UNIVERSIDADE AUTÓNOMA DE LISBOA

LUÍS DE CAMÕES

DEPARTAMENTO DE ENGENHARIAS E CIÊNCIAS DA COMPUTAÇÃO

LICENCIATURAS ENGENHARIA INFORMÁTICA E ENGENHARIA DE INFORMÁTICA DE GESTÃO

Sistema de Iluminação Inteligente com Arduíno

Relatório de atividade de laboratório de projeto para a obtenção dos graus de licenciaturas em engenharia informática e informática de gestão.

Autores: Daniel Silvestre, Hélder Oliveira, Luís Finuras, Rúben Rebelo

Orientador: Professor Hector Orrillo

Número dos candidatos: 30003099, 30006391, 30006396, 30006431

Junho de 2023

Lisboa

Resumo

O trabalho propõe um sistema de iluminação inteligente baseado em Arduíno. O objetivo é ajustar a intensidade da iluminação artificial com base na luz natural medida por um *photoresistor*, detetar altas temperaturas e humidade por meio de sensores e enviar sinais de alerta. O sistema é controlado remotamente por meio de uma aplicação móvel com conexão Bluetooth.

A arquitetura do protótipo consiste em diversos componentes integrados ao Arduíno. Um sensor de luminosidade fornece informações sobre a iluminação natural ao Arduíno, que automaticamente acende ou apaga as luzes com base num limite predefinido. Um sensor de humidade e temperatura emite sinais luminosos de alerta quando a temperatura é alta. Um *display* LCD que exibe informações sobre a divisão da casa que está ligada, bem como dados de humidade e temperatura. Um módulo Bluetooth que permite a conexão entre o telemóvel e o Arduíno. Uma aplicação desenvolvida em Android Studio permite o controlo total do sistema e para o controlo do Arduíno, é utilizado um módulo Bluetooth HC-05. O trabalho descreve a criação de uma maquete para o sistema de iluminação inteligente, ao utilizar uma impressora 3D para criar uma caixa retangular que abriga o Arduíno, a Breadboard, o display e os sensores. O sistema é construído na Breadboard para facilitar a montagem e demonstração das funcionalidades. A aplicação Android desenvolvida permite ligar e desligar as luzes das diferentes divisões da casa, ler os valores dos sensores de humidade e temperatura e configurar automatismos, como acender as luzes no fim do dia e apagá-las no início do dia. O sistema também permite a configuração de limites de temperatura, emitindo um alarme sonoro de alerta quando o limite é ultrapassado.

Os testes e análise de resultados foram realizados após a implementação dos componentes, ao incluir outros sensores e a configuração de alarmes e limites de temperatura. Alguns ajustes foram necessários para resolver falhas identificadas durante os testes. No geral, o trabalho propõe um sistema de iluminação inteligente controlado por Arduíno e uma aplicação móvel, que oferece funcionalidades como controlo remoto, monitorização de sensores e automatismos.

Palavras chave:

Arduino, Bluetooth, Breadboard, HC-05, conexão e aplicação móvel.

Abstract

The work proposes an intelligent lighting system based on Arduino. The objective is to adjust the intensity of artificial lighting based on natural light measured by a photoresistor, detect high temperatures and humidity using sensors and send alert signals. The system is remotely controlled via a mobile application with a Bluetooth connection.

The prototype architecture consists of several components integrated into the Arduino. A brightness sensor provides information about natural lighting to the Arduino, which automatically turns the lights on or off based on a predefined threshold. A humidity and temperature sensor emits light warning signals when the temperature is high. An LCD display that displays information about the room in the house that is turned on, as well as humidity and temperature data. A Bluetooth module that allows the connection between the mobile phone and the Arduino. An application developed in Android Studio allows total control of the system and for controlling the Arduino, an HC-05 Bluetooth module is used. The work describes the creation of a model for the intelligent lighting system, using a 3D printer to create a rectangular box that houses the Arduino, the Breadboard, the display and the sensors. The system is built on Breadboard to facilitate assembly and demonstration of features. The developed Android application allows you to turn on and off the lights in the different rooms of the house, read the values of the humidity and temperature sensors and configure automatisms, such as turning on the lights at the end of the day and turning them off at the beginning of the day. The system also allows the configuration of temperature limits, emitting an alert sound alarm when the limit is exceeded.

The tests and analysis of results were carried out after the implementation of the components, by including other sensors and the configuration of alarms and temperature limits. Some adjustments were necessary to solve failures identified during the tests. In general, the work proposes an intelligent lighting system controlled by Arduino and a mobile application, which offers features such as remote control, sensor monitoring and automation.

Keywords:

Arduino, Bluetooth, Breadboard, HC-05, Connection and mobile Application.

Índice

Lista de figuras

Lista de siglas e acrónimo

IoT	Internet of things
BLE	Bluetooth Low Energy
GPIO	General Purpose Input/Output
I2C	Inter-Integrated Circuit
IDE	Integrated Development Environment
LCD	Bluetooth low energy
LDR	Light Dependent Resistor
LED	Light-emitting diode
MCU	multipoint control unit
RX	Receive
SPI	Serial Peripheral Interface
TX	Transmit
UART	Universal asynchronous receiver/transmitter
USB	Universal Serial Bus

1 - Introdução

Nos últimos anos, o desenvolvimento tecnológico tem impulsionado a criação de soluções inteligentes que visam otimizar o consumo de energia e melhorar a eficiência. Um desses avanços é o Sistema de Iluminação Inteligente, que utiliza a integração de dispositivos eletrónicos e sensores para controlar a iluminação de forma mais eficiente. O objetivo principal de um sistema de iluminação inteligente é proporcionar iluminação adequada e personalizada, ao adaptar-se às necessidades dos ambientes e dos utilizadores. Ao contrário dos sistemas convencionais, que funcionam de maneira fixa e estática, um sistema inteligente é capaz de ajustar a intensidade da luz, horários de funcionamento, operar de forma automática à presença de pessoas, ter uma luminosidade ambiente ou até mesmo com o clima.

Nesse contexto, o Arduíno surge como uma plataforma de prototipagem eletrónico, que permite a criação de sistemas de iluminação inteligente. O Arduíno uno que irá ser utilizado é um microcontrolador que oferece uma variedade de recursos, como entradas e saídas digitais e analógicas.

Neste trabalho, exploraremos os princípios e benefícios de um sistema de iluminação inteligente baseado em Arduíno. Será discutido os componentes necessários, as etapas de configuração e programação, assim como os resultados esperados. Positivos e menos positivos. Além disso, abordaremos algumas aplicações práticas, como o controlo de iluminação numa residência, constituída apenas por quarto, sala de estar e sala de jantar. Irão ser destacado os impactos positivos que a adoção desses sistemas pode trazer, como a redução do consumo de energia e a criação de ambientes mais confortáveis e eficientes.

Em resumo, o Sistema de Iluminação Inteligente com Arduíno representa uma abordagem inovadora para o controlo da iluminação, ao promover assim para a sustentabilidade, a economia de energia e uma experiência aprimorada para os moradores. Ao explorar as possibilidades dessa tecnologia, estamos diante de uma solução que pode transformar a forma como interagimos com a iluminação no nosso dia-a-dia.

1.1 - Descrição do problema

Os sistemas de iluminação convencionais são estáticos e pouco flexíveis, então cada vez menos se adaptam às necessidades específicas dos utilizadores ou às condições do ambiente. Isso resulta num desperdício de energia e na falta de conforto. Portanto, há uma procura crescente por sistemas de iluminação inteligentes que sejam capazes de tornar automático ou mais simples o seu funcionamento. Como tal este trabalho visa desenvolver um sistema de iluminação inteligente capaz de resolver essas questões.

1.2 - Objetivos

1.2.1 - Objetivo geral

O objetivo geral deste trabalho é implementar um sistema inteligente de iluminação para uma *Smart Home*, com um Arduíno uno e a ligação com uma aplicação em sistema Android via Bluetooth. Esse sistema permitirá o controlo personalizado da iluminação, ao levar em consideração as preferências dos utilizadores e as condições do ambiente.

1.2.2 - Objetivos específicos

- Integrar um sensor de luminosidade para ajustar automaticamente a intensidade da luz com base na luz ambiente;
- Implementar um módulo Bluetooth para permitir o controlo do sistema através de uma aplicação móvel em android;
- Desenvolver uma aplicação móvel para controlar o sistema de iluminação, ao incluir funcionalidades como, ligar/desligar luzes e monitorizar a humidade e a temperatura;
- Incorporar um LCD para exibir mensagens de informação relevantes sobre o ambiente em que se encontra;
- Implementação de um *relay* para utilização de lâmpadas de 220V.

1.3 – Justificativa

A implementação de um sistema inteligente de iluminação para uma *Smart Home* apresenta benefícios significativos. Além de proporcionar maior conforto e conveniência aos utilizadores, esse sistema tem o potencial para reduzir o consumo de energia, ao mesmo tempo contribuir para a sustentabilidade ambiental. Ao permitir o controlo personalizado da iluminação por meio de uma aplicação móvel, os utilizadores podem ajustar a iluminação de acordo com as suas preferências individuais. A utilização do Arduíno como plataforma de desenvolvimento oferece flexibilidade e acessibilidade, ao permitir assim a criação de um sistema eficiente e de baixo custo. Além disso, a integração com uma aplicação móvel torna a interação com o sistema mais intuitiva. Espera-se que este trabalho contribua para a compreensão dos sistemas de iluminação inteligente baseados em IoT, ao demonstrar os benefícios e possibilidades práticas desta tecnologia.

2 - Fundamentação Teórica
2.1 - Internet das coisas e Sistema inteligentes de luzes
2.1.1 - Conceito do Internet das coisas (IoT)

A Internet das Coisas, frequentemente abreviada como IoT, é um conceito que descreve a conexão de dispositivos físicos à Internet, permitindo que eles comuniquem entre si e com os utilizadores. Este conceito emergiu na sequência do desenvolvimento e expansão da Internet, que deixou de ser apenas uma rede de computadores e passou a conectar uma vasta gama de dispositivos, desde eletrodomésticos e carros até infraestruturas urbanas e dispositivos médicos.

Os dispositivos IoT são equipados com sensores e/ou atuadores, bem como com tecnologia de comunicação sem fios, permitindo-lhes recolher, enviar e receber dados. Estes dispositivos podem ser controlados remotamente e podem também responder automaticamente a certas condições ou eventos, graças à sua capacidade de processar e analisar os dados recolhidos.

A IoT tem um amplo leque de aplicações, desde a criação de casas inteligentes e cidades inteligentes, à indústria 4.0, cuidados de saúde, agricultura, entre outros. Com a IoT, é possível automatizar e otimizar processos, melhorar a eficiência energética, monitorizar e melhorar a saúde, e proporcionar uma maior conveniência e conforto para os utilizadores.

No entanto, a IoT também apresenta desafios, sobretudo no que diz respeito à segurança e privacidade dos dados, uma vez que a recolha e análise de grandes volumes de dados pode levar a preocupações sobre a forma como esses dados são usados e protegidos.

Em resumo, a IoT representa uma transformação significativa na forma como interagimos com o mundo à nossa volta, oferecendo oportunidades para melhorar a nossa qualidade de vida, mas também levantando questões importantes que precisam de ser abordadas.

2.1.2 - Conceito de Sistema Inteligente de Luzes

Um Sistema Inteligente de Luzes é uma tecnologia que permite o controlo automatizado e personalizado da iluminação de um ambiente. Esta tecnologia, que está muitas vezes interligada com o conceito da Internet das Coisas (IoT), tem como objetivo principal otimizar o consumo de energia e proporcionar um ambiente de iluminação mais confortável e adaptável às necessidades dos utilizadores.

Ao contrário dos sistemas de iluminação convencionais que têm uma operação fixa e estática, os sistemas inteligentes de luzes são capazes de ajustar a intensidade da luz, os horários de funcionamento, e até mesmo operar de forma automática à presença de pessoas, proporcionando uma luminosidade ambiente ou até mesmo ajustando-se com o clima.

Estes sistemas utilizam uma variedade de sensores e atuadores, juntamente com algoritmos de processamento de dados, para responder de forma inteligente às mudanças nas condições ambientais e comportamentais. Por exemplo, pode-se utilizar sensores de luz para ajustar automaticamente a intensidade da iluminação com base na quantidade de luz natural disponível, ou sensores de movimento para ligar ou desligar as luzes quando as pessoas entram ou saem de uma sala.

Os sistemas inteligentes de luzes podem ser controlados de várias formas, incluindo através de aplicações de dispositivos móveis, assistentes de voz, ou até mesmo através de programações pré-definidas. Além disso, podem também ser integrados com outros sistemas de automação doméstica, permitindo a criação de cenários de iluminação personalizados que se adaptam às rotinas diárias dos utilizadores.

Em suma, os sistemas inteligentes de luzes representam uma forma inovadora de gestão da iluminação, que pode trazer benefícios significativos em termos de eficiência energética, conforto e conveniência.

2.1.3 - Benefícios proporcionados pelos Sistemas Inteligentes de Luzes

Os sistemas inteligentes de luzes oferecem diversos benefícios, entre os quais destacam-se:

- Economia de energia: ao ajustar a iluminação com base na luz ambiente e na presença de pessoas, os sistemas inteligentes de luzes podem reduzir significativamente o consumo de energia;
- Conforto e personalização: com a capacidade de adaptar a iluminação às preferências do utilizador e às condições do ambiente, os sistemas inteligentes de luzes proporcionam maior conforto e uma experiência personalizada;
- Fácil controlo: o controlo remoto através de aplicações móveis ou dispositivos de automação residencial permite aos utilizadores gerir facilmente a iluminação, mesmo à distância;
- Segurança: ao simular a presença de pessoas e controlar a iluminação externa, os sistemas inteligentes de luzes podem aumentar a segurança da residência ou estabelecimento.

2.1.4 - Trabalhos relacionados com Sistemas Inteligentes de Luzes

Diversos trabalhos abordam o desenvolvimento e a implementação de sistemas inteligentes de luzes baseadas em Arduíno. Alguns exemplos de trabalhos relacionados são:

- *Connection between 3D engine unity and microcontroller arduino: A virtual smart house by Kučera,E. et al.* – O artigo ilustra a criação de um percurso virtual através de uma casa inteligente, desenvolvido no motor gráfico Unity 3D. Relata uma experiência imersiva interligada com um microcontrolador Arduino, que tem integrados uma série de sensores (nomeadamente, o sensor de fogo, o sensor de luz, o sensor de temperatura, entre outros) e atuadores (como por exemplo o buzzer). A aplicação do artigo tem diversas funções, nomeadamente, ligar/desligar luzes com base em sons altos por exemplo palmas, disparar alarme ao detetar fogo, ver a temperatura atual, entre outras funções;

- *Low Cost Arduino/Android-Based Energy-Efficient Home Automation System with Smart Task Scheduling by Baraka, K. et al.* – Neste artigo, foram usadas técnicas de automação residencial para projetar e implementar uma casa inteligente controlada remotamente, energeticamente eficiente e altamente escalável, com recursos básicos que protegem o conforto e a segurança dos residentes. O sistema proposto consiste numa rede doméstica (sensores e atuadores de aparelhos para, respetivamente, obter informações e controlar o ambiente da casa). Como controlador central, foi utilizado um microcontrolador Arduino que comunica com um aplicativo Android. A rede doméstica reúne as tecnologias sem fio Zigbee e com fio X10. Os eventos podem ser programados para serem acionados em condições específicas, o que pode ter um grande papel na redução do consumo total de energia de alguns eletrodomésticos. Além disso, o sistema proposto sugere o agendamento inteligente de tarefas.

Estes trabalhos fornecem *insights* valiosos para o projeto e a implementação do sistema de iluminação inteligente proposto neste estudo.

2.2 - Ambiente de Desenvolvimento (IDE)

Para o desenvolvimento do sistema de iluminação inteligente proposto neste trabalho, foram utilizadas duas ferramentas principais. O primeiro é o Arduíno, uma plataforma de prototipagem eletrónica de código aberto baseada em hardware e software flexíveis e fácil utilização. O Arduíno é amplamente utilizado em projetos de automação, robótica e IoT, devido à sua simplicidade, acessibilidade e vasta comunidade de apoio.

O Ambiente de Desenvolvimento Integrado (IDE) do Arduíno é uma aplicação de código aberto que facilita a escrita, compilação e carregamento de código para a placa Arduíno. A IDE do Arduíno inclui diversas funcionalidades, como a possibilidade de programar em C++ e a disponibilização de várias bibliotecas para facilitar o desenvolvimento de projetos.

Para o desenvolvimento da aplicação móvel do sistema, foi utilizado o Android Studio. O Android Studio é um Ambiente de Desenvolvimento Integrado (IDE) oficial para o desenvolvimento de aplicações Android. Este ambiente fornece uma série de ferramentas que auxiliam no desenvolvimento de aplicações robustas e de alta qualidade para todos os dispositivos Android. Além disso, o Android Studio oferece recursos que aceleram a construção de aplicações em todos os tipos de dispositivos Android e que permitem a visualização do projeto com uma variedade de pré-visualizações e ferramentas de depuração.

A escolha do Arduíno e do Android Studio como ambientes de desenvolvimento para este projeto deve-se à sua vasta adoção na indústria, à sua robustez e à grande comunidade de desenvolvedores que oferecem suporte e contribuem para a sua evolução contínua.

Figura 1-Arduíno Uno & Android Studio

2.2.1 - Características do Arduíno

O Arduíno possui diversos modelos de placas, cada um com suas especificações e recursos. Algumas características comuns incluem:

- Microcontrolador: cada placa Arduíno possui um microcontrolador que executa o código carregado na placa;
- Portas de Entrada/Saída (GPIO): as placas Arduíno têm pinos de Entrada/Saída Generalizados (GPIO), que permitem a conexão com componentes externos, como sensores, atuadores e módulos de comunicação;
- Alimentação: as placas Arduíno podem ser alimentadas por uma fonte de alimentação externa ou pela porta USB;
- Comunicação: as placas Arduíno possuem comunicação serial (UART) e, dependendo do modelo, podem suportar outros protocolos, como I2C e SPI.

2.2.2 – Bibliotecas

Neste projeto, várias bibliotecas do Arduíno foram utilizadas para facilitar o trabalho com diferentes componentes e funcionalidades. A seguir, apresentamos uma breve descrição de cada uma das bibliotecas utilizadas:

2.2.2.1 - Timer.h

A biblioteca Timer.h fornece uma maneira fácil de criar rotinas de temporização. Permite executar uma função em intervalos regulares sem o uso da função delay(), que bloqueia todo o código durante o período de atraso. É especialmente útil em projetos onde várias tarefas precisam de ser executadas simultaneamente.

2.2.2.2 - Arduino.h

Arduíno.h é a biblioteca principal do Arduíno que inclui definições básicas e funções que são comumente usadas na programação do Arduíno. Esta biblioteca é automaticamente incluída

em todos os programas Arduíno, fornecendo acesso a funções básicas como digitalRead(), digitalWrite(), delay(), entre outras.

2.2.2.3. Wire.h

A biblioteca Wire.h é usada para comunicação entre dispositivos que usam o protocolo I2C (Inter-Integrated Circuit). Esta biblioteca permite que o Arduíno se comunique com um ou mais dispositivos I2C, enviando e recebendo dados.

2.2.2.4. SoftwareSerial.h

A biblioteca SoftwareSerial.h permite a comunicação serial em pinos digitais do Arduíno, além dos pinos de RX e TX padrão. Isso pode ser útil em projetos que requerem mais de uma porta serial.

2.2.2.5. DHT11.h

A biblioteca DHT11.h é usada para trabalhar com o sensor de temperatura e humidade DHT11. Ela fornece funções para ler facilmente a temperatura e a humidade do sensor DHT11.

2.2.2.6. LiquidCrystal.h

A biblioteca LiquidCrystal.h é usada para controlar displays LCD que são compatíveis com o controlador Hitachi HD44780. Esta biblioteca facilita a exibição de texto e números no LCD.

Em resumo, estas bibliotecas fornecem uma variedade de funções que facilitam a programação do Arduíno e a interação com vários componentes de hardware. Elas permitem que o programador se concentre na lógica principal do projeto, sem ter que se preocupar com os detalhes de baixo nível da comunicação com o hardware.

```
#include "Timer.h"
#include <Arduino.h>
#include <Wire.h>
#include <SoftwareSerial.h>
#include <DHT11.h>
#include <LiquidCrystal.h>
LiquidCrystal lcd_1(8,9,10,11,12,13);
#define DHTPIN 7
```

Figura 2-Bibliotecas usadas

2.2.3 - Estrutura do código

O código em análise pode ser categorizado em duas componentes distintas: a componente Arduino e a aplicação Android.

