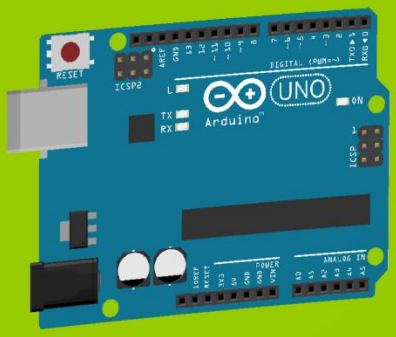
ANDROID APPS FOR ARDUINO

with MIT App Inventor 2



Build 8 Android applications that can interact with the Arduino board via bluetooth to control outputs, read sensors and display messages

Written by Rui Santos & Sara Santos

ANDROID APPS FOR ARDUINO

Security Notice

This is the kind of thing I hate having to write about, but the evidence is clear: piracy for digital products is over all the internet.

For that reason I've taken certain steps to protect my intellectual property contained in this eBook.

This eBook contains hidden random strings of text that only apply to *your* specific eBook version that is unique to your email address. You probably won't see anything different, since those strings are hidden in this PDF. I apologize for having to do that – **but it means if someone were to share this eBook I know exactly who shared it and I can take further legal consequences.**

You cannot redistribute this eBook. **This eBook is for personal use and** is only available for purchase at:

- http://randomnerdtutorials.com/products
- https://rntlab.com

Please send an email to the author (Rui Santos - hello@ruisantos.me), if you found this eBook anywhere else.

What I really want to say is thank you for purchasing this eBook and I hope you have fun with it!

Disclaimer

This eBook has been written for information purposes only. Every effort has been made to make this eBook as complete and accurate as possible. The purpose of this eBook is to educate. The authors (Rui Santos and Sara Santos) do not warrant that the information contained in this eBook is fully complete and shall not be responsible for any errors or omissions.

The authors (Rui Santos and Sara Santos) shall have neither liability nor responsibility to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by this eBook.

Throughout this eBook you will find some links and some of them are affiliate links. This means the authors (Rui Santos and Sara Santos) earn a small commission from each purchase with that link. Please understand that the authors have experience with all of these products, and he recommends them because they are useful, not because of the small commissions he makes if you decide to buy something. Please do not spend any money on these products unless you feel you need them.

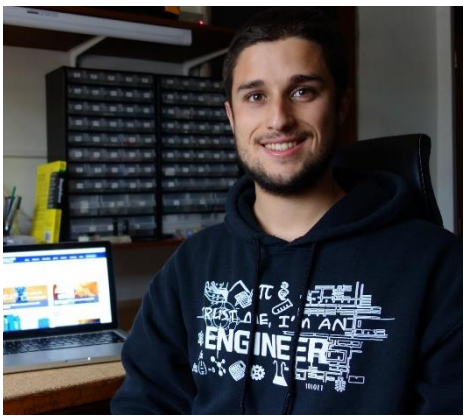
Other Helpful Links:

- Join Private Facebook Group
- Terms and Conditions

About the Authors

Hey there,

Thank you for reading the “Build Android Apps for Arduino with MIT App Inventor 2” course!



> *“I’m Rui Santos, an electrical engineer student and blogger. I have more than four years of experience teaching electronics and programming with the Random Nerd Tutorials blog. I’m also founder of RNTLab.com and author of BeagleBone For Dummies. Most of my projects and tutorials are related wiht Arduino, Home Automation, ESP8266, and Raspberry Pi.”*

> *“I’m Sara Santos, I have a master’s degree in Bioengineering and I’ve been working with Rui at Random Nerd Tutorials since 2015. My main tasks at Random Nerd tutorials include: recording and editing video, photo shooting, content creation, making projects, proofreading, etc… I also write and build courses with Rui.”*

Contents

Course Intro

Welcome to **Build Android Apps for Arduino** *with MIT App Inventor 2*. In this course you'll learn how to build Android apps that interact with your Arduino board.

This is a practical course in which you'll build 8 different Arduino projects.

Build Android Apps for Arduino starts with a quick introduction to Arduino and MIT App Inventor 2 and gets you building Android apps in no time by following step by step instructions with detailed screenshots, schematics and code.

Download Apps Resources

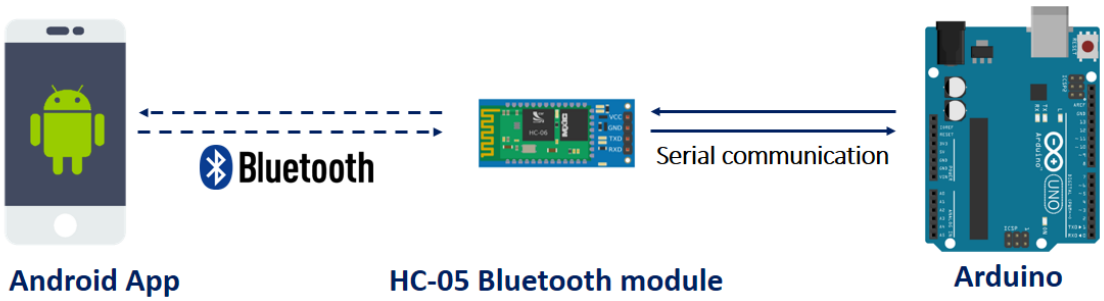
Each project contains everything you need:

- Step by step instructions for building the app design and logic
- Schematics
- Downloadable code
- Downloadable assets
- *.aia* files (files that you can upload to the MIT App Inventor 2 software to edit the app)
- *.apk* files (files that you can upload to your smartphone for quickly running the app)

Press the button below to download all the resources or open this link in your browser: https://github.com/RuiSantosdotme/arduino-mit-app-inventor/archive/master.zip

Click here to download resources

Introduction



Introduction

This quick introduction shows you what you need to make Android apps for Arduino with MIT App Inventor 2.

What do you need?

For making Android apps for Arduino you need these components:

- Computer with internet connection;
- Smartphone with internet/Wi-Fi connection to quickly test your apps;
- Arduino UNO or other similar Arduino board;
- HC-05/HC-04/HC-06 bluetooth module;
- Assorted electronics components:
    - Breadboard;
    - Jumper wires;
    - LEDs;
    - Resistors;
    - Temperature sensor;
    - Relay module;
    - ...

Each project contains a detailed list with all the components you need.

Introducing MIT App Inventor 2

You're going to use to a software called MIT App Inventor 2 to build the apps. You don't have to download or install a program in your computer as the software is cloud-based, so you build the apps directly in your browser (Chrome, Mozilla, Safari, Internet Explorer, etc). For instance, you only need an internet connection for building the apps.

You can install an app in your smartphone called **MIT AI2 Companion** to quickly test your apps (this is optional). This will be covered later in the projects section.

Why MIT App Inventor 2?

MIT App Inventor 2 is a simple and intuitive free service for creating Android applications. You don't have to be an expert in programming or design for building awesome apps that can do useful stuff.

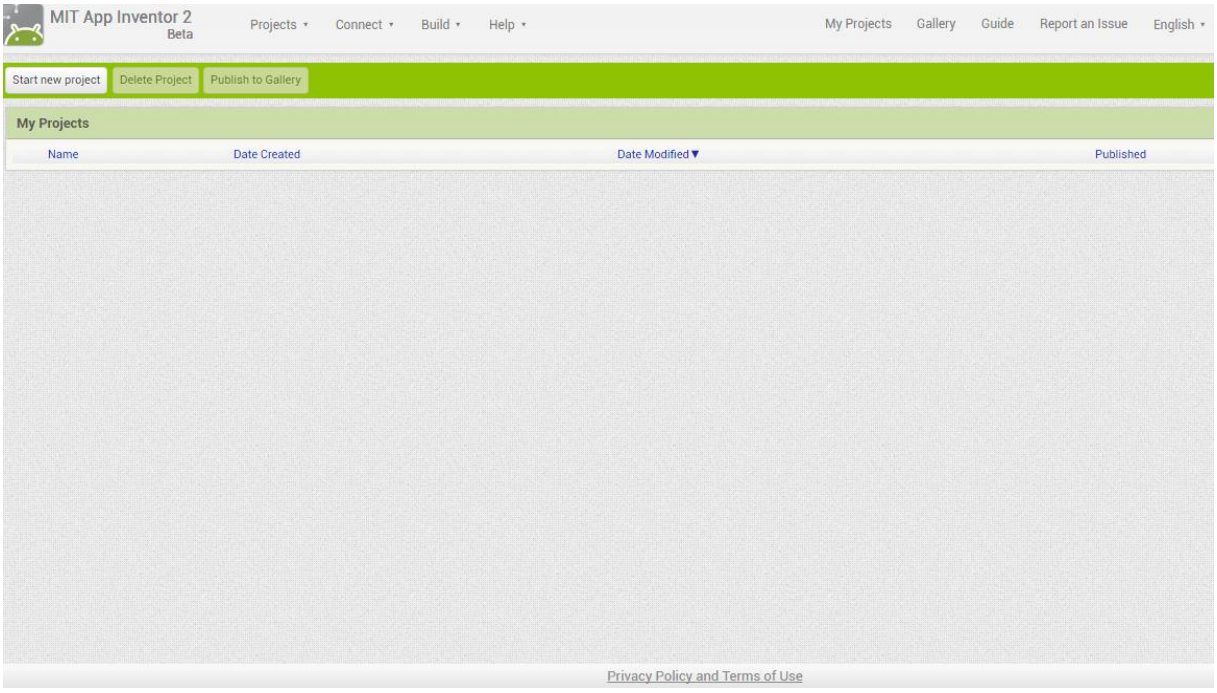
Creating the design is as easy as selecting and placing widgets in the smartphone screen. The coding is done with drag and drop puzzle blocks.

Anyone can learn how to build their own apps with MIT App Inventor 2 with a few hours of practice.

Accessing MIT App Inventor 2

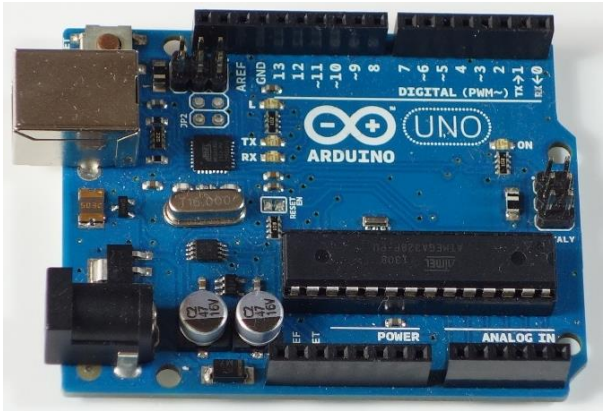
To access MIT App Inventor 2 go to http://appinventor.mit.edu/explore/ and press the orange **Create Apps** button.

To access the app builder, you need a Google account. Follow the on-screen steps to login into MIT App Inventor 2. After that, you'll be presented with the following dashboard (we'll cover how to use the dashboard in the **MIT App Inventor 2 Overview** section):



Introducing Arduino

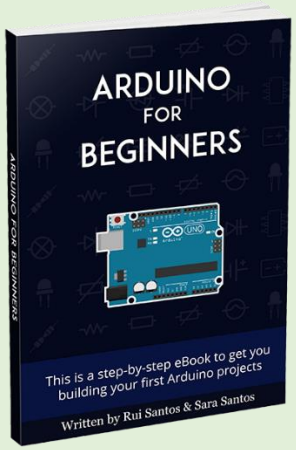
The Arduino is the board shown in the figure below.



The Arduino is a tiny computer with several digital and analog pins that you can connect to electrical circuits. You can read inputs from sensors and control outputs like an LED or a relay.

The code that tells the Arduino what to do is written with the Arduino IDE (Integrated Development Environment) software.

Arduino for Beginners



If you're new to the Arduino we recommend following the **Arduino for Beginners** eBook – a free bonus that comes with this course. **Arduino for Beginners** is a simple guide for quickly getting started with Arduino electronics projects.

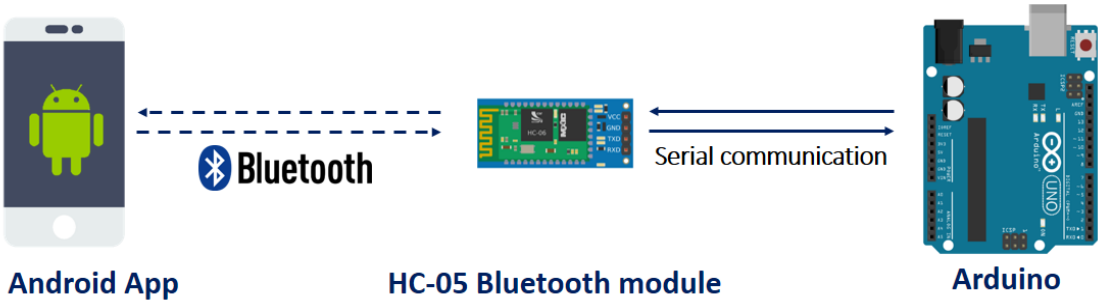
Bluetooth Module

To establish a connection between the Arduino and your Android app, we'll use the bluetooth communication protocol. For that, you need a bluetooth module.

The projects in this eBook use the HC-05 bluetooth module - shown in the figure below (HC-04 and HC-06 are also compatible).

This bluetooth module works with serial data. This means that the Arduino sends information and the bluetooth module receives it via serial (and vice-versa).

The following figure explains how the information flows from the Android app to the Arduino.

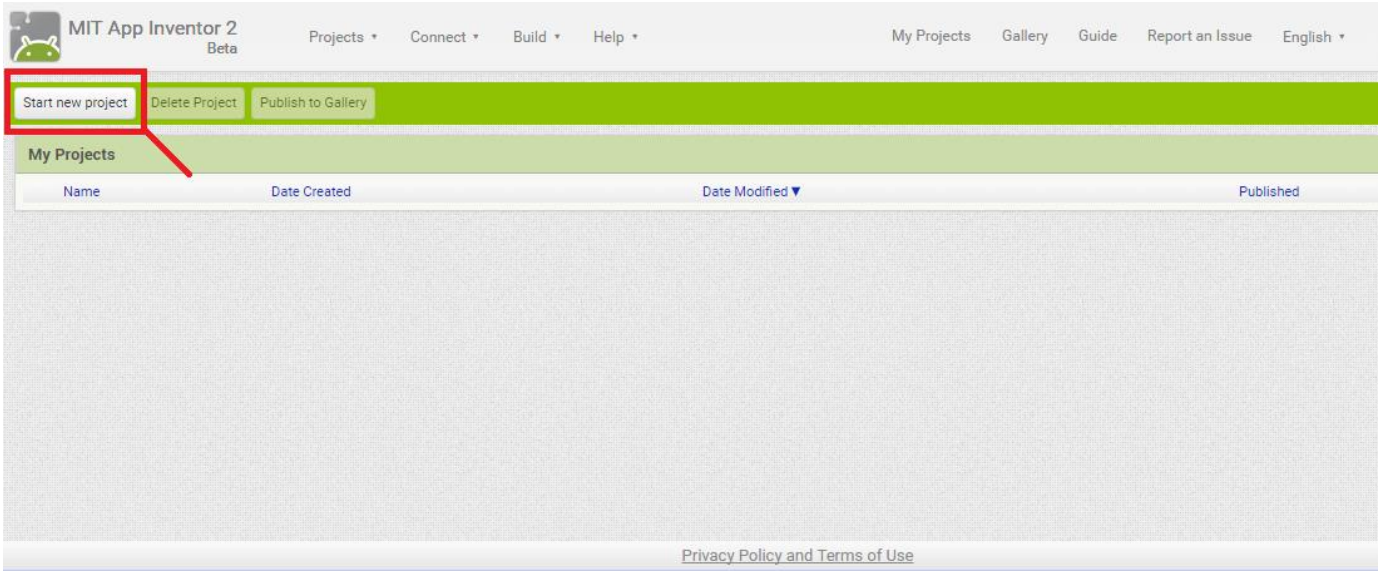


Your smartphone sends information to the bluetooth module via bluetooth. Then, the bluetooth module sends the information via serial communication to the Arduino. This flow also works the other way around: the Arduino sends information to the bluetooth module that sends it to the smartphone via bluetooth.

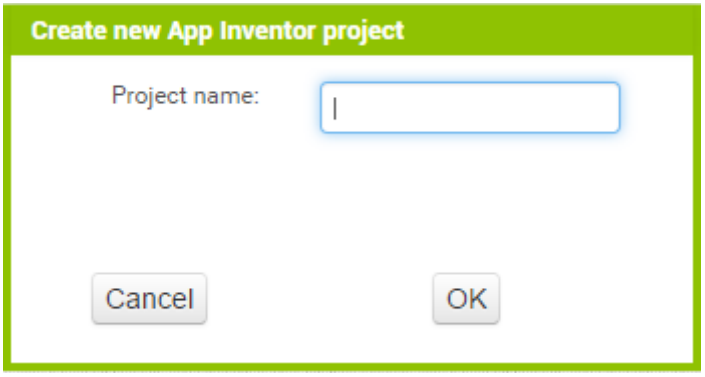
MIT App Inventor Overview

Go to http://appinventor.mit.edu/explore/ and press **Create Apps** button.

Next, click on **Start new project** as shown in figure below.

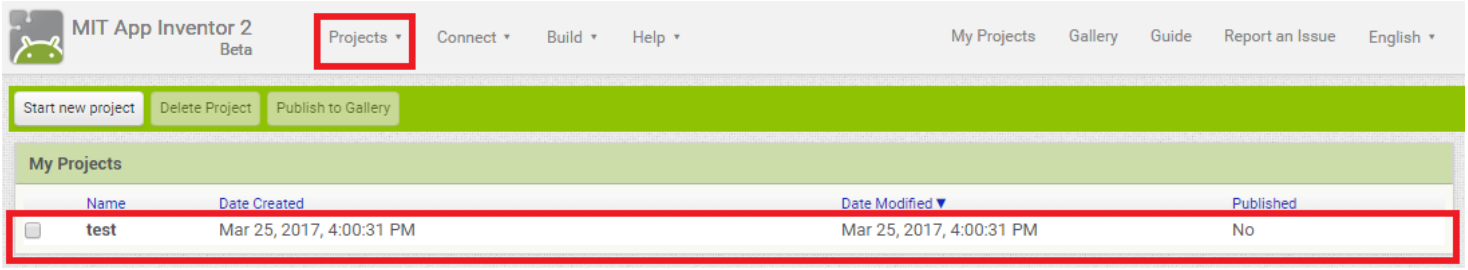


You'll be asked to give your project a name. By now, we're just exploring the MIT App Inventor 2 features, so you can name it **test**.



