herotech team, Romania
herotech.ro

Title: Smart House Lighting

Students Names: Daniel Marpozan, Codrin Muntean

School: National College “Octavian Goga”

City and Country: Sibiu, Romania

Category: Science – Innovation

The project description paper

Abstract

Smart House Lighting is a system that counts the number of people in every room of a house, using the infrared barrier method. Based on the number of people counted in every room, the luminaries are controlled automatically. This system is meant to help people save energy and make a better life. For the implementation of the project, the team appealed on electronics, mechanics and IT knowledge.

Introduction

Our world is continuously changing and the natural resources are limited. Unfortunately, in some countries the resources used for creating electricity are non-renewable, so we must act now to change the future of our planet. By making this project, we hope we could contribute with a solution that not only facilitates our lives, but will also help reduce the consumption of electricity.

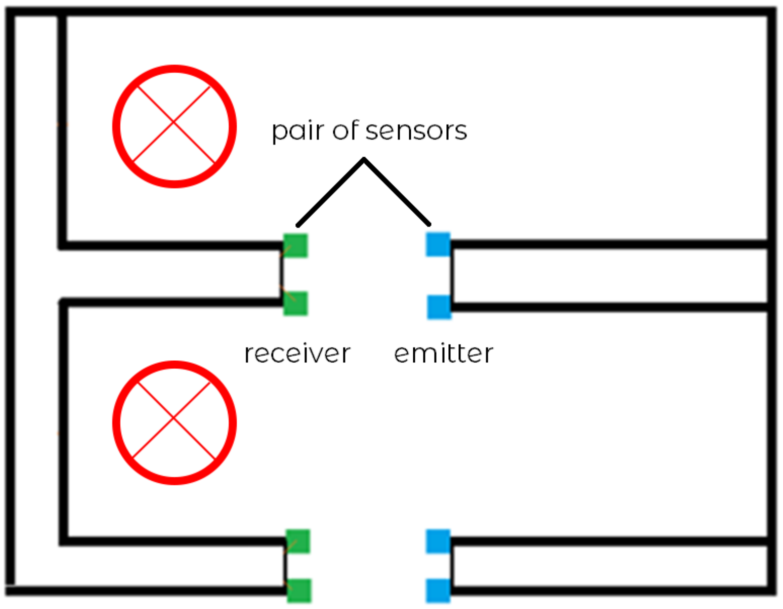
The goal of the project

The main aim of our application is to reduce energy consumption in a house by automatically switching on and off the lights. Another goal is to make our houses more comfortable. By doing this, we will be able to decrease pollution and usage of non-renewable resources. Needless to say, that all of these could contribute to a better life on Terra.

herotech team, Romania
herotech.ro

Description of the project

The system is composed of 3 modules: an emitter, a receiver and a controller module. An emitter and a receiver represent a pair, which is placed on the doorframe, so that our device is able to detect when the IR ray is interrupted. Every door requires two of these pairs to detect the way the person came from.



The emitter sends IR signal to the receiver module. The receiver detects when someone gets in or out of a room because the IR ray is blocked by that person. Therefore, based on the number of people in a room, the lights will be turned on or off automatically. Also, we make a cross – platform application that allow us to control every single light in case of the user does not want to use the automatic mode.

The team

Our team, *herotech*, is composed of three high school students (16 years old). After the school has begun, we decided to participate to *GENIUS Olympiad* with two projects, one in the *Robotics* category and this one in the *Science-Innovation* category.

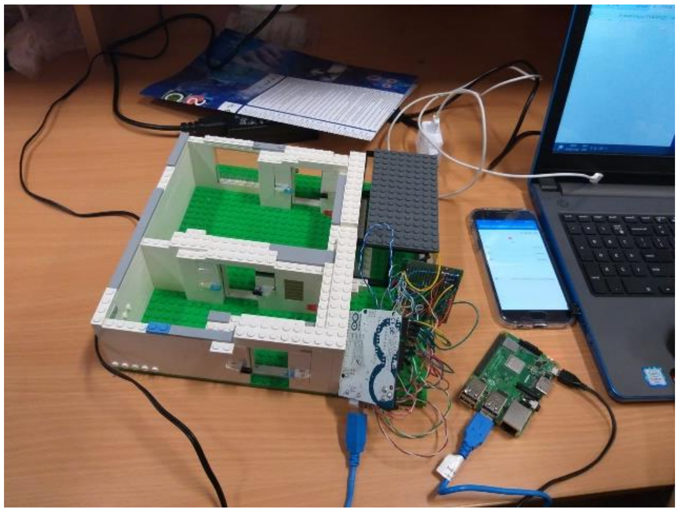
Each member was in charge of a specific part of the project as follows, but only the first two will represent the team:

- **Daniel Marpozan** – responsible for the electronic part.
- **Codrin Muntean** – responsible for the programming part.
- **Alexandru Radac** – responsible for the design of the project and the idea behind it.

herotech team, Romania
herotech.ro

The idea behind our system

The need for a smart energy-saver system led us to the idea to make these sensors. Firstly, we made a scale model of a house and sensors were implemented in it. The achieved result did not satisfy us, because there were a lot of wires which connected all sensors to the controller part, an Arduino board. To resolve this problem, we integrated WI-FI modules (ESP8266) in every sensor board, which are connected to a central server.



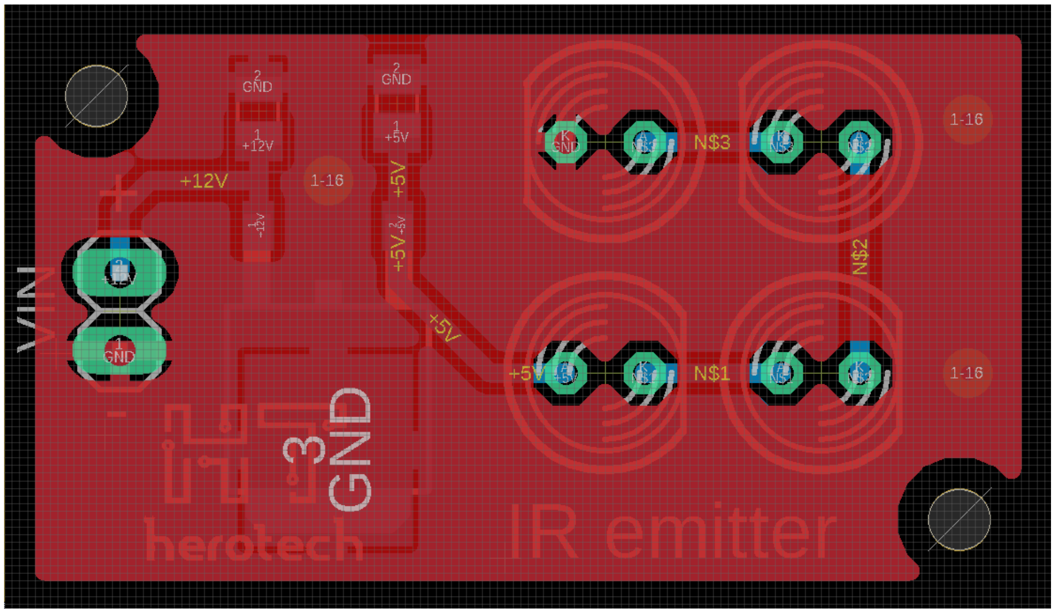
This was our first try: sensors integrated in a LEGO model