A parte do código associada ao Arduino é responsável por duas funções principais. Primeiramente, procede à leitura dos valores de input e à sua respetiva automatização. Em segundo lugar, é responsável pelo envio destes valores sempre que uma conexão Bluetooth é estabelecida.

Por sua vez, a aplicação Android tem como propósito principal estabelecer a ligação Bluetooth com o Arduino, receber os dados transmitidos, interpretá-los e apresentá-los de forma adequada ao utilizador. Adicionalmente, a aplicação Android permite ainda o envio de comandos ou informação ao Arduino, possibilitando a sua manipulação.

Nos parágrafos que se seguem, cada um destes componentes será descrito de forma mais aprofundada e detalhada.

2.2.3.1 - Estrutura do código - Arduino

O código pode ser dividido em várias partes, que incluem as declarações de variáveis, a inicialização, as funções de leitura de sensores, as funções de controlo de dispositivos e a função principal, chamada loop().

2.2.3.1 - Declaração de variáveis e inclusão de bibliotecas

No início do código, várias bibliotecas são incluídas. As bibliotecas incluem Arduino.h, Wire.h, SoftwareSerial.h, DHT11.h e LiquidCrystal.h. Estas bibliotecas são usadas para várias funções como a leitura de sensores, comunicação serial e controlo do ecrã LCD.

Várias variáveis são declaradas e iniciadas neste ponto. Estas variáveis incluem pinos de entrada e saída, valores de temporizador, limites de sensores e variáveis para armazenar valores de sensores. Além disso, são criados objetos para o sensor de temperatura e humidade DHT11, o ecrã LCD e o módulo Bluetooth.

2.2.3.2 - Função setup()

A função setup() é chamada uma vez quando o programa começa. Nesta função, o ecrã LCD e o módulo Bluetooth são iniciados e os pinos de entrada e saída são definidos. Além disso, o ecrã LCD é configurado para mostrar uma mensagem de inicialização.

2.2.3.3 - Funções de leitura de sensores

As funções tempHum() e lumsensor() são responsáveis por ler a temperatura, a humidade e a luminosidade, respetivamente. Estas funções fazem as leituras dos valores dos sensores, armazenam-nos em variáveis e enviam-nos através do módulo Bluetooth.

2.2.3.4 - Funções de controlo de dispositivos

Várias funções são usadas para controlar os dispositivos na casa. Estas funções incluem photoresistormeasure(), que controla as luzes com base na luminosidade, e o processCommand(), que processa os comandos recebidos através do módulo Bluetooth e controla os dispositivos com base nestes comandos.

2.2.3.5 - Função loop()

A função loop() é a função principal do programa e é executada repetidamente enquanto o programa estiver a funcionar. Nesta função, as funções de leitura de sensores são chamadas em intervalos regulares e a função de processamento de comandos Bluetooth é chamada sempre que um comando é recebido. Além disso, se a função de medição do *photoresistor* estiver ativa, a função photoresistormeasure() é chamada para controlar as luzes com base na luminosidade.

```
void sendData(String transmitData){

void setup(){

void tempHum(){

void lumsensor(){

void Lcdscroll(){

void photoresistormeasure(){

void bluetoothConnection() {

void processCommand(const String& command) {

int parseIntegerValue(const String& command) {

void loop(){
```

Figura 3-Estrutura do Código - Arduino

2.2.4 - Estrutura do código – Aplicação Android

Para o nosso projeto foi usado o Android Studio para o desenvolvimento da aplicação. Como primeiro passo e por se tratar de uma aplicação Android o ficheiro "manifest" é obrigatório, para identificar informações importantes sobre a aplicação, como o nome, a versão, as permissões necessárias entre outros.

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- Permissions for Bluetooth access -->
    <uses-permission android:name="android.permission.BLUETOOTH" />
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="LHDR Smart Home"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".SelectDeviceActivity" />
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Figura 4- Manifest Android Studio

O nosso código conta com quatro ficheiros mas divide-se em duas partes: a ligação ao Bluetooth e a *mainActivity* que controla e lê as informações do Arduíno.

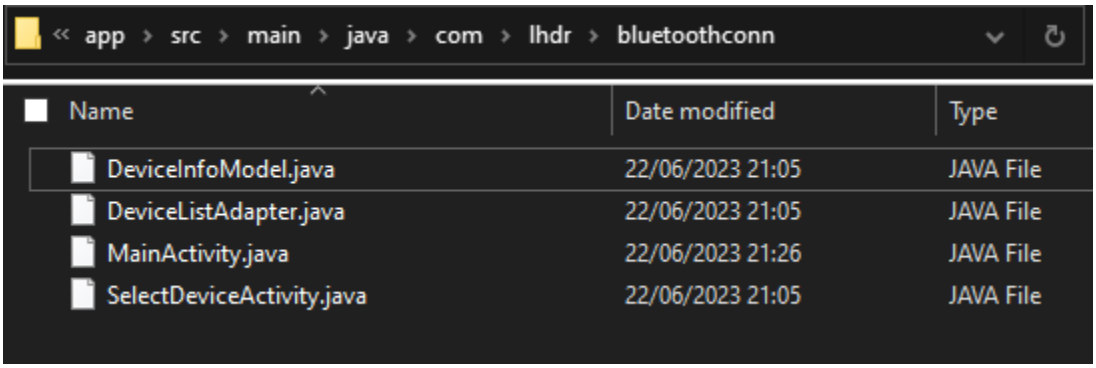
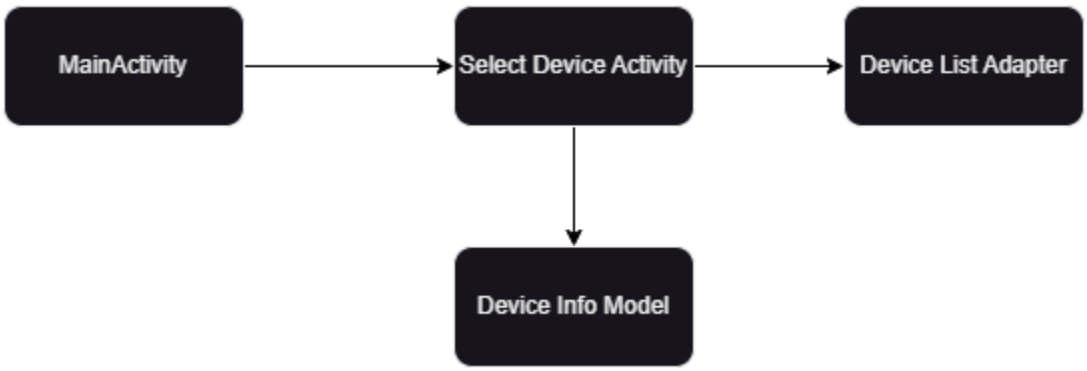


Figura 5-Ficheiros Java

O nosso ficheiro *mainActivity* é responsável por chamar o *selectDeviceActivity* que por sua vez irá chamar o *deviceListadapter* para retornar todos os dispositivos Bluetooth existentes e o *Deviceinfomodel*, para retornar o nome e o MAC do dispositivo Bluetooth. Após estes dados serem recolhidos, a aplicação retorna ao *MainActivity* de novo.



Dentro do *MainActivity*, podemos controlar e alterar parâmetros no Arduíno, podemos ligar e desligar a iluminação através dos botões existentes. Mas também podemos alterar parâmetros para tornar a iluminação automática. Aqui também são definidos valores de luminosidade e temperatura.

```
/* ============================ Light Control Screen Action ================================== */
buttonToggle4.setOnClickListener(new View.OnClickListener() {

buttonToggle5.setOnClickListener(new View.OnClickListener() {

buttonToggle7.setOnClickListener(new View.OnClickListener() {

buttonToggleAll.setOnClickListener(new View.OnClickListener() {

buttonToggleAutoLightsAll.setOnClickListener(new View.OnClickListener() {

dinTB.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {

bedTB.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {

livTB.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {

applytemp.setOnClickListener(new View.OnClickListener() {

applyhum.setOnClickListener(new View.OnClickListener() {

applylum.setOnClickListener(new View.OnClickListener() {
```

Figura 6-Controlo de luzes na aplicação

Na secção dos cases, é o onde a aplicação comunica com o Arduíno e lê o que o Arduino retorna. Leituras de sensores, indicações se as luzes estão desligadas ou ligadas, etc. Esta secção apenas serve de leitura, não são enviadas alterações para o Arduíno.

```
public void handleMessage(Message msg){
    switch (msg.what){
        case CONNECTING_STATUS:
            switch(msg.arg1){
            break;

        case MESSAGE_READ:
            String arduinoMsg = msg.obj.toString(); // Read message from Arduino
            String[] splitMsg = arduinoMsg.split(" ");


            switch (splitMsg[0]) {
                //Startup Configs
                case "configs":
                    tempmin.setText(splitMsg[2]);
                    tempmax.setText(splitMsg[3]);
                    hummin.setText(splitMsg[6]);
                    hummax.setText(splitMsg[7]);
                    lummin.setText(splitMsg[10]);
                    lummax.setText(splitMsg[11]);
                    break;
                case "Living":
                case "Bedroom":
                case "Dining":
                case "ALL":
                case "Auto":
                    textViewInfoLights.setText(arduinoMsg);
                    Timer timer = new Timer();
                    timer.schedule(new TimerTask() {
                    break;
                case "Temperature":
                    textViewTempValue.setText(splitMsg[1]);
                    textViewHumValue.setText(splitMsg[3]);
                    break;
                case "Luminosity":
                    textViewLumValue.setText(splitMsg[1]);
                    break;
            }
            break;
    }
}
```

Figura 7-Casos na aplicação

Por fim, de forma a tornar o protótipo mais inteligente e dinâmico, o código conta com multi-threading.

```
/* ============================== Thread for Data Transfer ========================================= */
public static class ConnectedThread extends Thread {
    private final BluetoothSocket mmSocket;
    private final InputStream mmInStream;
    private final OutputStream mmOutStream;

    public ConnectedThread(BluetoothSocket socket) {

    public void run() {

    /* Call this from the main activity to send data to the remote device */
    public void write(String input) {

    /* Call this from the main activity to shutdown the connection */
    public void cancel() {
}
```

Figura 8-Multi-threading

2.3 - Mecanismo de Comunicação do Sistema

Para o sistema de iluminação inteligente proposto neste trabalho, será utilizado o Bluetooth como mecanismo de comunicação sem fio. O Bluetooth é um protocolo de comunicação amplamente utilizado para a troca de dados entre dispositivos eletrônicos em curta distância, permitindo conexões rápidas e seguras. Algumas vantagens do uso do Bluetooth neste projeto incluem:

- Baixo consumo de energia: o Bluetooth Low Energy (BLE) é uma versão do protocolo Bluetooth que consome menos energia, o que é especialmente útil em projetos de IoT e automação;
- Facilidade de conexão: a maioria dos dispositivos móveis possui suporte nativo ao Bluetooth, facilitando a conexão com o sistema de iluminação inteligente e a implementação do aplicativo móvel;
- Ampla disponibilidade de módulos: existem diversos módulos Bluetooth disponíveis no mercado, como o HC-05 e o HM-10, que são compatíveis com o Arduíno e permitem a comunicação sem fio entre dispositivos.

2.3.1 - Android Studio

O Android Studio é uma plataforma de desenvolvimento integrado (IDE) para a criação de aplicativos Android. Desenvolvido pela Google, o Android Studio é baseado no IntelliJ IDEA e oferece recursos poderosos e abrangentes para facilitar o desenvolvimento de aplicativos móveis. Algumas funcionalidades do Android Studio incluem:

- Editor de código avançado: o Android Studio possui um editor de código inteligente com suporte a linguagens como Java, Kotlin e XML, além de recursos como auto completar, refatoração e análise de código em tempo real;
- Design de interface: o Android Studio fornece um editor visual de interface que permite criar e editar layouts de aplicativos de maneira rápida e intuitiva, utilizando arrastar e soltar;
- Emulador: o Android Studio inclui um emulador que permite testar aplicativos em diversos dispositivos e versões do Android, sem a necessidade de dispositivos físicos;
- Integração com o Firebase: o Android Studio oferece integração direta com o Firebase, uma plataforma de desenvolvimento de aplicativos da Google que fornece serviços como banco de dados, autenticação e notificações push;
- Controlo de versão: o Android Studio possui suporte integrado para sistemas de controlo de versão, como Git, facilitar a gestão e a colaboração no desenvolvimento de aplicativos.

Para o desenvolvimento da aplicação móvel que controlará o sistema de iluminação inteligente, o Android Studio será utilizado devido à sua facilidade de uso, recursos avançados e compatibilidade com dispositivos Android, que são amplamente utilizados em todo o mundo.

2.4 - Componentes Físicos

O sistema de iluminação inteligente proposto neste trabalho utilizará uma série de componentes eletrônicos, incluindo uma plataforma de desenvolvimento, sensores e módulos de comunicação. Nesta seção, descreveremos os principais componentes físicos e as suas características.

2.4.1 – DHT11

O DHT11 é um sensor digital de temperatura e humidade, comumente utilizado em projetos de eletrónica e automação. Devido ao seu baixo custo e à sua precisão moderada, tornou-se uma escolha popular para muitos entusiastas da eletrónica, incluindo aqueles que utilizam plataformas como o Arduíno.

Em termos de especificações, o DHT11 é capaz de medir a humidade no intervalo de 20% a 80% com uma precisão de 5%, e a temperatura de 0 a 50 graus Celsius com uma precisão de ±2 graus. Estes dados podem ser lidos através de uma interface digital, o que facilita a conexão com os microcontroladores.

O DHT11 tem uma taxa de atualização de um segundo, o que significa que pode fornecer novas leituras de temperatura e humidade a cada segundo. Além disso, o sensor é autocalibrado e não requer componentes externos para funcionar, tornando-o uma solução fácil de usar para medição de temperatura e humidade.

As suas principais características são:

- O DHT11 é um sensor de temperatura e humidade;
- Capacidade de medir a humidade no intervalo de 20% a 80%;
- A faixa de medição de temperatura é de 0 a 50 graus Celsius;
- A precisão da medição de humidade é de ±5%;
- A precisão da medição de temperatura é de ±2 graus Celsius;
- Ele opera a uma tensão de 3 a 5V e possui uma interface de sinal digital de um único fio, o que significa que apenas um pino GPIO é necessário para ler os dados do sensor.

2.4.2 – HC-05 módulo Bluetooth

É um componente amplamente utilizado para adicionar capacidades Bluetooth a dispositivos eletrónicos. É frequentemente utilizado em projetos Arduino devido à sua facilidade de uso, baixo custo e compatibilidade com uma ampla gama de dispositivos.

O módulo HC-05 usa a interface de comunicação serial (UART), tornando-o fácil de ligar a microcontroladores como o Arduino. Isto permite a comunicação sem fios entre um Arduino e outros dispositivos como computadores, dispositivos móveis ou outros microcontroladores.

As suas principais características são:

- Tensão de operação: 4V a 6V (tipicamente +5V);
- Corrente de operação: 30mA;
- Trabalha com comunicação serial (USART) e é compatível com TTL;
- Segue o protocolo padronizado IEEE 802.15.1;
- Tensão de fornecimento: 3.3V a 6.0V;
- Tensões de operação: 3.3V (todos os outros pinos, exceto VCC);
- Alcance operacional: máximo de 10m;
- Senha padrão: 0000 ou 1234 (depende do modelo/fabricante);
- Taxas de bandas suportadas: 9600, 19200, 38400, 57600, 115200, 230400, 460800;
- O módulo Bluetooth HC-05 é um módulo MASTER/SLAVE. Por padrão, a configuração de fábrica é SLAVE;
- Suporta o padrão Bluetooth 2.0+EDR e opera na banda de frequência ISM de 2.4GHz, usa a modulação de espectro espalhado de salto de frequência (FHSS) para transmissão de dados confiável e segura.

2.4.3 – Photoresistor

Os *photoresistor*, também conhecidos como LDR (resistores dependentes da luz), são componentes feitos de semicondutores. Um photoresistor é sensível à luz. A sua resistência diminui quando a iluminação aumenta.

As suas principais características são:

- Faixa de resistência: de 200KΩ (no escuro) a 10KΩ (brilho de 10 lux);
- Faixa de sensibilidade: as células respondem à luz entre os comprimentos de onda de 400nm (violeta) e 600nm (laranja), atingindo um pico de cerca de 520nm (verde);
- Fonte de alimentação: praticamente qualquer coisa até 100V, usa menos de 1mA de corrente em média (dependendo da tensão da fonte de alimentação).

2.4.4 – 8 channel relay

Placa com 8 relés individuais que podem ser controlados diretamente por uma grande variedade de microcontroladores, como Arduíno, AVR, PIC, ARM e outros.

Um relé é um interruptor que é operado eletronicamente. Ele pode ser usado para controlar circuitos de alta tensão e alta corrente a partir de um sinal de baixa tensão e baixa corrente. Cada canal num módulo de relé de 8 canais pode ser usado para controlar um dispositivo diferente.

O módulo de relé de 8 canais normalmente tem 8 entradas que correspondem a cada um dos 8 relés, uma entrada de energia para alimentar os relés, e uma entrada de controlo que pode ser usada para ativar ou desativar todos os relés simultaneamente.

Cada relé pode operar de forma independente, o que significa que se pode ter até 8 dispositivos diferentes controlados por um único módulo de relés de 8 canais.

O módulo de relé de 8 canais tem as seguintes características:

- Tensão de alimentação: 3,75V a 6V;
- Corrente de disparo: 5mA;
- Corrente quando o relé está ativo: cerca de 70mA (único), cerca de 600mA (todos os oito);
- Tensão de contato máxima do relé: 250VAC, 30VDC;
- Corrente máxima do relé: 10ª.

2.4.5 – LCD A1602

O A1602 é um módulo de ecrã de cristais líquidos (LCD) que é muito comum em vários tipos de projetos eletrónicos. Este módulo utiliza a tecnologia LCD para apresentar texto e caracteres simples.

A designação "A1602" refere-se ao facto de o ecrã poder mostrar 16 caracteres por linha, com um total de 2 linhas, dando-lhe uma capacidade total de exibir 32 caracteres de uma só vez. Os caracteres podem ser letras, números, símbolos ou uma combinação destes.

Este módulo tem a vantagem de ser bastante legível, mesmo à luz do dia, e a capacidade de exibir texto personalizado torna-o uma escolha popular para projetos que necessitam de uma interface do utilizador simples e eficaz.

As características do LCD A1602 são as seguintes:

- Tem um ecrã LCD de 16 caracteres por 2 linhas com luz de fundo azul e caracteres brancos. Cada um dos caracteres é composto por uma matriz de pontos 5x8 para uma boa representação do carácter;
- Opera a 5V;
- A luz de fundo não é controlável e fica ligada quando a energia é aplicada ao módulo;

- Possui interface I2C que requer apenas 2 pinos num MCU para a interface. Tem um bom suporte de biblioteca para uma rápida configuração. O endereço I2C destes ecrãs é normalmente 0x3F ou 0x27, mas pode ser ajustado para evitar conflitos;
- A ligação ao ecrã é feita através de um cabeçalho de 4 pinos (GND, VCC, SDA, SCL);
- A corrente de operação total é tipicamente de 30mA, enquanto que a corrente da luz de fundo é tipicamente de 20mA.

Em suma, todos estes componentes serão integrados na plataforma Arduíno, e a lógica de controlo será desenvolvida usando a IDE do Arduíno. A aplicação móvel, por sua vez, será desenvolvida no Android Studio e comunicará com o sistema de iluminação inteligente por meio do módulo Bluetooth.

A combinação destes componentes e plataformas permitirá a criação de um sistema de iluminação inteligente eficiente e versátil, capaz de se adaptar às condições ambientais e ser controlado remotamente por meio de uma aplicação móvel.