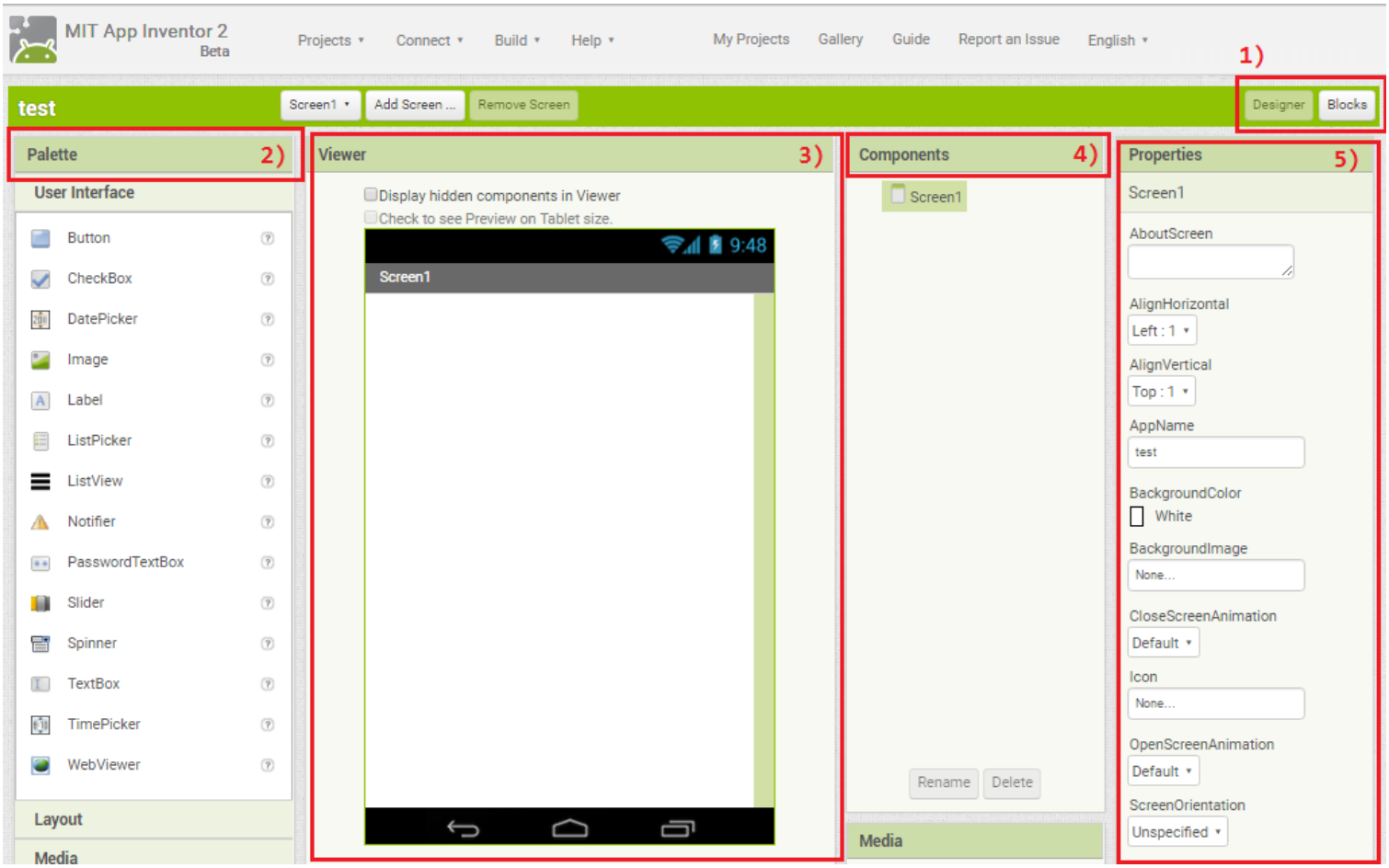
Click **OK.** Your project is automatically saved.

If you go to **Projects** ▸ **My Projects** you can see all your saved projects.



Click on the project name to open the app builder.

You'll be presented with the **Designer** tab as shown in the following figure.



At **1)** you select whether you are on the **Designer** or in the **Blocks Editor** tab. With MIT App Inventor you have 2 main sections: **Designer** and **Blocks**. The designer gives you the ability to add buttons, add text, add screens and edit the overall app look.

The **Blocks** section allows you to create custom functionality for your app, so when you press the buttons it actually does something with that event.

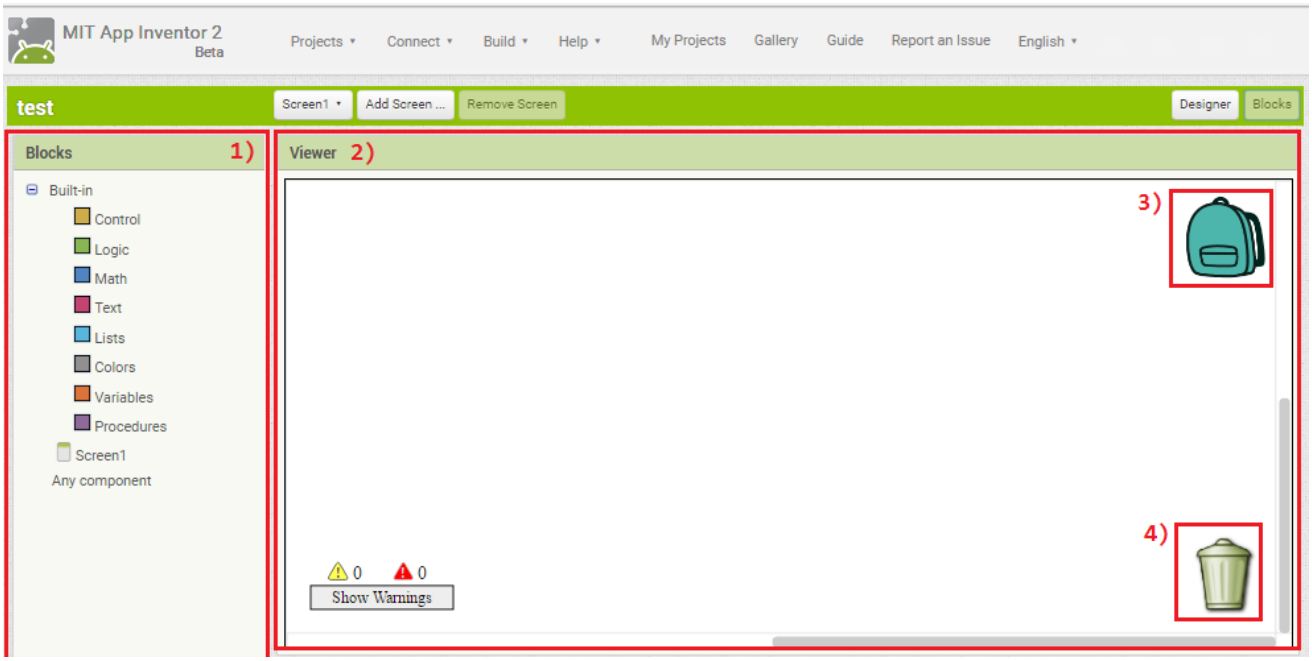
2) The **Palette** contains the components to build the app design like buttons, sliders, images, labels, etc...

3) It's the **Viewer**. This is where you drag the components to build the app look.

4) **Components**. You can see all the components added to your app and how they are organized hierarchically.

5) **Properties**. This is where you select your components' properties like color, size and orientation.

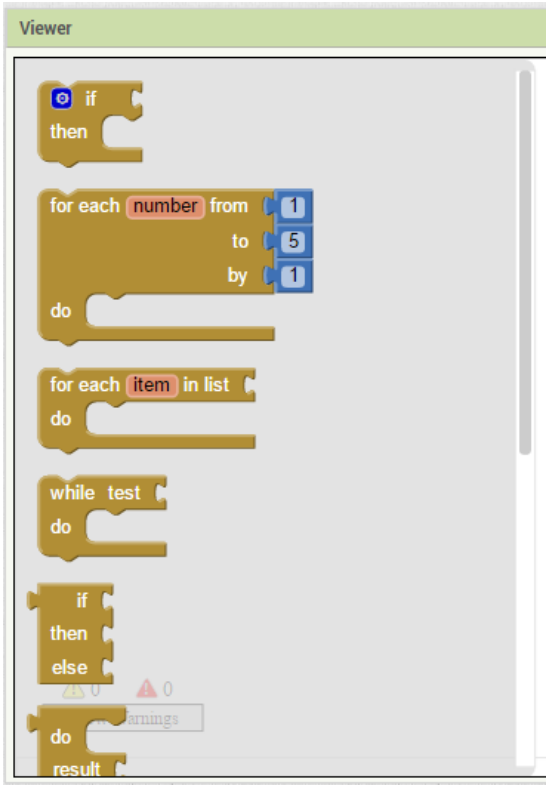
Open the **Blocks** editor tab**.**



In the **Blocks** editor tab, you have several sections:

1) contains the built-in blocks for creating the app's logic. This is what makes the app define the buttons functionalities, send commands to Arduino, connect to the bluetooth module, etc. You have several blocks grouped by categories:

- **Control**: *if/else* statements, *while loops*, etc...



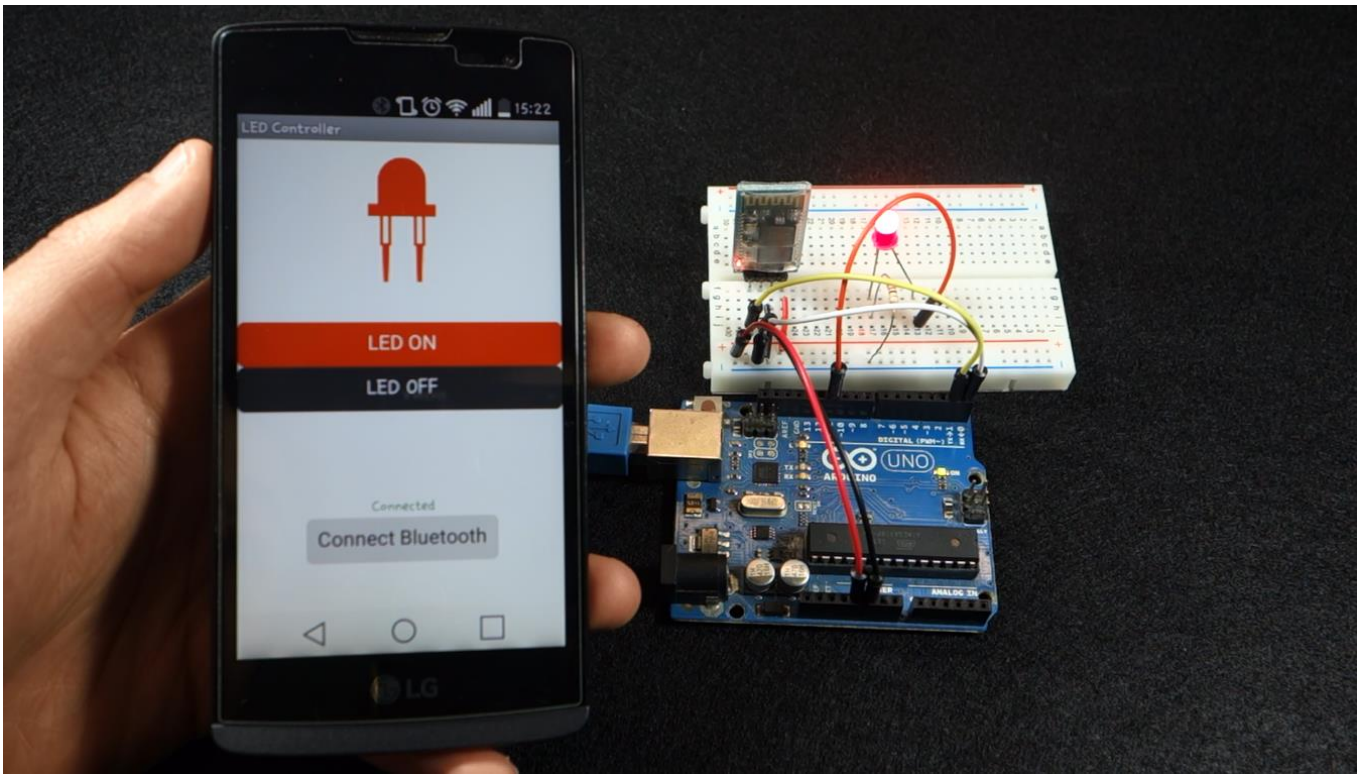
- **Logic**: *True, False, equal, not equal,* etc...
- **Math**: math operators.
- **Text**: blocks that deal with text.
- **Lists**: blocks for handling lists.
- **Colors**: blocks to handle colors, like choosing a color, make color and split colors.
- **Variables:** initialize variables, setting variables values, get variables values, etc...
- **Procedures:** procedures are like functions. A procedure is a sequence of code blocks with a given name. Later, you can call that sequence of blocks instead of creating the same long sequence.

Inside each group, you have blocks that you can drag to the **Viewer 2)**. In the **Viewer**, you drag the blocks and join them in a specific way to make something happen.

We recommend that you navigate inside the blocks section and explore what's inside. The blocks look like puzzle pieces that fit into each other or not. If you can't do something with certain blocks, they won't fit.

In the **backpack 3)** you save code blocks to use later. You move blocks to the **dustbin 4)** to delete them.

Project 1: LED Controller



Project 1: LED Controller

In this project you're going to create an Android app that turns an LED on and off.

We highly recommend that you follow this project first, because you'll learn how to make the app's design, and how to build the blocks that send information to the Arduino to turn an LED on and off. We'll also cover how to program the Arduino to receive the information from the app and how to test your app in real time as you build it.

Finally, we'll show you how to install the app on your smartphone. We hope you're excited to build your first Android application.

Parts Required

Here's a complete list of parts you need to do this project.

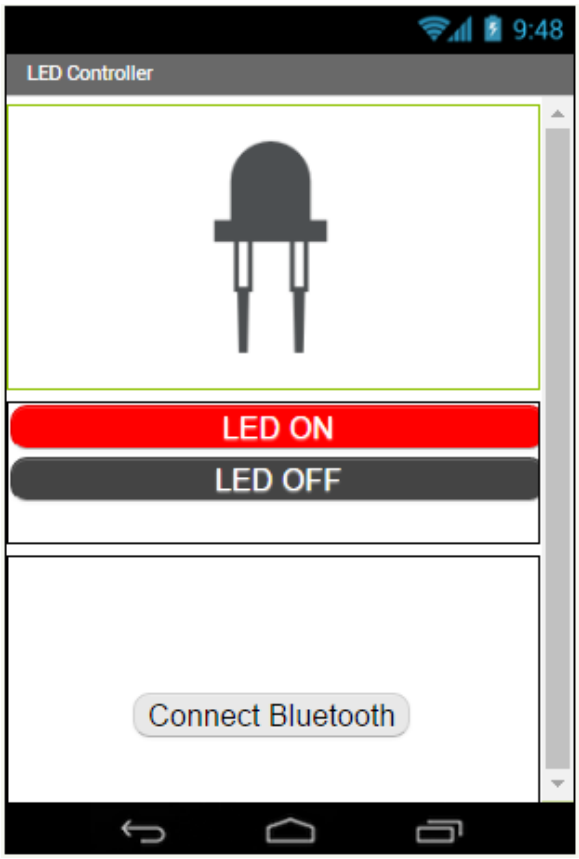
Figure	Name	eBay
	Arduino UNO	http://ebay.to/1SQda0R
	HC-04 or HC-05 or HC-06 Bluetooth module	http://ebay.to/2dt1b6M
	LED	http://ebay.to/20H2Oyy
	Breadboard	http://ebay.to/21bEojM
	220Ohm Resistor	http://ebay.to/1KsMYFP
	Jumper Wires	http://ebay.to/1PXeaJz

Building the App

Go to http://ai2.appinventor.mit.edu for opening the MIT App Inventor 2 software. Go to **Projects** ▸ **Start New Project** and call it *Led_Controller*.

Alternatively, if you prefer, you can upload the *.aia* file and edit the app the way you like. You just have to go to **Projects** ▸ **Import project (.aia) from my computer** and upload the *LED_Controller.aia* file that comes with the eBook's resources.

Before building the app, look at the final result below, so that you have clear picture of what you're going to build.



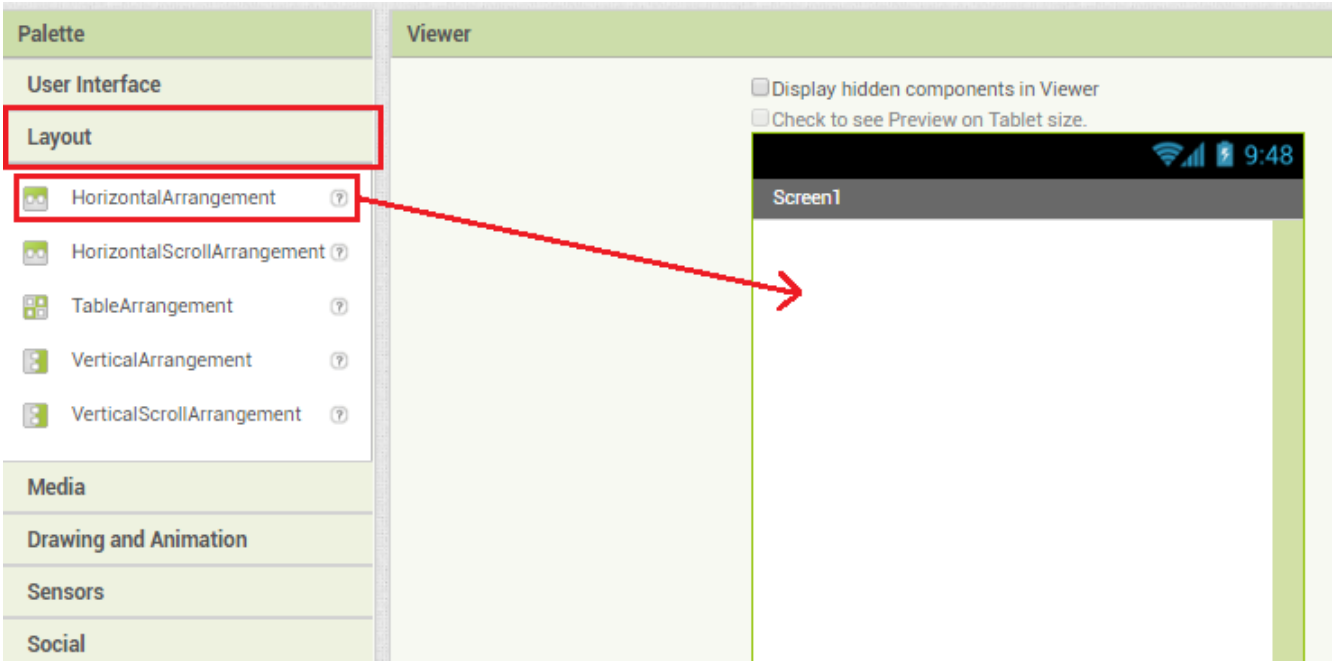
Here's the application core features:

- It has a **Connect Bluetooth** button that connects your smartphone to the bluetooth module;
- It has two buttons: **LED ON** and **LED OFF**;
- It has an LED image. The LED image changes from gray to red when you light up the LED and vice-versa.

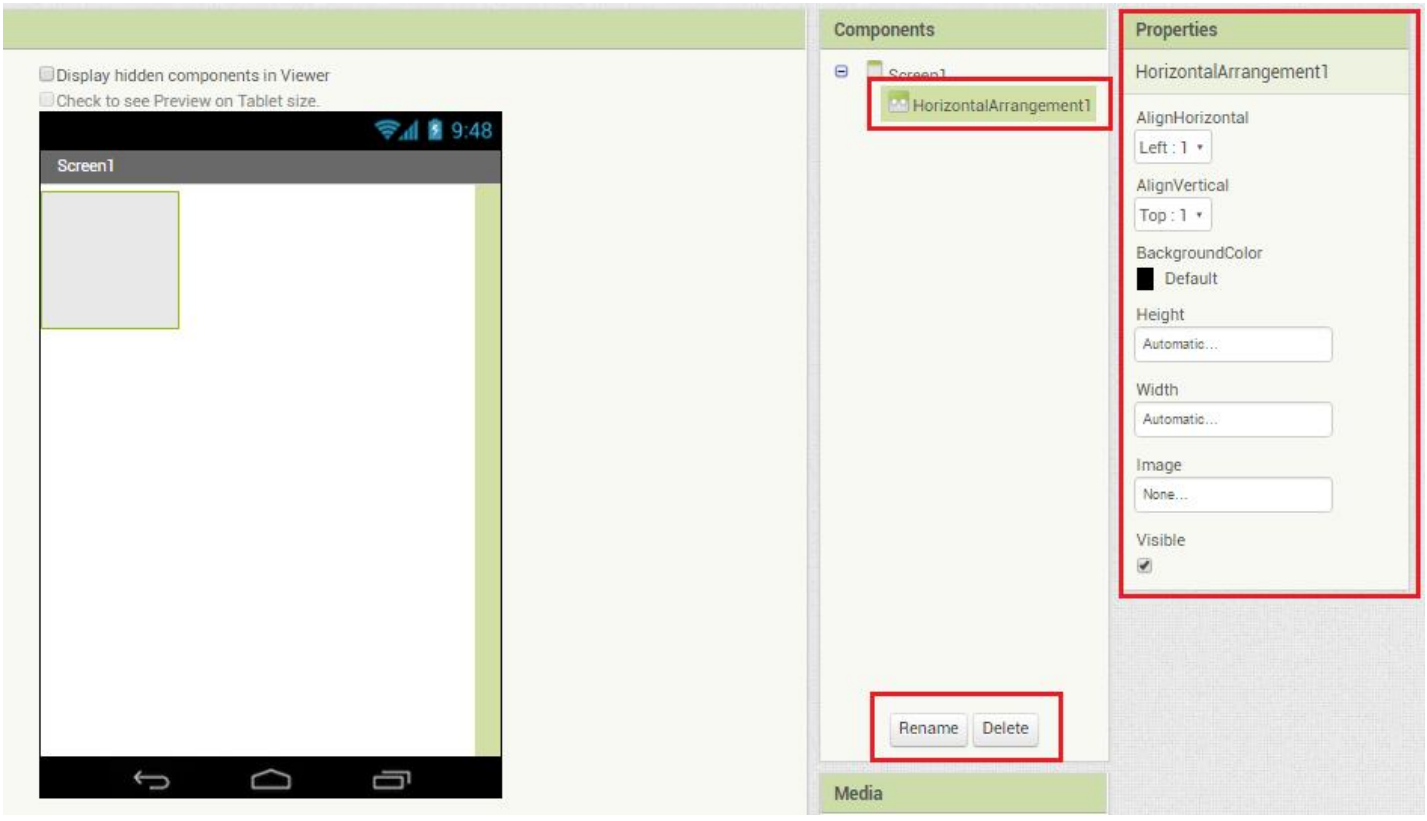
Now, that you know how the app is structured, let's build it!

