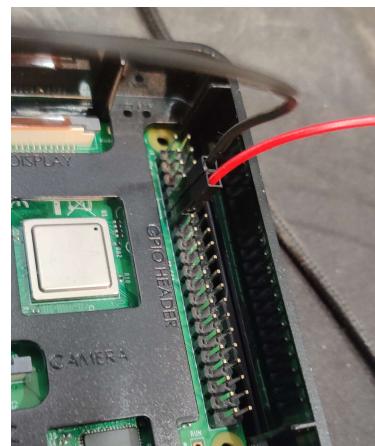
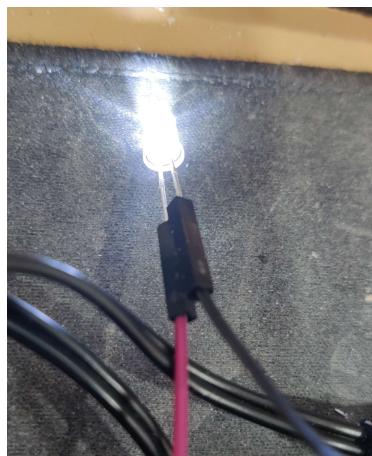
Latest Light Intensity: 11000

Control: MANUAL

LED: OFF

[Automatic](#) [Toggle](#)

Simulating (due to faulty photoresistor sensor) a high value (darkness) resulting in MANUAL mode with the ability to toggle the light



LED connected to Raspberry Pi with a 200 ohm resistor

```

<body>
    <h1>Light Intensity Display</h1>
    <p>Latest Light Intensity: {{ latest_intensity }}</p>

    <form method="POST" action="{% url 'index' %}">
        {% csrf_token %}
        <p>Control: {{ mode }}</p>
        <p>LED: {{ led_value }}</p>

        {% if mode != "MANUAL" %}
            <button type="submit" name="mode-manual" value="manual">Manual</button>
        {% endif %}
        {% if mode != "AUTOMATIC" %}
            <button type="submit" name="mode-automatic" value="automatic">Automatic</button>
        {% endif %}
        {% if mode != "AUTOMATIC" %}
            <button type="submit" name="toggle" value="toggle">Toggle</button>
        {% endif %}
    </form>
</body>

```

Updated HTML template with conditional visibility of buttons and display of state values

```

if request.method == 'POST':
    global led_value
    global mode

    if 'mode-manual' in request.POST:
        mode = "MANUAL"
    if 'mode-automatic' in request.POST:
        mode = "AUTOMATIC"

    if 'toggle' in request.POST:
        global led_state
        led_state = not led_state
        if mode == "MANUAL":
            if led_state:
                print("[LED-MANUAL] Attempting to turn LED on!")
                led_value = "ON"
                GPIO.output(led_pin, GPIO.HIGH)
            else:
                print("[LED-MANUAL] Attempting to turn LED off!")
                led_value = "OFF"
                GPIO.output(led_pin, GPIO.LOW)

```

Handling POST requests to change AUTO/MANUAL as well as toggle LED requests

```
# high values are low light, low values are high light
LI_LOWER_LIMIT = 100      # when BELOW this value, turn the light off (when automatic is enabled)
LI_UPPER_LIMIT = 10000    # when ABOVE this value, turn the light on (when automatic is enabled)

# USED TO CHECK AUTOMATIC BEHAVIOUR TO THRESHOLD VALUES
SIMULATE = False
SIMULATE_MDOE = "DARK"
# SIMULATE_MDOE = "LIGHT"
if SIMULATE_MDOE == "DARK":
    SIM_VALUE = 11000
elif SIMULATE_MDOE == "LIGHT":
    SIM_VALUE = 50
```

Declaration of example threshold values for low/high light and simulation code for faulty sensor

```
if mode == "AUTOMATIC":
    if latest_intensity > LI_UPPER_LIMIT:  # SENSOR IS READING DARK - TURN ON LED
        led_state = not led_state
        print("[LED-AUTO] Attempting to turn LED on!")
        led_value = "ON"
        GPIO.output(led_pin, GPIO.HIGH)
    elif latest_intensity < LI_LOWER_LIMIT:  # SENSOR IS READING LIGHT - TURN OFF LED
        led_state = not led_state
        print("[LED-AUTO] Attempting to turn LED off!")
        led_value = "OFF"
        GPIO.output(led_pin, GPIO.LOW)
```

Code for AUTO handling sensor readings outside of thresholds and actuating LED accordingly