Methods and experiments

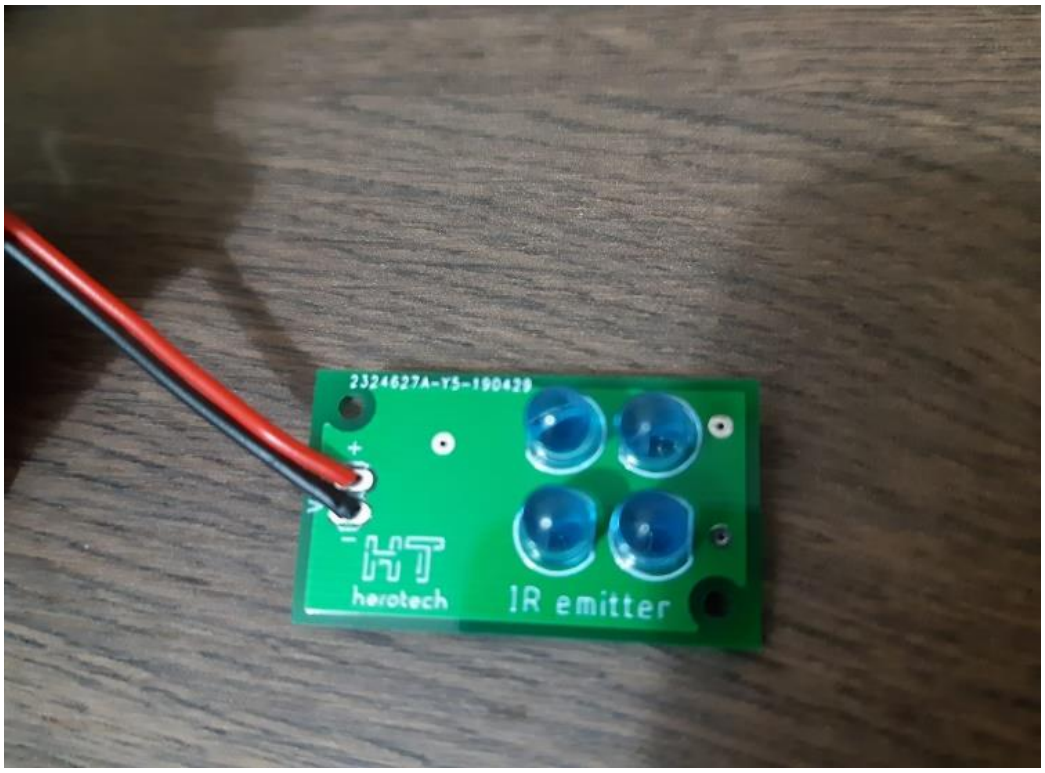
1. The hardware part

Firstly, we designed the electronics parts, using the Autodesk Eagle software.

- **The emitter module**
  It is a simple electronic circuit composed of four IR emitter diodes, connected in series. We used four IR diodes to relieve the calibration process (between emitter and receptor). This module is supplied at a voltage greater than 5V and it broadcasts IR ray continuously. We integrated a 5V regulator, because every led has the forward voltage at 1.25 V. Also, we used two 100 nF capacitors in the event of a current fluctuation.

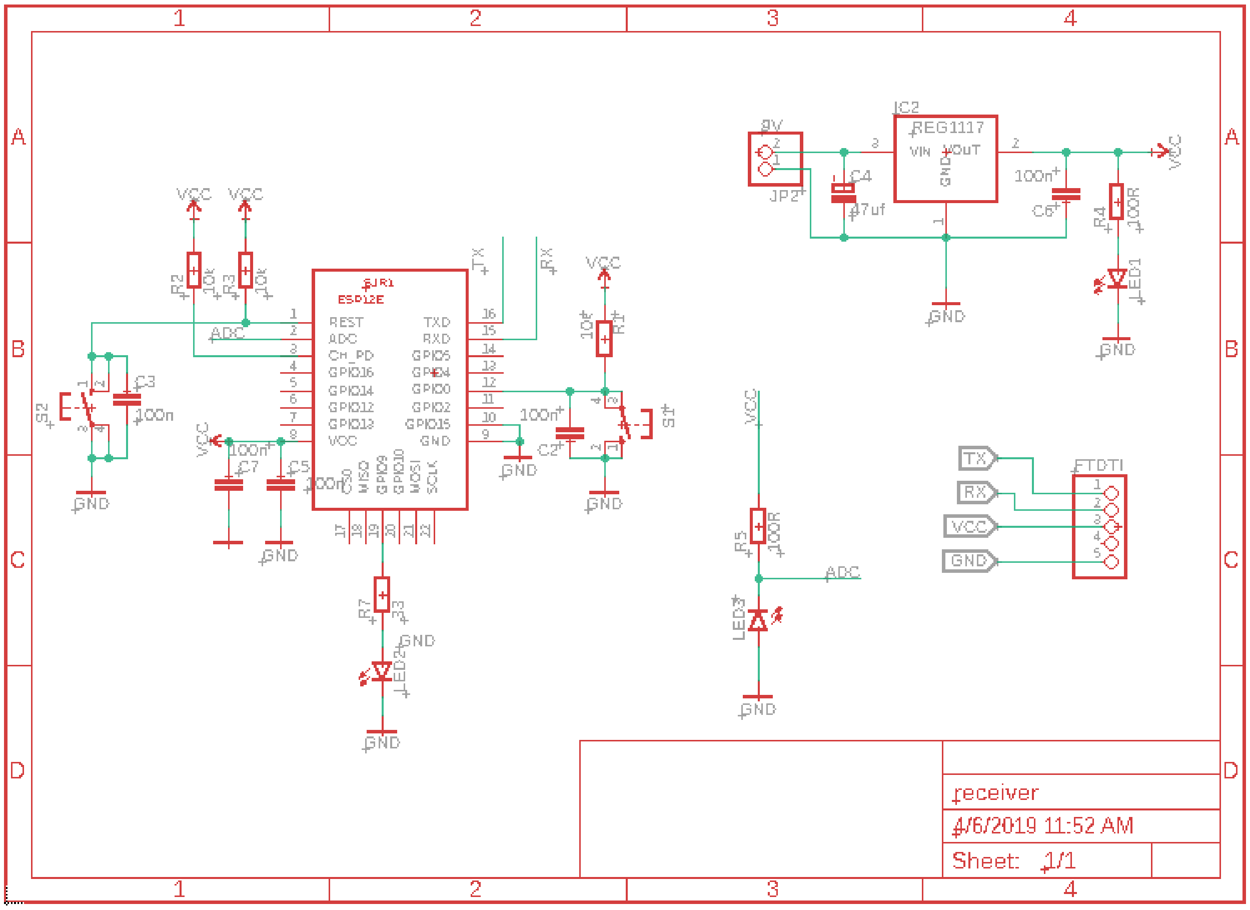


Layout of the emitter module



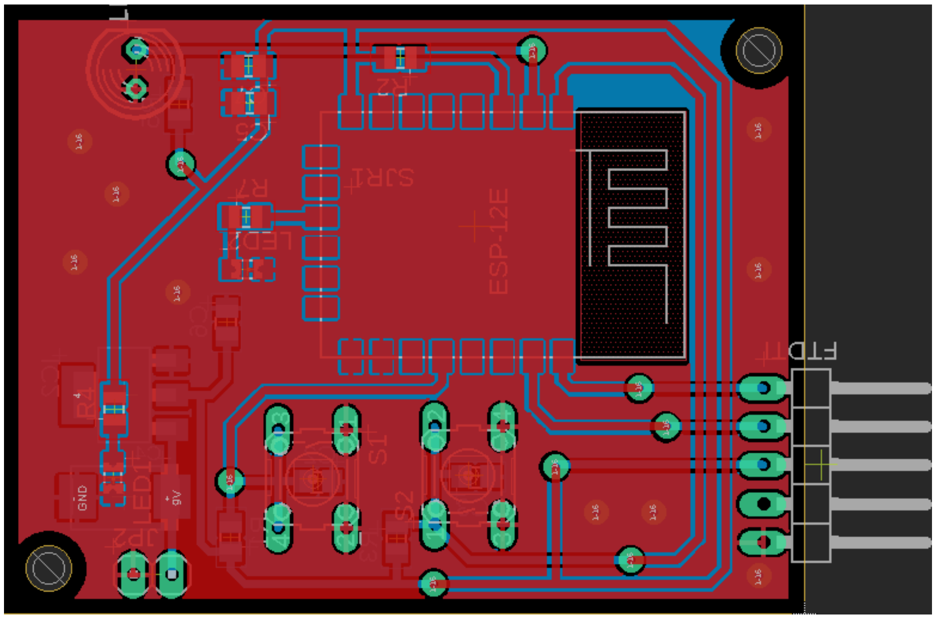
The printed circuit board

- **The receiver module**
  It is designed around the ESP8266 chip, that sends information based on data read on the ADC pin. The IR led receptor acts like a resistor whose value is changing depending on the received light. Based on it, we made a voltage divider with this led and a resistor.