Designer

When creating a new app, you always start in the **Designer** tab. Inside the **Palette**, select **Layout** and drag an **HorizontalArrangement** into the **Viewer** as shown in figure below.



The **HorizontalArrangement** allows you to align several widgets horizontally.



After adding the **HorizontalArrangement**, it shows up in the **Components** section.

Adding LED images

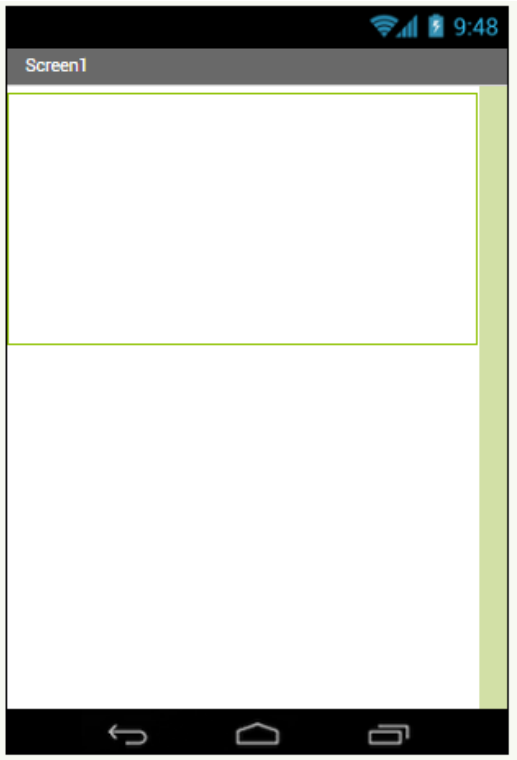
Select the **HorizontalArrangement** in the **Components** section. You can rename it or delete it by clicking the **Rename** or **Delete** button. You can also select the **HorizontalArrangement** properties at the right as shown in previous figure.

Click the **Rename** button to change the name to **ledHorizontalArrangement** - this is where you are going to place the LED images.

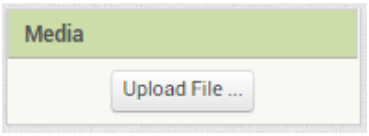
In the **Properties** section select the following properties:

- **AlignHorizontal:** Center : 3
- **AlignVertical:** Center : 2
- **BackroungColor**: none
- **Height**: 40 percent
- **Width**: Fill parent
- **Image:** none
- **Visible**: ✓

The **ledHorizontalArrangement** occupies 40% of the screen and fills the whole screen from left to right. The **ledHorizontalArrangement** is shown in the following figure – it's the rectangle with a green border.



You need to upload the LED images. The images are available within the book's resources **Assets** folder. Below the **Components** section, you have a section called **Media.** Click on **Upload file** and upload the led_gray.png and led_red.png images.



To insert an image in your application, go to **Palette** ▸ **User Interface** and drag an **Image** inside the **ledHorizontalArrangement**. Do this twice because we want two images.

Rename the **Images** to **ledGrayImage** and **ledRedImage**.

Select the **ledGrayImage** and set the following properties:

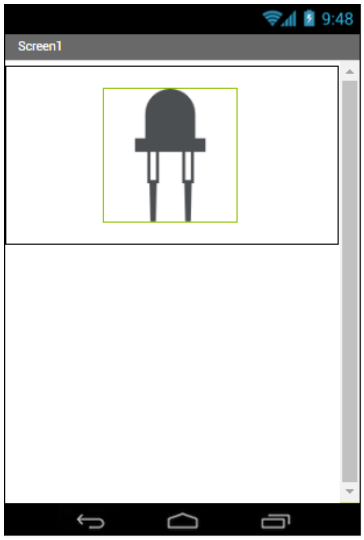
- **Height:** Automatic
- **Width:** Automatic
- **Picture:** led_gray.png
- **RotationAgle:** 0.0
- **ScalePictureToFit:** ✓
- **Visible:** ✓

Select the **ledRedImage** and set the following properties:

- **Height:** Automatic
- **Width:** Automatic
- **Picture:** led_red.png
- **RotationAgle:** 0.0
- **ScalePictureToFit:** ✓
- **Visible:** ☐

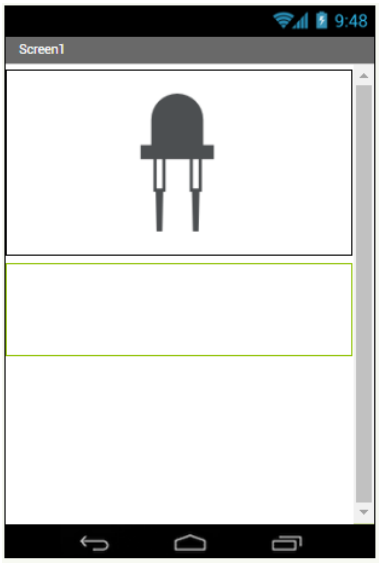
Notice that we set the **ledRedImage** to not visible (invisible). We are doing this, because initially our app shows the gray LED image. Then, when we light up the LED, the red LED image switches to visible and the gray image becomes invisible. This makes the app more user friendly.

The figure at the right shows how your app should look like at this moment.



Adding LED control buttons

Let's design the on and off buttons. Go to **Palette** ▸ **Layout** and drag a **VerticalArrangement**. You use a **VerticalArrangement** to place two buttons aligned vertically. Rename the **VerticalArrangement** to **buttonVerticalArrangement**.



The **buttonVerticalArrangement** should have the following properties**:**

- **AlignHorizontal:** Center : 3
- **AlignVertical:** Top : 1
- **BackroungColor**: none
- **Height**: 20 percent
- **Width**: Fill parent
- **Image:** none
- **Visible**: ✓

Your app should look like the figure at the right.

Go to **Palette** ▸ **User Interface** and drag two **Buttons** inside the **buttonVerticalArrangement.**

Rename the **Buttons**. The one at the top to **ledOnButton** and the one at the bottom to **ledOffButton.**

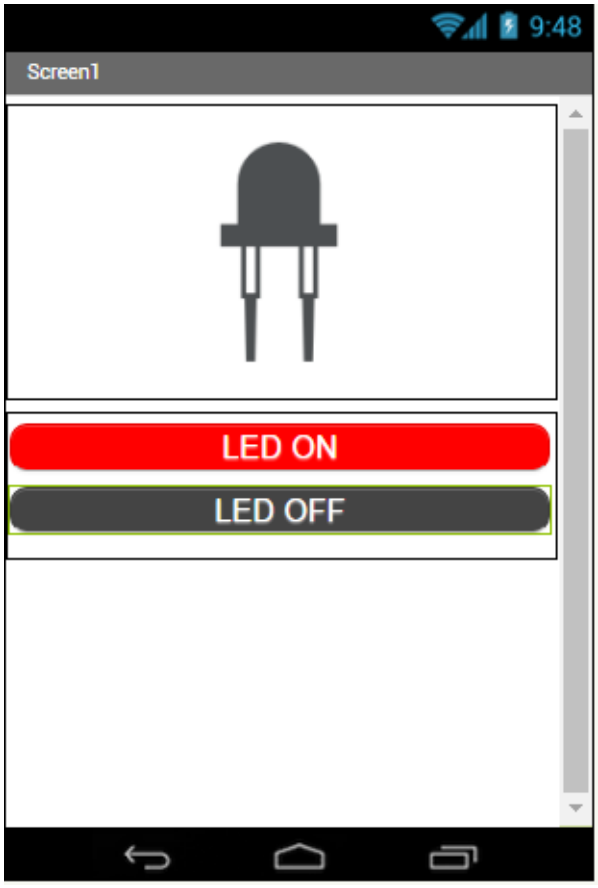
Edit the **ledOnButton** properties:

- **BackgroundColor**: **Red**
- **Enabled:** ✓
- **FontBold:** ☐
- **FonItalic:** ☐
- **FontSize:** 20
- **FontTypeface:** sans serif
- **Height:** Automatic
- **Width:** Fill parent
- **Image:** none
- **Shape:** rounded
- **Show Feedback:** ✓
- **Text:** LED ON
- **TextAlignment:** center : 1
- **TextColor:** white
- **Visible:** ✓

Edit the **ledOffButton** properties:

- **BackgroundColor**: **Dark Gray**
- **Enabled:** ✓
- **FontBold:** ☐
- **FonItalic:** ☐
- **FontSize:** 20
- **FontTypeface:** sans serif
- **Height:** Automatic
- **Width:** Fill parent
- **Image:** none
- **Shape:** rounded
- **Show Feedback:** ✓
- **Text:** LED OFF
- **TextAlignment:** center : 1
- **TextColor:** white
- **Visible:** ✓

Here's how your app should look with the buttons:



Adding connect bluetooth button

We also need a button to connect to the Arduino via bluetooth. When the connection is established, a message saying "Connected" should appear above the button. Also, when you click the button to connect the bluetooth, a list picker should pop up for choosing between the available bluetooth devices.

Go to **Palette ▸ Layout** and drag a **Vertical Arrangement** below the **buttonVerticalArrangement**. Rename the arrangement to **connectVerticalArrangement.**

The **connectVerticalArrangement** should have the following properties:

- **AlignHorizontal:** Center : 3
- **AlignVertical:** Center : 2
- **BackroungColor**: none
- **Height**: 40 percent
- **Width**: Fill parent
- **Image:** none
- **Visible**: ✓

Go to **Palette ▸ User Interface** and drag a **Label** and a **ListPicker** into the **connectVerticalArrangement.** Rename the **Label** to **bluetoothConnectionLabel** and the **ListPicker** to **bluetoothListPicker.**

The **bluetoothConnectionLabel** should have the following properties:

- **BackgroundColor**: none
- **FontBold:** ☐
- **FontItalic:** ☐
- **FontSize:** 14
- **FontType:** default
- **HTMLFormat:** ☐
- **HasMargins:** ✓
- **Height:** Automatic
- **Width:** Automatic
- **Text:** Disconnected
- **TextAlignment:** center : 1
- **TextColor:** none
- **Visible:** ✓

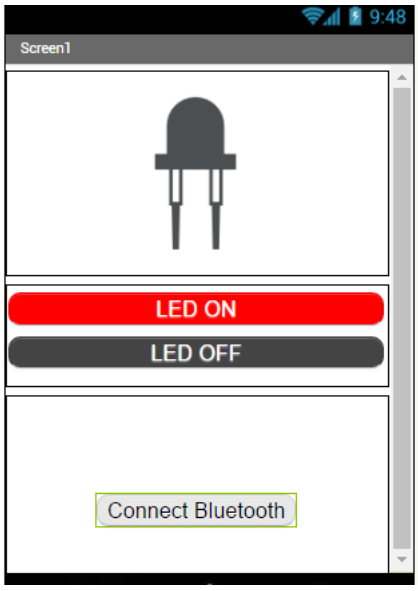
The **bluetoothConnectionLabel** shows your bluetooth's current connection state – if it is connected or not. Notice that we're setting the **TextColor** to **none** when you initialize the app. After initializing, the app checks whether the connection is

established or not and sets the color accordingly – green if the connection is established, and red if it isn't. We'll do this in the **Blocks** editor section.

The **bluetoothListPicker** should have the following properties:

- **BackgroundColor:** Default
- **ElementsFromString:**
- **Enabled:** ✓
- **FontBold:** ☐
- **FontItalic:** ☐
- **FontSize:** 20
- **FontTypeface:** sans serif
- **Height:** Automatic
- **Width:** Automatic
- **Image:** none
- **ItemBackgroundColor:** Black
- **ItemTextColor:** White
- **Selection:**
- **Shape:** rounded
- **ShowFeedBack:** ✓
- **ShowFitterBar:** ☐
- **Text:** Connect Bluetooth
- **TextAlignment:** center : 1
- **TextColor:** Default
- **Title:**
- **Visible:** ✓

Here's how your app looks:



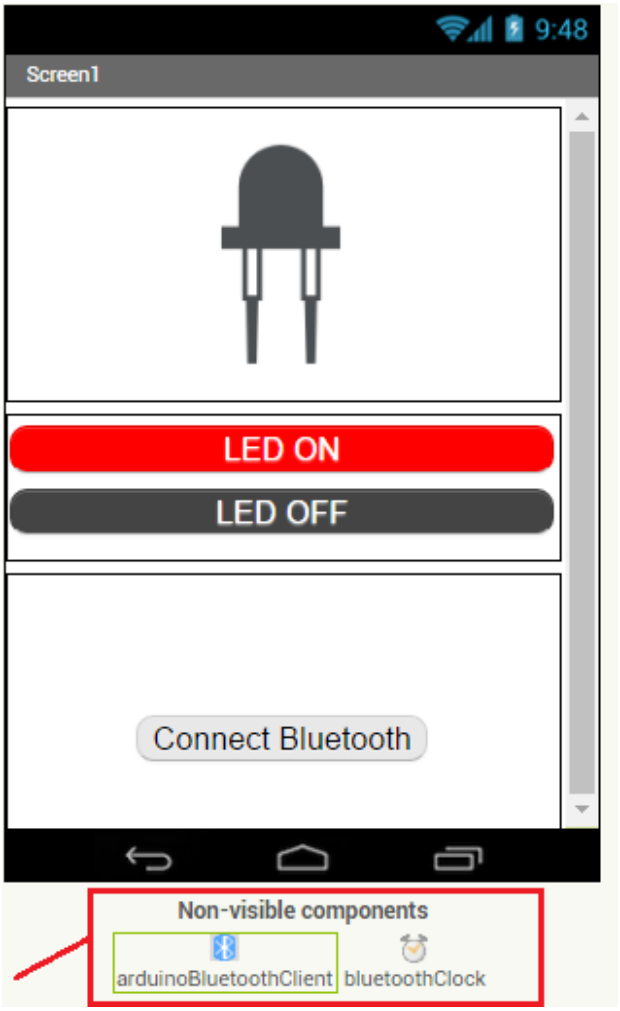
Adding non-visible components

To complete your app design, you need to add **Non-visible components**: a **Bluetooth Client** and a **Clock**.

Go to **Palette** ▸ **Connectivity** and drag a **BluetoothClient** into the **Viewer.** Rename the **BluetoothClient** to **arduinoBluetoothClient.** You don't need to edit its properties, you'll use the default settings.

Go to **Palette** ▸ **Sensor** and drag a **Clock** to the **Viewer.** Rename the **Clock** to **bluetoothClock.** You don't need to edit its properties, they are also properly set by default.

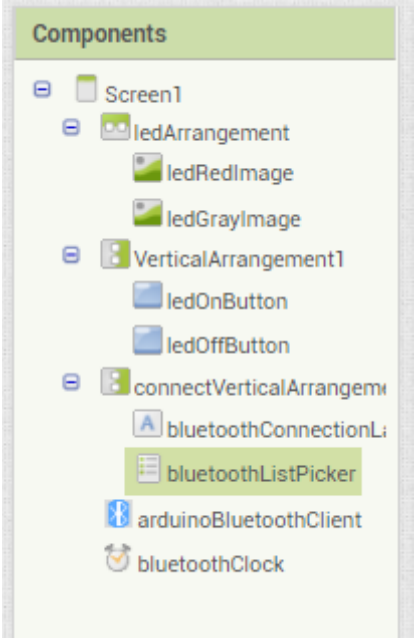
The **arduinoBluetoothClient** and the **bluetoothClock** should appear below the app screen as shown in the figure below:



Finally, the app design is ready.

Notice that in the **Components** section you have all your components organized as shown in figure below. It is good to rename the components as we did, so that when

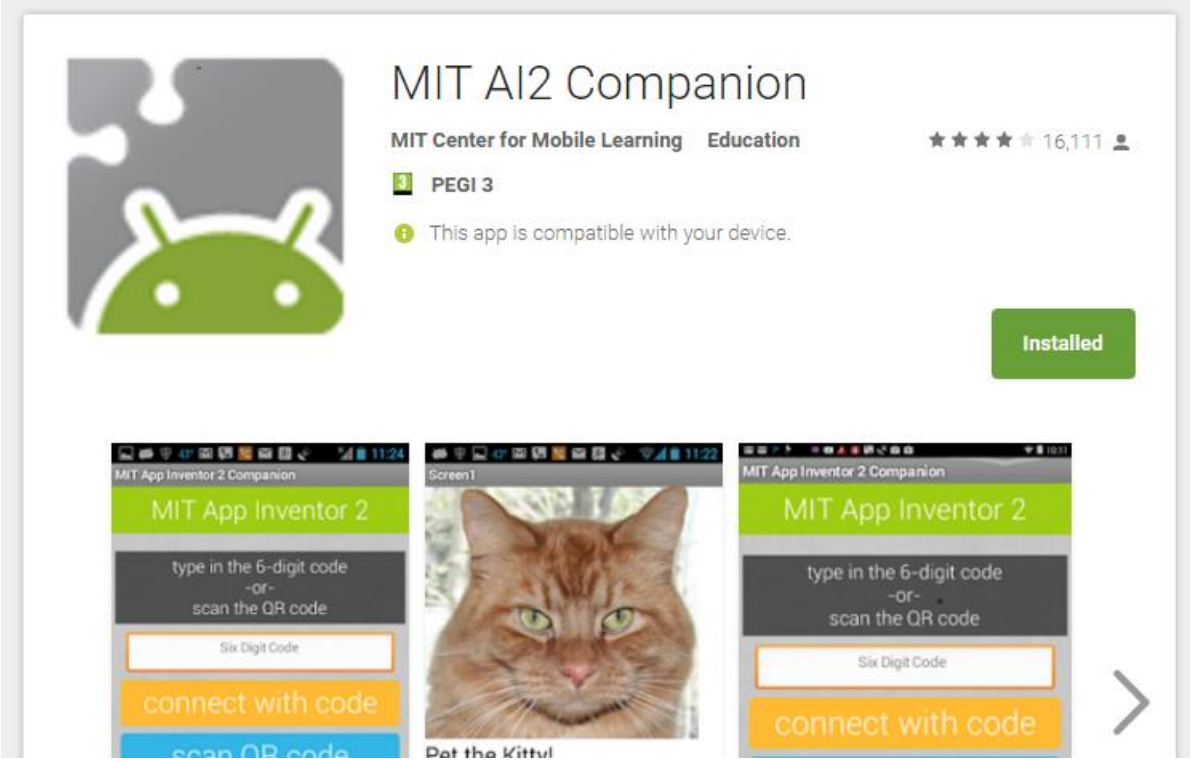
you look at the **Components** section you can quickly understand how the design is made and how the components are placed hierarchically.