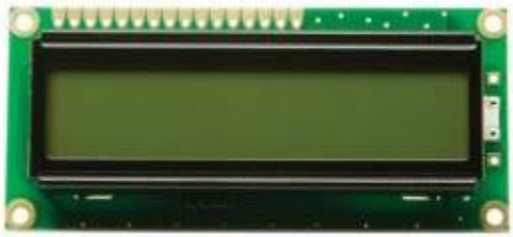


Figura 9-LCD A1602

2.5 - Metodologia de medição

Neste projeto, trabalhamos com correntes de valores muito baixos, o que requer um cuidado especial para a medição e tratamento das mesmas. De tal forma, escolhemos trabalhar com um multímetro que possui uma maior precisão com relação as medições. O instrumento usado para a realização das medições executadas neste projeto consiste em um multímetro digital, da marca Fluke modelo 187 True RMS. Sua resolução de precisão é de 0.001mV para tensões contínuas e resolução de 0.001mA para correntes contínuas. Para evitar problemas com electroestática e variações nas leituras, serão utilizadas ponteiras do tipo *alligator* onde exclui a necessidade de contactos físicos diretos entre o utilizador e o circuito. Seguindo o padrão de ambiente de teste da grande maioria dos componentes, todas as medições foram realizadas em ambiente com temperatura controlada de 25ºC, garantindo assim, maior fiabilidade e estabilidade dos dados coletados.

Figura 10-Múltimetro Fluke

3 - Metodologia do modelo proposto

Nesta seção, iremos apresentar a metodologia e a arquitetura geral do modelo proposto para o sistema de iluminação inteligente. O objetivo é criar um sistema capaz de ajustar a intensidade da iluminação artificial com base na luz natural medida por um *photoresistor*, detetar através de sensores quando a temperatura e humidade forem elevadas e enviar sinais de alerta. Por fim, tudo ser controlado remotamente através de uma aplicação móvel com ligação Bluetooth.

De maneira a facilitar a consulta de humidade e temperatura, o protótipo terá um *display* associado que irá demonstrar os valores atuais na habitação. Para se aproximar ainda mais da realidade será ligado um *relay* de 8 portas, em que neste caso só iremos usar 3, para a ligação de lâmpadas 220V.

3.1 - Apresentação geral do modelo proposto

A arquitetura do protótipo do sistema de iluminação inteligente consiste em diversos componentes integrados ao Arduíno. A Figura 1 mostra a arquitetura geral do modelo proposto, aqui pode ser observado o Arduíno com ligações ao *display*, ao sensor de temperatura, e a simulação de três LED's como divisões da casa.

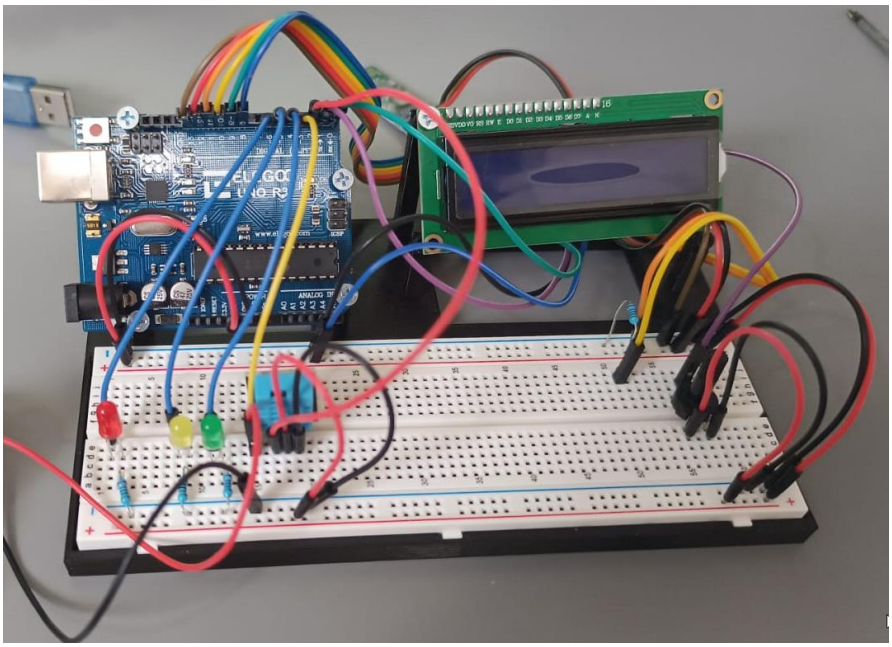


Figura 11-Arduíno geral

Serão desenvolvidos no sistema as seguintes funções:

- Um sensor de luminosidade que irá fornecer informações ao Arduíno acerca da iluminação natural, se esta for abaixo de um determinado limite, o Arduíno de forma automática acende ou apaga as luzes;
- Um sensor de humidade e temperatura, que assim que a temperatura for demasiado elevada, emite sinais de luzes para alertar;
- Um LCD com informações, sempre que alguma parte da casa for ligada, o LCD informa a divisão. Também servirá para dar informações relativas à humidade e temperatura. O LCD também terá um potenciómetro para dar mais ou menos brilho;
- Um módulo Bluetooth para assegurar a ligação entre o telemóvel e o Arduíno;
- Uma aplicação desenvolvida em Android para o controlo total do sistema.

Para a programação do Arduíno foi utilizada a linguagem de programação C++. Para a montagem dos componentes foi usada uma Breadboard. A aplicação foi desenvolvida em Android Studio e a sua interface será projetada para permitir a conexão com o módulo Bluetooth, a visualização das informações do sistema e o controlo das luzes.

Para o tratamento de informações entre os sensores e a aplicação será executado da seguinte forma:

1. Os sensores medem os parâmetros de luminosidade, humidade e temperatura e enviam para o Arduíno;
2. O Arduíno processa os dados dos sensores e ajusta a intensidade das luzes com base nas condições ambientais e nas configurações da aplicação móvel;
3. O Arduíno comunica com o módulo Bluetooth, transmite as informações sobre o estado atual do sistema e recebe comandos da aplicação se forem dados pelo utilizador;
4. A aplicação exibe as informações do sistema e permite assim o controlo das luzes, ao enviar comandos para o Arduíno através do módulo Bluetooth.

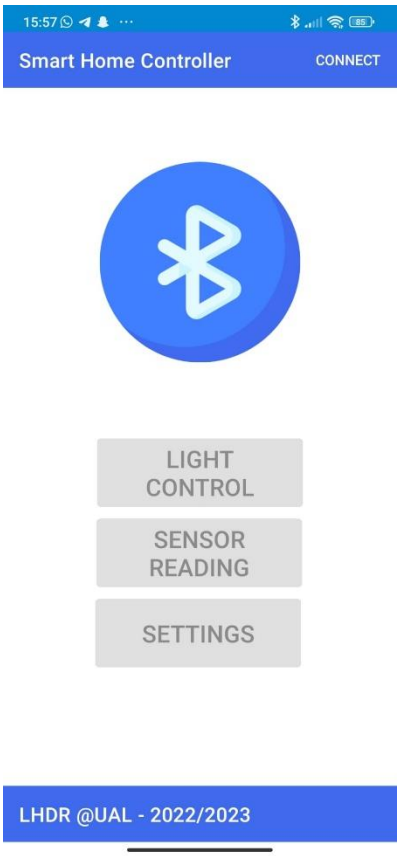


Figura 12-Aplicação desenvolvida

3.2 - Desenvolvimento do Modelo Proposto

Para o desenvolvimento do sistema de iluminação inteligente foi necessário o desenvolvimento de uma maquete, inicialmente seria a adaptação do nosso Arduíno a uma casa em pequena escala, mas como tal não foi possível, optou-se por uma impressão em 3D onde conseguiríamos integrar todo o equipamento e com um espaço mais reduzido.

O protótipo foi todo construído sobre uma Breadboard, de forma a preservar os componentes de possíveis acidentes, foi uma forma mais segura e mais fácil para montar e demonstrar todas as funcionalidades. Todas as conexões foram executas com cabos específicos para a Breadboard de forma a não se soltarem.

A aplicação Android desenvolvida pelo grupo com a ajuda do Android Studio, visa facilitar a interação com o sistema de iluminação e o controlo de todo o equipamento.

3.2.1 - Construção da Maquete

A maquete usada como mencionado anteriormente, foi elaborado numa impressora 3D. Utilizou-se um filamento de 1.75mm de cor azul para a sua construção.

Figura 13-Filamento de 1.75mm de cor azul

A maquete foi impressa com um preenchimento de 15% e 0.3mm de espaçamento por *layer* em Z. Uma caixa retangular onde será encaixado Arduíno, a Breadboard e o *display*. Terá 3 LED's na parte superior da caixa que simularão as diferentes divisões da casa, esses LED's irão acender em conjunto com as luzes de 220V.

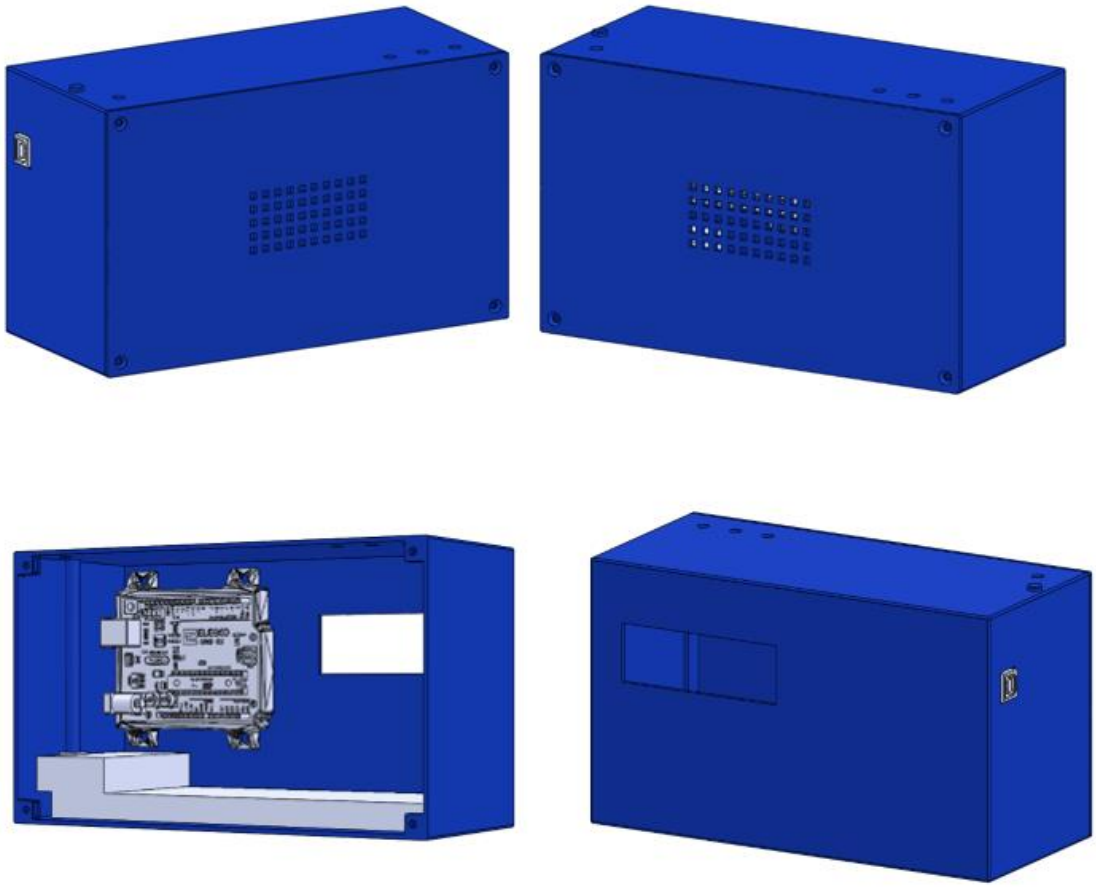


Figura 14-Maquete 3D

Para a fixação das luzes de 220V e do *relay* foi usado um painel de circuitos elétricos em madeira, foram usados 3 focos brancos de maneira a cada um simular uma divisão da casa.

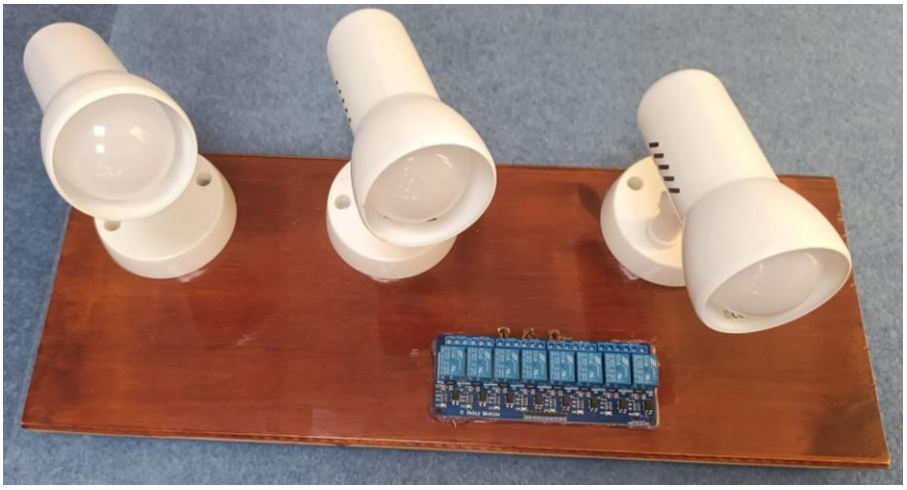


Figura 15-Painel das luzes

Nos 4 cantos do painel foram impressos 4 suportes de 10 cm de altura cada, de forma a facilitar a sua estabilidade. A impressão foi executada com um preenchimento 15%, 0.3 mm de espaçamento por *layer* e *Support in fill* orgânico em árvore. Para todas as impressões foi usado o mesmo filamento do Arduíno.

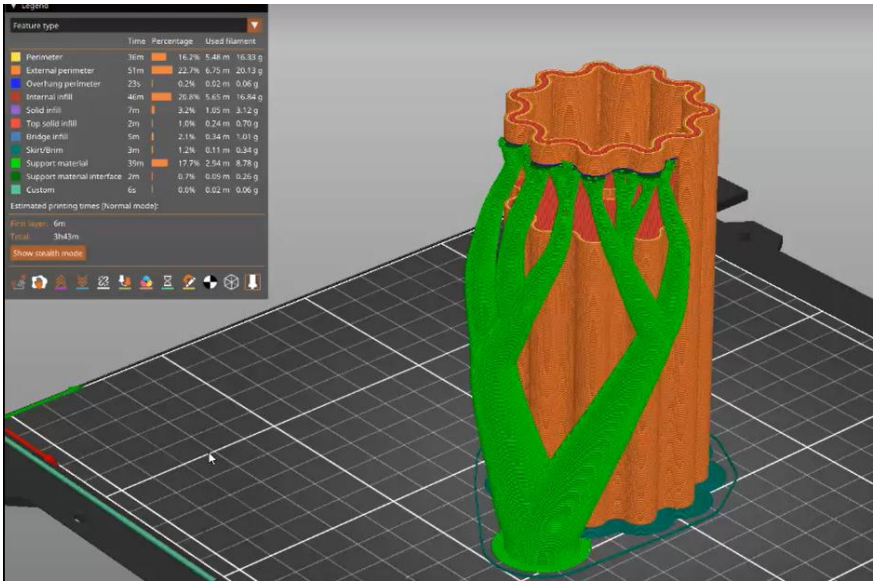


Figura 16-Suportes do painel em 3D

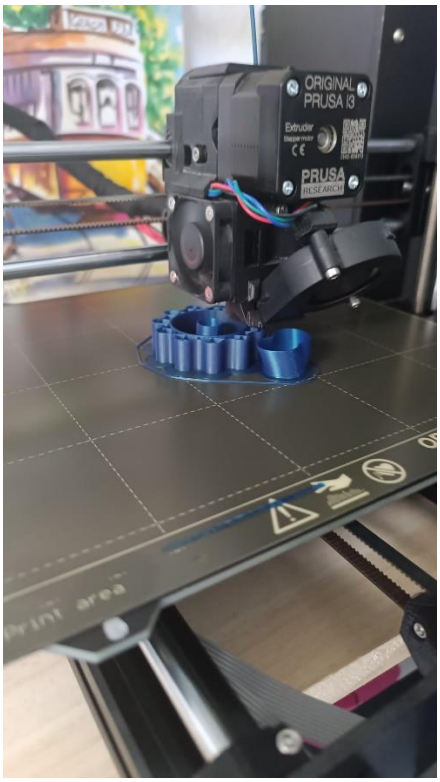


Figura 17-Impressão 3D dos suportes

Figura 18-Suporte do painel

3.2.2 - Implementação física do Protótipo

Para este projeto, utilizou-se um Arduíno UNO e o módulo Bluetooth HC-05 que será usado para estabelecer a ligação com o telemóvel através da aplicação. O LCD A1602 também será conectado ao Arduíno ao utilizar os pinos de comunicação I2C e estará programado para exibir informações relevantes sobre o estado do sistema. O LCD tem associado um potenciómetro para ser exibido mais ou menos brilho.

Um sensor de humidade e temperatura DHT11 para medir os níveis no ambiente em que se encontra e se ultrapassar um certo limite, emite um som através do buzzer de alerta.

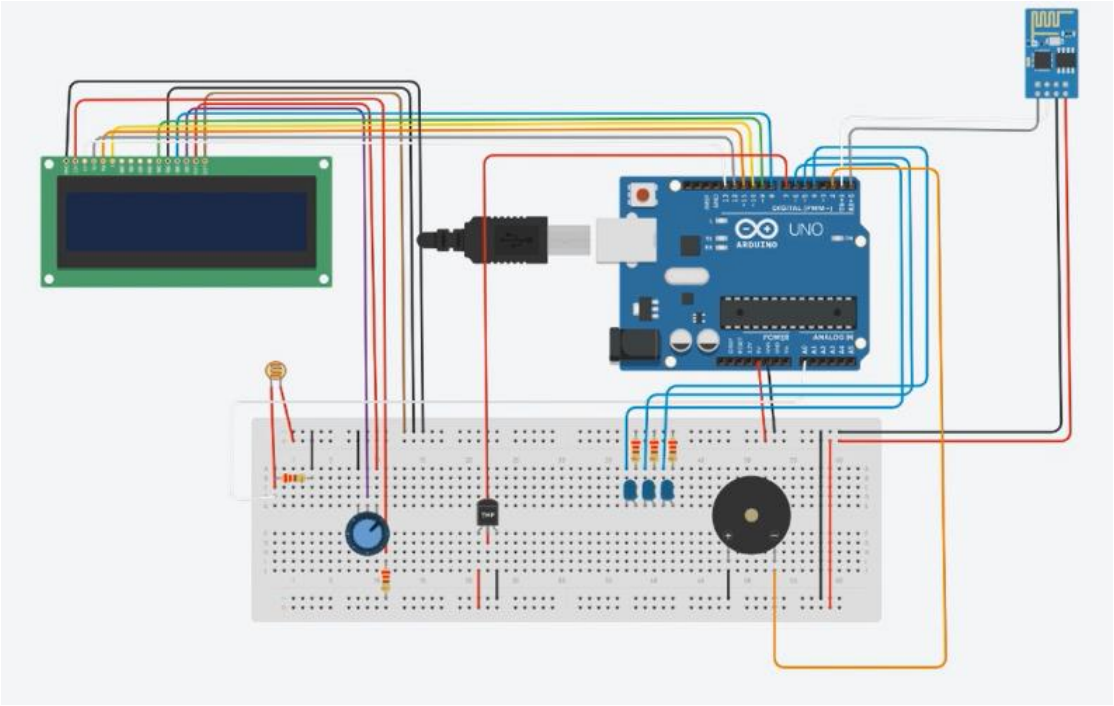


Figura 19-Desenho da implementação do sistema

Após os diagramas de ligação completos e verificação dos mesmos, procedeu-se à impressão do protótipo em 3D. Após a impressão estar concluída, os componentes foram instalados no seu interior.