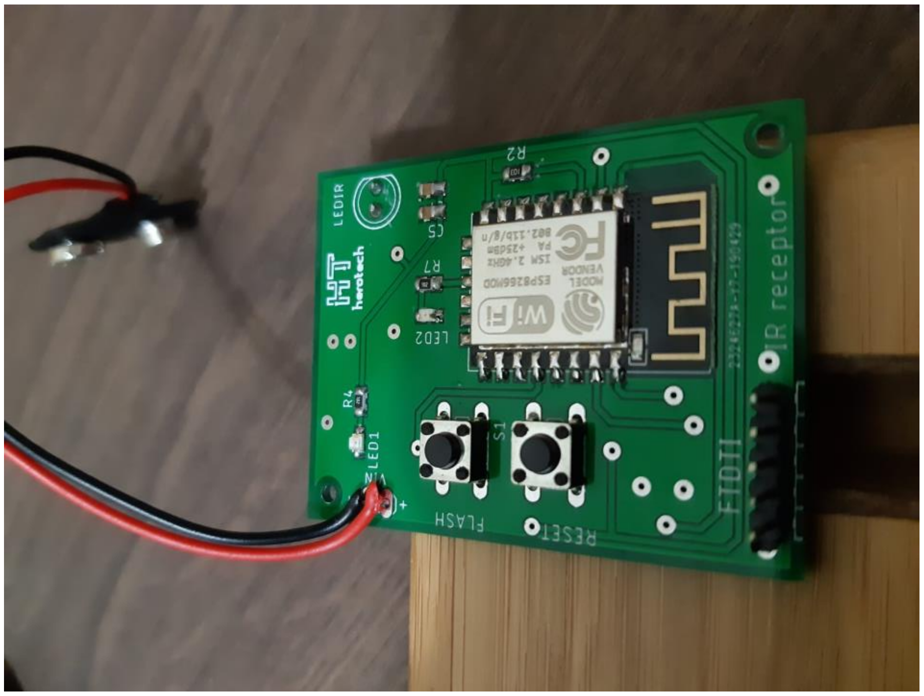


The schematic of the receiver part

This module is powered by a minimum 3.3 V power supply and has a 1117 voltage regulator. The ESP module (ESP 12F) is programmed via RX and TX pins (serial communication), using a FTDI programmer. To flash the chip, is required to use *FLASH* and *RESET* push buttons.

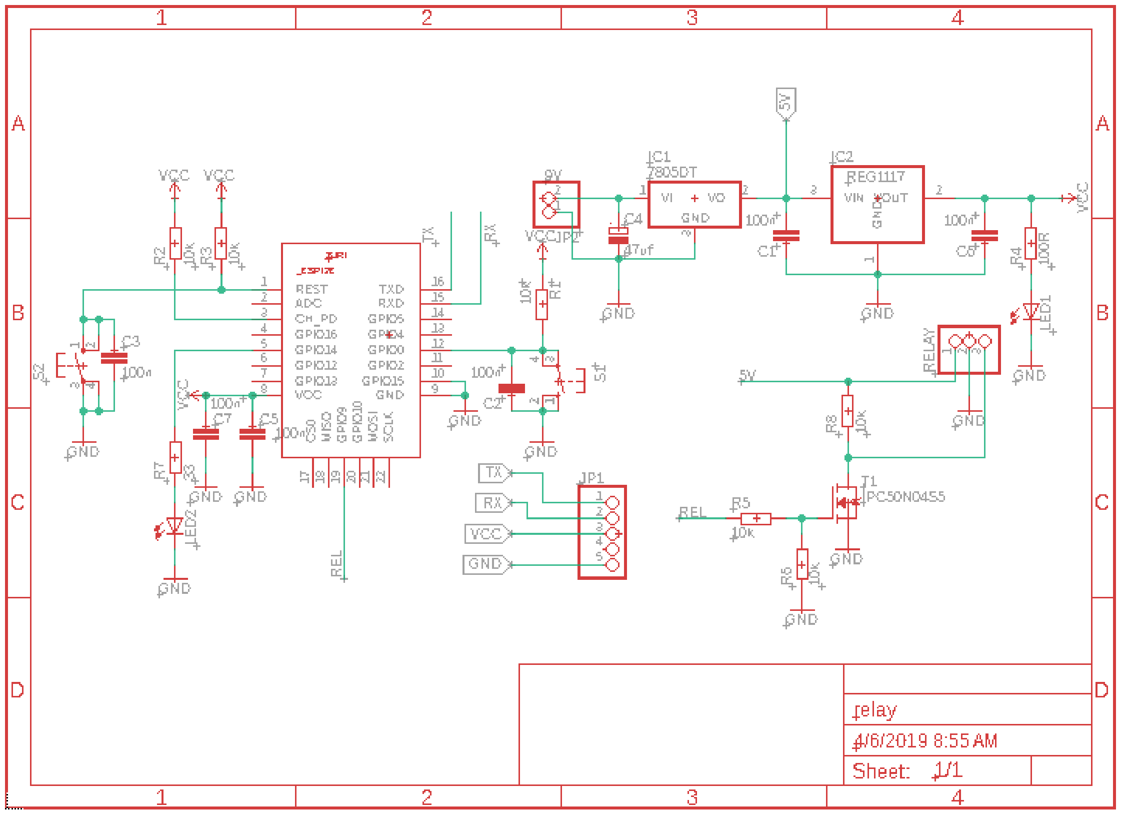


The layout of the receiver module

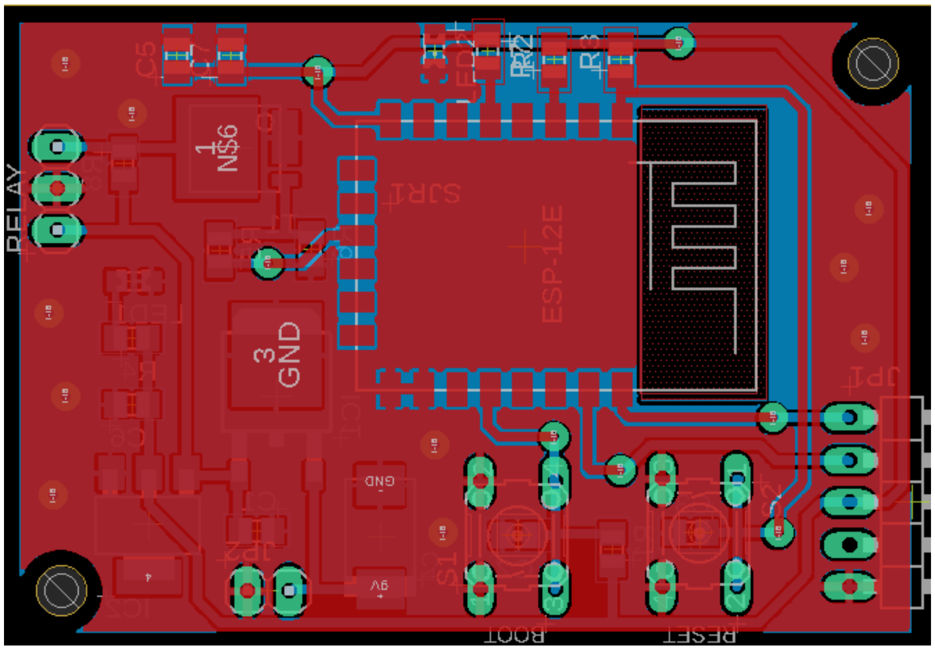


The PCB for this part

- **The relay controller**
  This last module switches on or off the lights, based on some information received from the server. This module is controlled by the ESP8266 microcontroller and is programmed via *TX* and *RX* pins, using a FTDI programmer. This circuit requires a minimum 5V power, because this is the specific voltage for the relay. To drop the voltage, we integrated 5V and 3.3V regulators. The relay is connected to the controller using 3 pins: *GND*, *5V* and the *signal* pin. The signal pin is the drain of a MOSFET transistor which acts like a switch. The gate receives a logic signal and the source is connected to GND, while the drain is connected to the 5V signal, using a pull-up resistor. The best way to mount this module, is to incorporate it in the luminaries.



The schematic designed in Autodesk Eagle



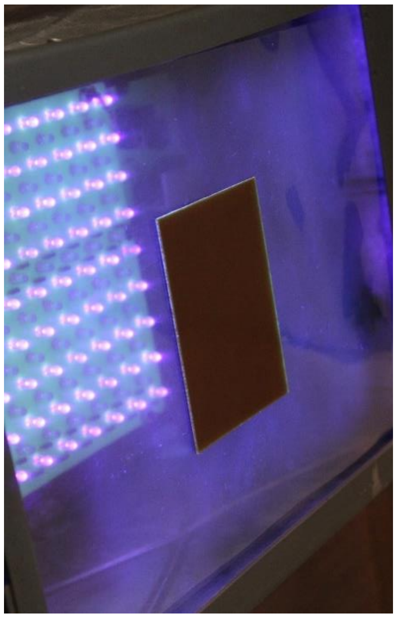
Layout of the relay controller

To power these sensors, we used a 9V Lithium-Ion battery which has a bigger yield. The current consumption is a disadvantage, because with a 600 mAh capacity, the batteries will last approximative 6 hours, in case of continuously working. For the competition, we will use a plugged power supply, to ensure that we will not run out of electricity. In the future, we want to implement the sleep mode to reduce the electricity consumption up to 90%. The best solution regarding the high energy consumption is to power the modules from a linear power supply (in this case, the user has to bring a minimum 5V electric lines at every doorframe). We programmed the ESP8266 microcontrollers in Arduino IDE, using C++ language.

After we had designed these, we made a simulation, using Proteus software, to be sure that all will work properly. By doing this, we realized that we had to change some resistor values and other electronics parts in the schematic.