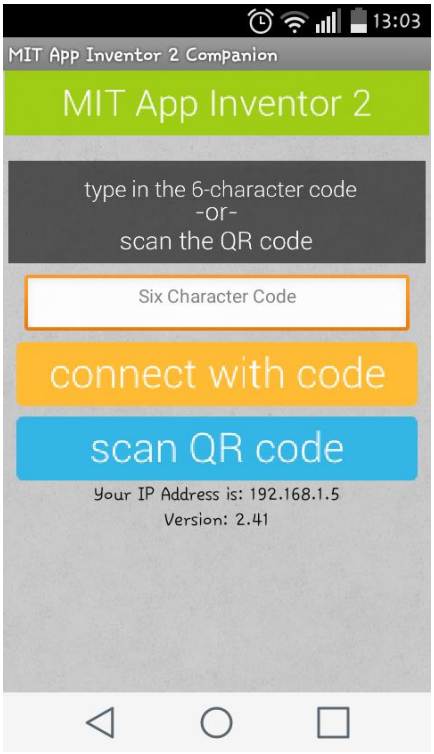
Testing the App

You can test the app in your smartphone while you're working on it in real time. You need to install the **MIT AI2 Companion** app in your smartphone.

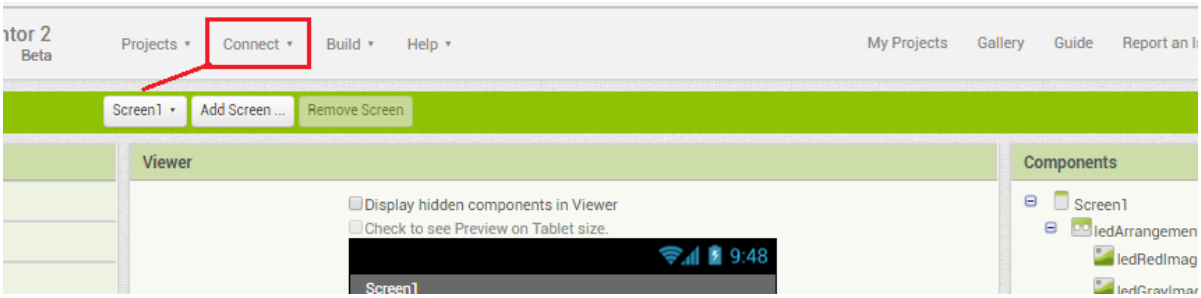
Go to the Google Play Store and search for **MIT AI2 Companion** and install it on your Android phone.



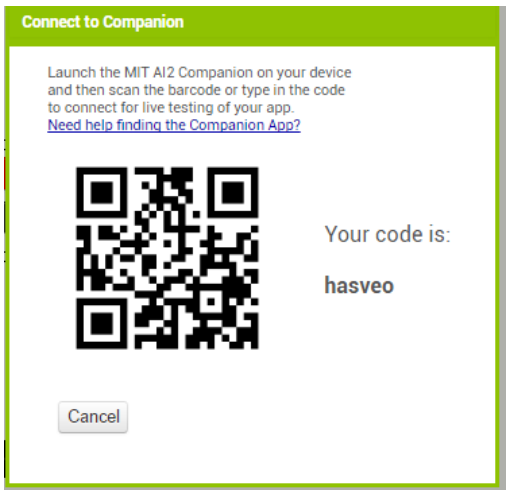
Open the app. You'll be presented with the following screen.



To test your app, in the MIT App Inventor 2 software go to **Connect ▸ AI Companion** as shown in following figure.



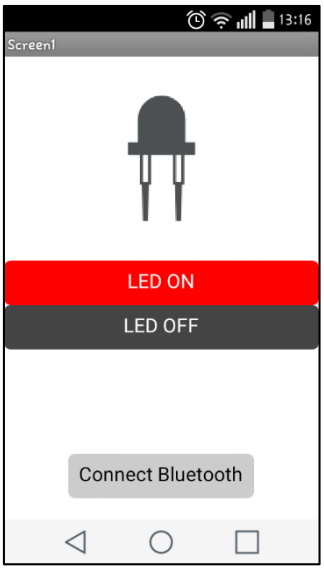
A QR code like the one in the following figure pops up:



In your smartphone, you can either enter the code or scan the QR code. After having the code, click on **connect with code**.

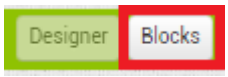
Your smartphone shows how your app looks. It updates in real time, so every time you make a change, you can instantly see the result. The figure at the right shows how the app looks like in the smartphone.

If you tap the buttons, nothing happens, because you haven't added any functionalities to the app in the **Blocks** editor**.** That's what you're going to do in the next section.



Blocks Editor

Once the design is ready, it's time to make the app do something. Let's go to the **Blocks** editor tab. Click **Blocks** in the upright corner.

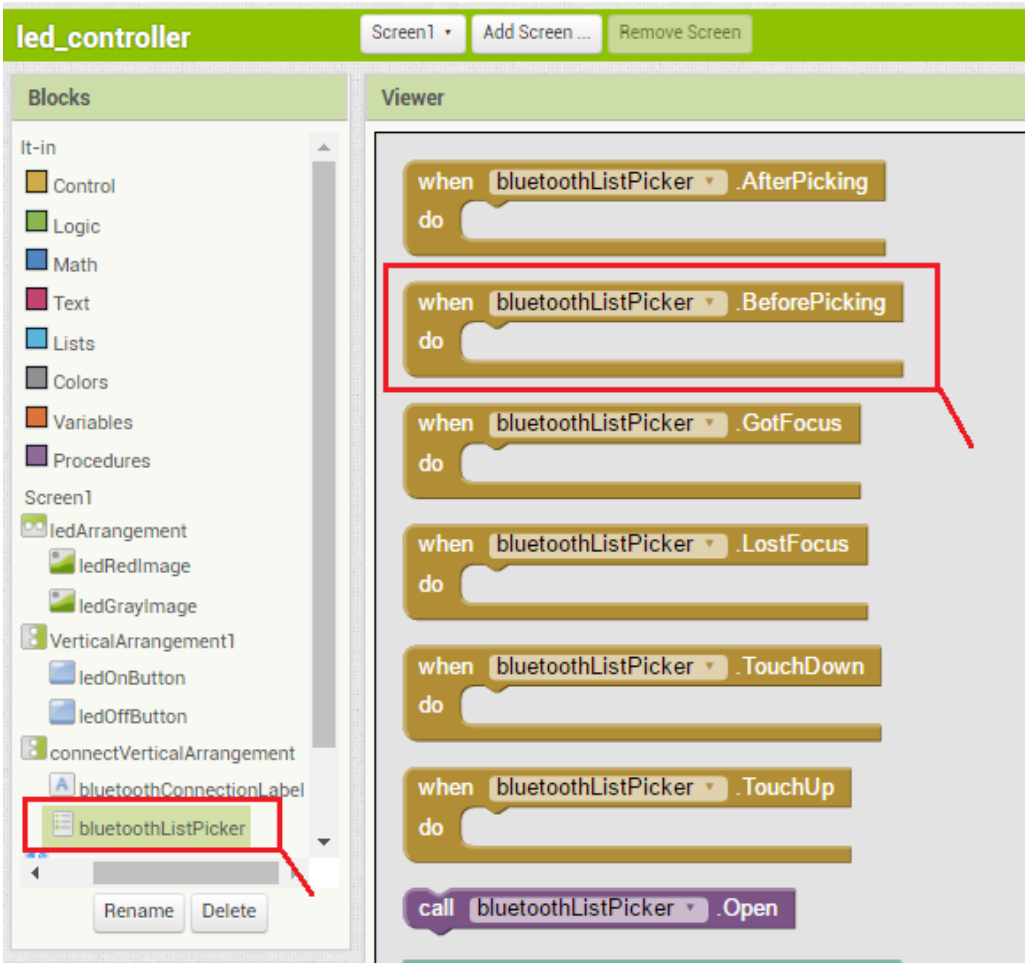


List picker – connecting bluetooth

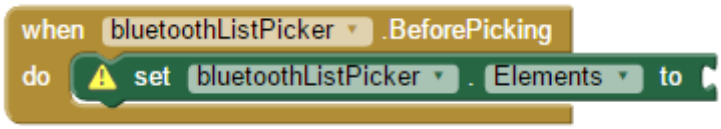
The app needs to establish a connection with the bluetooth module. To connect the bluetooth we will use the **bluetoothListPicker**. We need to:

- Define what the list picker shows;
- Define what happens when we select an element from the list picker;
- Change the **bluetoothLabel** text to "Connected".

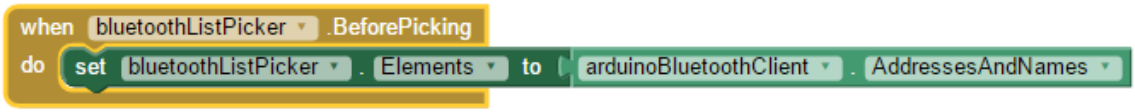
Go to **Blocks ▸ bluetoothListPicker** and drag a puzzle block that says "**when bluetoothListPicker.BeforePicking"** to the **Viewer.**



Go to the **bluetoothListPicker** again and drag the block that says "**set bluetoothListPicker.Elements to"** inside the previous block, so that they connect.



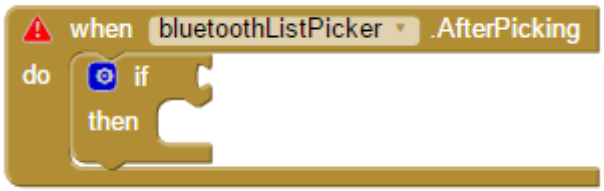
Then, go to **arduinoBluetoothClient** and drag the **arduinoBluetoothClient.AddressesAndNames** in the previous block, so it stays as follows:



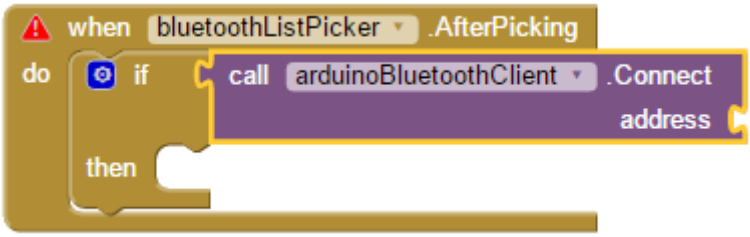
This block sets the elements of the list picker to the devices paired with the smartphone.

Then, go to **Blocks ▸ bluetoothListPicker** and drag the block that says **"when bluetoothListPicker.AfterPicking"** into the **Viewer**. With this, you'll create a block that says what the app is going to do after picking a list picker item.

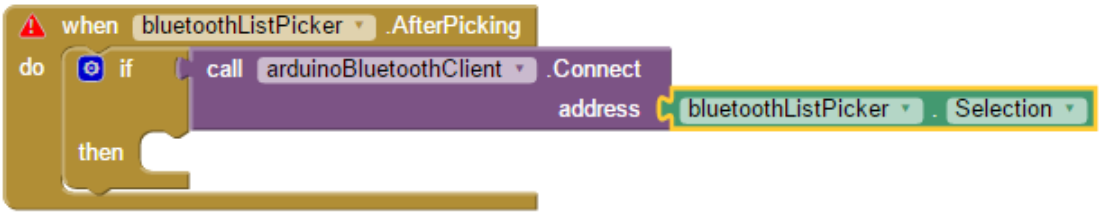
Go to **Blocks** ▸ **Control** and drag an **"if then"** block into the previous block.



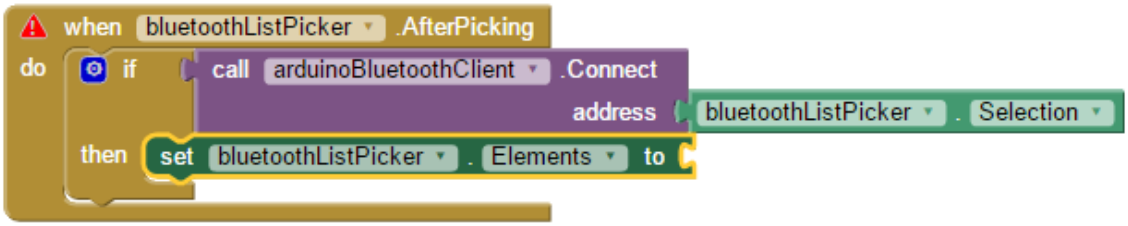
Go to **arduinoBluetoothClient** and select the block **call arduinoBluetoothClient.Connect address** and fit it in the **if** space**.**



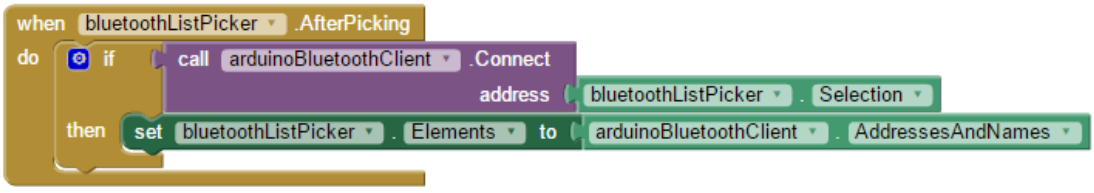
From the **bluetoothListPicker** select the **bluetoothListPicker.Selection** and drag it into the purple block.



Go to the **bluetoothListPicker** and drag the **set bluetoothLisPicker.Elements to** into the **then.**



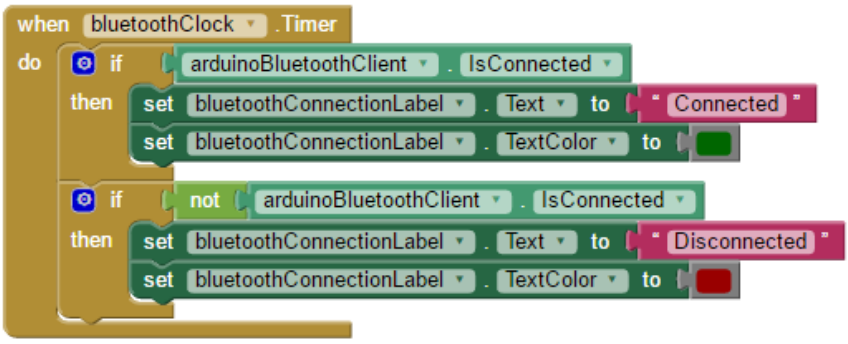
Complete the previous block with a **arduinoBluetoothClient.AddressesAndNames** block. These blocks connect your app to the list picker chosen device.



Clock

You need to make a new block that tells your app to display a message saying "Connected" when the bluetooth connection is established or a "Disconnected" message if the connection is not established.

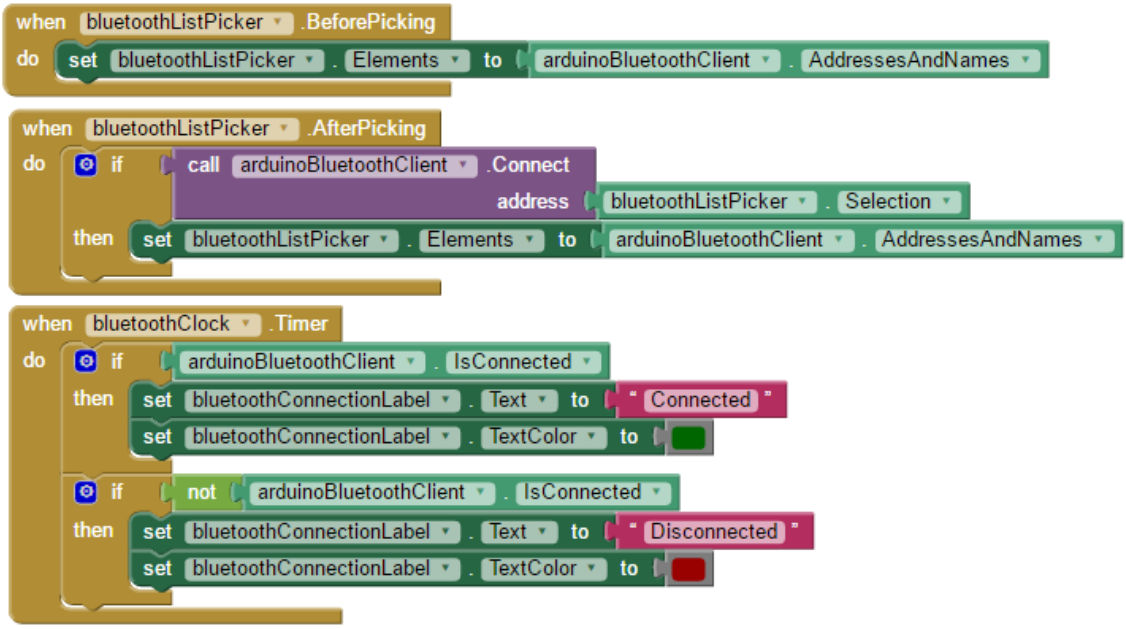
Go to **Blocks ▸ bluetoothClock** and drag a block **whenbluetoothClock.Timer** into the **Viewer**. Then, add the following blocks, so that you have a full block like this:



This code block checks if your bluetooth communication is established and if true, it changes the **bluetoothConnectionLabel** text to "**Connected"** in **green**. If not, it shows a "**Disconnected**" message in **red.**

You find the text pink boxes in **Blocks ▸ Text.** The **not** block is found in the **Logic** section.

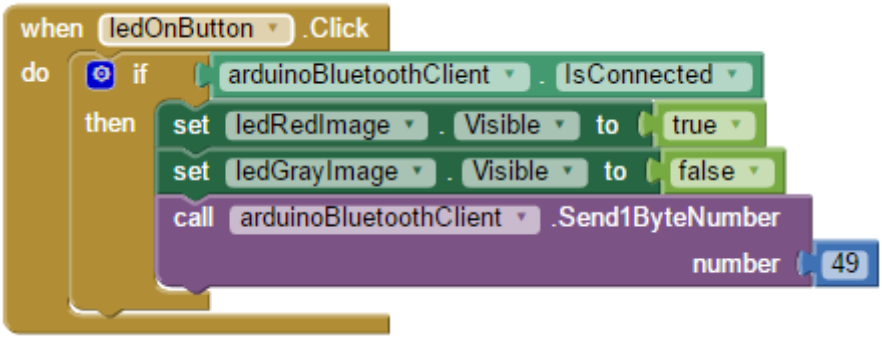
If you've followed all the steps, you should have the following code blocks:



LED buttons

The next step is to make something happen when you tap the **LED ON** and **LED OFF** buttons.

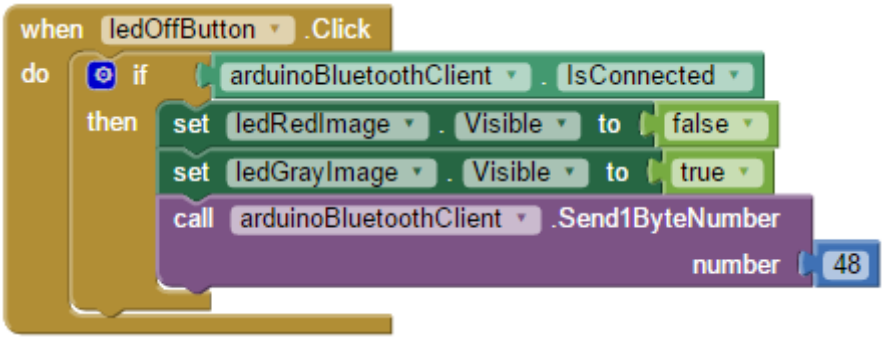
Go to **Blocks** ▸ **ledOnButton** and drag the block that says **when ledOnButton.Click** into the **Viewer**. Add the following blocks:



This code block says that when you press the **LED ON** button, if the bluetooth communication is established, the app sets the **ledRedImage** to visible and the **ledGrayImage** to invisible. That changes the LED color on the app depending if the LED is on or off giving visual feedback.