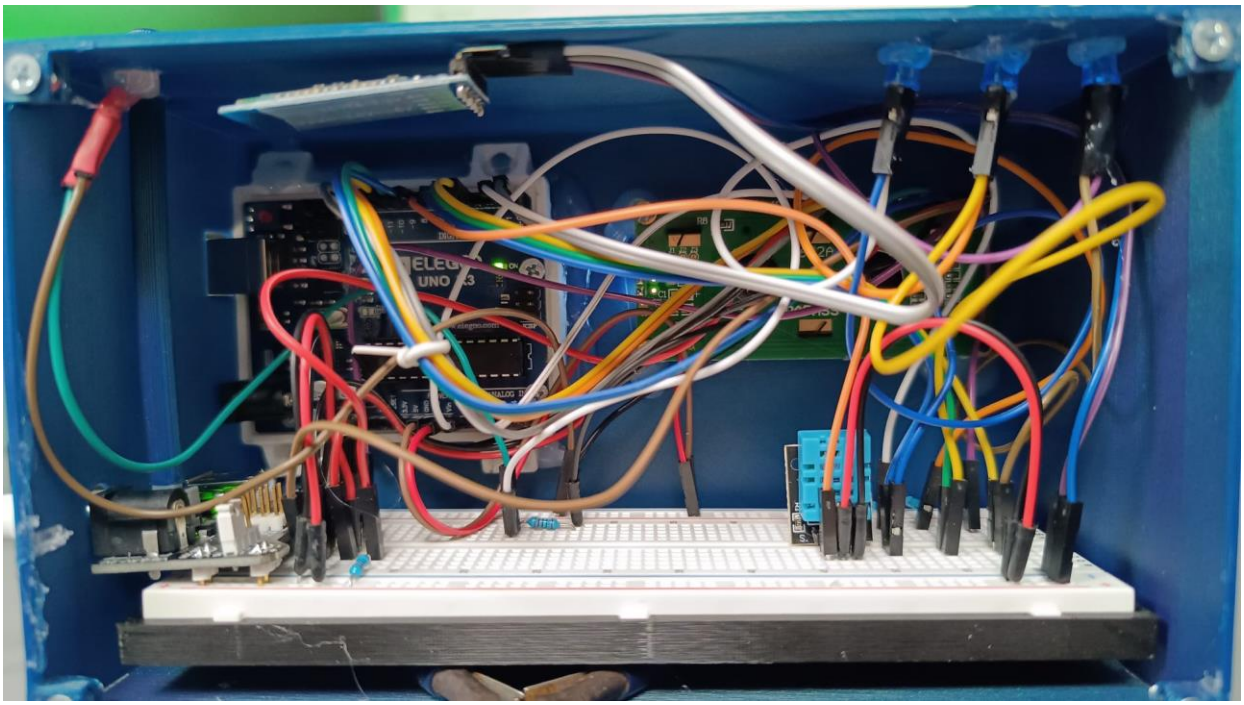


Figura 20-Ligações do sistema

Por fim, após estar ligado, obteve-se o resultado final que era expectável, o Arduíno ligado com todos os componentes a funcionar.

Figura 21-Iniciar do sistema

Foram também adaptadas saídas externas para a ligação ao painel de circuitos elétricos de forma a facilitar as ligações ao Arduíno.

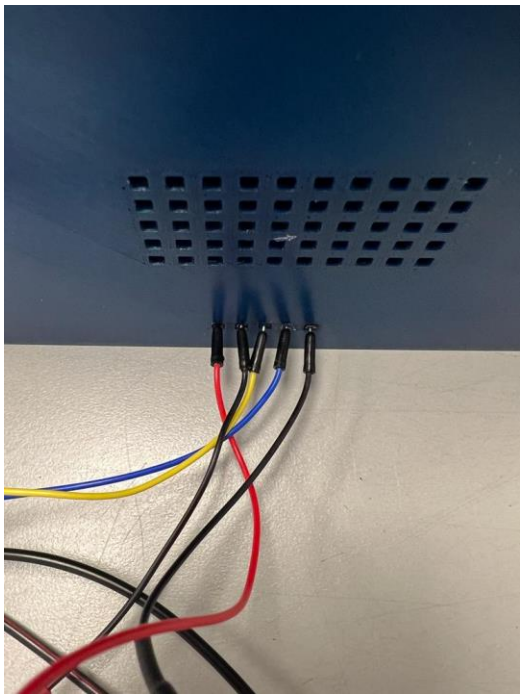


Figura 22-Ligações ao painel de circuitos elétricos

Estas ligações físicas estão conectadas ao *Relay* de oito portas para ligar e desligar as luzes de 220V.

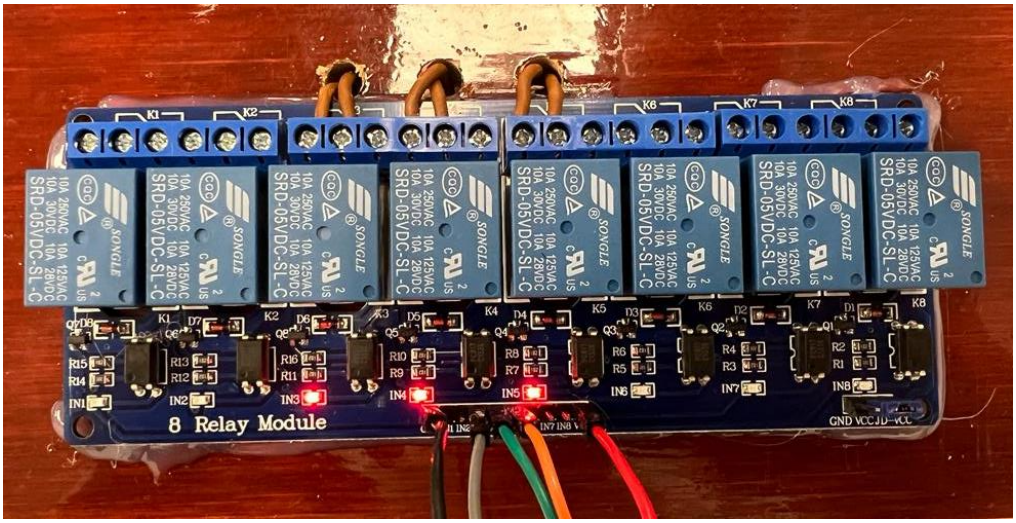


Figura 23-Ligações ao Relay

3.2.3 - Implementação da Aplicação de Controlo

Para o controlo do Arduíno, usamos uma ligação Bluetooth com o nosso telemóvel. De início a aplicação que foi desenvolvida em android Studio, servia apenas para fazer a ligação ao módulo HC-05 ligar os LEDs.

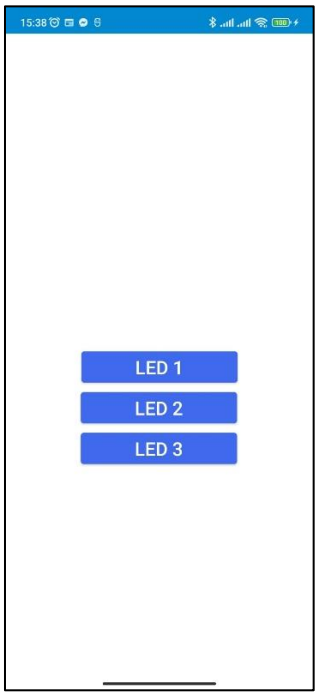


Figura 24-UI da aplicação móvel inicial

A aplicação tem a função de ligar ou desligar as luzes consoante a preferência do utilizador. Para tal na aplicação usamos *strings* para enviar a informação para o Arduíno.

As *strings* usadas na aplicação são:

- **0** - Todas as divisões
- **4** - Living room
- **5** - BedRoom
- **6** - Dining room

Assim que o Arduíno recebe os *inputs* da aplicação, trata a informação e transforma em ações.

Figura 25-Inputs da aplicação

Para os automatismos temos:

- **M** – Deteta a luminosidade máxima para acender as luzes (Fim do dia)
- **m –** Deteta a luminosidade mínima para apagar as luzes (Inicio do dia)

De forma a tornar a iluminação automática, que se inicia ao entardecer com o aceder das luzes e termina ao amanhecer com o seu desligar das mesmas, o utilizador tem de programar previamente no telemóvel.

Tudo isso é possível programar na aplicação na janela das *settings*, basta definir os valores máximos e mínimos de luminosidade.

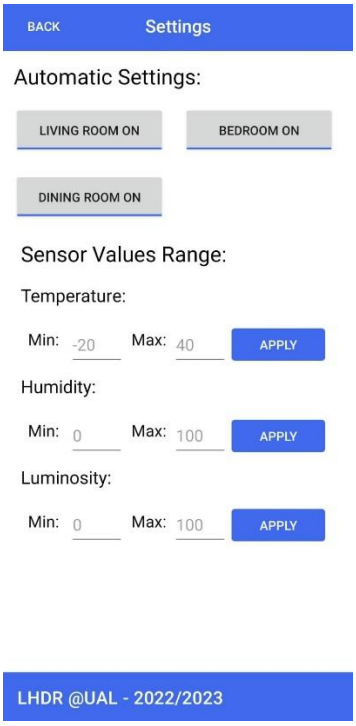


Figura 26-Definições de limites

Após a definição dos limites, o sistema de iluminação começa a agir automaticamente. Para isso basta ativar a opção “auto-lights” na aplicação.

Figura 27-Opções da aplicação

3.3 - Iluminação Automática

Iluminações automáticas são cada vez mais recorrentes no nosso dia-a-dia, seja por mero comodismo ou por necessidade. Por exemplo, em montras de lojas este sistema já é usado, para a uma certa hora, todas as luzes se apagarem e contribuir assim para a poupança de energia. Outra vantagem pode ser para pessoas com mobilidade reduzida acender ou apagar uma luz de forma automática.

3.3.1 - Controlo de iluminação automática

A iluminação automática funciona através de um *photoresistor* situado à esquerda na imagem abaixo. Sempre que a iluminação for inferior ao *threshold* de 200, as luzes acendem automaticamente. Se for superior a 700, irão apagar de forma automática.

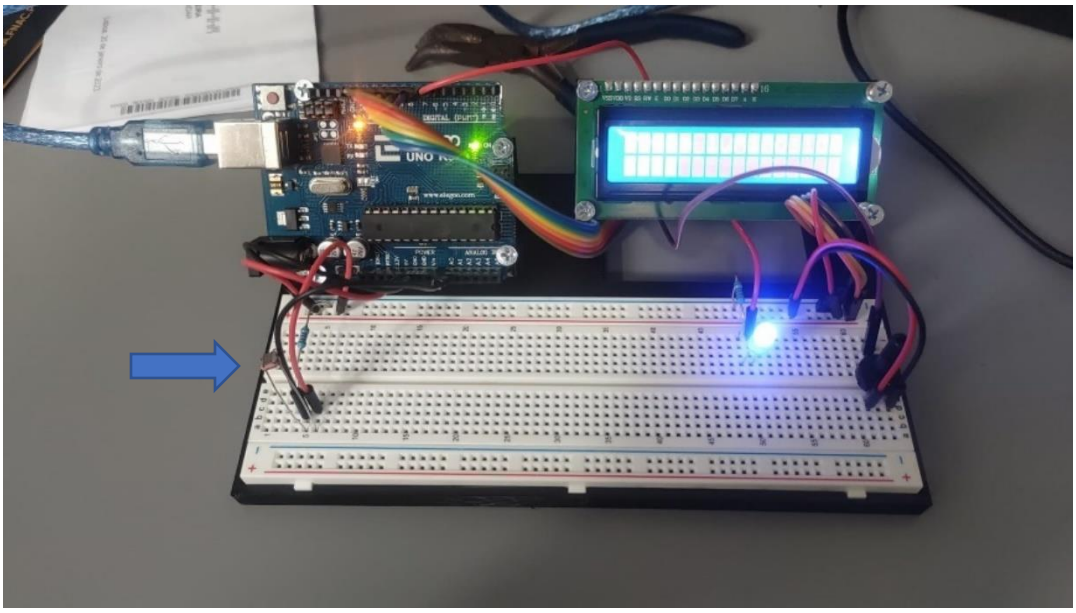


Figura 28-Teste com iluminação abaixo do limite de 200

De seguida, segue o diagrama que demonstra a simulação para apenas uma LED, como a iluminação está baixa, o LED encontra-se aceso. Esta informação estará sempre disponível no display para consulta.

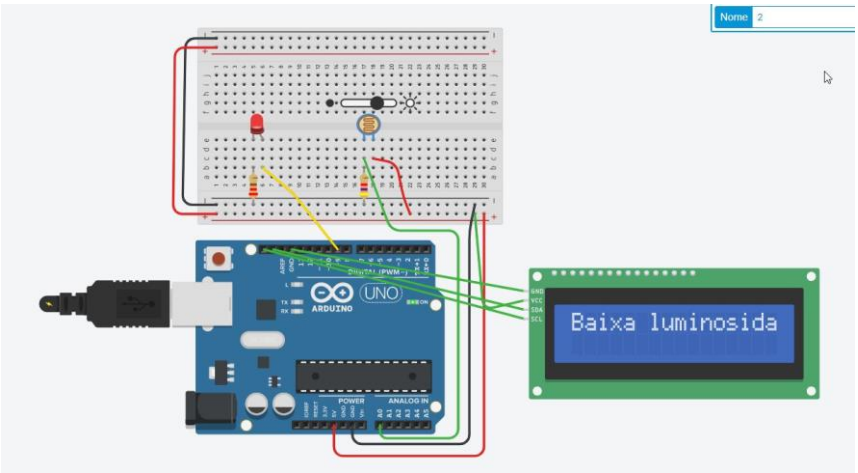


Figura 29-Simulação com iluminação baixa

3.3.2 – Sensor de humidade e temperatura

O módulo DHT11 é responsável por ler os parâmetros de humidade e temperatura dentro de casa, mostrar no LCD ligado ao Arduíno, no entanto irá funcionar como um sistema de alerta. Sempre que se verifique uma temperatura acima dos 50 graus, o Arduíno irá emitir um alerta sonoro com recurso a um *buzzer*.

Figura 30-Módulo DHT11

Os valores são programáveis, a temperatura de 50 graus foi apenas um valor de referência para efeitos de teste.

3.3.3 – Módulo Bluetooth

O módulo Bluetooth HC-05 ligado ao Arduíno, funciona como a base do controlo do sistema de iluminação, é possível simplesmente com este módulo estabelecer uma ligação e controlar todas as luzes.

Figura 31-Módulo Bluetooth HC-05

Com o alcance de apenas 10 metros, este módulo foi suficiente para o desenvolvimento deste projeto além de ser uma opção económica.

3.3.4 - Integração com a aplicação de Controlo

O módulo Bluetooth anteriormente mencionado é a peça chave para a nossa aplicação comunicar com o sistema de iluminação. Desenvolvida em Android Studio e com a base em JAVA, a aplicação permite controlar a casa e também ler os valores dos sensores.

Sempre que o Arduino inicia o Bluetooth, fica disponível para receber conexões como demonstrado com o seguinte código:

```
void loop(){
  t.update();
  if (Bluetooth.available()>0){
    bluetoothConnection();
  }

  if (photoResistorActivation ==1 ){
    photoresistormeasure();
  }
}
```

Figura 32-Bluetooth disponível

Quando conectada a aplicação apresenta três opções:

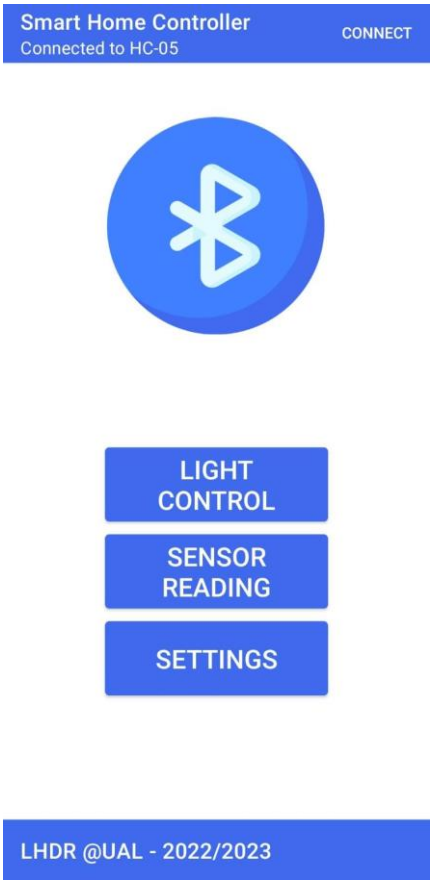


Figura 33-Opções da aplicação móvel

Caso a opção seja gerir a iluminação, recolher leituras dos sensores ou programar funcionalidades de forma automática utilizando as configurações predefinidas, a aplicação apresenta diversas opções para atender a estas necessidades.

Quando selecionamos o menu de controlo, será mostrada uma nova janela que irá apresentar as divisões da casa, ao clicar na divisão correspondente a luz acenderá ou apagar-se-á consoante o estado em que estiver.

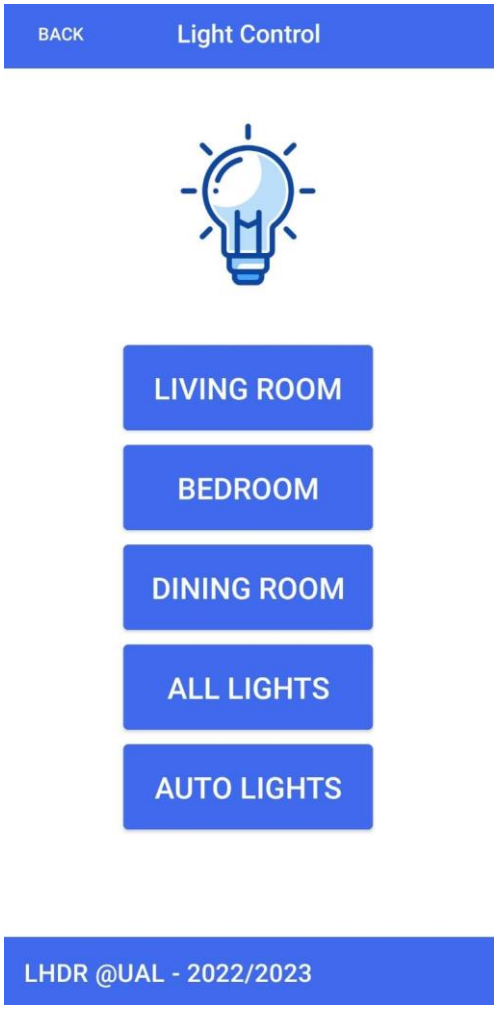


Figura 34-Divisões da casa na aplicação móvel

Na secção dos sensores e na sua leitura, serão apresentados os dados relativos à humidade, temperatura e luminosidade existente no momento. Facilitando assim a leitura dos mesmos e definirmos a partir de que valores queremos tornar o sistema automático.

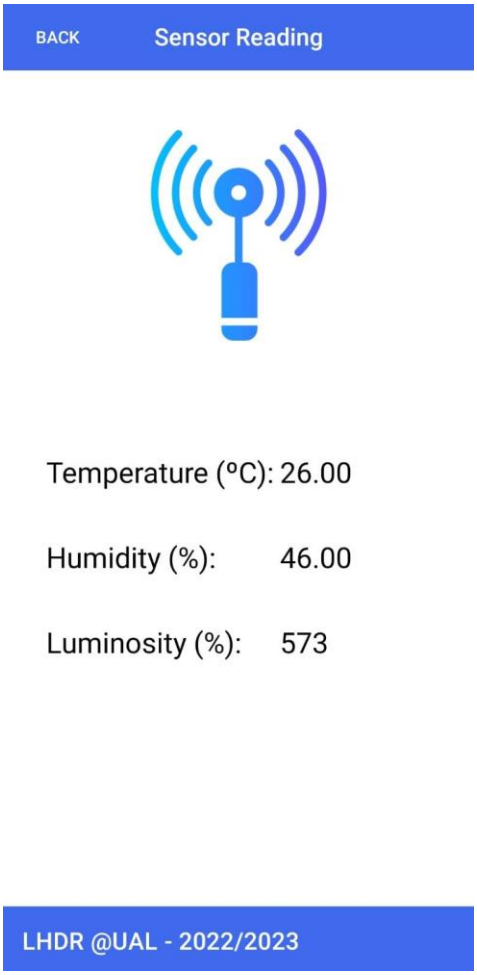


Figura 35-Informações dos sensores na aplicação móvel

Por fim, nas definições/*settings* é permitido configurar o sistema de iluminação inteligente. Encontramos nesta janela, duas opções, sendo a primeira, a definição dos limites no qual o sistema inteligente irá funcionar e caso sejam ultrapassados, as luzes de forma automática irão acender ou apagar, depende do caso. A segunda opção da aplicação é a definição de limites de temperatura, que resultará numa emissão de um alarme sonoro aquando estes forem ultrapassados.