The next step was the actual PCB realization, using UV exposure. Firstly, we exposed the PCB with a photoresist layer for 4 minutes and developed it with *NaOH*. Then, we inserted the board in a *$FeCl_3$* solution.

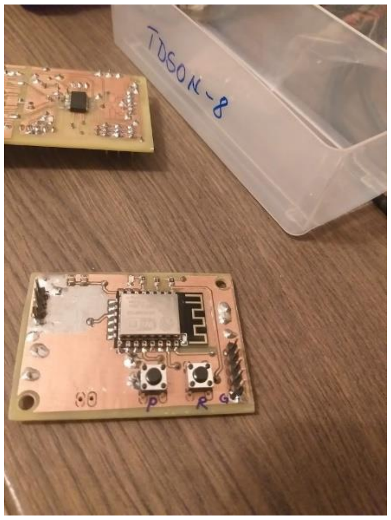
After making the drills for via-s and *THT* components (through-hole technology), we solder all *THT* and *SMD* components. Finally, we uploaded the code on ESP chips and we tested its functionality.


UV exposure

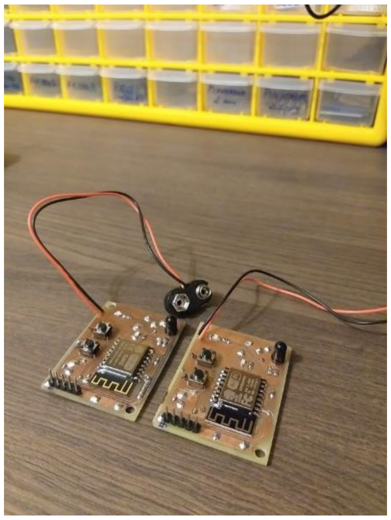
the PCB in NaOH solution

Daniel soldering the components

PCB results:

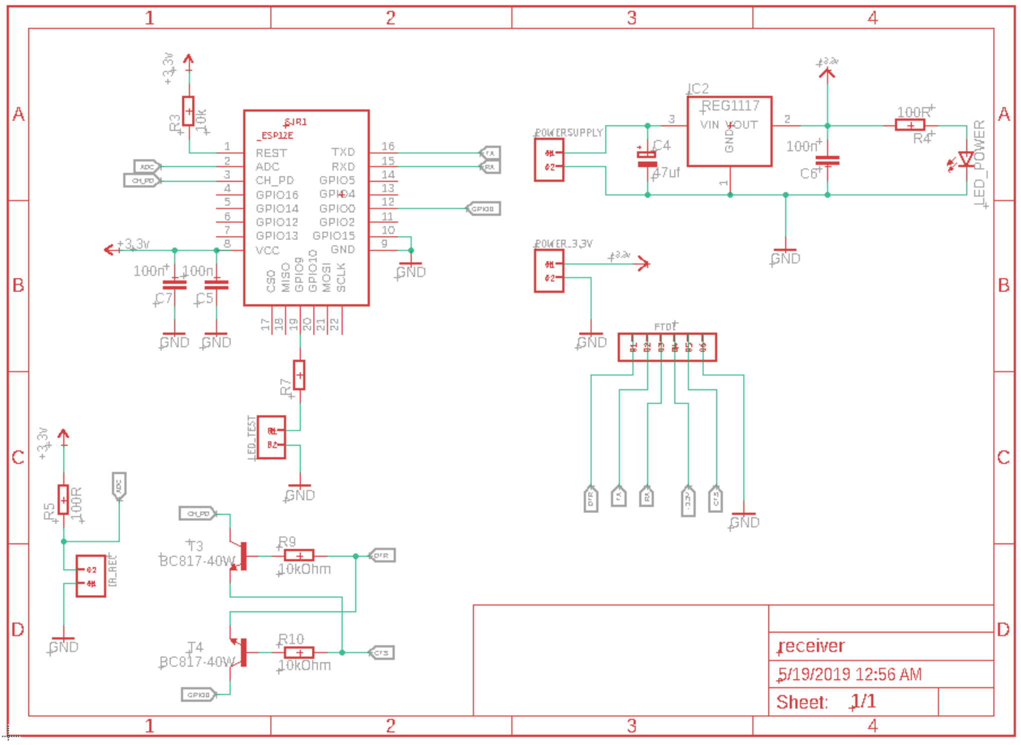


The relay controller


The receiver module

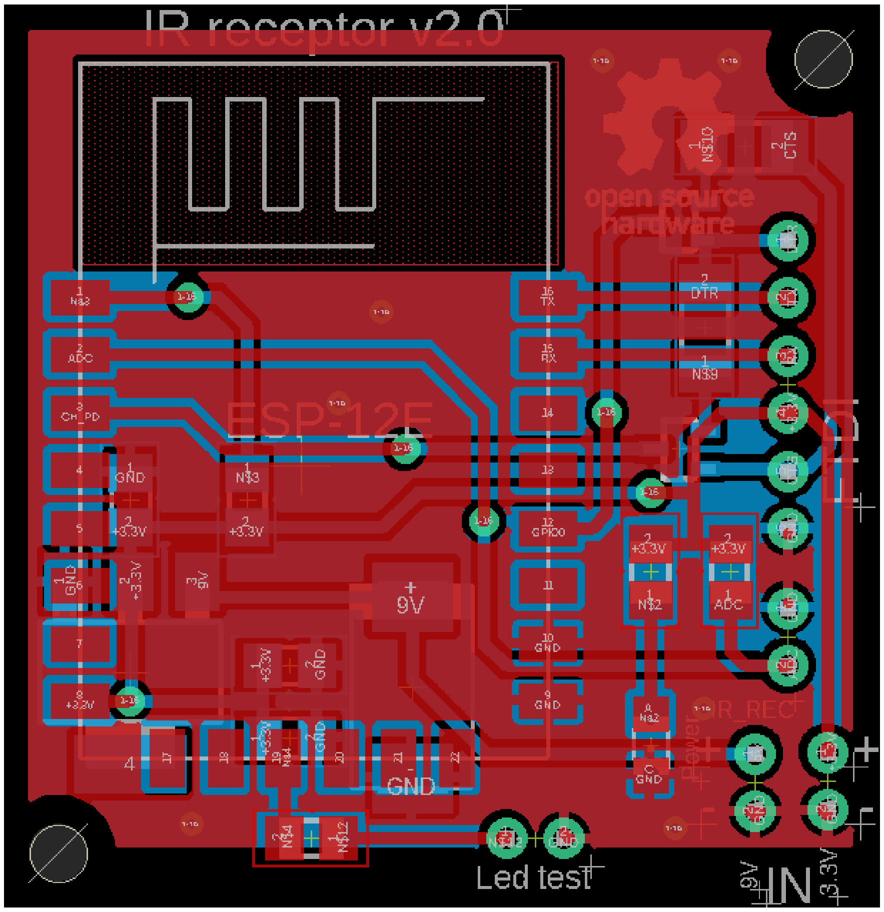
To implement this system in a house, we should make many circuit boards, which is not very easy, given our tools, therefore we paid a specialized company to do them for us. As a result, we got high quality PCB-s, with high resolution traces.

After we have made the circuits and we have tested them, we decided to change a part of the schematic, so we designed new boards, without push buttons. We replaced them with a circuit based on two transistors, controlled by the *DTR* and *CTS* pins from the FTDI programmer.