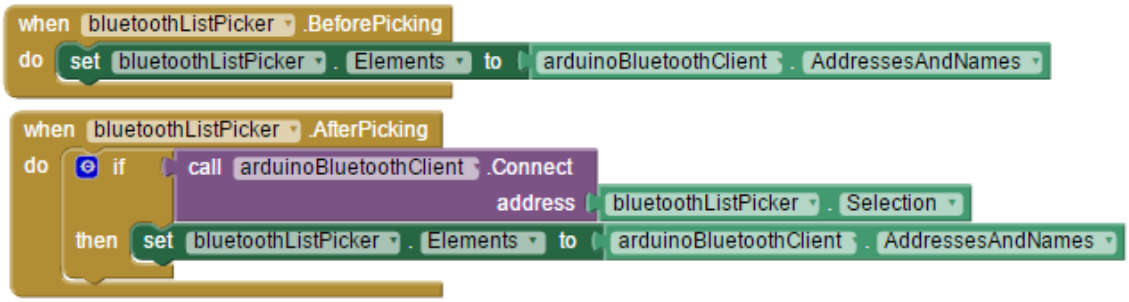
Then, you send a **49** decimal value via bluetooth that corresponds to **1** in char.

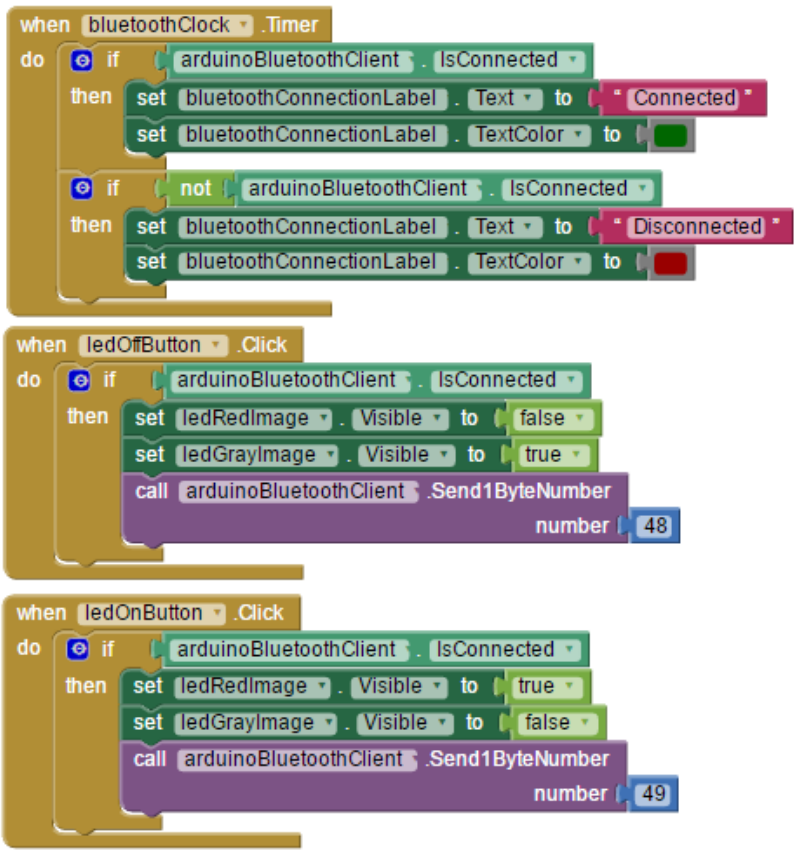
Following the same concept, build a block for the **LED OFF** button.



In this case, you're sending **48** decimal value that is **0** in char.

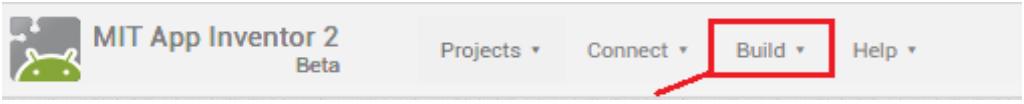
Congratulations! Your code blocks are ready. You should have the following:





Installing the LED Controller App

You're ready to install the app in your smartphone. To install the app, in the MIT App Inventor 2 software go to **Build** tab.



You can choose one of the following options:

1. **App (provide QR code for *.apk*)**: generates a QR code that you can scan with your smartphone and automatically install the app in your smartphone.
2. **App (save *.apk* to my computer):** downloads the *.apk* file to your computer. You connect your smartphone to your computer and move the file to the phone. Then, simply follow the installation wizard to install the app.

Alternatively, if you prefer, you can use the *.apk* file provided. Move the *LED_Controller.apk* file that comes with the eBook's resources to your Android phone and follow the on-screen instructions to install the application.

Writing the Sketch

Having the app completed and installed, the following step is writing the Arduino sketch. The HC-05 bluetooth module receives information from the app via bluetooth. Then, the information is forwarded to the Arduino board via serial communication.

The following Arduino sketch receives the bluetooth module information and lights up an LED when you tap the **LED ON** button or turns off the LED when you tap the **LED OFF** button. Upload the following code to your Arduino board. Make sure you have the right board and COM port selected.

```
/*
 * Rui Santos
 * Complete Project Details http://randomnerdtutorials.com
 */

int led = 11;      // pin Digital 11
int state;           // saves the state
int flag=0;    // makes sure that the serial only prints once the state

void setup() {
    // sets the led as output:
    pinMode(led, OUTPUT);
    digitalWrite(led, LOW);

    // initialize serial communication at 9600 bits per second:
    Serial.begin(9600);
}

void loop() {
    //if some date is sent, reads it and saves in state
    if(Serial.available() > 0){
      state = Serial.read();
      flag=0;
    }
    // if the state is '0' the led will turn OFF
    if (state == '0') {
        digitalWrite(led, LOW);
        if(flag == 0){
          Serial.println("LED Off!");
          flag=1;
        }
    }

    // if the state is '1' the led will turn ON
    else if (state == '1') {
        digitalWrite(led, HIGH);
        if(flag == 0){
          Serial.println("LED On!");
          flag=1;
        }
    }
    //Uncomment For debugging purpose
    //Serial.println(state);
}
```

Briefly, here's how the code works:

1. You start initializing three variables.

```
int led = 11;      // pin Digital 11
int state;           // saves the state
int flag = 0;  // makes sure that the serial only prints once the state
```

The `led` variable refers to the Arduino digital pin 11, this where the LED is connected to.

The `state` variable saves the information received from the bluetooth module. It receives either 1 or 0 because we set that in the app's blocks.

The `flag` variable is used to ensure that the current state is only printed once in the serial monitor. Printing the LED state is helpful for debugging purposes.

2. Then, you set the LED as an output and set the LED off by default. You also initialize the serial communication at a baud rate of 9600 bits per second.

```
void setup() {
    // sets the led as output:
    pinMode(led, OUTPUT);
    digitalWrite(led, LOW);
    // initialize serial communication at 9600 bits per second
    Serial.begin(9600);
}
```

3. In the `loop` is where you read the data sent via serial communication and decide whether to turn the LED on or off. The next snippet of code, checks if the serial communication is available. If it is, it gathers the data and saves it in the `state` variable using `Serial.read()`. Then, `flag` is set to `0`.

```
void loop() {
    //if some date is sent, reads it and saves in state
    if(Serial.available() > 0){
      state = Serial.read();
      flag=0;
    }
```

4. Then, you decide what to do, depending on the `state` variable value.

```
    // if the state is '0' the led will turn OFF
    if (state == '0') {
        digitalWrite(led, LOW);
        if(flag == 0){
          Serial.println("LED Off!");
          flag=1;
        }
    }
    // if the state is '1' the led will turn ON
    else if (state == '1') {
        digitalWrite(led, HIGH);
        if(flag == 0){
          Serial.println("LED On!");
          flag=1;
        }
    }
```

If the state is '0' (48 in decimal), this means we tapped the LED OFF button and so, the LED is turned off with `digitalWrite(led, LOW)`.

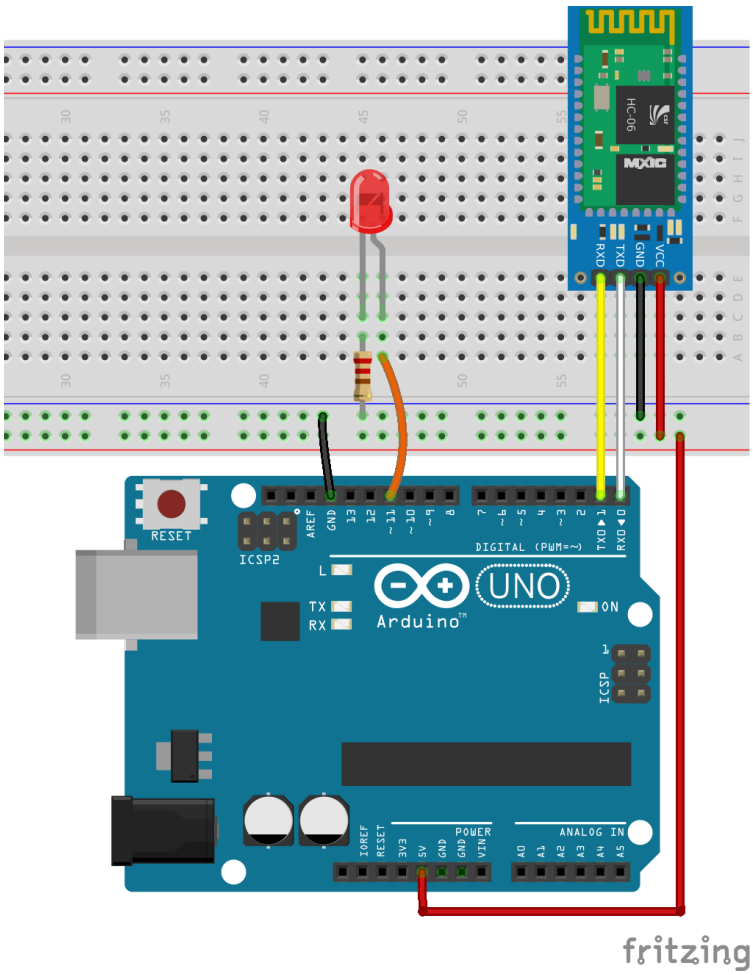
Else if the state is '1' (49 in decimal), we turn the LED on with `digitalWrite(LED, HIGH)`.

Wiring the Circuit

The last step is wiring the circuit. Wiring this circuit is straightforward, follow the next schematic diagram.

Warning: When you're uploading code to your Arduino you should disconnect the TX and RX cables from the bluetooth module. These pins are needed for serial communication between the Arduino and your computer.

Tip: People often make the wrong connections between the Arduino and bluetooth module. **TX** from the bluetooth module connects to **RX** pin of the Arduino. **RX** from the bluetooth module connects to **TX** pin of the Arduino.



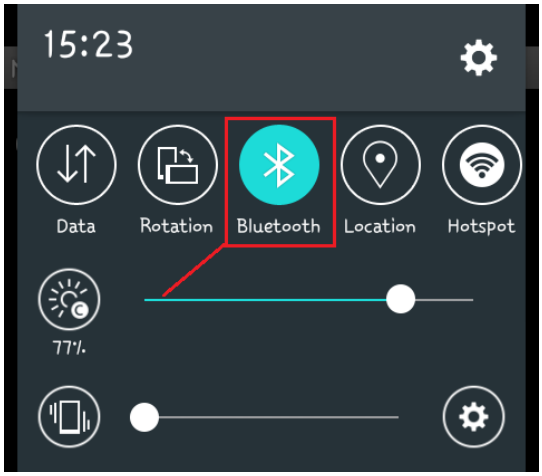
Here's the bluetooth module connections:

1. Bluetooth module **TX** connects to Arduino **RX**
2. Bluetooth module **RX** connects to Arduino **TX**

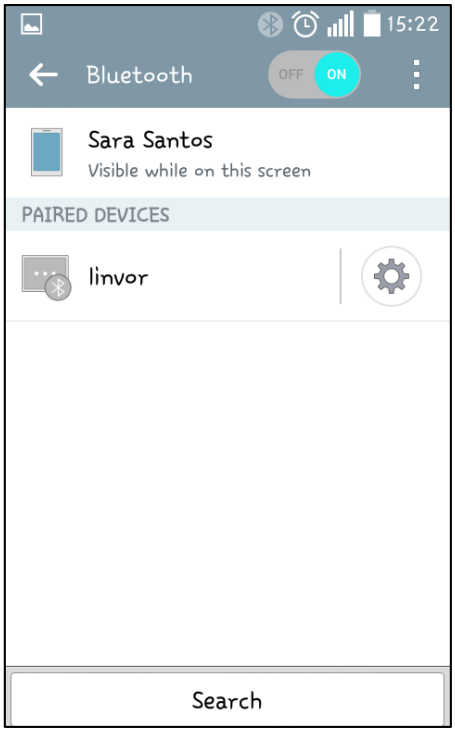
Demonstration

Congratulations! You've made it, your first Arduino Android app project.

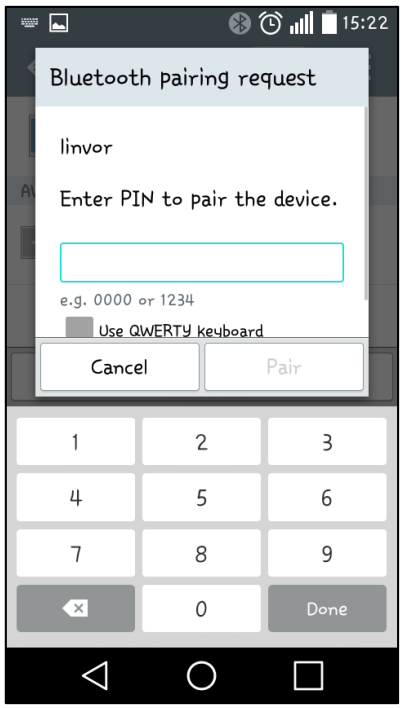
1. First, you need to enable your smartphone's bluetooth.



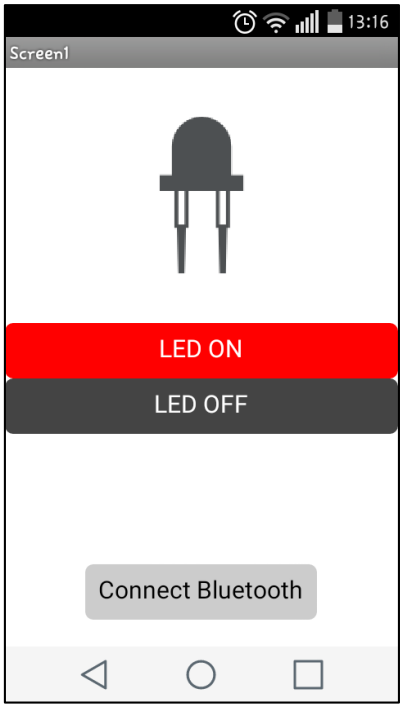
2. Then, you need to pair your smartphone with the bluetooth module. Go to your bluetooth settings and search for the available devices. Your bluetooth module device should appear (it's often called: linvor, HC-06, HC-04, HC-05, ...). Pair with it.



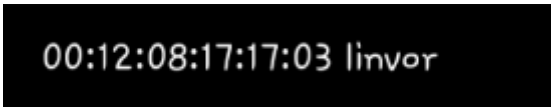
3. If it asks for a password, type 1234.



4. Open the app. The bluetooth connection is not established.

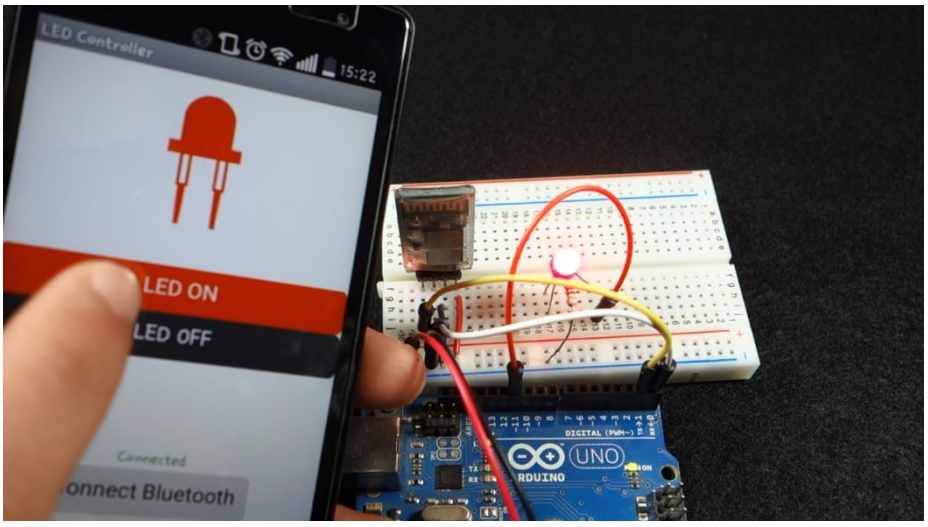


5. Click on the **Connect Bluetooth** button. A list picker shows up. Choose the ***linvor*** device and wait a few seconds for the communication to be established.

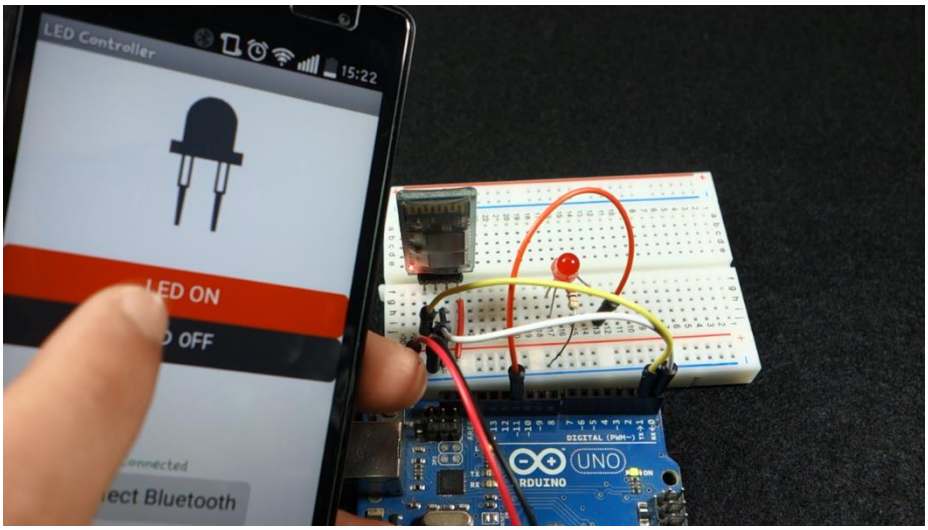


6. If the communication is well established you'll see a green message saying **Connected**. Your app is ready to control the LED!

Tapping the **LED ON** button, changes the LED image from gray to red in the app, and the physical LED lights up.

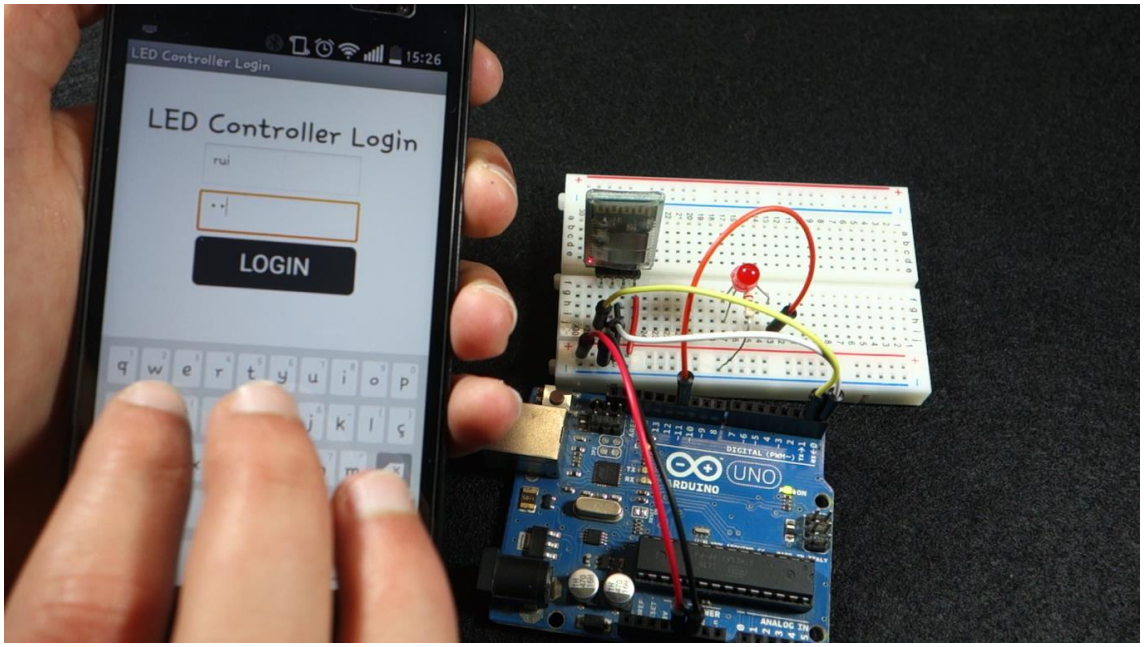


Tapping the **LED OFF** button, change the LED image back to gray, and the physical LED turns off.