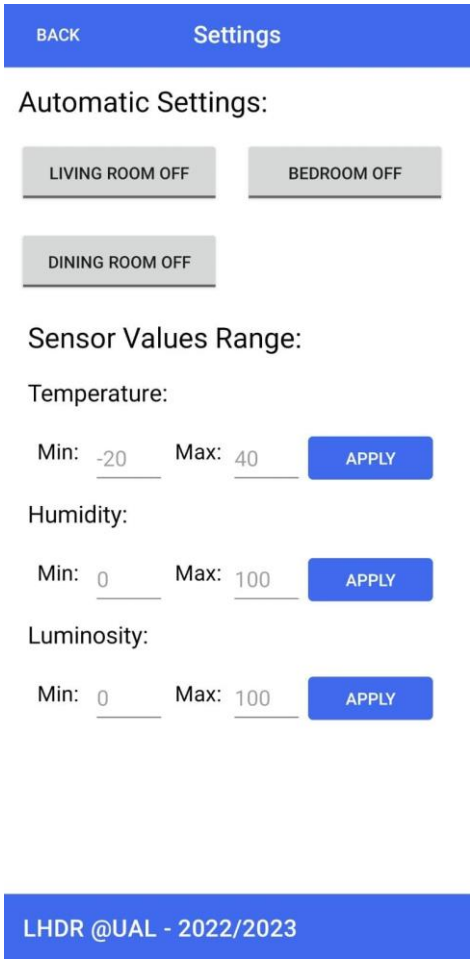


Figura 36-Definições da aplicação móvel

Este protótipo funciona com aplicações já presentes na *AppStore* da Google.

3.3.5 - Ajuste de Parâmetros e Configurações

O ajuste de parâmetros pode ser feito de duas formas distintas: modo programador com recurso a um computador e o ajuste no próprio código no Arduíno. Ou via aplicação de parâmetros de automatismo apenas. Na página das *settings* na aplicação, definimos os valores referência para os automatismos. Sempre que uma configuração é definida na aplicação, fica ativa até o Arduíno ser reiniciado. Uma vez reiniciado, volta ao código inicial e todos os automatismos têm de ser repetidos. Se for programado via computador, todas as definições ficam guardadas independentemente de ser reiniciado ou não.

4 - Testes e análise de resultados

Após concluída a implementação dos componentes, vários testes foram executados. Desde a implementação de outros sensores até a configuração de novos alarmes ou limites de temperatura excessivos. Contudo, nem todos os testes correram bem e foram precisas adaptações ao sistema.

4.1 - Testes e Tratamento de Falhas do Protótipo

4.1.1 - Teste de IOT

Foi testado e implementado um módulo IoT no projeto. No entanto, foi retirado devido a limitações. Para um módulo IoT funcionar corretamente, o utilizador teria de ter uma conta associada numa *Cloud*, o que envolvia mais custos além precisar de um maior conhecimento de como trabalhar com a aplicação. Por fim, a segurança, visto que este protótipo se destina a pessoas com uma mobilidade reduzida ou mais idosas, poderia ser um fator a ter em conta.

4.1.2 – Teste de humidade e temperatura

O teste realizado com o sensor de humidade e temperatura consistia na seguinte premissa: sempre que um determinado valor fosse ultrapassado, as luzes acendiam de forma intermitente. Contudo, esse teste não foi bem-sucedido, visto que o Arduíno teria de enviar sempre o código para as 3 luzes em simultâneo, fazendo com que o *display* ficasse com caracteres especiais. Mais tarde, optou-se pela remoção do código das luzes intermitente, passando a existir apenas o alarme sonoro com recurso a mais um elemento, o buzzer.

4.1.3 – Teste de luminosidade

Foi testada a luminosidade ambiente com o sensor de luminosidade e, baseado nos valores obtidos, foram definidos os limites máximos e mínimos, para que, de forma automática as luzes ligassem e desligassem. O teste foi considerado bem-sucedido.

4.1.2 - Limitação de portas disponíveis no Arduíno

Visto ser um projeto de escala reduzida, a limitação de portas foi um problema que se verificou. Todas as portas disponíveis no Arduíno foram ocupadas, o que impossibilita a adição de novos sensores. Contudo, se o objetivo fosse numa escala maior, poderia ter sido opção o Arduíno mega ou até mesmo colocar vários Arduíno uno em paralelo.

4.1.3 – Display com caracteres especiais

Durante a implementação, verificou-se que ao usar uma corrente de 220V no circuito existia ruido. A solução encontrada foi conectar uma fonte de energia externa. Contudo, não foi possível a sua implementação.

Figura 37-LCD com caracteres especiais

Uma outra solução passava por introduzir um *delay* quando fossem submetidos os comandos para acender ou apagar todas as luzes em simultâneo, de forma a fechar o circuito e não existir ruído.

O problema só se verifica quando é dada a instrução para acender ou apagar todas as luzes em simultâneo, individualmente não se verifica qualquer anomalia.

4.2 – Casos de uso

Para o protótipo foram executados vários casos de uso que visam facilitar a vida dos utilizadores ou sentirem-se mais seguros nas suas habitações. Este protótipo é também adequado para pessoas com capacidades reduzidas ou mais idosas, face à simplicidade de usar a aplicação e tornar o sistema automático.

4.2.1 - Ligar e desligar luzes com a aplicação

Com recurso à aplicação, é possível ter controlo total de toda a habitação para poder ligar ou desligar as luzes. A aplicação comunica por Bluetooth com o Arduíno e dá o comando de ligar ou desligar uma determinada divisão da casa.

Figura 38-LCD com informação sobre luz e divisão da casa

Ao ligar uma luz, um LED na caixa acende, assim como no painel de iluminação simulando uma divisão da casa que está a ser controlada remotamente.

Figura 39-diagrama de controlo de luzes

4.2.2 - Luminosidade baixa ou alta com o sensor

Com o anoitecer, vem a necessidade de ter iluminação artificial, como tal, podemos automatizar o Arduíno para após um limite mínimo de luminosidade, este emita um comando para a divisão que queremos que ligue automaticamente. Para ser automático, terá de ser previamente configurado na aplicação.

Os limites são lidos pelo sensor de luminosidade presente na parte superior do protótipo.

Figura 40-Sensor de luminosidade no protótipo

Também funciona no sentido inverso, sempre que as luzes estiverem ligadas e a luminosidade for superior, as luzes desligam automaticamente.

Figura 41-Diagrama de iluminação automática

4.2.3 - Temperatura alta e sinal sonoro

Uma das funções adicionais será o aviso em caso de temperatura alta, por exemplo num cenário de incêndio. Sempre que a temperatura for muito alta, um alarme sonoro é emitido através de um *buzzer* instalado no protótipo.

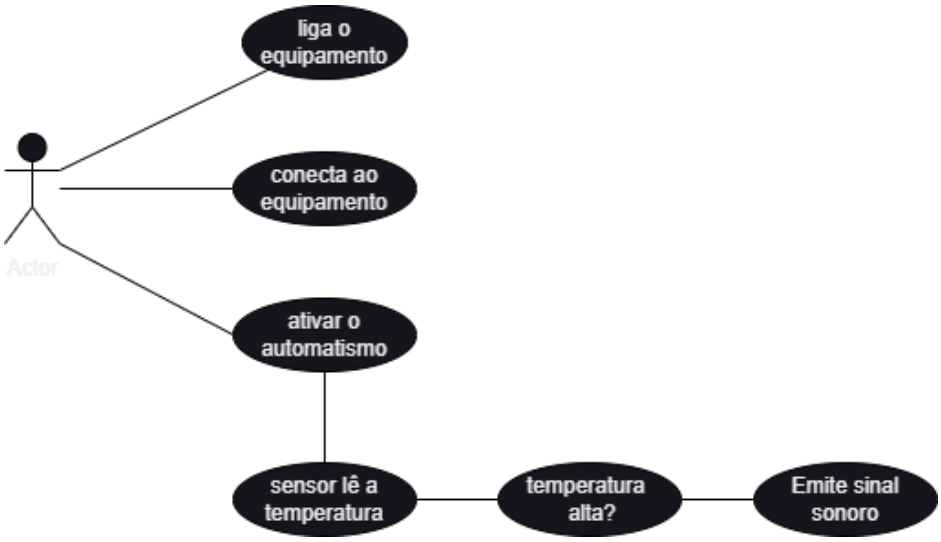


Figura 42-Diagrama de sensor de temperatura

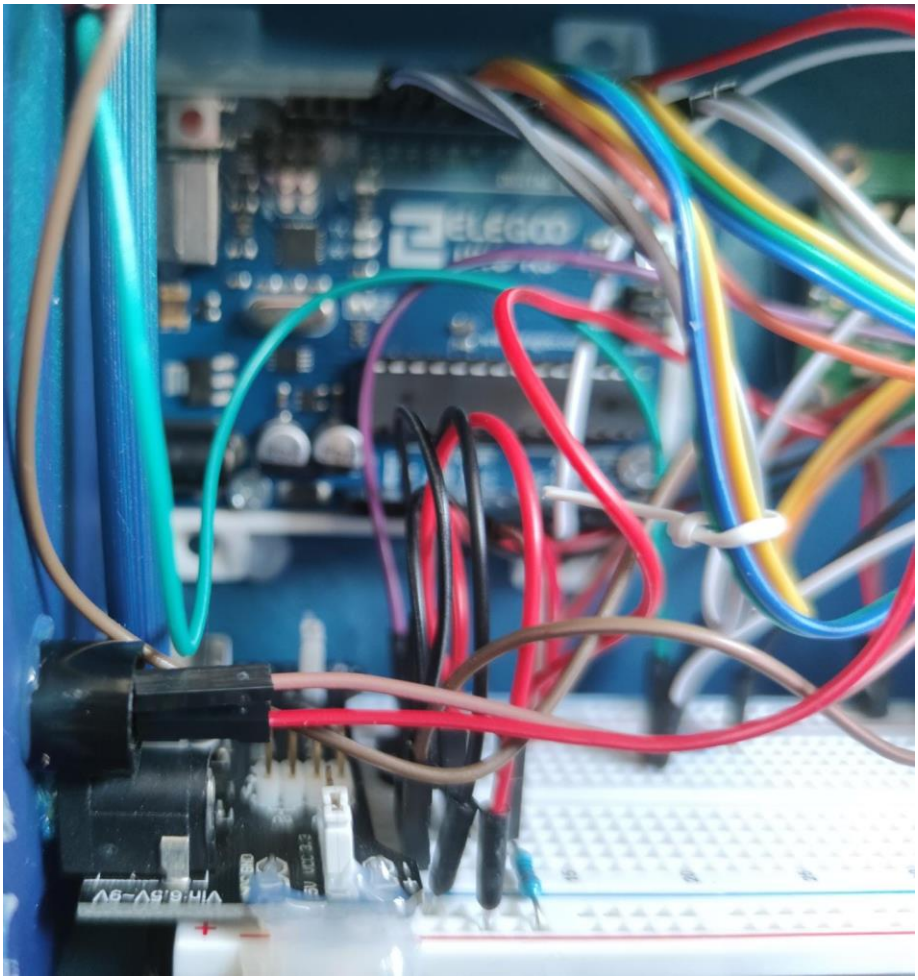


Figura 43-Buzzer no protótipo

5 – Conclusão

O trabalho apresentado demonstra um esforço significativo na criação de um sistema de iluminação inteligente baseado em Arduíno. Foi proposta uma metodologia e arquitetura para o sistema, foi feita construção de uma maquete impressa em 3D, implementou-se os componentes físicos do protótipo e desenvolveu-se uma aplicação móvel para o controlo do sistema.

A integração de diferentes componentes, como sensores de luminosidade, temperatura e humidade, juntamente com a comunicação Bluetooth e o uso de um display LCD, permitiu que o sistema monitorizasse e ajustasse a iluminação artificial com base nas condições ambientais. Além disso, a inclusão de um módulo de alerta sonoro em caso de temperaturas elevadas demonstra uma preocupação com a segurança e o conforto dos utilizadores. A criação da maquete impressa em 3D proporcionou uma representação visual realista do sistema, facilitou a demonstração e a visualização das funcionalidades. A utilização de uma Breadboard para a montagem dos componentes também foi uma escolha adequada para garantir a segurança e a facilidade na conexão dos elementos. A implementação da aplicação móvel, desenvolvida no Android Studio, permitiu o controlo remoto do sistema de iluminação. A possibilidade de configurar parâmetros e automatismos através da aplicação é uma vantagem, pois oferece flexibilidade aos utilizadores para personalizar o funcionamento do sistema de acordo com as suas preferências.No entanto, alguns pontos podem ser considerados para melhorias futuras. A limitação das portas disponíveis no Arduíno pode restringir a adição de novos sensores ou componentes. Seria interessante explorar alternativas para expandir as capacidades do sistema. Além disso, a inclusão de um módulo IoT poderia proporcionar uma maior conectividade e a facilidade de controlar remotamente, embora possa implicar desafios adicionais, como a segurança da rede.

No geral, o trabalho apresentado mostra um bom entendimento dos conceitos e tecnologias envolvidas na criação de um sistema de iluminação inteligente. A implementação física do protótipo, juntamente com a aplicação móvel de controlo, demonstra uma abordagem prática e funcional para atingir os objetivos propostos.

6 – Trabalhos Futuros

Deteção de presença: Adicionar sensores de movimento para detetar a presença de pessoas nas diferentes divisões da casa. Quando o sensor detetar movimento, as luzes podem ser automaticamente ligadas e, caso não haja movimento por um determinado período, as luzes podem ser desligadas.

Integração com assistentes de voz: Implementação com assistentes de voz, como o Amazon Alexa ou o Google assistant. Isso permitiria que o utilizador controle as luzes por meio de comandos de voz, tornando o sistema ainda mais conveniente e fácil de usar.

Programação horária: Acrescenta a capacidade de programar horários específicos para ligar e desligar as luzes automaticamente. Isso permitiria que o utilizador defina diferentes cenários de iluminação ao longo do dia, de acordo com suas preferências e necessidades.

Sensor de fumo: Além do sensor de temperatura atualmente existente, adicionar um sensor de fumo ao sistema para Deteção de incêndios. Quando o sensor detetar fumo, as luzes podem piscar ou emitir também um sinal sonoro para alerta.

Integração com sistemas de segurança: Explorar a possibilidade de integrar o sistema de iluminação inteligente com sistemas de segurança, como câmaras de vigilância. Por exemplo, quando uma câmara detetar um movimento suspeito, o sistema de iluminação pode ser configurado para ligar todas as luzes externas, .

7 – Bibliografia

- Baraka, K. et al (2013). Low Cost Arduino/Android-Based Energy-Efficient Home Automation System with Smart Task Scheduling - https://ieeexplore.ieee.org/document/6571382
- Kučera,E. et al (2018). Connection between 3D engine unity and microcontroller arduino: A virtual smart house - https://ieeexplore.ieee.org/abstract/document/8337531
- Söderby, K. (2023). Getting Started with Arduino. Arduino Documentation. - https://docs.arduino.cc/learn/starting-guide/getting-started-arduino
- Copes, F. (2020). Introduction to the Arduino Programming Language. Retrieved from https://flaviocopes.com/arduino-programming-language/
- Datasheet de Arduino https://docs.arduino.cc/static/cb6f768710d6e86336c223eae0f1ee2b/A000066-datasheet.pdf
- Instalação de IDE Arduino https://docs.arduino.cc/learn/starting-guide/the-arduino-software-ide
- Datasheet de DHT-11 https://www.arduino.cc/reference/en/libraries/dfrobot_dht11/
- Android Studio - temas https://m2.material.io/design/color/the-color-system.html#color-theme-creation
- Android Studio – layouts https://m2.material.io/design/layout/understanding-layout.html#principles
- Android Studio - guide app https://developer.android.com/guide/app-compatibility?hl=pt-br
- Android Studio - permissões Android https://developer.android.com/guide/topics/permissions/overview?hl=pt-br
- Android Studio – conectividade https://developer.android.com/guide/topics/connectivity/bluetooth?hl=pt-br

8 – Apêndice

Os guias de instalação e utilização para o funcionamento deste projeto.

1 - Manual de instalação do Hardware

LHDR – Smart Home

- Instalação do equipamento

Procedimento – Conexão à energia

1- Ligar o dispositivoà energia;

2- Ligar o dispositivo no botao On/Off.

Procedimento – Ligar o dispositivo

1- Verificar que o dispositivo está a funcionar;

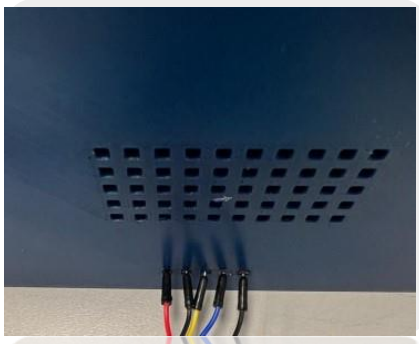
2- Ligar o dispositivo no botão On/Off.

Procedimento – Conexão ao painel de iluminação

1- Ligar o fio 5v à esquerda;

2- Ligar o fio GND à direita;

3- Ligar o trigger das lampadas nas 3 ligações centrais.



Procedimento – Manuseamento

Após o hardware estar instalado, proceder à ligação do Bluetooth através do dispositivo android.

LHDR – Smart Home

Obrigado

LHDR – Smart Home

Manual de instalação

Requisitos

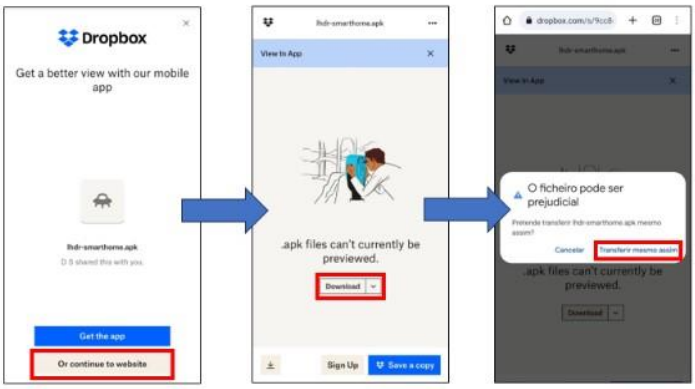
- Para instalar a aplicação “LHDR – Smart Home” o utilizador deverá possuir:

- Dispositivo Android com Bluetooth (recomendado versão de android superior ou igual a 10).
- Ligação à Internet.

Procedimento

Fazer download da aplicação (.apk) na hiperligação abaixo e seguir os passos conforme as imagens:

https://www.dropbox.com/s/9cc8qouwxmmlxzj/lhdrsmarthome.apk?dl=0



Procedimento

De seguida, o utilizador deverá procurar o ficheiro no "gestor de ficheiros" do equipamento e abrir a pasta "Downloads". Seguir imagens abaixo:

Procedimento

Na pasta Downloads o utilizador encontrará o ficheiro descarregado e deverá abri-lo, conforme nas imagens abaixo:

Procedimento

Ao abrir deverá surgir um aviso, pelo que deverá ser permitida instalação de fontes desconhecidas:

Após a instalação terminar, o utilizador poderá executar a aplicação através do seu ecrã principal ou menu de aplicações.

LHDR – Smart Home

Obrigado

3 - Manual de utilização do Software

LHDR – Smart Home

Manual do utilizador

Funcionalidades

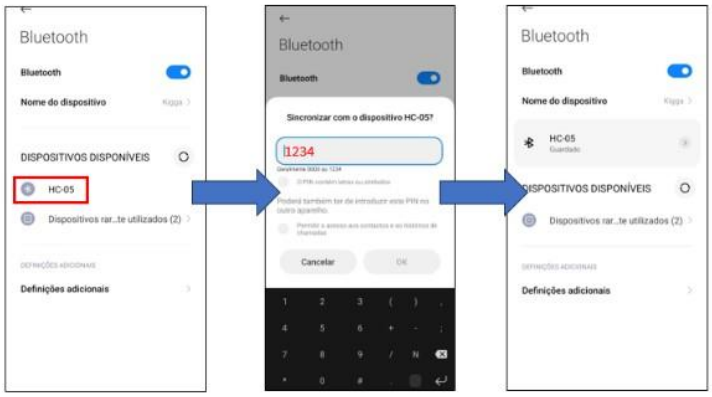
- Ligar e Desligar Luzes: Funcionalidade manual, em que o utilizador poderá acender ou apagar as luzes das divisões disponíveis.

- Luzes Automáticas: Funcionalidade em que as luzes se irão acender automaticamente a partir de um certo nível de luminosidade. A ativação e configuração desta opção é feita no menu "Settings". É permitido nesta opção a definição do mínimo e máximo de luminosidade ao qual o sensor irá reagir para ligar ou desligar as luzes.