The new schematic design for the receiver module

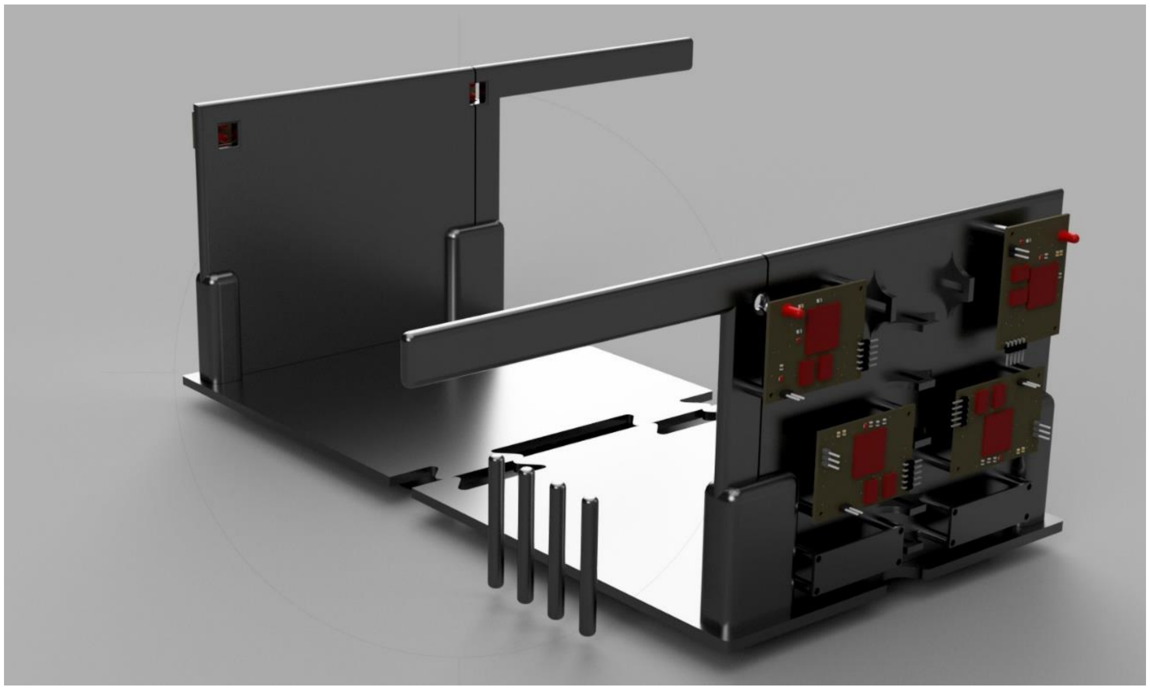
Also, we improved a lot the layout part, making this as small as we could, in the idea of having a smaller case for every module. We are planning to manufacture the new boards in the summer holiday.



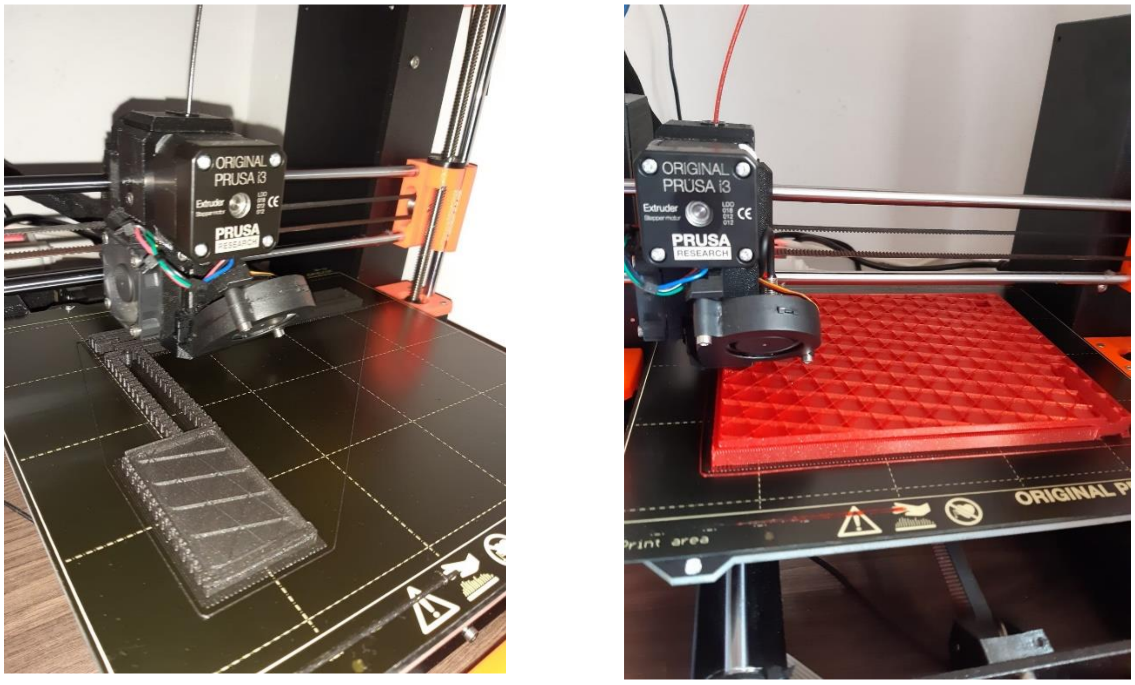
The new layout design for the receiver module

2. The design part

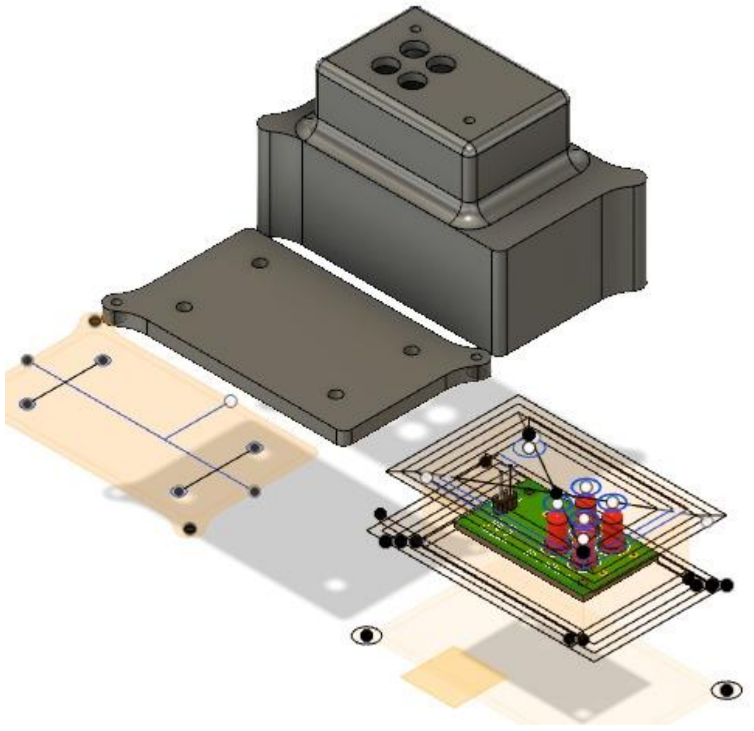
Using Autodesk Fusion360 software, we designed some cases for our sensors. We exported our PCB-s layout from EAGLE to Fusion360 and we integrated them in simple boxes which we then printed, using 3D technology. For this competition, we created a support to simulate a doorframe. The mounting support was designed to be as modular as possible, allowing us to transport it easily.

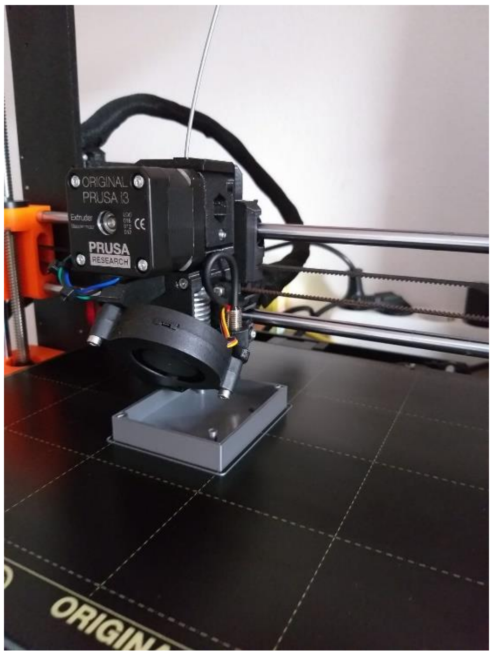


The 3D design of a mounting support for sensors



3D printing of the mounting support


The 3D design of the case for the emitter module



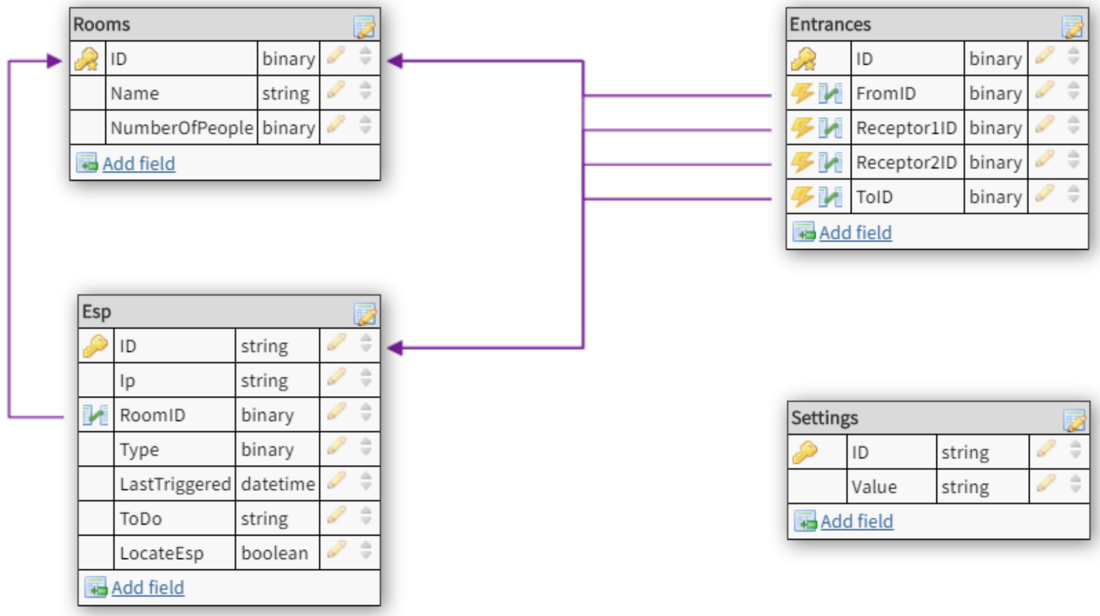
The 3D printing process and the final results

3. The programming part

- Server code:

The server is an ASP.NET CORE application which runs on a machine connected to the local network. The machine is meant to be assigned a local domain or to have a static IP, so the ESP can hardcode that value.