We suggest that you edit the app and change the layout, colors and images to practice.

Project 2: Login Protected LED Controller



Project 2: Login protected LED controller

This project is a continuation of Project 1 with an additional feature. The LED controller screen is password protected. To control the LED, you must enter a valid username and password.

After learning how to make a login screen, you can take this concept and apply it to your other applications. You'll also learn how to make an app with MIT App Inventor 2 that has multiple screens.

Parts Required

Here's a complete list of parts you need to do this project – the same parts of the previous project.

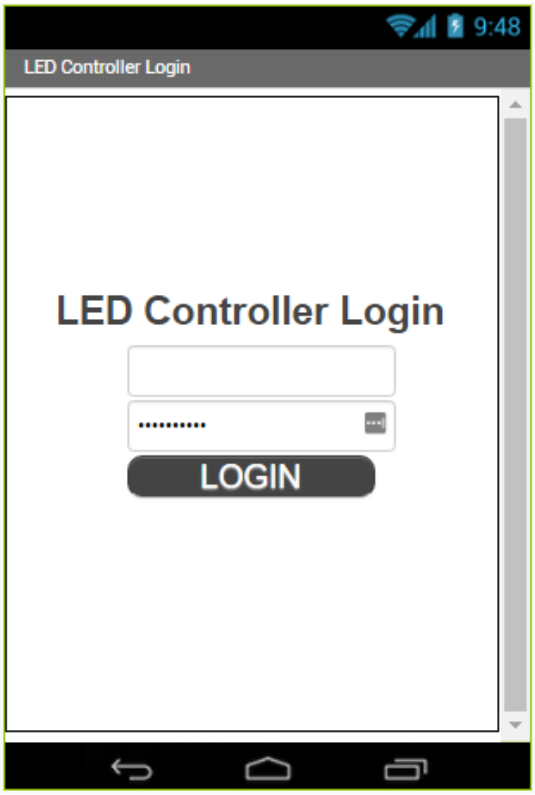
Figure	Name	eBay
	Arduino UNO	http://ebay.to/1SQda0R
	HC-04 or HC-05 or HC-06 Bluetooth module	http://ebay.to/2dt1b6M
	LED	http://ebay.to/20H2Oyy
	Breadboard	http://ebay.to/21bEojM
	220Ohm Resistor	http://ebay.to/1KsMYFP
	Jumper Wires	http://ebay.to/1PXeaJz

Building the App

Go to http://ai2.appinventor.mit.edu for opening the MIT App Inventor 2 software. Go to **Projects** ▸ **Start New Project** and call it *Led_Controller_Login_Protected*.

Alternatively, if you prefer, you can upload the *.aia* file and edit the app the way you like. You just have to go to **Projects** ▸ **Import project (.aia) from my computer** and upload the *LED_Controller_Login_Protected.aia* file that comes with the eBook's resources.

Before building the app, let's look at the result, so that you know what you're going to build. Here's how the login screen looks like:



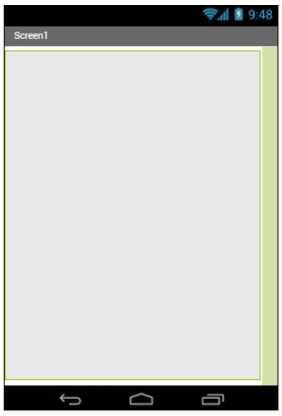
Here's what you have in this screen:

- A **Label** saying **LED Controller Login**
- A **TextBox** to insert the username
- A **PasswordTextBox** to insert the password
- A **Button** with the text **LOGIN**

With what you've learn in the previous project about app design, try to build this layout by yourself. Don't forget to rename the components with elucidative names. When you're done, proceed to the next section.

Designer

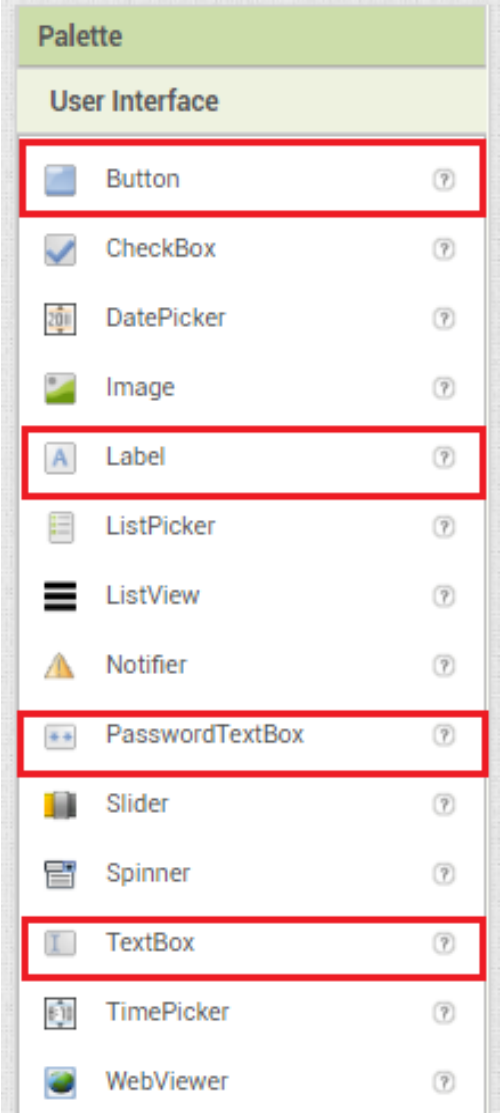
Go to **Pallet ▸ Layout** and drag a **VerticalArrangement** into the **Viewer**. Rename it **loginVerticalArrangement.**

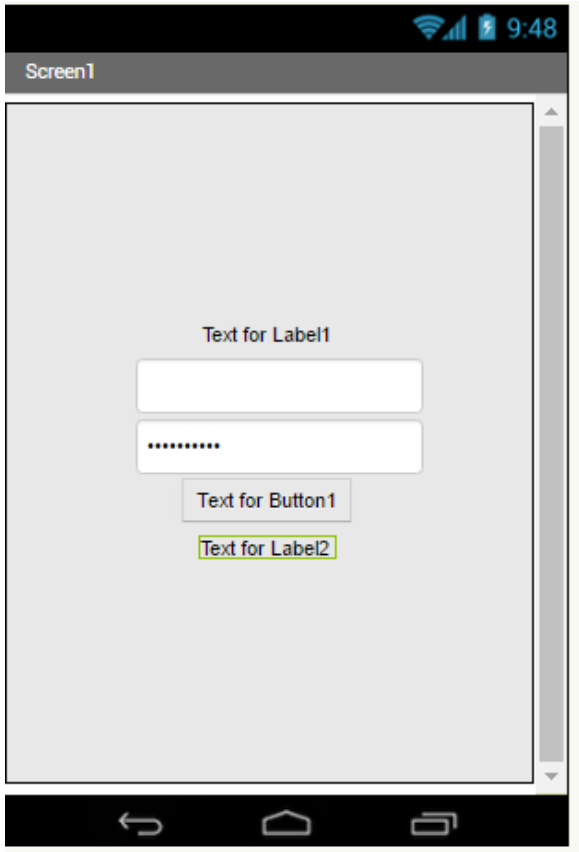


The **loginVerticalArrangement** has the following properties:

- **AlignHorizontal:** Center : 3
- **AlignVertical:** Center : 2
- **BackgroundColor:** white
- **Width:** fill parent
- **Image:** none
- **Visibel:** ✓

Next, go to **Pallet ▸ User Interface** and drag a **Label**, a **TextBox**, a **PasswordTextBox,** a **Button** and another **Label** into the **loginVerticalArrangement**.





Rename the components to **titleLabel**, **usernameTextBox**, **passPasswordTextBox**, **loginButton** and **statusLabel**, respectively. Then, change the components' properties as follows.

The **titleLabel** has the following properties:

- **BackgroundColor:** none
- **FontBold:** ✓
- **FonItalic:** ☐
- **FontSize:** 28
- **FontTypeFace:** default
- **HTML Formar:** ☐
- **HasMargins:** ✓
- **Height:** Automatic
- **Width:** Automatic
- **Text:** LED Controller Login
- **TextAlignment:** center : 1
- **TextColor:** **Dark Gray**
- **Visible:** ✓

The **usernameTextBox** has these next properties:

- **BackgroundColor:** Default
- **Enabled:** ✓
- **FontBold:** ☐
- **FontItalic:** ☐
- **FontSize:** 14
- **FontTypeface:** default
- **Height:** Automatic
- **Width:** Automatic
- **Hint:** Type your username
- **MultiLine:** ☐
- **NumberOnly:** ☐
- **Text:**
- **TextAlignment:** left : 0
- **TextColor:**
- **Visible:** ✓

Edit the **passPasswordTextBox** with the following properties:

- **BackgroundColor:** Default
- **Enabled:** ✓
- **FontBold:** ☐

- **FontItalic:** ☐
- **FontSize:** 14
- **FontTypeface:** default
- **Height:** Automatic
- **Widht:** Automatic
- **Hint:** Your password
- **Text:**
- **TextAlignment:** left : 0
- **TextColor:** Black
- **Visible:** ✓

Edit the **loginButton** as follows:

- **BackgroundColor**: **Dark Gray**
- **Enabled:** ✓
- **FontBold:** ☐
- **FonItalic:** ☐
- **FontSize:** 24
- **FontTypeface:** sans serif
- **Height:** Automatic
- **Width:** 50 percent
- **Image:** none
- **Shape:** rounded
- **Show Feedback:** ✓
- **Text:** LOGIN
- **TextAlignment:** center : 1
- **TextColor:** white
- **Visible:** ✓

Edit the **statusLabel:**

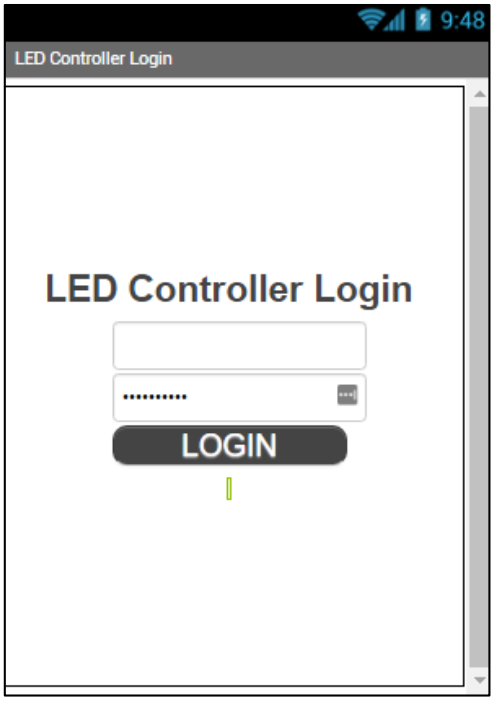
- **BackgroundColor**: none
- **FontBold:** ☐
- **FontItalic:** ☐
- **FontSize:** 16
- **FontType:** sans serif
- **HTMLFormat:** ☐
- **HasMargins:** ✓
- **Height:** Automatic
- **Width:** Automatic
- **Text:**

- **TextAlignment:** center : 1
- **TextColor:** red
- **Visible:** ✓

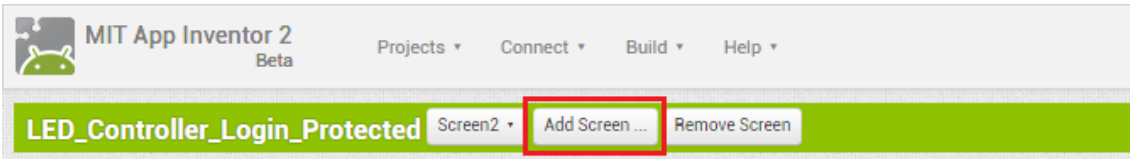
Finally, edit the **screen1** component:

- **AboutScreen**
- **AlignHorizontal:** Center : 3
- **AlignVertical:** Center : 2
- **AppName:** LED Controller Login
- **BackgroundColor:** White
- **BackgroundImage:** None
- **CloseScreenAnimation:** Default
- **Icon:** None
- **OpenScreenAnimation:** Default
- **ScreenOrientation:** Portrait
- **Scrollable:** ☐
- **ShowListsAsJson:** ☐
- **ShowStatusBar:** ✓
- **Sizing:** Responsive
- **Title:** LED Controller Login
- **TitleVisible:** ✓
- **VersionCode:** 1
- **VersionName:** 1.0

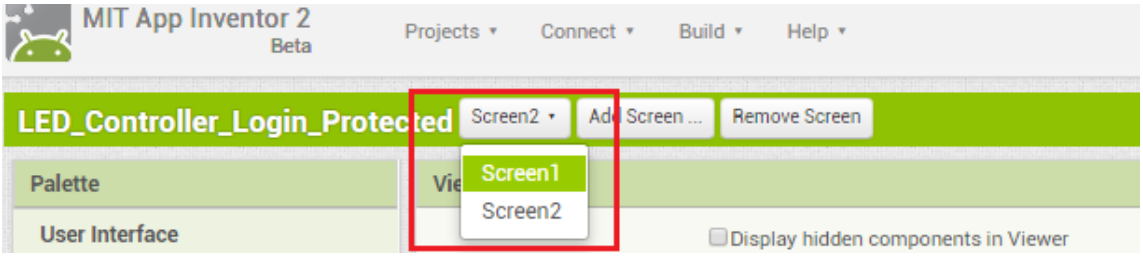
Your app layout should look like the figure below.



This is the login screen. You should create another screen for the controller LED app. For creating a new screen go to **Add Screen** and then click **ok**.



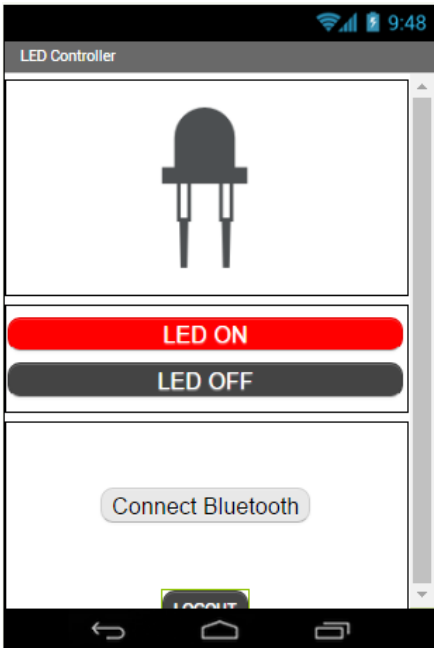
Now, you have two screens. To change back and forth between screens, you select the screen you want to edit as shown in figure below.



Select **Screen2**. This is where your app's core stays. In this case, the design is the same as the LED Controller app but with an extra button called **LOGOUT**.

Click here to be redirected to the PDF page that contains the instructions to design the LED Controller app. Once you have the layout completed, add a new **Button** with the following properties:

- **BackgroundColor**: **Dark Gray**
- **Enabled:** ✓
- **FontBold:** ☐
- **FonItalic:** ☐
- **FontSize:** 14
- **FontTypeface:** sans serif
- **Height:** Automatic
- **Width:** Automatic
- **Image:** none
- **Shape:** rounded
- **Show Feedback:** ✓
- **Text:** LOGOUT
- **TextAlignment:** center : 1
- **TextColor:** white
- **Visible:** ✓



Your **Screen2** should look as the figure at the right.

Blocks Editor

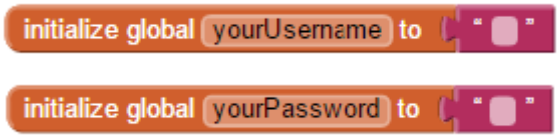
Go to the **Blocks** editor and select **Screen1.**

Adding global variables

In the **Blocks** section go to **Variables** and drag two **initialize global name to** blocks into the **Viewer**. These blocks initialize global variables.

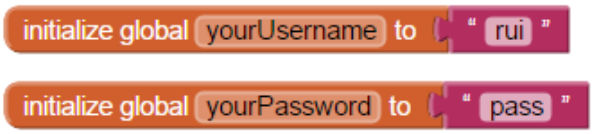
Click on **name** and change the variables' names to **yourUsername** and **yourPassword.**

Next, go to the **Text** section and drag a **" "** block into each of the previous blocks. So that you have the following.



Inside the **" "** you should write the username and password you want for your login.

These blocks create two global text variables that save the username and password of your application. In this example we are using username ***rui*** and password ***pass***.



Login logic

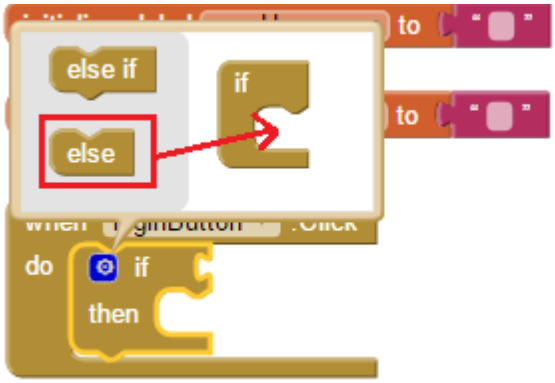
The next step is creating the blocks that decide what happens when you tap the **LOGIN** button.

When you tap the **LOGIN** button, the blocks should compare the username and password written with the actual username and password you've defined in the global variables blocks. We will consider these conditions:

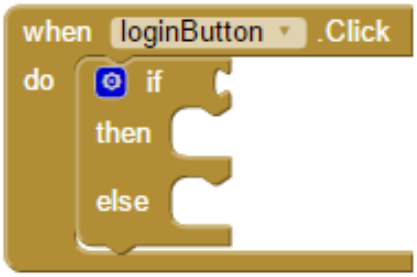
- If you don't fill the fields, it displays a message saying "Username or password is empty";
- If you don't enter the right username and/or password you'll get "Invalid username or password";
- If both the username and password are correct, it opens **Screen2** where you can control the LED.

To make this happen, in the **Blocks** editor go to **loginButton** and drag a **when loginButton.Click** block. Go to **Control** and drag an **if then** into the previous block.

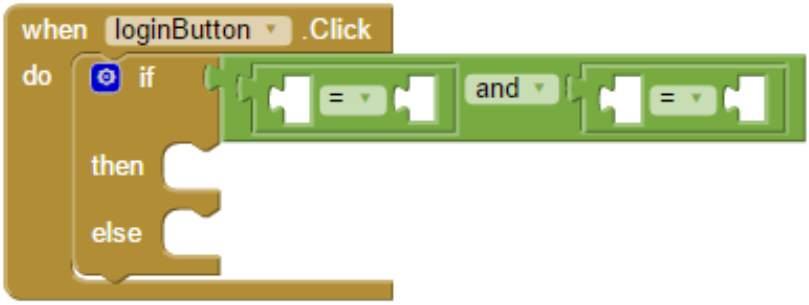
Then, click on the blue square inside the **if then** button and drag the **else** block into the **if** block.



Your block is mutated to an **if... then... else** block.



The first condition we're setting is: password and username empty. Go to **Logic** and drag an **and** block into the **if** space. Then, add two **"="** comparative blocks on each side of the **and** block.