- Alerta Incêndio: Funcionalidade automática em que o sistema irá soar um alarme, devido à temperatura e luminosidade estarem com os valores acima dos máximos configurados.

Procedimento - Conexão

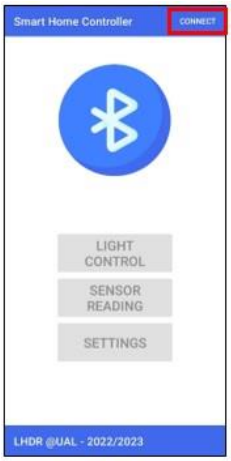
Antes de executar a aplicação, deverá emparelhar o seu equipamento ao Arduíno no menu de Bluetooth, utilizando o pin "1234":



Após emparelhar, poderá abrira aplicação "LHDR – Smart Home".

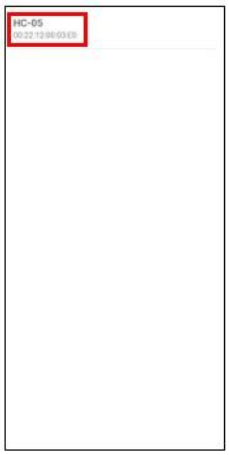
Procedimento - Conexão

Ao executar a aplicação, irá surgir o menu principal da aplicação, pelo que para utilizar as funções e potencialidades da aplicação o utilizador deverá clicar no botão " **Connect".**



Procedimento - Conexão

Neste ecrã deverá selecionar o equipamento com nome “HC -05”. Após clicar, deverá ser redirecionado para a página anterior e deverá ter todos os botões desbloqueados. Caso falhe, o emparelhamento deverá validar se o Arduíno está ligado e/ou deverá tentar nova ligação.



Procedimento - Conexão

Quando conectado, todos os botões ficam azuis e prontos para serem escolhidos; temos 3 opções: Light Control; sensor Reading e settings.

Light Control

No primeiro botão "Light Control", o utilizador irá ser redirecionado para uma página onde o utilizador poderá ligar e desligar as luzes de qualquer divisão.

Botão "**Living Room**" – Ao clicar, o utilizador irá ligar ou desligar a luz referente à Sala de Estar.

Botão "**Bedroom**" – Ao clicar, o utilizador irá ligar ou desligar a luz referente ao Quarto.

Botão "**Dining Room**" – Ao clicar, o utilizador irá ligar ou desligar a luz referente à Sala de Jantar.

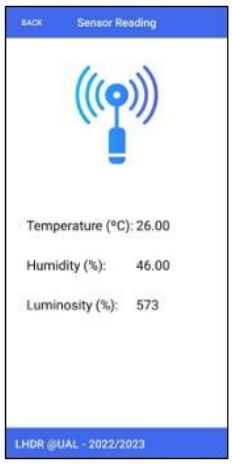
Botão "**All Lights**" – Ao clicar, o utilizador irá ligar ou desligar todas as luzes em simultâneo.

Botão "**Auto Lights**" – Ao clicar, o utilizador irá ligar ou desligar a funcionalidade de Luzes Automáticas.

Botão "**Back**" – Ao clicar, o utilizador será redirecionado para a página inicial.

Sensor Reading

No segundo botão "Sensor Reading", o utilizador irá ser redirecionado para uma página onde o utilizador poderá consultar os valores fornecidos pelos sensores implementados no Arduíno (Temperatura, Humidade e Luminosidade).



Temperature (ºC) – Valor captado pelo Arduíno relativamente à temperatura atual, em graus Celsius.

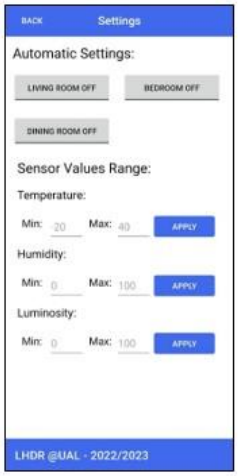
Humidity (%) – Valor captado pelo Arduíno relativamente à humidade atual, em percentagem.

Luminosity (ºC) – Valor captado pelo Arduíno relativamente à luminosidade atual, em percentagem.

Botão "**Back**" – Ao clicar, o utilizador será redirecionado para a página inicial.

Settings

No terceiro botão “ Settings”, o utilizador irá ser redirecionado para uma página onde o utilizador poderá configurar em que divisões pretende ter ativa a funcionalidade de luzes automáticas e configurar os valores máximos e mínimos, para a Temperatura, Humidade e Luminosidade.



Automatic Settings – Configuração estado ON/OFF o botão referente a cada divisão, conforme o utilizador pretenda que as divisões apresentadas sejam utilizadas ou não pela funcionalidade de “LuzesAutomáicas”.

Sensor Values Range – Configuração dos valores mínimos e máximos da temperatura, humidade e luminosidade. Após editar os valores que o utilizador pretende, deverá clicar no botão “Apply”, para enviara informação para o Arduíno.

Botão “**Back**” – Ao clicar, o utilizador será redirecionado para a página inicial.

LHDR – Smart Home

Obrigado

4 – Código da Aplicação Android

4.1 – Código de configuração Manifest

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- Permissions for Bluetooth access -->
    <uses-permission android:name="android.permission.BLUETOOTH" />
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"
/>

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="LHDR Smart Home"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".SelectDeviceActivity" />
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

4.2 – Código de Aplicação MainActivity

```
package com.lhdr.bluetoothconn;
import androidx.appcompat.app.AppCompatActivity;
import androidx.appcompat.widget.Toolbar;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
import android.content.Intent;
import android.os.Bundle;
import android.os.Handler;
import android.os.Looper;
import android.os.Message;
import android.util.Log;
import android.view.View;
import android.view.animation.Animation;
import android.view.animation.AnimationUtils;
import android.widget.Button;
import android.widget.CompoundButton;
import android.widget.EditText;
import android.widget.ProgressBar;
import android.widget.TextView;
```

```
import android.widget.ToggleButton;
import android.widget.ViewFlipper;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.Timer;
import java.util.TimerTask;
import java.util.UUID;
import static android.content.ContentValues.TAG;

public class MainActivity extends AppCompatActivity {

    private String deviceName = null;
    private String deviceAddress;
    public static Handler handler;
    public static BluetoothSocket mmSocket;
    public static ConnectedThread connectedThread;
    public static CreateConnectThread createConnectThread;
    private final static int CONNECTING_STATUS = 1; // used in bluetooth
handler to identify message status
    private final static int MESSAGE_READ = 2; // used in bluetooth handler
to identify message update

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // UI Initialization
        final ViewFlipper viewFlipper = findViewById(R.id.viewFlipper);
        Animation slideInAnimation = AnimationUtils.loadAnimation(this,
R.anim.slide_in);
        Animation slideOutAnimation = AnimationUtils.loadAnimation(this,
R.anim.slide_out);
        viewFlipper.setInAnimation(slideInAnimation);
        viewFlipper.setOutAnimation(slideOutAnimation);
        viewFlipper.setDisplayedChild(0);

        //First Screen Elements
        final Button buttonConnect = findViewById(R.id.buttonConnect);
        final Toolbar toolbar = findViewById(R.id.toolbar);
        final ProgressBar progressBar = findViewById(R.id.progressBar);
        progressBar.setVisibility(View.GONE);
        final Button lightsButton = findViewById(R.id.lightsButton);
        lightsButton.setEnabled(false);
        final Button buttonToggle2 = findViewById(R.id.buttonToggle2);
        buttonToggle2.setEnabled(false);
        final Button buttonToggle3 = findViewById(R.id.buttonToggle3);
        buttonToggle3.setEnabled(false);
        //First Screen Elements End

        //Second Screen Elements
        final Button buttonBack4 = findViewById(R.id.buttonBack4);
        final Button buttonToggle4 = findViewById(R.id.buttonToggle4);
        final Button buttonToggle5 = findViewById(R.id.buttonToggle5);
        final Button buttonToggle7 = findViewById(R.id.buttonToggle7);
        final Button buttonToggleAll = findViewById(R.id.buttonToggleA);
```

```
        final TextView textViewInfoLights = findViewById(R.id.textViewInfo3);
        final Button buttonToggleAutoLightsAll =
findViewById(R.id.buttonToggleAutoLightsAll);
        //Second Screen Elements End

        //Third Screen Elements
        final Button buttonBack2 = findViewById(R.id.buttonBack2);
        final TextView textViewTempValue =
findViewById(R.id.textViewTempValue);
        final TextView textViewHumValue =
findViewById(R.id.textViewHumValue);
        final TextView textViewLumValue =
findViewById(R.id.textViewLumValue);
        //Third Screen Elements End

        //Fourth Screen Elements
        final Button buttonBackSettings =
findViewById(R.id.buttonBackSettings);;
        final Button applytemp = findViewById(R.id.applytemp);
        final Button applyhum = findViewById(R.id.applyhum);
        final Button applylum = findViewById(R.id.applylum);
        final EditText tempmin = findViewById(R.id.tempmin);
        final EditText tempmax = findViewById(R.id.tempmax);
        final EditText hummin = findViewById(R.id.hummin);
        final EditText hummax = findViewById(R.id.hummax);
        final EditText lummin = findViewById(R.id.lummin);
        final EditText lummax = findViewById(R.id.lummax);
        final ToggleButton dinTB = findViewById(R.id.dinTB);
        final ToggleButton bedTB = findViewById(R.id.bedTB);
        final ToggleButton livTB = findViewById(R.id.livTB);
        //Fourth Screen Elements End


        // If a bluetooth device has been selected from SelectDeviceActivity
        deviceName = getIntent().getStringExtra("deviceName");
        if (deviceName != null){
            // Get the device address to make BT Connection
            deviceAddress = getIntent().getStringExtra("deviceAddress");
            // Show progress and connection status
            toolbar.setSubtitle("Connecting to " + deviceName + "...");
            progressBar.setVisibility(View.VISIBLE);
            buttonConnect.setEnabled(false);

            /*
            When "deviceName" is found
            the code will call a new thread to create a bluetooth connection
to the
            selected device
             */
            BluetoothAdapter bluetoothAdapter =
BluetoothAdapter.getDefaultAdapter();
            createConnectThread = new
CreateConnectThread(bluetoothAdapter,deviceAddress);
            createConnectThread.start();
        }
```

```
        //GUI Handler
        handler = new Handler(Looper.getMainLooper()) {
            @Override
            public void handleMessage(Message msg){
                switch (msg.what){
                    case CONNECTING_STATUS:
                        switch(msg.arg1){
                            case 1:
                                toolbar.setSubtitle("Connected to " +
deviceName);
                                progressBar.setVisibility(View.GONE);
                                buttonConnect.setEnabled(true);
                                lightsButton.setEnabled(true);
                                buttonToggle2.setEnabled(true);
                                buttonToggle3.setEnabled(true);
                                connectedThread.write("k;");
                                break;
                            case -1:
                                toolbar.setSubtitle("Device fails to
connect");
                                progressBar.setVisibility(View.GONE);
                                buttonConnect.setEnabled(true);
                                break;
                        }
                        break;

                    case MESSAGE_READ:
                        String arduinoMsg = msg.obj.toString(); // Read
message from Arduino
                        String[] splitMsg = arduinoMsg.split(" ");


                        switch (splitMsg[0]) {
                            //Startup Configs
                            case "configs":
                                tempmin.setText(splitMsg[2]);
                                tempmax.setText(splitMsg[3]);
                                hummin.setText(splitMsg[6]);
                                hummax.setText(splitMsg[7]);
                                lummin.setText(splitMsg[10]);
                                lummax.setText(splitMsg[11]);
                                break;
                            case "Living":
                            case "Bedroom":
                            case "Dining":
                            case "ALL":
                            case "Auto":
                                textViewInfoLights.setText(arduinoMsg);
                                Timer timer = new Timer();
                                timer.schedule(new TimerTask() {
                                    @Override
                                    public void run() {
                                        textViewInfoLights.setText("");
                                    }
                                }, 2000);
                                break;
```

```
                            case "Temperature":
                                textViewTempValue.setText(splitMsg[1]);
                                textViewHumValue.setText(splitMsg[3]);
                                break;
                            case "Luminosity":
                                textViewLumValue.setText(splitMsg[1]);
                                break;
                        }
                        break;
                    }
                }
            };
            // Select Bluetooth Device
            buttonConnect.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View view) {
                    // Move to adapter list
                    Intent intent = new Intent(MainActivity.this,
SelectDeviceActivity.class);
                    startActivity(intent);
                }
            });

            // Back Buttons Action
            buttonBack2.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View view) {
                    viewFlipper.setDisplayedChild(0);
                }
            });
            buttonBack4.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View view) {
                    viewFlipper.setDisplayedChild(0);
                }
            });
            buttonBackSettings.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View view) {
                    viewFlipper.setDisplayedChild(0);
                }
            });

            //Main Page Buttons Action
            lightsButton.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View view) {
                    viewFlipper.setDisplayedChild(1);
                }
            });
            buttonToggle2.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View view) {
                    viewFlipper.setDisplayedChild(2);
                }
            });
            buttonToggle3.setOnClickListener(new View.OnClickListener() {
```

```
            @Override
            public void onClick(View view) {
                viewFlipper.setDisplayedChild(3);
            }
        });

        /* ============================ Light Control Screen Action
================================== */
        buttonToggle4.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                String cmdText = "4;";
                connectedThread.write(cmdText);
            }
        });

        buttonToggle5.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                String cmdText = "5;";
                connectedThread.write(cmdText);
            }
        });

        buttonToggle7.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                String cmdText = "6;";
                connectedThread.write(cmdText);
            }
        });
        buttonToggleAll.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                String cmdText = "0;";
                connectedThread.write(cmdText);
            }
        });
        buttonToggleAutoLightsAll.setOnClickListener(new
View.OnClickListener() {
            @Override
            public void onClick(View view) {
                String cmdText = "1;";
                connectedThread.write(cmdText);
            }
        });

        dinTB.setOnCheckedChangeListener(new
CompoundButton.OnCheckedChangeListener() {
            @Override
            public void onCheckedChanged(CompoundButton buttonView, boolean
isChecked) {
                // Check if the toggle button is checked
                if (isChecked) {
                    String cmdText = "D 1;";
                    connectedThread.write(cmdText);
                } else if (!isChecked){
```

```
                String cmdText = "D 0;";
                connectedThread.write(cmdText);
            }
        }
    });
    bedTB.setOnCheckedChangeListener(new
CompoundButton.OnCheckedChangeListener() {
        @Override
        public void onCheckedChanged(CompoundButton buttonView, boolean
isChecked) {
            // Check if the toggle button is checked
            if (isChecked) {
                String cmdText = "E 1;";
                connectedThread.write(cmdText);
            } else if (!isChecked){
                String cmdText = "E 0;";
                connectedThread.write(cmdText);
            }

        }
    });
    livTB.setOnCheckedChangeListener(new
CompoundButton.OnCheckedChangeListener() {
        @Override
        public void onCheckedChanged(CompoundButton buttonView, boolean
isChecked) {
            // Check if the toggle button is checked
            if (isChecked) {
                String cmdText = "F 1;";
                connectedThread.write(cmdText);
            } else if (!isChecked){
                String cmdText = "F 0;";
                connectedThread.write(cmdText);
            }
        }
    });

    applytemp.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            String min = tempmin.getText().toString();
            String max = tempmax.getText().toString();
            String toSend = "A" + "," + min + "," + max + ";";
            connectedThread.write(toSend);
        }
    });
    applyhum.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            String min = hummin.getText().toString();
            String max = hummax.getText().toString();
            String toSend = "B" + "," + min + "," + max + ";";
            connectedThread.write(toSend);
        }
    });
    applylum.setOnClickListener(new View.OnClickListener() {
        @Override
```

```
            public void onClick(View view) {
                String min = lummin.getText().toString();
                String max = lummax.getText().toString();
                String toSend = "C" + "," + min + "," + max + ";";
                connectedThread.write(toSend);
            }
        });


    }

    /* ============================ Thread to Create Bluetooth Connection
================================== */
    public static class CreateConnectThread extends Thread {

        public CreateConnectThread(BluetoothAdapter bluetoothAdapter, String
address) {
            /*
            Use a temporary object that is later assigned to mmSocket
            because mmSocket is final.
             */
            BluetoothDevice bluetoothDevice =
bluetoothAdapter.getRemoteDevice(address);
            BluetoothSocket tmp = null;
            UUID uuid = bluetoothDevice.getUuids()[0].getUuid();

            try {
                /*
                Get a BluetoothSocket to connect with the given
BluetoothDevice.
                 */
                tmp =
bluetoothDevice.createInsecureRfcommSocketToServiceRecord(uuid);

            } catch (IOException e) {
                Log.e(TAG, "Socket's create() method failed", e);
            }
            mmSocket = tmp;
        }

        public void run() {
            // Cancel discovery because it otherwise slows down the
connection.
            BluetoothAdapter bluetoothAdapter =
BluetoothAdapter.getDefaultAdapter();
            bluetoothAdapter.cancelDiscovery();
            try {
                // Connect to the remote device through the socket. This call
blocks
                // until it succeeds or throws an exception.
                mmSocket.connect();
                Log.e("Status", "Device connected");
                handler.obtainMessage(CONNECTING_STATUS, 1, -
1).sendToTarget();
            } catch (IOException connectException) {
                // Unable to connect; close the socket and return.
                try {
```

```
                    mmSocket.close();
                    Log.e("Status", "Cannot connect to device");
                    handler.obtainMessage(CONNECTING_STATUS, -1, -
1).sendToTarget();
                } catch (IOException closeException) {
                    Log.e(TAG, "Could not close the client socket",
closeException);
                }
                return;
            }

            // The connection attempt succeeded. Perform work associated with
            // the connection in a separate thread.
            connectedThread = new ConnectedThread(mmSocket);
            connectedThread.run();
        }

        // Closes the client socket and causes the thread to finish.
        public void cancel() {
            try {
                mmSocket.close();
            } catch (IOException e) {
                Log.e(TAG, "Could not close the client socket", e);
            }
        }
    }

    /* ============================== Thread for Data Transfer
========================================= */
    public static class ConnectedThread extends Thread {
        private final BluetoothSocket mmSocket;
        private final InputStream mmInStream;
        private final OutputStream mmOutStream;

        public ConnectedThread(BluetoothSocket socket) {
            mmSocket = socket;
            InputStream tmpIn = null;
            OutputStream tmpOut = null;

            // Get the input and output streams, using temp objects because
            try {
                tmpIn = socket.getInputStream();
                tmpOut = socket.getOutputStream();
            } catch (IOException e) { }

            mmInStream = tmpIn;
            mmOutStream = tmpOut;
        }

        public void run() {
            byte[] buffer = new byte[1024];  // buffer store for the stream
            int bytes = 0; // bytes returned from read()
            // Keep listening to the InputStream until an exception occurs
            while (true) {
                try {
```

```
 /* from the InputStream from Arduino until termination character is reached.
                    Then send the whole String message to GUI Handler.
                     */
                    buffer[bytes] = (byte) mmInStream.read();
                    String readMessage;
                    if (buffer[bytes] == '\n'){
                        readMessage = new String(buffer,0,bytes);
                        Log.e("Arduino Message",readMessage);

handler.obtainMessage(MESSAGE_READ,readMessage).sendToTarget();
                        bytes = 0;
                    } else {
                        bytes++;
                    }
                } catch (IOException e) {
                    e.printStackTrace();
                    break;
                }
            }
        }

        /* Call this from the main activity to send data to the remote device
*/
        public void write(String input) {
            byte[] bytes = input.getBytes(); //converts entered String into
bytes
            try {
                mmOutStream.write(bytes);
            } catch (IOException e) {
                Log.e("Send Error","Unable to send message",e);
            }
        }

        /* Call this from the main activity to shutdown the connection */
        public void cancel() {
            try {
                mmSocket.close();
            } catch (IOException e) { }
        }
    }

    /* ============================ Terminate Connection at BackPress
====================== */
    @Override
    public void onBackPressed() {
        // Terminate Bluetooth Connection and close app
        if (createConnectThread != null){
            createConnectThread.cancel();
        }
        Intent a = new Intent(Intent.ACTION_MAIN);
        a.addCategory(Intent.CATEGORY_HOME);
        a.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
        startActivity(a);
    }
}
```