The application is connected to a local DB using the Entity Framework CORE. We have 4 tables: *Esp*, *Rooms*, *Entrances* and *Settings* which are displayed in the following schematic.



Rooms

ID	binary
Name	string
NumberOfPeople	binary

Entrances

ID	binary
FromID	binary
Receptor1ID	binary
Receptor2ID	binary
ToID	binary

Esp

ID	string
Ip	string
RoomID	binary
Type	binary
LastTriggered	datetime
ToDo	string
LocateEsp	boolean

Settings

ID	string
Value	string

Database schema

The API exposes some controllers such as: *Connections*, *Esp* and *Rooms*. The Esp controller is used for server-module communication while the others for server-mobile application.

- The ESP code:

This code runs on our modules and it differs a little bit between the controller and the receiver. When the module is powered on, it connects to a local network with the hardcoded SSID and password.

After the connection is established it checks for a locally saved GUID. If it finds one it makes a PUT request on /api/Esp/UpdateIp for updating the ip address stored in the server's database. But if there isn't one, it makes a POST request on /api/Esp/Register that will both register the module in the database and will return a GUID that will be saved locally.

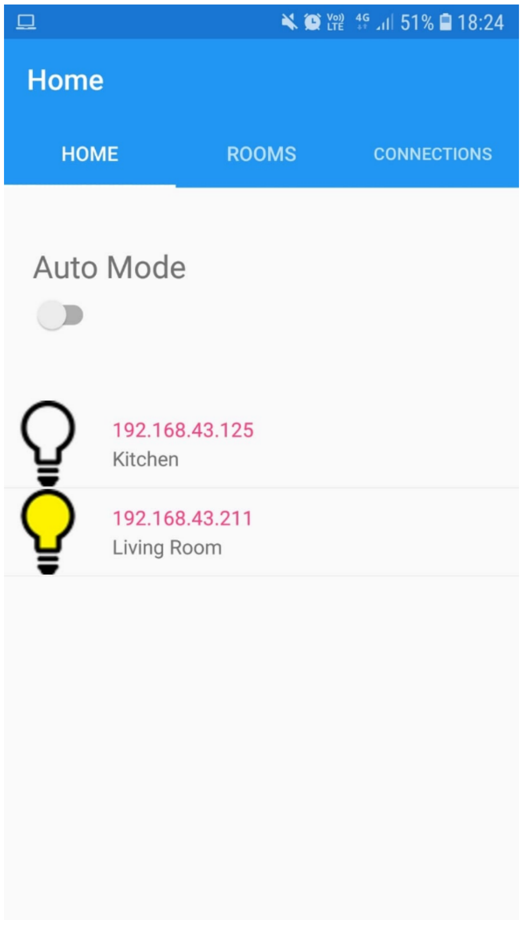
In the loop function, the controller makes a GET request on /api/Esp/ that will return the state of the light, while the receiver makes a POST request on /api/Esp/ReceptorTriggered when the IR ray is interrupted.

- Application code:

We made a cross-platform application using the Xamarin.Forms Framework that uses tabs to navigate between 3 child pages.

1. The home page

The home page has a switch for setting the mode of our system: *automatic* or *manual*. When the manual mode is selected, the application displays a list of every controller module that is connected to the server, and it lets the user select which lights should be on.


Home

Auto Mode

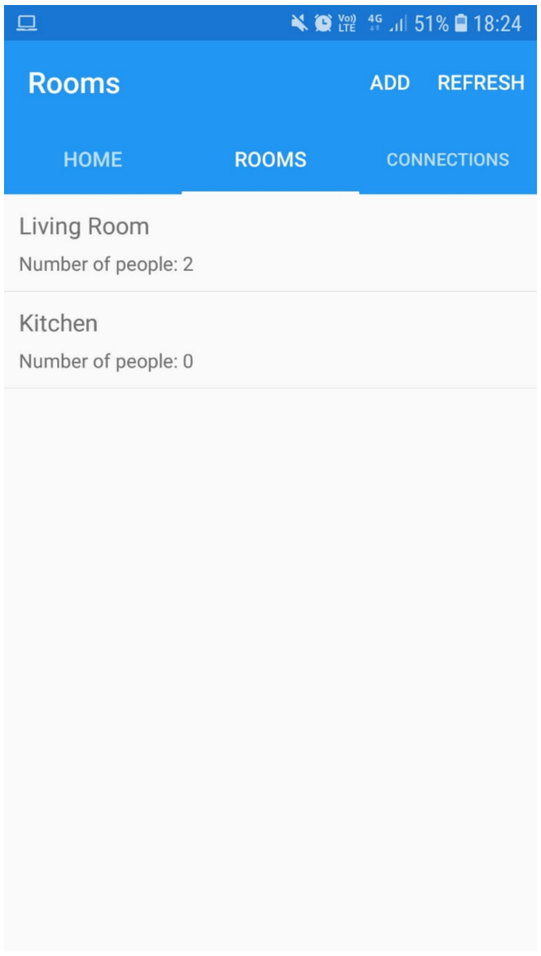
192.168.43.125
Kitchen

192.168.43.211
Living Room

The home page

2. The rooms page

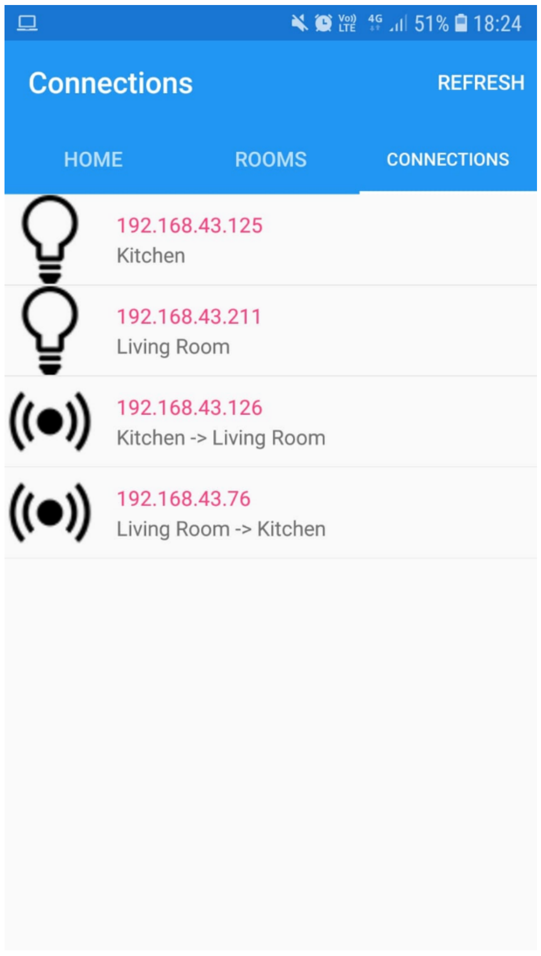
The rooms page displays a list of the rooms from the database, each showing the current number of people. We can also create a new room by clicking a button from the toolbar.