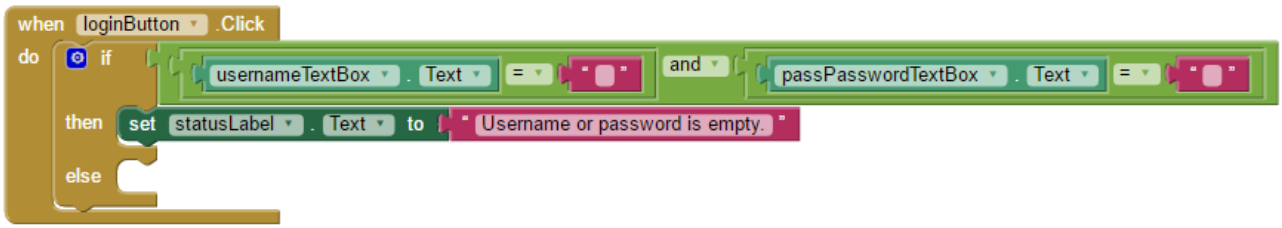


We need to compare if the **usernameTextBox** text is equal to empty **" "** and if the **passPasswordTextBox** is also equal to empty **" ".** In the **Blocks** section go to **usernameTextBox** and drag the **usernameTextBox.Text** into the left side of the **"="** box. Go to text and drag a **" "** box on the right side of the **"="** box**.** Repeat this process with the **passPasswordTextBox.**

You should have the following:

If this condition is true, the **statusLabel** is set to "Username or password is empty".

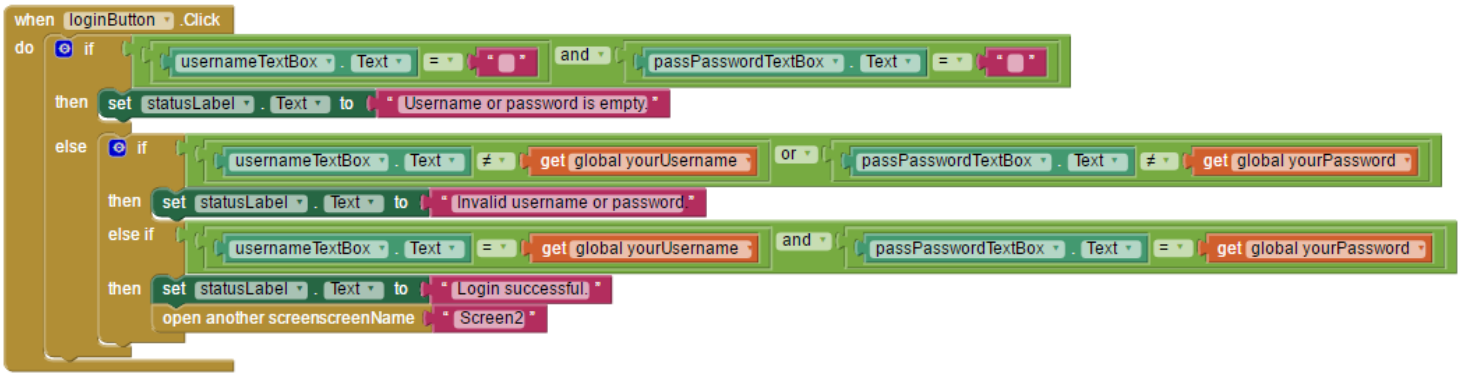
Go to **statusLabel** and drag the **set statusLabel.Text** box into the **then** space. In the **Text** category, drag the first block **" "** into the **set statusLabel.Text.** In the **" "** block write the following message: "**Username or password is empty"**.



You need to compare if the **usernameTextBox.Text** is different from the **globalUsername** or if the **passPassword** is different from the **globalPasword**. If one of them is different, we want our label to show "**Invalid username or password**".

So, you need to build the following block:

Finally, we need to set the condition when both the password and username are correct. If both are correct we want our label to display "**Login successful**" and to open **Screen2**.

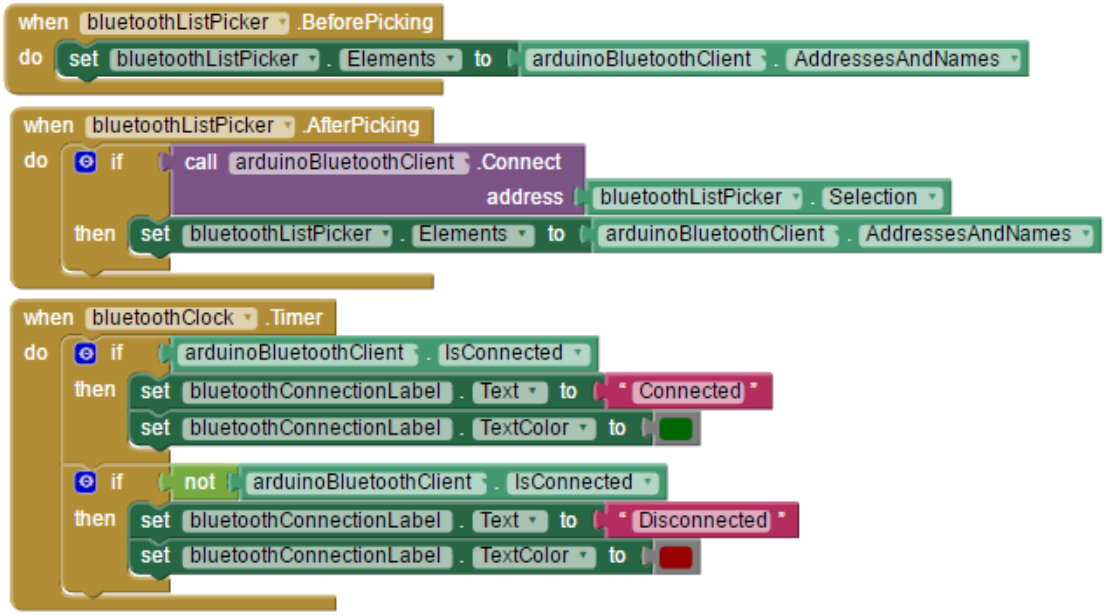


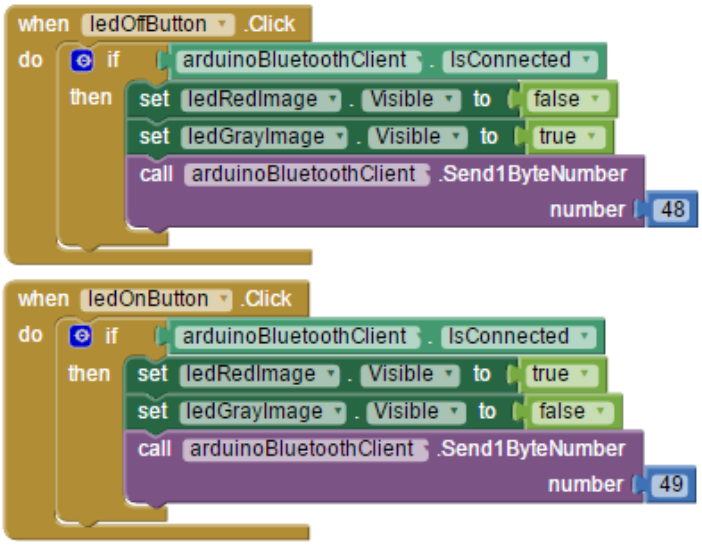
The **open another screenscreenName** block opens **Screen2**. You should write in the pink box the name of the second screen. In our case, it is called **Screen2.**

Screen2 blocks

Having the login screen completed and **Screen2** selected, it is time to add the code blocks to **Screen2**. The code blocks are the same as in Project 1, but you need to set what happens when you tap the **LOGOUT** button.

Your blocks should look like these:





Lastly, you need to add a function to your **logoutButton** button that logs out from the LED Controller screen, **Screen2**.

These are the blocks that do that:

Congratulations! Your app is ready. Go to the **Build** tab to build the app and download the *.apk* file to your phone. If you don't know how to do this, go to Project 1. Here's how the app looks in an Android phone.

Writing the Sketch

The Arduino sketch is the same as in Project 1. Upload the following code to your Arduino board. Make sure you have the right board and COM port selected.

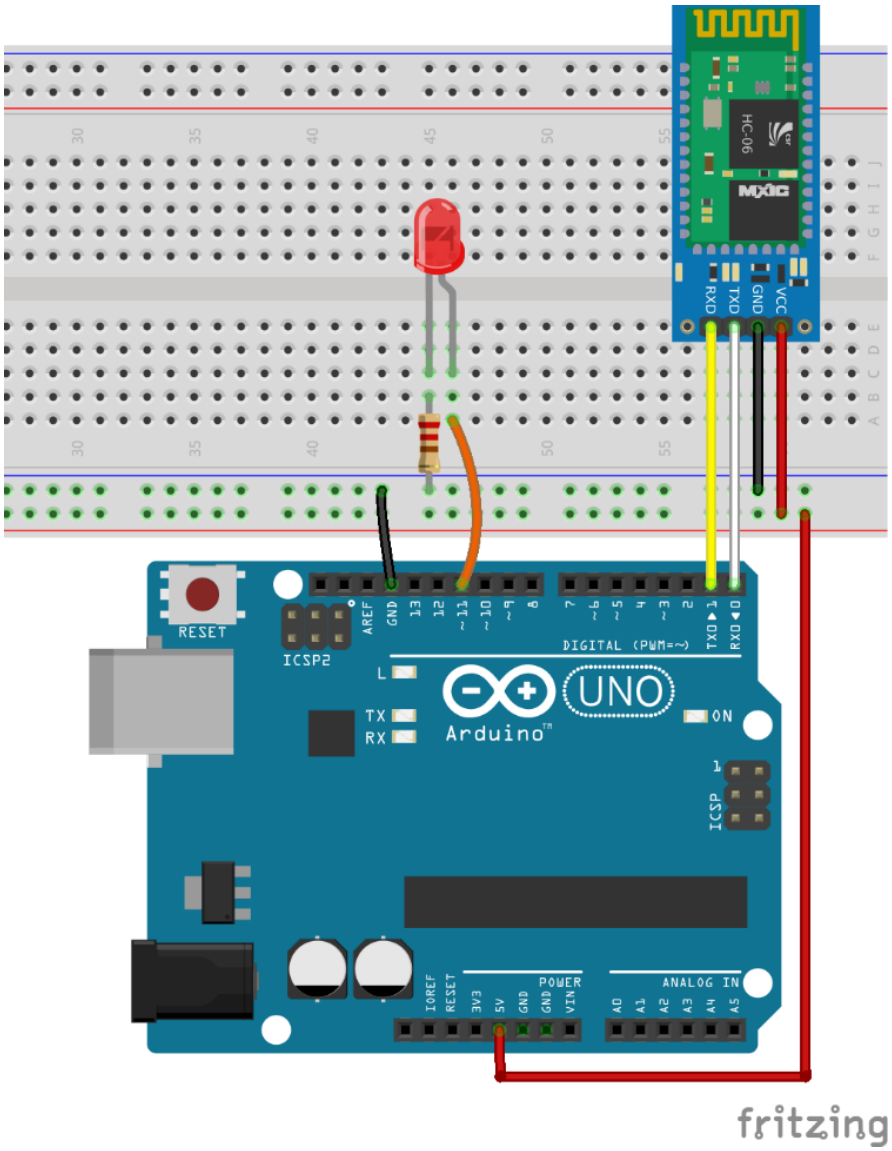
```
/*
 * Rui Santos
 * Complete Project Details http://randomnerdtutorials.com
 */

int led = 11;     // pin Digital 11
int state;          // saves the state
int flag=0;  // makes sure that the serial only prints once the state

void setup() {
    // sets the led as output:
    pinMode(led, OUTPUT);
    digitalWrite(led, LOW);

    // initialize serial communication at 9600 bits per second:
    Serial.begin(9600);
}

void loop() {
    //if some date is sent, reads it and saves in state
    if(Serial.available() > 0){
      state = Serial.read();
      flag=0;
    }
    // if the state is '0' the led will turn OFF
    if (state == '0') {
        digitalWrite(led, LOW);
        if(flag == 0){
          Serial.println("LED Off!");
          flag=1;
        }
    }

    // if the state is '1' the led will turn ON
    else if (state == '1') {
        digitalWrite(led, HIGH);
        if(flag == 0){
          Serial.println("LED On!");
          flag=1;
        }
    }
    //Uncomment For debugging purpose
    //Serial.println(state);
}
```

Wiring the Circuit

The circuit can be quickly wired with the next schematics:

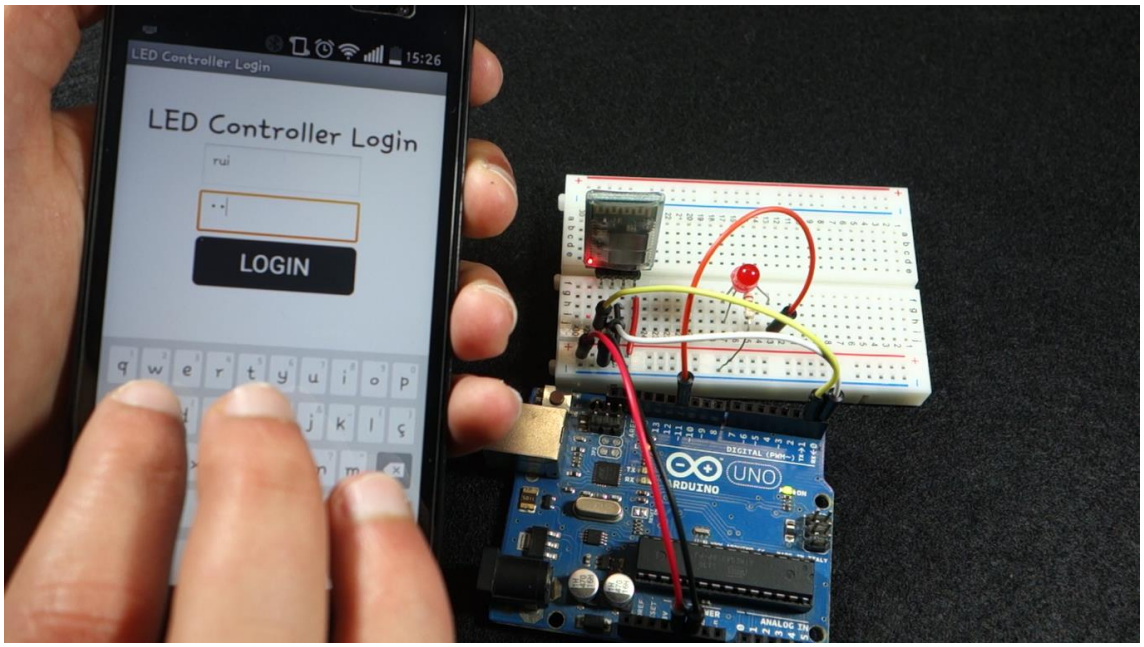
Warning: When you're uploading code to your Arduino you should disconnect the TX and RX cables from the bluetooth module. These pins are needed for serial communication between the Arduino and your computer.

Tip: People often make the wrong connections between the Arduino and bluetooth module. **TX** from the bluetooth module connects to **RX** pin of the Arduino. **RX** from the bluetooth module connects to **TX** pin of the Arduino.

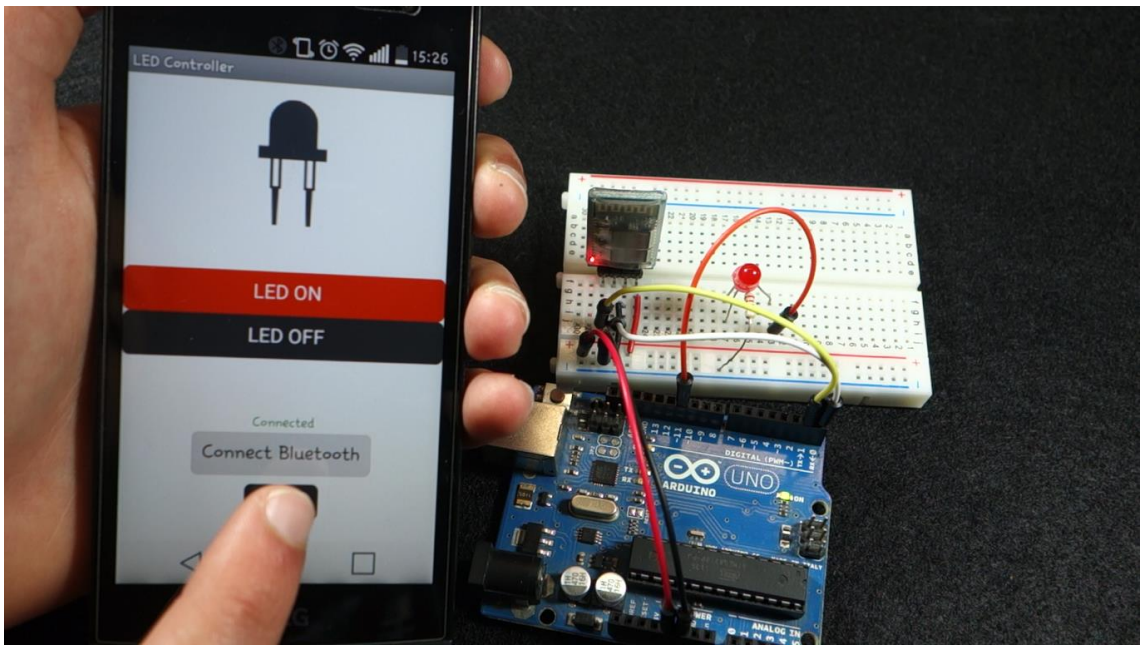


Demonstration

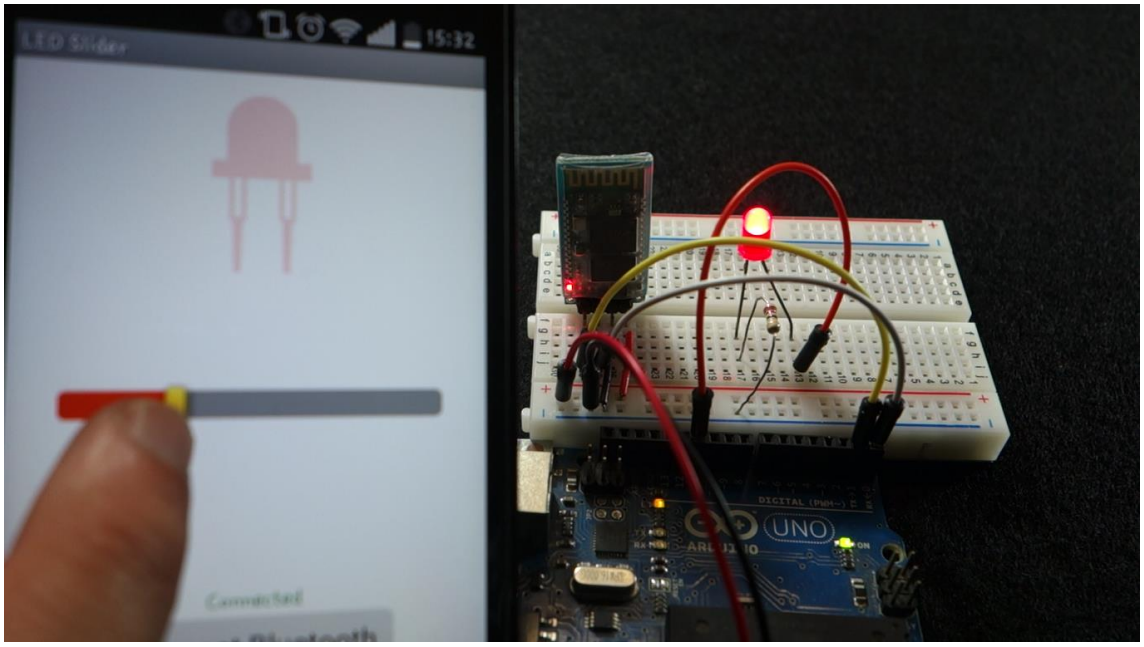
Try your app! When you first open the app, you need to type your username and password.



Then, if the credentials are correct, you can access the LED controller screen. To go back to the login screen, simply tap the **LOGOUT** button.



Project 3: LED Slider



Project 3: LED Slider

This project shows you how to build a slider that controls the LED brightness. The app contains an LED image that changes color accordingly to the slider position. The slider at its minimum displays a gray LED image, and the slider at its maximum shows a red LED image. Intermediate slider values show intermediate color intensity.

Parts Required

Here's a complete list of parts you need to do this project.