4.3 – Código de Bluetooth Device Info

```
package com.lhdr.bluetoothconn;

public class DeviceInfoModel {

    private String deviceName, deviceHardwareAddress;

    public DeviceInfoModel(){}

    public DeviceInfoModel(String deviceName, String deviceHardwareAddress){
        this.deviceName = deviceName;
        this.deviceHardwareAddress = deviceHardwareAddress;
    }

    public String getDeviceName(){return deviceName;}

    public String getDeviceHardwareAddress(){return deviceHardwareAddress;}

}
```

4.4 – Código de Bluetooth Device List

```
package com.lhdr.bluetoothconn;
import android.content.Intent;
import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.LinearLayout;
import android.widget.TextView;
import androidx.recyclerview.widget.RecyclerView;
import java.util.List;

public class DeviceListAdapter extends
RecyclerView.Adapter<RecyclerView.ViewHolder> {
    private Context context;
    private List<Object> deviceList;

    public static class ViewHolder extends RecyclerView.ViewHolder {
        TextView textName, textAddress;
        LinearLayout linearLayout;

        public ViewHolder(View v) {
            super(v);
            textName = v.findViewById(R.id.textViewDeviceName);
            textAddress = v.findViewById(R.id.textViewDeviceAddress);
            linearLayout = v.findViewById(R.id.linearLayoutDeviceInfo);
        }
    }
    public DeviceListAdapter(Context context, List<Object> deviceList) {
        this.context = context;
        this.deviceList = deviceList;
```

```
    public RecyclerView.ViewHolder onCreateViewHolder(ViewGroup parent, int
viewType) {
        View v =
LayoutInflater.from(parent.getContext()).inflate(R.layout.device_info_layout,
parent, false);
        ViewHolder vh = new ViewHolder(v);
        return vh;
    }
    @Override
    public void onBindViewHolder(final RecyclerView.ViewHolder holder, final
int position) {
        ViewHolder itemHolder = (ViewHolder) holder;
        final DeviceInfoModel deviceInfoModel = (DeviceInfoModel)
deviceList.get(position);
        itemHolder.textName.setText(deviceInfoModel.getDeviceName());

itemHolder.textAddress.setText(deviceInfoModel.getDeviceHardwareAddress());

        // When a device is selected
        itemHolder.linearLayout.setOnClickListener(new View.OnClickListener()
{
            @Override
            public void onClick(View view) {
                Intent intent = new Intent(context,MainActivity.class);
                // Send device details to the MainActivity
                intent.putExtra("deviceName",
deviceInfoModel.getDeviceName());

intent.putExtra("deviceAddress",deviceInfoModel.getDeviceHardwareAddress());
                // Call MainActivity
                context.startActivity(intent);
            }
        });
    }

    @Override
    public int getItemCount() {
        int dataCount = deviceList.size();
        return dataCount;
    }
}
```

4.5 – Código de Bluetooth Select Device

```
package com.lhdr.bluetoothconn;
import androidx.appcompat.app.AppCompatActivity;
import androidx.recyclerview.widget.DefaultItemAnimator;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.os.Bundle;
import android.view.View;
import com.google.android.material.snackbar.Snackbar;
```

```
import java.util.ArrayList;
import java.util.List;
import java.util.Set;

public class SelectDeviceActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_select_device);

        // Bluetooth Setup
        BluetoothAdapter bluetoothAdapter =
BluetoothAdapter.getDefaultAdapter();

        // Get List of Paired Bluetooth Device
        Set<BluetoothDevice> pairedDevices =
bluetoothAdapter.getBondedDevices();
        List<Object> deviceList = new ArrayList<>();
        if (pairedDevices.size() > 0) {
            // There are paired devices. Get the name and address of each
paired device.
            for (BluetoothDevice device : pairedDevices) {
                String deviceName = device.getName();
                String deviceHardwareAddress = device.getAddress(); // MAC
address
                DeviceInfoModel deviceInfoModel = new
DeviceInfoModel(deviceName,deviceHardwareAddress);
                deviceList.add(deviceInfoModel);
            }
            // Display paired device using recyclerView
            RecyclerView recyclerView =
findViewById(R.id.recyclerViewDevice);
            recyclerView.setLayoutManager(new LinearLayoutManager(this));
            DeviceListAdapter deviceListAdapter = new
DeviceListAdapter(this,deviceList);
            recyclerView.setAdapter(deviceListAdapter);
            recyclerView.setItemAnimator(new DefaultItemAnimator());
        } else {
            View view = findViewById(R.id.recyclerViewDevice);
            Snackbar snackbar = Snackbar.make(view, "Activate Bluetooth or
pair a Bluetooth device", Snackbar.LENGTH_INDEFINITE);
            snackbar.setAction("OK", new View.OnClickListener() {
                @Override
                public void onClick(View view) { }
            });
            snackbar.show();
        }
    }
}
```

5 – Código do Arduino

```
#include "Timer.h"
#include <Arduino.h>
#include <Wire.h>
#include <SoftwareSerial.h>
#include <DHT11.h>
#include <LiquidCrystal.h>
LiquidCrystal lcd_1(8,9,10,11,12,13);
#define DHTPIN 7


DHT11 dht(DHTPIN);
int livingroom = 4;
int bedroom = 5;
int diningroom = 6;
int fan = 3;
int timer=5000;
int sensorValue = 0;
int photoResistorActivation =0;
int lumSensorMin=200;
int lumSensorMax=700;
int auto4=1;
int auto5=1;
int auto6=1;
int tempmin=-1000;
int tempmax=60;
int hummin =0;
int hummax=100;
int alarm =3;
String inputBuffer = "";
Timer t;

SoftwareSerial Bluetooth(0, 1);
char Data;
void sendData(String transmitData){
Bluetooth.println(transmitData);}

void setup(){
  // set up the LCD's number of columns and rows:
  lcd_1.begin(16, 2);
  Bluetooth.begin(9600);
  pinMode(A0, INPUT);
```

```
  pinMode(livingroom,OUTPUT);
  pinMode(bedroom,OUTPUT);
  pinMode(fan,OUTPUT);
  pinMode(diningroom,OUTPUT);
  pinMode(alarm,OUTPUT);
  lcd_1.setCursor(0, 0);
  lcd_1.print("  Initializing  ");
  for (int a = 5; a <= 10; a++) {
    lcd_1.setCursor(a, 1);
    lcd_1.print(".");
    delay(500);
  }


  //Call the function
  t.every(10000, tempHum);
  t.every(7000, lumsensor);

}

void tempHum(){
  float h = dht.readHumidity();
  float t = dht.readTemperature();

  if (isnan(h) || isnan(t)) {
    Serial.println(F("Failed to read from DHT sensor!"));
    return;
  }else if ((t >= tempmax || t <= tempmin) && analogRead(A0) >= lumSensorMax){
    digitalWrite(alarm,1);
  }else if ((t <= tempmax || t >= tempmin) && analogRead(A0) <= lumSensorMax){
    digitalWrite(alarm,0);
  }



  Bluetooth.println("Temperature "+ String(t)+ " Humidity " + String(h) );
  lcd_1.clear();
  lcd_1.setCursor(0, 0);
  lcd_1.print("T:");
  lcd_1.print(t);

  lcd_1.setCursor(8, 0);
  lcd_1.print("H:");
  lcd_1.print(h);
```

```
}

void lumsensor(){
  sensorValue = analogRead(A0);
  Bluetooth.println("Luminosity " +String(sensorValue));
}


void Lcdscroll(){
  for (int positionCounter = 0; positionCounter < 13; positionCounter++) {
      // scroll one position left:
      lcd_1.scrollDisplayLeft();
      // wait a bit:
      delay(500);
    }
}

void photoresistormeasure(){
  sensorValue = analogRead(A0);
  if(sensorValue<lumSensorMax && (digitalRead(livingroom)==0
||digitalRead(bedroom)==0 || digitalRead(diningroom)==0 )){
 //   Bluetooth.println("sensor"+sensorValue);
//    Bluetooth.println("treshold"+lumSensorMax);
    if(auto4==1){
      digitalWrite(livingroom,1);
    }
    if(auto5==1){
      digitalWrite(bedroom,1);}
    if(auto6==1){
      digitalWrite(diningroom,1);}
    lcd_1.setCursor(0, 1);
    lcd_1.print("                ");
    lcd_1.setCursor(0, 1);
    lcd_1.print("Auto Lights ON");
    sendData("Auto Lights ON");
    return;
  }else if(sensorValue>lumSensorMax && (digitalRead(livingroom)==1
||digitalRead(bedroom)==1 || digitalRead(diningroom)==1 ) ){
        if(auto4==1){
      digitalWrite(livingroom,0);
    }
    if(auto5==1){
      digitalWrite(bedroom,0);}
    if(auto6==1){
```

```
      digitalWrite(diningroom,0);}
    lcd_1.setCursor(0, 1);
    lcd_1.print("                ");
    lcd_1.setCursor(0, 1);
    lcd_1.print("Auto Lights OFF");
    sendData("ALL LIGHTS OFF");
    return;
  }
}

void bluetoothConnection() {
  while (Bluetooth.available() > 0) {
    char c = Bluetooth.read();

    // Check for termination character
    if (c == ';') {
      inputBuffer.trim();  // Remove leading/trailing whitespace
      Bluetooth.println("Received command: " + inputBuffer);
      processCommand(inputBuffer);
      inputBuffer = "";  // Reset input buffer
    } else {
      inputBuffer += c;
    }
  }
}


void processCommand(const String& command) {
  Data = command[0];
  Bluetooth.println("Received command: " + command);
    if (Data == '4') {
    //  Bluetooth.println(digitalRead(livingroom));
      if(digitalRead(livingroom)==0){
        digitalWrite(livingroom,1);
        lcd_1.setCursor(0, 1);
        lcd_1.print("                ");
        lcd_1.setCursor(0, 1);
        lcd_1.print("Living Room ON");
        sendData("Living Room Light ON");

      } else{
        digitalWrite(livingroom,0);
        lcd_1.setCursor(0, 1);
        lcd_1.print("                ");
        lcd_1.setCursor(0, 1);
```

```
        lcd_1.print("Living Room OFF");
        sendData("Living Room Light OFF");

      }

    } else if (Data == '5') {
    if(digitalRead(bedroom)==0){
    digitalWrite(bedroom,1);
    lcd_1.setCursor(0, 1);
    lcd_1.print("                ");
    lcd_1.setCursor(0, 1);
    lcd_1.print("Bedroom ON ");
    sendData("Bedroom ON");
    return;
    } else{
    digitalWrite(bedroom,0);
    lcd_1.setCursor(0, 1);
    lcd_1.print("                ");
    lcd_1.setCursor(0, 1);
    lcd_1.print("Bedroom OFF");
    sendData("Bedroom OFF");
    }

    } else if (Data == '6') {
      if(digitalRead(diningroom)==0){
        digitalWrite(diningroom,1);
        lcd_1.setCursor(0, 1);
        lcd_1.print("                ");
        lcd_1.setCursor(0, 1);
        lcd_1.print("Dining Room ON");
        sendData("Dining Room ON");
        return;
      } else{
        digitalWrite(diningroom,0);
        lcd_1.setCursor(0, 1);
        lcd_1.print("                ");
        lcd_1.setCursor(0, 1);
        lcd_1.print("Dining Room OFF");
        sendData("Dining Room OFF");
      }

    } else if (Data == '0') {
      if(digitalRead(livingroom)==1 || digitalRead(bedroom)==1 ||
digitalRead(diningroom)==1){
        digitalWrite(livingroom,0);
```

```
    digitalWrite(bedroom,0);
    digitalWrite(diningroom,0);
    lcd_1.setCursor(0, 1);
    lcd_1.print("                ");
    lcd_1.setCursor(0, 1);
    lcd_1.print("ALL LIGHTS OFF");
    sendData("ALL LIGHTS OFF");
    return;
  }else{
    digitalWrite(livingroom,1);
    digitalWrite(bedroom,1);
    digitalWrite(diningroom,1);
    lcd_1.setCursor(0, 1);
    lcd_1.print("                ");
    lcd_1.setCursor(0, 1);
    lcd_1.print("ALL LIGHTS ON");
    sendData("ALL LIGHTS ON");

  }


} else if (Data == '1') {
  if(photoResistorActivation==1){
    photoResistorActivation = 0;
    lcd_1.setCursor(0, 1);
    lcd_1.print("                ");
    lcd_1.setCursor(0, 1);
    lcd_1.print("Auto Light OFF");
    sendData("Auto Light OFF");
    return;
  }
  else{
    photoResistorActivation = 1;
    lcd_1.setCursor(0, 1);
    lcd_1.print("                ");
    lcd_1.setCursor(0, 1);
    lcd_1.print("Auto Light ON");
    sendData("Auto Light ON");
    return;
  }



}
```

```
    else if (Data == 'M') {
      int maxvalue=parseIntegerValue(command);
      if(maxvalue <= lumSensorMin){
        sendData("ERROR! Max value below min value. Your Min value is:" +
String(lumSensorMin)+ " Please use a value above that!");

      }else{
        lumSensorMax = parseIntegerValue(command);
        sendData("Max value updated: " + String(lumSensorMax));
        sendData("Your lights will turn on when lumminance is below " +
String(lumSensorMax) + " units");
      }

    } else if (Data == 'm') {
      int minvalue=parseIntegerValue(command);
      if(minvalue >= lumSensorMin){
        sendData("ERROR! Min value above Max value. Your Max value is:" +
String(lumSensorMax)+ " Please use a value above that!");

      }else{
        lumSensorMin = parseIntegerValue(command);
        sendData("Min value updated: " + String(lumSensorMin));
        sendData("Your lights will turn off when lumminance is above " +
String(lumSensorMin) + " units");
      }
    }else if(Data == 'k'){
      //send data temp
      sendData ("configs temp " + String(tempmin) +" " +String(tempmax)+ "
configs hum "+  String(hummin) +" "+  String(hummax)+" configs lum " +
String(lumSensorMin) +" " + String(lumSensorMax) +" configs auto " +
String(auto4) + " " + String(auto5)+" "+ String(auto6)+ " end");
    }else if(Data == 'A'){
      int ind1 = command.indexOf( ',');
      int ind2 = command.indexOf(',', ind1+1 );
      tempmin = command.substring(ind1+1, ind2).toInt();
      tempmax = command.substring(ind2+1).toInt();

    }else if(Data == 'B'){
      int ind3 = command.indexOf( ',');
      int ind4 = command.indexOf(',', ind3+1 );
      hummin = command.substring(ind3+1, ind4).toInt();
      hummax = command.substring(ind4+1).toInt();
    }
    else if(Data == 'C'){
```

```
      int ind5 = command.indexOf( ',');
      int ind6 = command.indexOf(',', ind5+1 );
      lumSensorMin = command.substring(ind5+1, ind6).toInt();
      lumSensorMax= command.substring(ind6+1).toInt();
    }
    else if(Data == 'D'){
       auto4 = parseIntegerValue(command);
    }
    else if(Data == 'E'){
      auto5 = parseIntegerValue(command);
    }
    else if(Data == 'F'){
      auto6 = parseIntegerValue(command);
    }



    else{sendData("Invalid Command please check again!");

    }

}

int parseIntegerValue(const String& command) {
  String valueStr = command.substring(1);  // Extract value part of the command
  return valueStr.toInt();
}



void loop(){
  t.update();
 // Bluetooth.println("hello word");
  if (Bluetooth.available()>0){
    bluetoothConnection();
  }
  if (photoResistorActivation ==1 ){
    photoresistormeasure();
  }
}
```

9 - Anexos

Datasheet do Arduíno

Product Reference Manual

SKU: A000066

Description

The Arduino UNO R3 is the perfect board to get familiar with electronics and coding. This versatile microcontroller is equipped with the well-known ATmega328P and the ATMega 16U2 Processor. This board will give you a great first experience within the world of Arduino.

Target areas:

Maker, introduction, industries

Features

- **ATMega328P** Processor
  - **Memory**
    - AVR CPU at up to 16 MHz
    - 32KB Flash
    - 2KB SRAM
    - 1KB EEPROM
  - **Security**
    - Power On Reset (POR)
    - Brown Out Detection (BOD)
  - **Peripherals**
    - 2x 8-bit Timer/Counter with a dedicated period register and compare channels
    - 1x 16-bit Timer/Counter with a dedicated period register, input capture and compare channels
    - 1x USART with fractional baud rate generator and start-of-frame detection
    - 1x controller/peripheral Serial Peripheral Interface (SPI)
    - 1x Dual mode controller/peripheral I2C
    - 1x Analog Comparator (AC) with a scalable reference input
    - Watchdog Timer with separate on-chip oscillator
    - Six PWM channels
    - Interrupt and wake-up on pin change
  - **ATMega16U2 Processor**
    - 8-bit AVR® RISC-based microcontroller
  - **Memory**
    - 16 KB ISP Flash
    - 512B EEPROM

512B SRAM

debugWIRE interface for on-chip debugging and programming

Power

2.7-5.5 volts

CONTENTS

The UNO board is the flagship product of Arduino. Regardless if you are new to the world of electronics or will use the UNO as a tool for education purposes or industry-related tasks.

First entry to electronics: If this is your first project within coding and electronics, get started with our most used and documented board; Arduino UNO. It is equipped with the well-known ATmega328P processor, 14 digital input/output pins, 6 analog inputs, USB connections, ICSP header and reset button. This board includes everything you will need for a great first experience with Arduino.

Industry-standard development board: Using the Arduino UNO board in industries, there are a range of companies using the UNO board as the brain for their PLC's.

Education purposes: Although the UNO board has been with us for about ten years, it is still widely used for various education purposes and scientific projects. The board's high standard and top quality performance makes it a great resource to capture real time from sensors and to trigger complex laboratory equipment to mention a few examples.