Living Room
Number of people: 2

Kitchen
Number of people: 0

The rooms page



192.168.43.125
Kitchen

192.168.43.211
Living Room

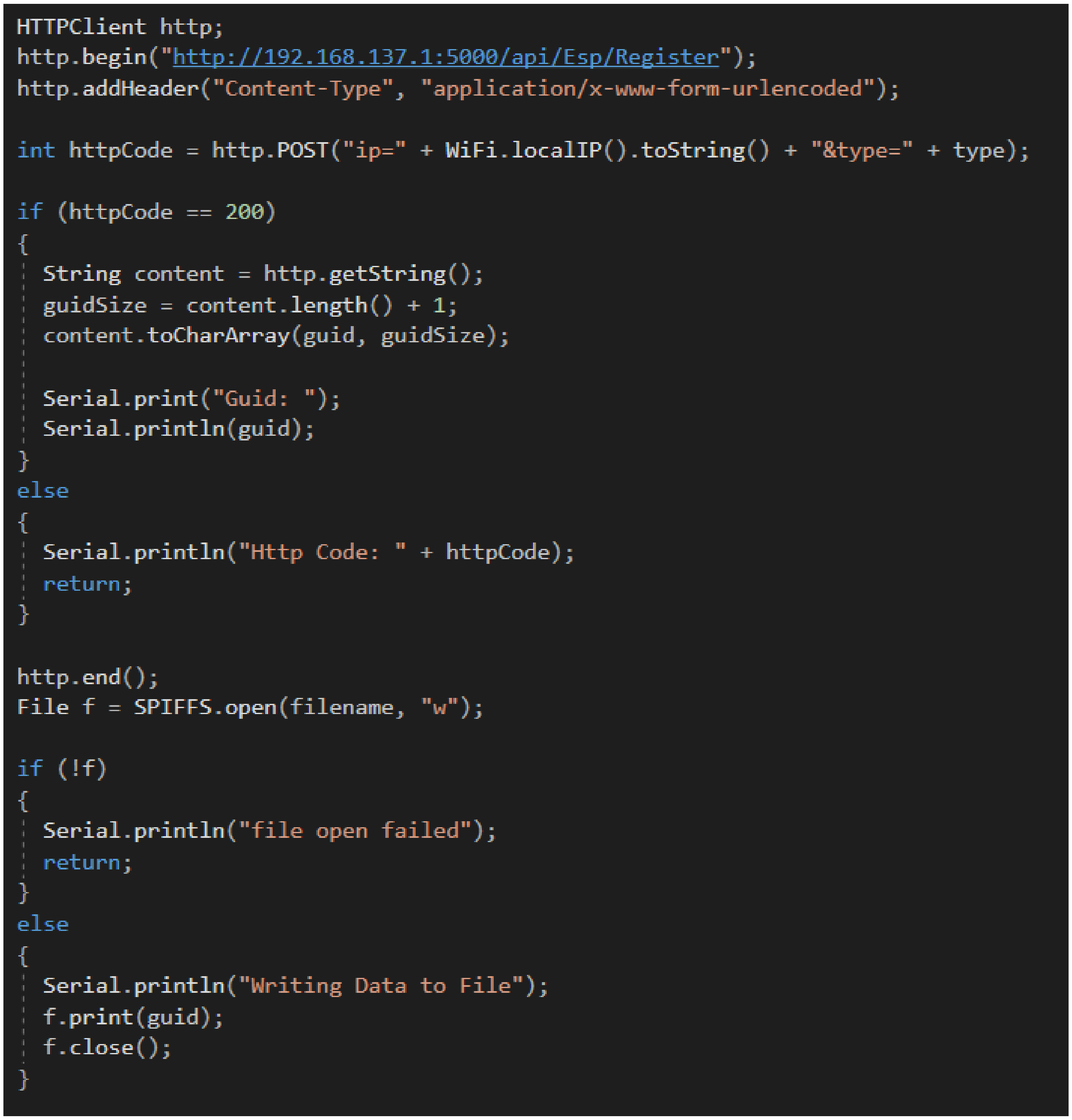
192.168.43.126
Kitchen -> Living Room

192.168.43.76
Living Room -> Kitchen

The connections page

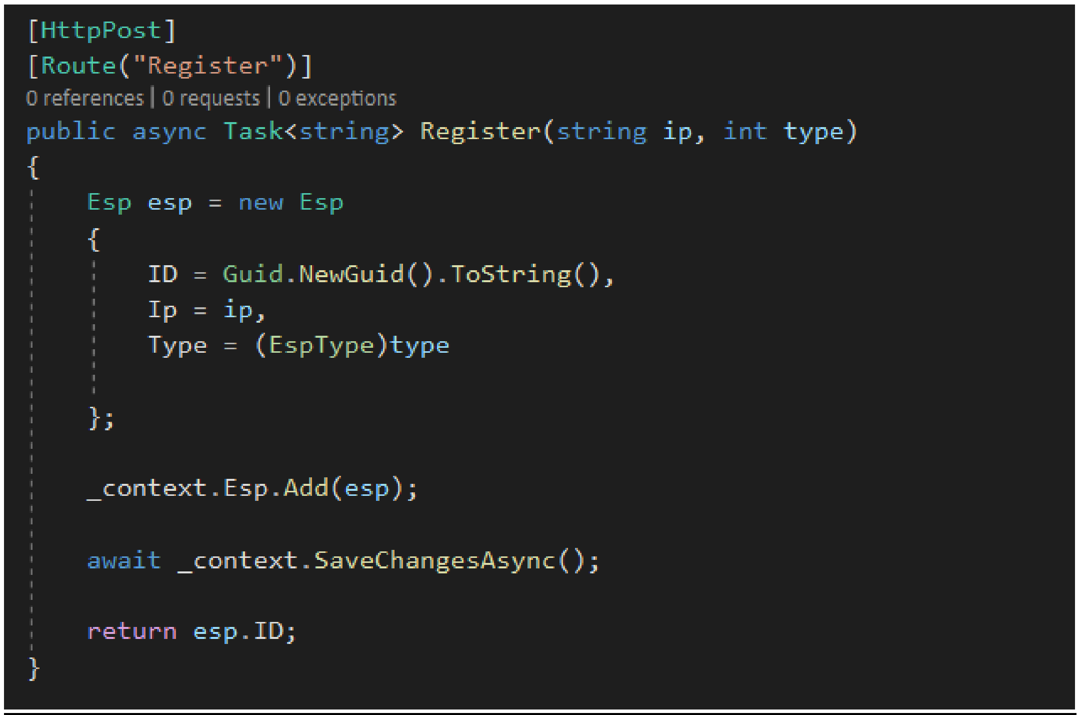
3. The connections page

The connections page consists of a list of all the modules connected to the server. A light bulb icon is used for controllers and a sensor icon for the receivers. By tapping an item on the list, we can configure the location of that module. If we don't know its location, we can press a button that will make a led on our module blink for a couple of seconds.



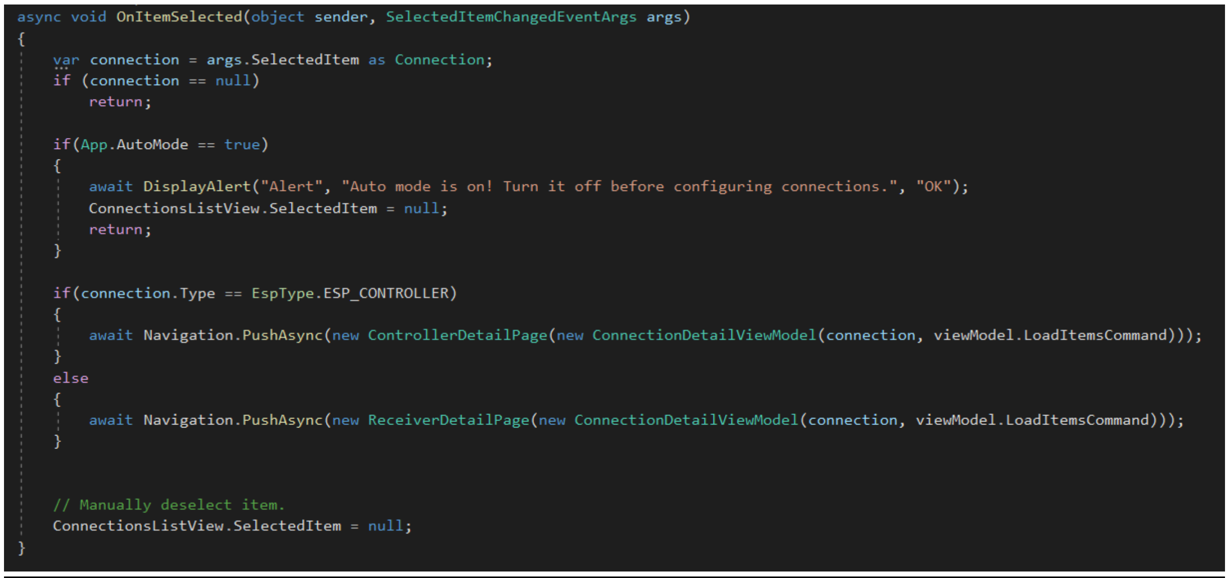
```
HTTPClient http;
http.begin("http://192.168.137.1:5000/api/Esp/Register");
http.addHeader("Content-Type", "application/x-www-form-urlencoded");

int httpCode = http.POST("ip=" + WiFi.localIP().toString() + "&type=" + type);

if (httpCode == 200)
{
  String content = http.getString();
  guidSize = content.length() + 1;
  content.toCharArray(guid, guidSize);

  Serial.print("Guid: ");
  Serial.println(guid);
}
else
{
  Serial.println("Http Code: " + httpCode);
  return;
}

http.end();
File f = SPIFFS.open(filename, "w");

if (!f)
{
  Serial.println("file open failed");
  return;
}
else
{
  Serial.println("Writing Data to File");
  f.print(guid);
  f.close();
}
```