Related Products

- Starter Kit
- Tinkerkit Braccio Robot
- Example

Ratings

Recommended Operating Conditions

Symbol	Description	Min	Max
	Conservative thermal limits for the whole board:	-40 °C (-40°F)	85 °C (185°F)

NOTE: In extreme temperatures, EEPROM, voltage regulator, and the crystal oscillator, might not work as expected due to the extreme temperature conditions

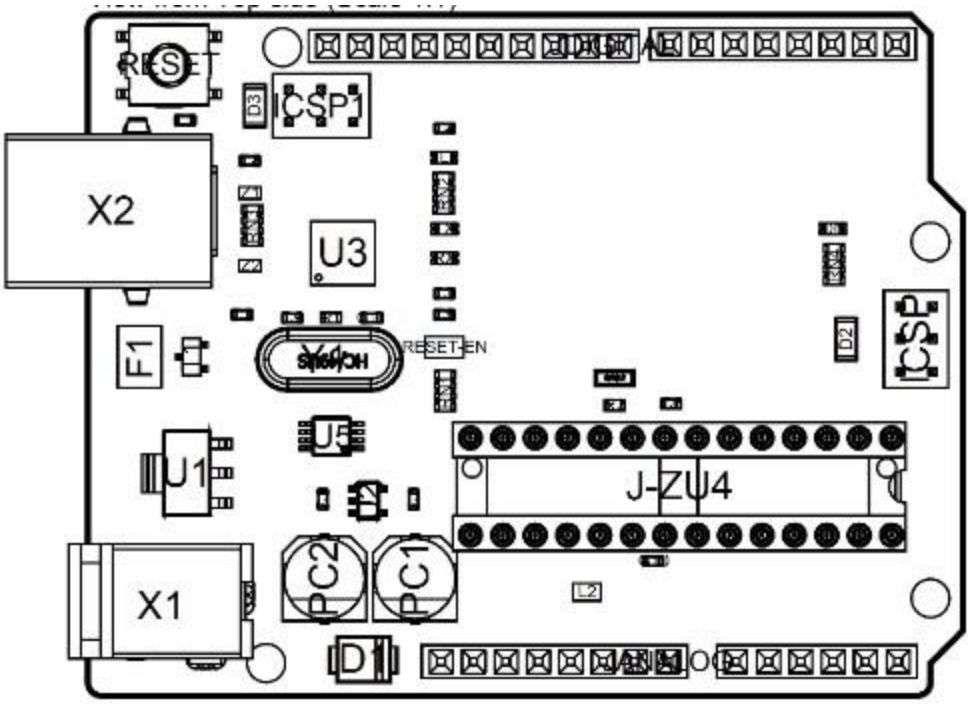
Power Consumption

Symbol	Description	Min	Typ	Max	Unit
VINMax	Maximum input voltage from VIN pad	6	-	20	V
VUSBMax	Maximum input voltage from USB connector		-	5.5	V
PMax	Maximum Power Consumption	-	-	xx	mA

Functional Overview

Board Topology

Top view



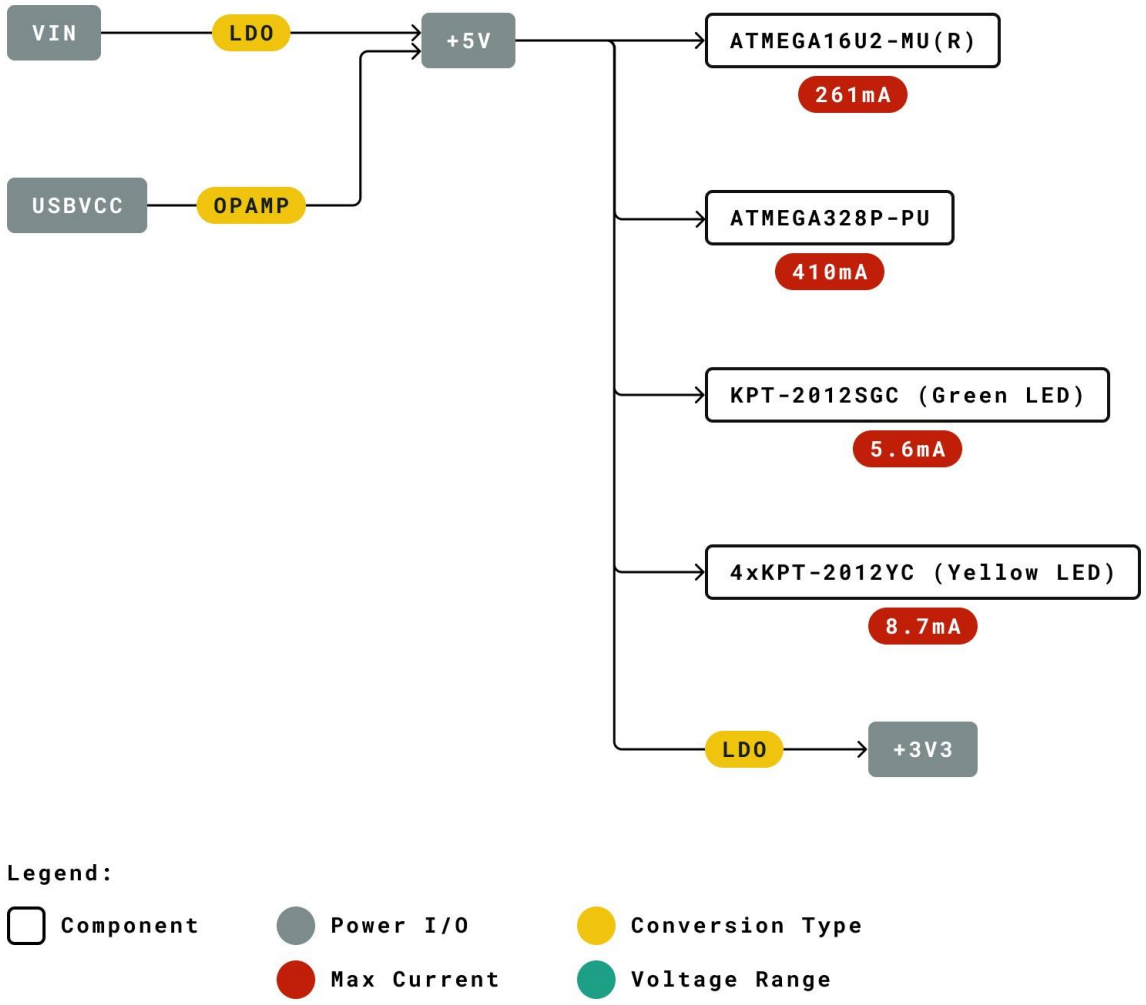
Board topology

Ref.	Description	Ref.	Description
X1	Power jack 2.1x5.5mm	U1	SPX1117M3-L-5 Regulator
X2	USB B Connector	U3	ATMEGA16U2 Module
PC1	EEE-1EA470WP 25V SMD Capacitor	U5	LMV358LIST-A.9 IC
PC2	EEE-1EA470WP 25V SMD Capacitor	F1	Chip Capacitor, High Density
D1	CGRA4007-G Rectifier	ICSP	Pin header connector (through hole 6)
J-ZU4	ATMEGA328P Module	ICSP1	Pin header connector (through hole 6)
Y1	ECS-160-20-4X-DU Oscillator		

Processor

The Main Processor is a ATmega328P running at up tp 20 MHz. Most of its pins are connected to the external headers, however some are reserved for internal communication with the USB Bridge coprocessor.

Power Tree



Power tree

Board Operation

Getting Started - IDE

If you want to program your Arduino UNO while offline you need to install the Arduino Desktop IDE [1] To connect the Arduino UNO to your computer, you'll need a Micro-B USB cable. This also provides power to the board, as indicated by the LED.

Getting Started - Arduino Web Editor

All Arduino boards, including this one, work out-of-the-box on the Arduino Web Editor [2], by just installing a simple plugin.

The Arduino Web Editor is hosted online, therefore it will always be up-to-date with the latest features and support for all boards. Follow **[3]** to start coding on the browser and upload your sketches onto your board.

Getting Started - Arduino IoT Cloud

All Arduino IoT enabled products are supported on Arduino IoT Cloud which allows you to Log, graph and analyze sensor data, trigger events, and automate your home or business.

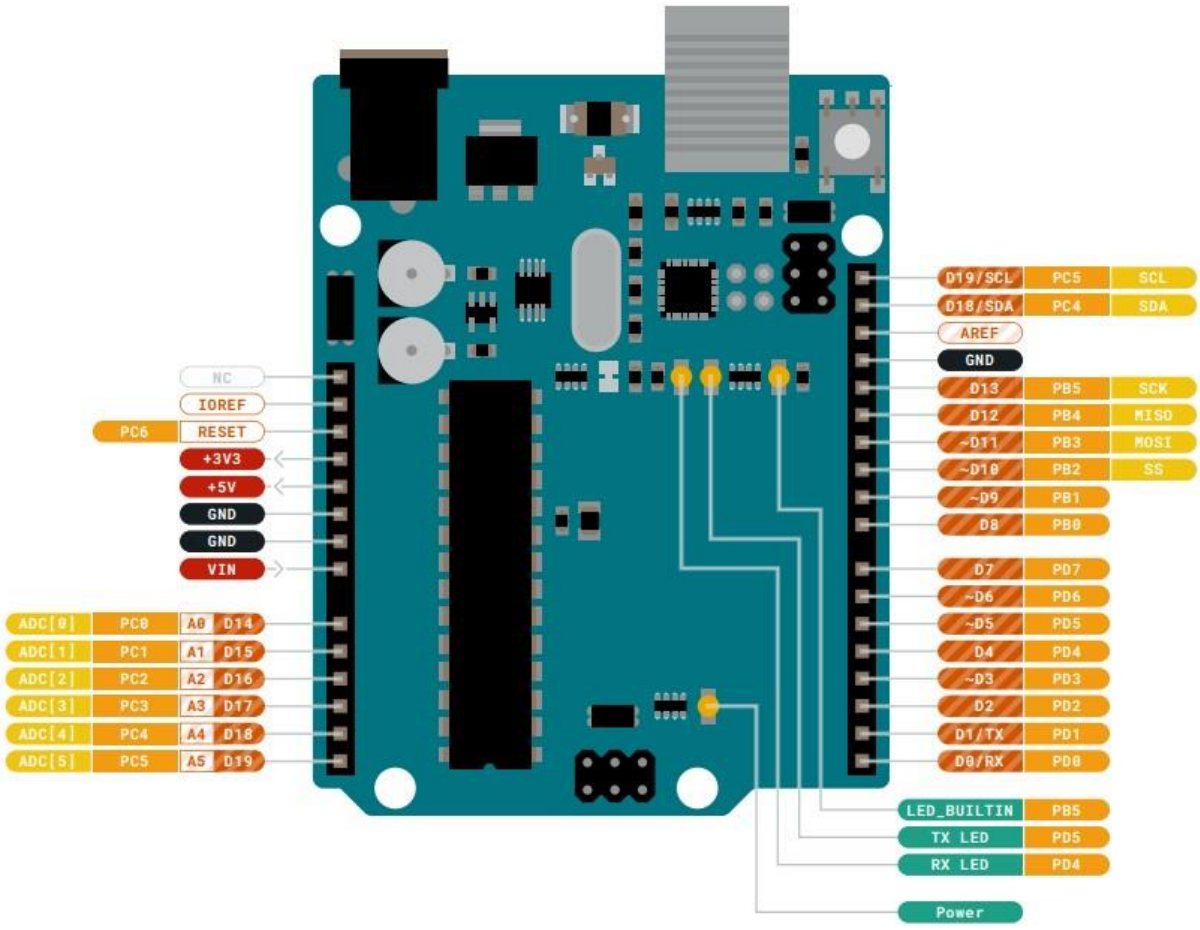
Sample Sketches

Sample sketches for the Arduino XXX can be found either in the “Examples” menu in the Arduino IDE or in the “Documentation” section of the Arduino Pro website [4]

Online Resources

Now that you have gone through the basics of what you can do with the board you can explore the endless possibilities it provides by checking exciting projects on ProjectHub **[5]**, the Arduino Library Reference **[6]** and the online store **[7]** where you will be able to complement your board with sensors, actuators and more

Connector Pinouts



Pinout

JANALOG

Pin	Function	Type	Description
1	NC	NC	Not connected
2	IOREF	IOREF	Reference for digital logic V - connected to 5V
3	Reset	Reset	Reset
4	+3V3	Power	+3V3 Power Rail
5	+5V	Power	+5V Power Rail
6	GND	Power	Ground
7	GND	Power	Ground
8	VIN	Power	Voltage Input
9	A0	Analog/GPIO	Analog input 0 /GPIO
10	A1	Analog/GPIO	Analog input 1 /GPIO

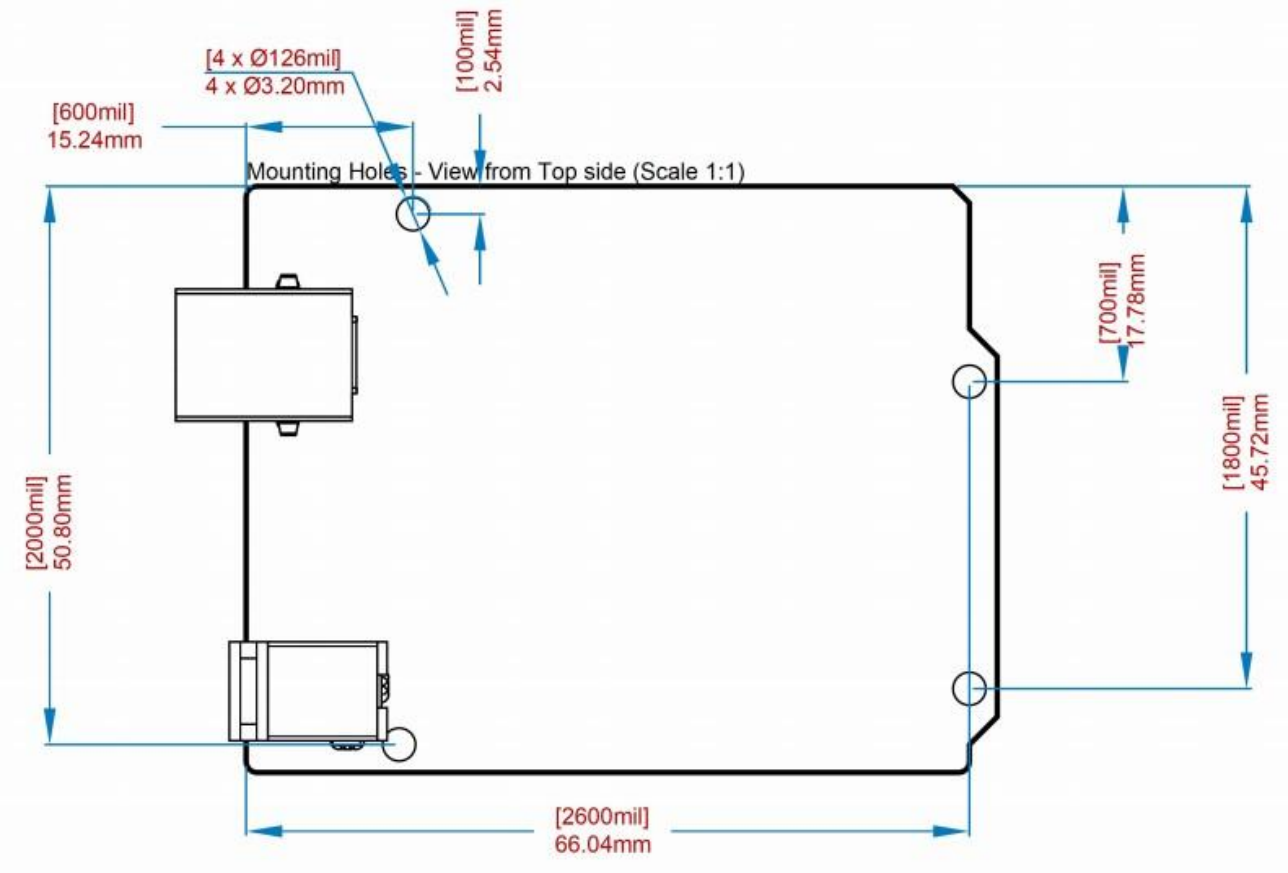
11	A2	Analog/GPIO	Analog input 2 /GPIO
12	A3	Analog/GPIO	Analog input 3 /GPIO
13	A4/SDA	Analog input/I2C	Analog input 4/I2C Data line
14	A5/SCL	Analog input/I2C	Analog input 5/I2C Clock line

JDIGITAL

Pin	Function	Type	Description
1	D0	Digital/GPIO	Digital pin 0/GPIO
2	D1	Digital/GPIO	Digital pin 1/GPIO
3	D2	Digital/GPIO	Digital pin 2/GPIO
4	D3	Digital/GPIO	Digital pin 3/GPIO
5	D4	Digital/GPIO	Digital pin 4/GPIO
6	D5	Digital/GPIO	Digital pin 5/GPIO
7	D6	Digital/GPIO	Digital pin 6/GPIO
8	D7	Digital/GPIO	Digital pin 7/GPIO
9	D8	Digital/GPIO	Digital pin 8/GPIO
10	D9	Digital/GPIO	Digital pin 9/GPIO
11	SS	Digital	SPI Chip Select
12	MOSI	Digital	SPI1 Main Out Secondary In
13	MISO	Digital	SPI Main In Secondary Out
14	SCK	Digital	SPI serial clock output
15	GND	Power	Ground
16	AREF	Digital	Analog reference voltage
17	A4/SD4	Digital	Analog input 4/I2C Data line (duplicated)
18	A5/SD5	Digital	Analog input 5/I2C Clock line (duplicated)

Mechanical Information

Board Outline & Mounting Holes



Board outline

Certifications

Declaration of Conformity CE DoC (EU)

We declare under our sole responsibility that the products above are in conformity with the essential requirements of the following EU Directives and therefore qualify for free movement within markets comprising the European Union (EU) and European Economic Area (EEA).

ROHS 2 Directive 2011/65/EU	
Conforms to:	EN50581:2012
Directive 2014/35/EU. (LVD)	
Conforms to:	EN 60950-1:2006/A11:2009/A1:2010/A12:2011/AC:2011
Directive 2004/40/EC & 2008/46/EC & 2013/35/EU, EMF	

Conforms to:	EN 62311:2008

Declaration of Conformity to EU RoHS & REACH 211 01/19/2021

Arduino boards are in compliance with RoHS 2 Directive 2011/65/EU of the European Parliament and RoHS 3 Directive 2015/863/EU of the Council of 4 June 2015 on the restriction of the use of certain hazardous substances in electrical and electronic equipment.

Substance	**Maximum limit (ppm)**
Lead (Pb)	1000
Cadmium (Cd)	100
Mercury (Hg)	1000
Hexavalent Chromium (Cr6+)	1000
Poly Brominated Biphenyls (PBB)	1000
Poly Brominated Diphenyl ethers (PBDE)	1000
Bis(2-Ethylhexyl} phthalate (DEHP)	1000
Benzyl butyl phthalate (BBP)	1000
Dibutyl phthalate (DBP)	1000
Diisobutyl phthalate (DIBP)	1000

Exemptions: No exemptions are claimed.

Arduino Boards are fully compliant with the related requirements of European Union Regulation (EC) 1907 /2006 concerning the Registration, Evaluation, Authorization and Restriction of Chemicals (REACH). We declare none of the SVHCs (https://echa.europa.eu/web/guest/candidate-list-table), the Candidate List of Substances of Very High Concern for authorization currently released by ECHA, is present in all products (and also package) in quantities totaling in a concentration equal or above 0.1%. To the best of our knowledge, we also declare that our products do not contain any of the substances listed on the "Authorization List" (Annex XIV of the REACH regulations) and Substances of Very High Concern (SVHC) in any significant amounts as specified by the Annex XVII of Candidate list published by ECHA (European Chemical Agency) 1907 /2006/EC.

Conflict Minerals Declaration

As a global supplier of electronic and electrical components, Arduino is aware of our obligations with regards to laws and regulations regarding Conflict Minerals, specifically the Dodd-Frank Wall Street Reform and Consumer Protection Act, Section 1502. Arduino does not directly source or process conflict minerals such as Tin, Tantalum, Tungsten, or Gold. Conflict minerals are contained in our products in the form of solder, or as a component in metal alloys. As part of our reasonable due diligence Arduino has contacted component suppliers within our supply chain to verify their continued compliance with the

regulations. Based on the information received thus far we declare that our products contain Conflict Minerals sourced from conflict-free areas.

FCC Caution

Any Changes or modifications not expressly approved by the party responsible for compliance could void the user's authority to operate the equipment.

This device complies with part 15 of the FCC Rules. Operation is subject to the following two conditions:

(1) This device may not cause harmful interference

(2) this device must accept any interference received, including interference that may cause undesired operation.

FCC RF Radiation Exposure Statement:

1. This Transmitter must not be co-located or operating in conjunction with any other antenna or transmitter.
2. This equipment complies with RF radiation exposure limits set forth for an uncontrolled environment.
3. This equipment should be installed and operated with minimum distance 20cm between the radiator & your body.

English: User manuals for license-exempt radio apparatus shall contain the following or equivalent notice in a conspicuous location in the user manual or alternatively on the device or both. This device complies with Industry Canada license-exempt RSS standard(s). Operation is subject to the following two conditions:

(1) this device may not cause interference

(2) this device must accept any interference, including interference that may cause undesired operation of thedevice.

French: Le présent appareil est conforme aux CNR d'Industrie Canada applicables aux appareils radio exempts de licence. L'exploitation est autorisée aux deux conditions suivantes :

(1) l' appareil nedoit pas produire de brouillage

(2) l'utilisateur de l'appareil doit accepter tout brouillage radioélectrique subi, même si le brouillage est susceptibled'en compromettre le fonctionnement.

IC SAR Warning:

English This equipment should be installed and operated with minimum distance 20 cm between the radiator and your body.

French: Lors de l' installation et de l' exploitation de ce dispositif, la distance entre le radiateur et le corps est d 'au moins 20 cm.

Important: The operating temperature of the EUT can't exceed 85℃ and shouldn't be lower than -40℃.

Hereby, Arduino S.r.l. declares that this product is in compliance with essential requirements and other relevant provisions of Directive 2014/53/EU. This product is allowed to be used in all EU member states.

Company Information

Company name	**Arduino S.r.l**
Company Address	Via Andrea Appiani 25 20900 MONZA Italy

Reference Documentation

Reference	**Link**
Arduino IDE (Desktop)	https://www.arduino.cc/en/Main/Software
Arduino IDE (Cloud)	https://create.arduino.cc/editor
Cloud IDE Getting Started	https://create.arduino.cc/projecthub/Arduino_Genuino/getting-started-with-arduinoweb-editor-4b3e4a
Arduino Pro Website	https://www.arduino.cc/pro
Project Hub	https://create.arduino.cc/projecthub?by=part&part_id=11332&sort=trending
Library Reference	https://www.arduino.cc/reference/en/
Online Store	https://store.arduino.cc/

Revision History

Date	**Revision**	**Changes**
xx/06/2021	1	Datasheet release