Esp: Making a request to register the module and then saving the GUID locally



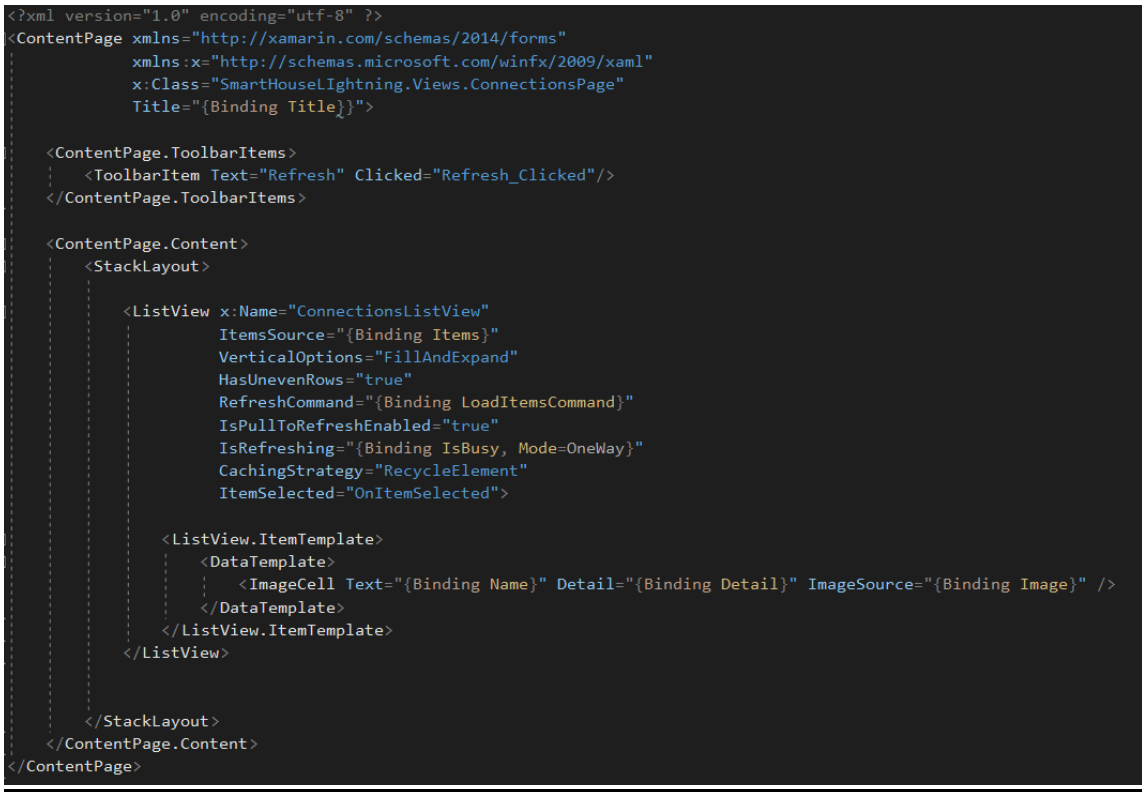
```
[HttpPost]
[Route("Register")]
0 references | 0 requests | 0 exceptions
public async Task<string> Register(string ip, int type)
{
    Esp esp = new Esp
    {
        ID = Guid.NewGuid().ToString(),
        Ip = ip,
        Type = (EspType)type

    };

    _context.Esp.Add(esp);

    await _context.SaveChangesAsync();

    return esp.ID;
}
```

Server: The Register action from the Esp controller



```
async void OnItemSelected(object sender, SelectedItemChangedEventArgs args)
{
    var connection = args.SelectedItem as Connection;
    if (connection == null)
        return;

    if(App.AutoMode == true)
    {
        await DisplayAlert("Alert", "Auto mode is on! Turn it off before configuring connections.", "OK");
        ConnectionsListView.SelectedItem = null;
        return;
    }

    if(connection.Type == EspType.ESP_CONTROLLER)
    {
        await Navigation.PushAsync(new ControllerDetailPage(new ConnectionDetailViewModel(connection, viewModel.LoadItemsCommand)));
    }
    else
    {
        await Navigation.PushAsync(new ReceiverDetailPage(new ConnectionDetailViewModel(connection, viewModel.LoadItemsCommand)));
    }


    // Manually deselect item.
    ConnectionsListView.SelectedItem = null;
}
```

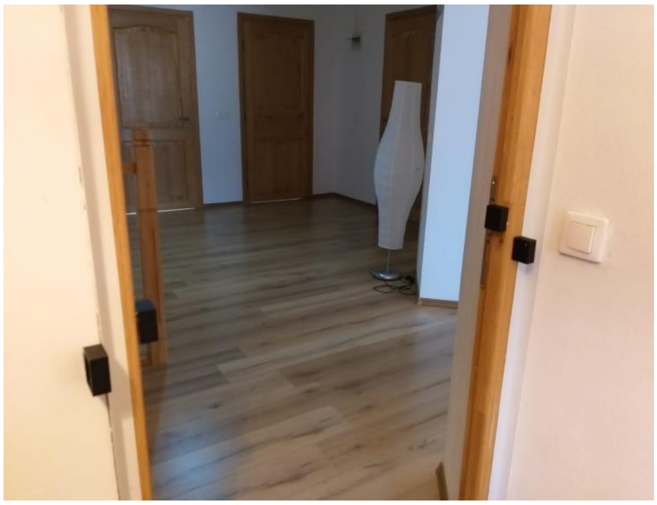
Application: OnSelectedItem event from the connections page

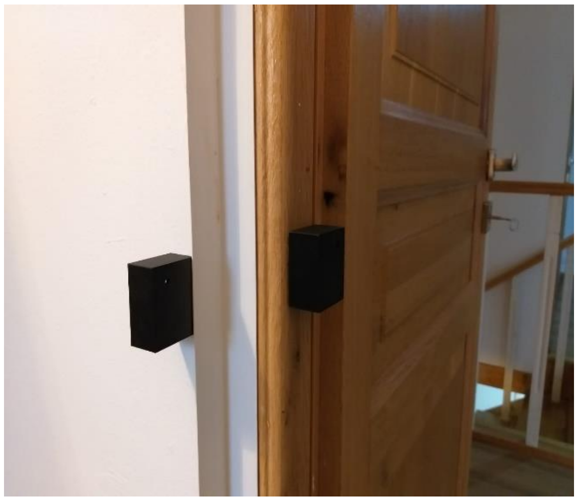


```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="SmartHouseLIghtning.Views.ConnectionsPage"
             Title="{Binding Title}">

    <ContentPage.ToolbarItems>
        <ToolbarItem Text="Refresh" Clicked="Refresh_Clicked"/>
    </ContentPage.ToolbarItems>

    <ContentPage.Content>
        <StackLayout>

            <ListView x:Name="ConnectionsListView"
                      ItemsSource="{Binding Items}"
                      VerticalOptions="FillAndExpand"
                      HasUnevenRows="true"
                      RefreshCommand="{Binding LoadItemsCommand}"
                      IsPullToRefreshEnabled="true"
                      IsRefreshing="{Binding IsBusy, Mode=OneWay}"
                      CachingStrategy="RecycleElement"
                      ItemSelected="OnItemSelected">

                <ListView.ItemTemplate>
                    <DataTemplate>
                        <ImageCell Text="{Binding Name}" Detail="{Binding Detail}" ImageSource="{Binding Image}" />
                    </DataTemplate>
                </ListView.ItemTemplate>
            </ListView>


        </StackLayout>
    </ContentPage.Content>
</ContentPage>
```

Application: User interface definition for the connections page using XAML

Results:





Our sensors on the door frame

Conclusion

In conclusion, we hope our project is useful and that it will have a positive effect on our lives. By making this system, we want to digitalize our houses and make them more comfortable. Besides, a very important aim is having a positive impact on the environment, by diminishing energy consumption. In the future, our team will improve this system, by making it easier to use, with a more attractive interface. We already have a lot in mind, like changing the communication protocol to one that is better suited for IoT devices resulting in reduced electricity consumption.

References

- http://www.esp8266learning.com/
- https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-2.1
- http://www.university.xamarin.com/