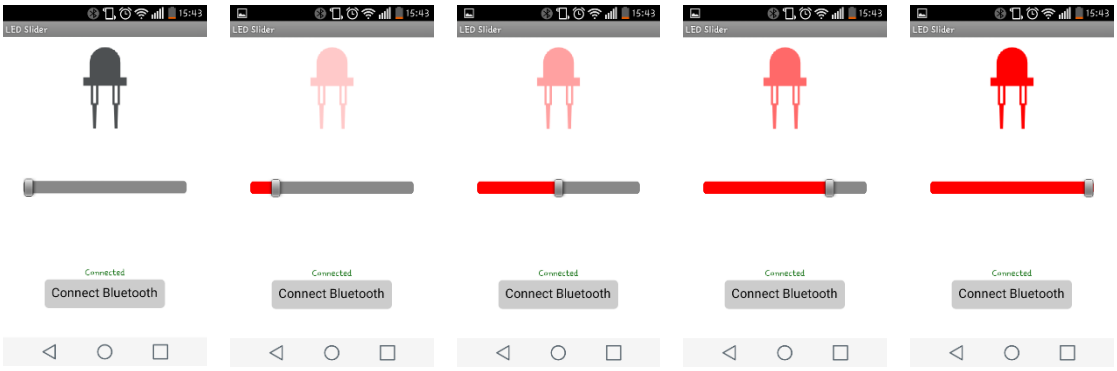
Figure	Name	eBay
	Arduino UNO	http://ebay.to/1SQda0R
	HC-04 or HC-05 or HC-06 Bluetooth module	http://ebay.to/2dt1b6M
	LED	http://ebay.to/20H2Oyy
	Breadboard	http://ebay.to/21bEojM
	220Ohm Resistor	http://ebay.to/1KsMYFP
	Jumper Wires	http://ebay.to/1PXeaJz

Building the App

Go to http://ai2.appinventor.mit.edu for opening the MIT App Inventor 2 software.
Go to **Projects** ▸ **Start New Project** and call it *LED_Slider*

> Alternatively, if you prefer, you can upload the *.aia* file and edit the app the way you like. You just have to go to **Projects** ▸ **Import project (.aia) from my computer** and upload the *LED_Slider.aia* file that comes with the eBook's resources.

Before building the app, let's see how it looks.



As you move the slider, the user interface changes the LED image color accordingly.

Designer

In the **Designer** tab, drag a **HorizontalArrangement** into the **Viewer**, that's where you're going to place the LED images. Rename it to **ledImageArrangement** and make sure it has the following properties:

- **AlignHorizontal:** Center : 3
- **AlignVertical:** Center : 2
- **BackgroundColor:** none
- **Height:** 40 percent...
- **Width:** fill parent...
- **Image:** none...
- **Visible:** ✓

Then, drag another **HorizontalArrangement** into the **Viewer**. This is where you're going to place your slider. Rename the **HorizontalArrangement** to **sliderHorizontalArrangement** and edit its properties:

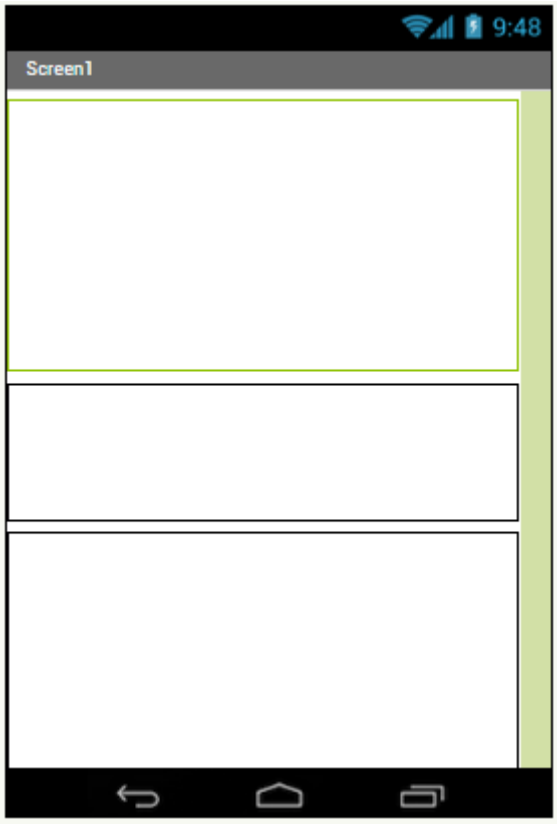
- **AlignHorizontal:** Center : 3
- **AlignVertical:** Center : 2
- **BackgroundColor:** none

- **Height:** 20 percent…
- **Width:** fill parent…
- **Image:** none…
- **Visible:** ✓

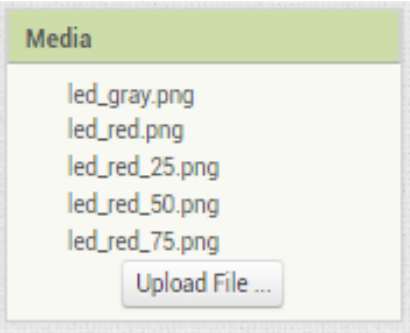
Finally, drag a **VerticalArrangement** into the **Viewer** where you're going to place the **Connect Bluetooth** button and the bluetooth label. Rename the arrangement to **connectVerticalArrangement** and edit its properties:

- **AlignHorizontal:** Center : 3
- **AlignVertical:** Center : 2
- **BackgroundColor:** none
- **Height:** 40 percent…
- **Width:** fill parent…
- **Image:** none…
- **Visible:** ✓

You should have the following layout:



Then, go to the **Assets** folder and upload the LED images that come in this eBook's resources.

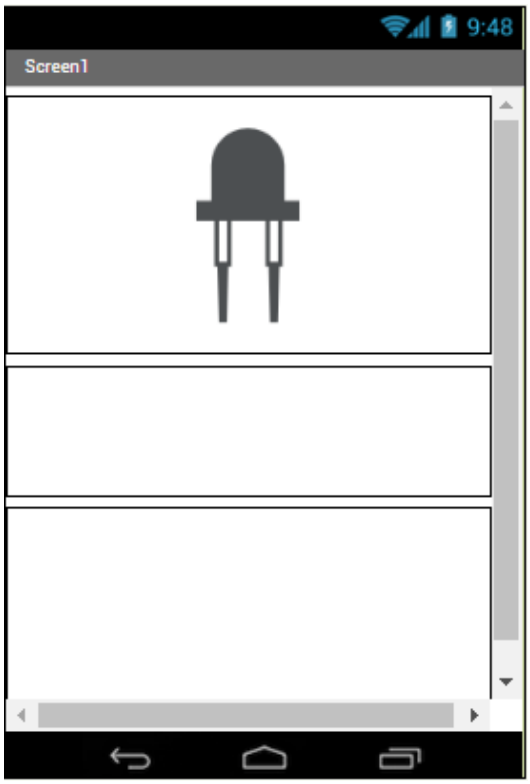


In the **Pallete**, go to the **User Interface** section and drag 5 **Images** into the **ledImageArrangement**. These **Images** will contain the LED images. Rename the **Images** with the following names that should contain the following images:

- ledRedImage: led_red.png
- ledGrayImage: led_gray.png
- ledRed75Image: led_red_75.png
- ledRed50Image: led_red_50.png
- ledRed25Image: led_red_25.png

In the images properties, only the **ledGrayImage** should have a ✓ in the **Visible** option. All other images should have a blank square ☐. When the application starts, it shows the gray LED image, meaning the LED is turned off. Then, in the **Blocks** section, we'll set the other images to visible accordingly to the slider position.

Your app should look like this:



Inside the **Pallete**, go to the **User Interface** and drag a **Slider** into the **sliderHorizontalArrangement.** Rename it to **ledSlider** and edit its properties:

- **ColorLeft:** Red
- **ColorRight:** Gray
- **Width:** 80 percent
- **MaxValue:** 255
- **MinValue: 0**
- **ThumbEnaled:** ✓
- **ThumbPosition:** 1
- **Visible:** ✓

The **MaxValue** is set to 255 because that's the maximum value the Arduino sends to its output PWM pins.

Note: Remember that you can test your app as you build it using the **MIT AI2 Companion**. You just need to have the MIT AI2 Companion app installed in your smartphone. In the MIT App Inventor 2 software go to **Connect ▸ AI Companion.**

Using the AI Companion app to preview how your app looks in your smartphone is particularly important in this project. What you see in the MIT App Inventor 2 software is slightly different from what you get in the final app.

In the **connectVerticalArrangement** drag a **label** and a **listpicker.** Rename them to **bluetoothConnectionLabel** and to **bluetoothListpicker.**

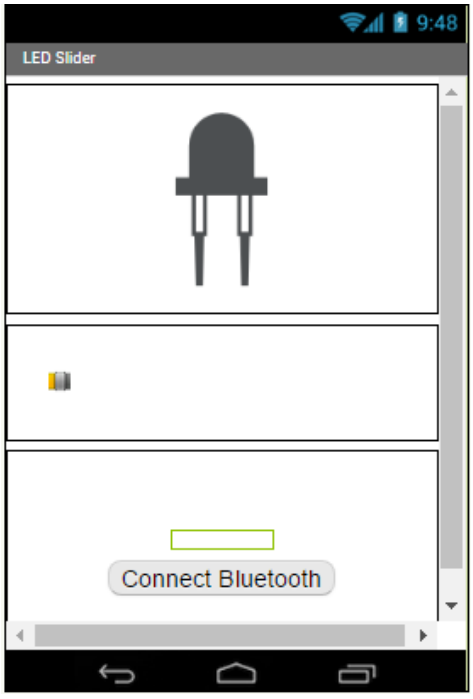
The **bluetoothConnectionLabel** should have the following properties:

- **BackgroundColor**: none
- **FontBold:** ☐
- **FontItalic:** ☐
- **FontSize:** 14
- **FontType:** default
- **HTMLFormat:** ☐
- **HasMargins:** ✓
- **Height:** Automatic
- **Width:** Automatic
- **Text:** Disconnected
- **TextAlignment:** center : 1
- **TextColor:** none
- **Visible:** ✓

The **bluetoothListPicker** should have the following properties**:**

- **BackgroundColor:** Default
- **ElementsFromString:**
- **Enabled:** ✓
- **FontBold:** ☐
- **FontItalic:** ☐
- **FontSize:** 20
- **FontTypeface:** sans serif
- **Height:** Automatic
- **Width:** Automatic
- **Image:** none
- **ItemBackgroundColor:** Black
- **ItemTextColor:** White
- **Selection:**
- **Shape:** rounded
- **ShowFeedBack:** ✓
- **ShowFitterBar:** ☐
- **Text:** Connect Bluetooth
- **TextAlignment:** center : 1
- **TextColor:** Default
- **Title:**
- **Visible:** ✓

This is how your app looks like:



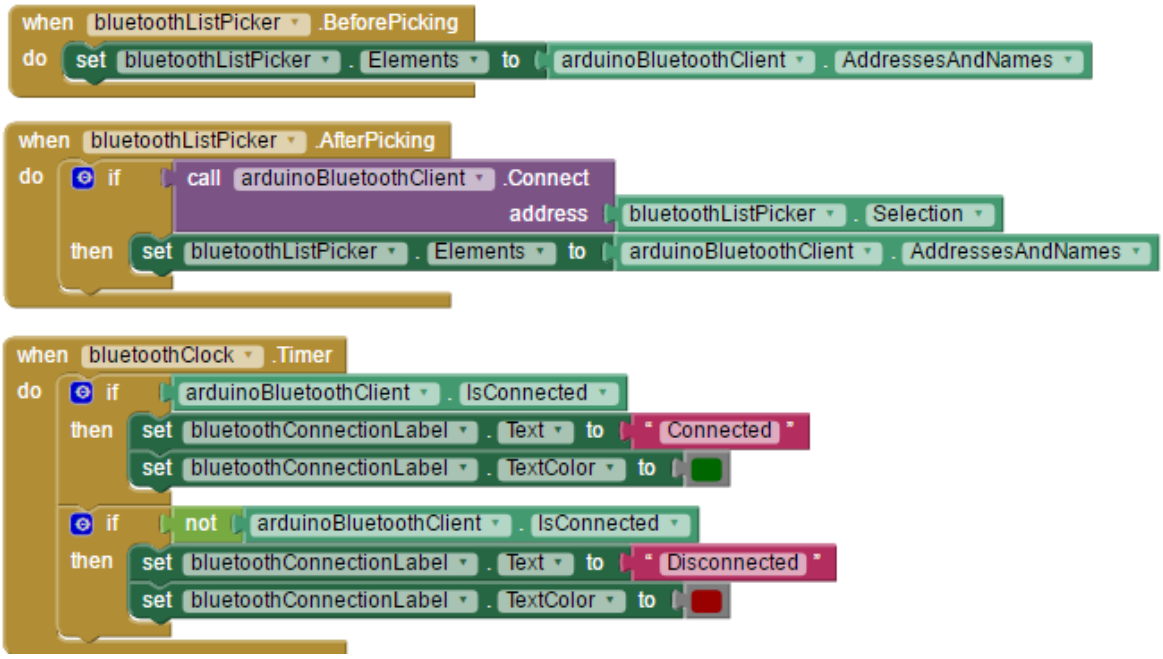
To finish the app design, you need to add a **Bluetooth Client** and a **Clock**.

In the **Palette,** go to **Connectivity** and drag a **BluetoothClient** into the app. Go to **Sensors** and drag a **Clock.** Rename the components to **arduinoBluetoothClient** and **bluetoothClock.**

The app design is ready. Let's move on to the **Blocks** editor.

Blocks Editor

The first thing to do is to connect your smartphone to the bluetooth module via bluetooth. For that, you need these blocks.



Slider

The next step is sending values to the Arduino bluetooth module accordingly to the slider position. Check the blocks on the next page.

Those blocks send the thumb position to the Arduino bluetooth module. Then, we set which LED image should be displayed on the user interface accordingly to the thumb position on the slider.

- < 1 : ledGrayImage
- ≥ 1 and < 80 : ledRed25Image
- ≥ 80 and < 190 : ledRed50Image
- ≥ 190 and < 250 : ledRed75Image
- ≥ 250: ledRedImage

To create the Android app, go to the **Build** tab and generate a new *.apk* file. Alternatively, you can use the *.apk* file that comes with the eBook's resources folder. Move the *.apk* file to your Android phone and follow the on-screen instructions to install it.

Writing the Sketch

Here's the sketch for this project. Upload it to your Arduino board – make sure you have the right board and COM port selected.

Warning: When you're uploading code to your Arduino you should disconnect the TX and RX cables from the bluetooth module. These pins are needed for serial communication between the Arduino and your computer.

Tip: People often make the wrong connections between the Arduino and bluetooth module. **TX** from the bluetooth module connects to **RX** pin of the Arduino. **RX** from the bluetooth module connects to **TX** pin of the Arduino.

```
/*
 * Rui Santos
 * Complete Project Details http://randomnerdtutorials.com
 */

int ledPWM = 11;

int state;
int flag = 0;   //makes sure that the serial only prints once the state

void setup() {
  // sets the pin as output:
  pinMode(ledPWM, OUTPUT);

  analogWrite(ledPWM, LOW);

  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
}

void loop() {

  //if some date is sent, reads it and saves in state
  if (Serial.available() > 0) {
    state = Serial.read();
    flag = 0;
  }

  if (flag == 0) {
    analogWrite(ledPWM, state);
    flag = 1;
  }

}
```

The code starts by initializing the `ledPWM` variable to 11, which is the pin the LED is connected to. You initialize a `state` variable as an integer, which is the variable that saves the information received via bluetooth.

In the `setup()` you set the LED as an output:

```
pinMode(ledPWM, OUTPUT);
```

And you set it to LOW, so that when the code starts the LED is off.

```
analogWrite(ledPWM, LOW);
```

Then, you initialize the serial communication at a baud rate of 9600 bits per second:

```
Serial.begin(9600);}
```

In the `loop()`, you read the information received from the bluetooth module with the following line:

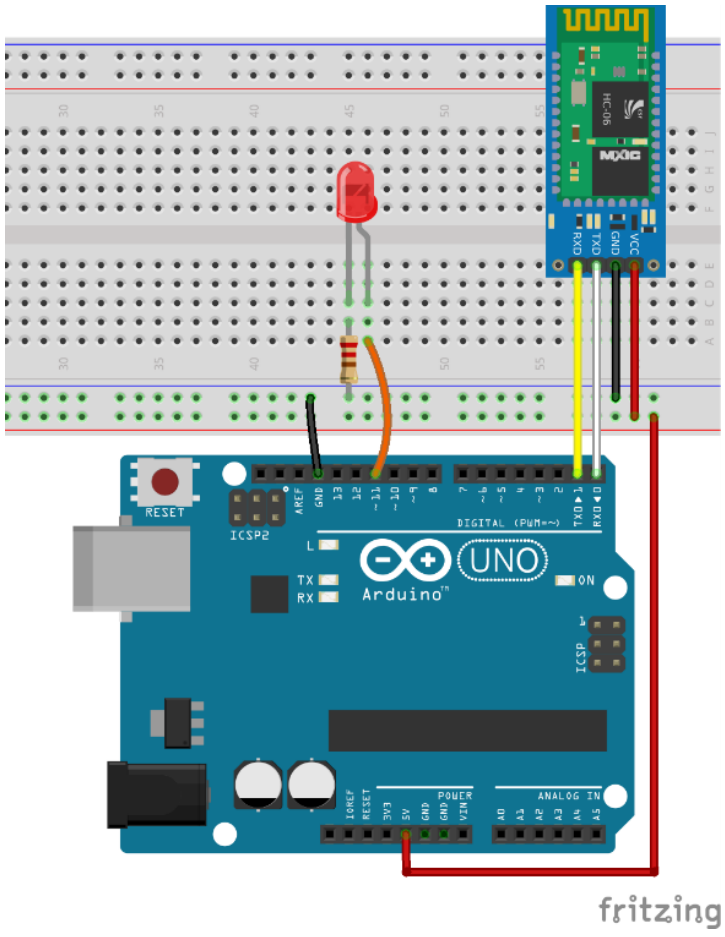
```
state = Serial.read();
```

Finally, the LED brightness is set to the value received via bluetooth:

```
analogWrite(ledPWM, state);
```

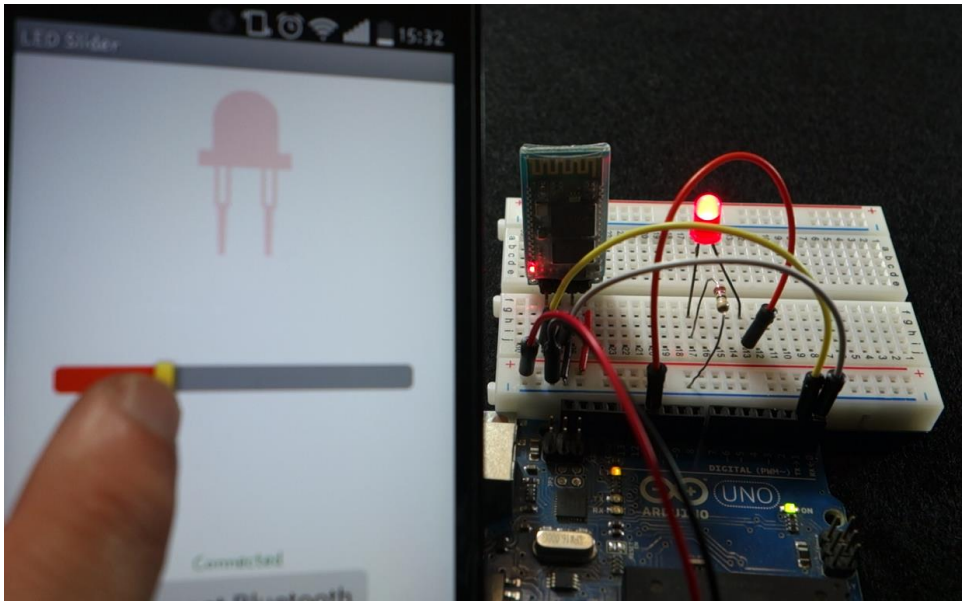
Wiring the Circuit

Here's the circuit for this project.

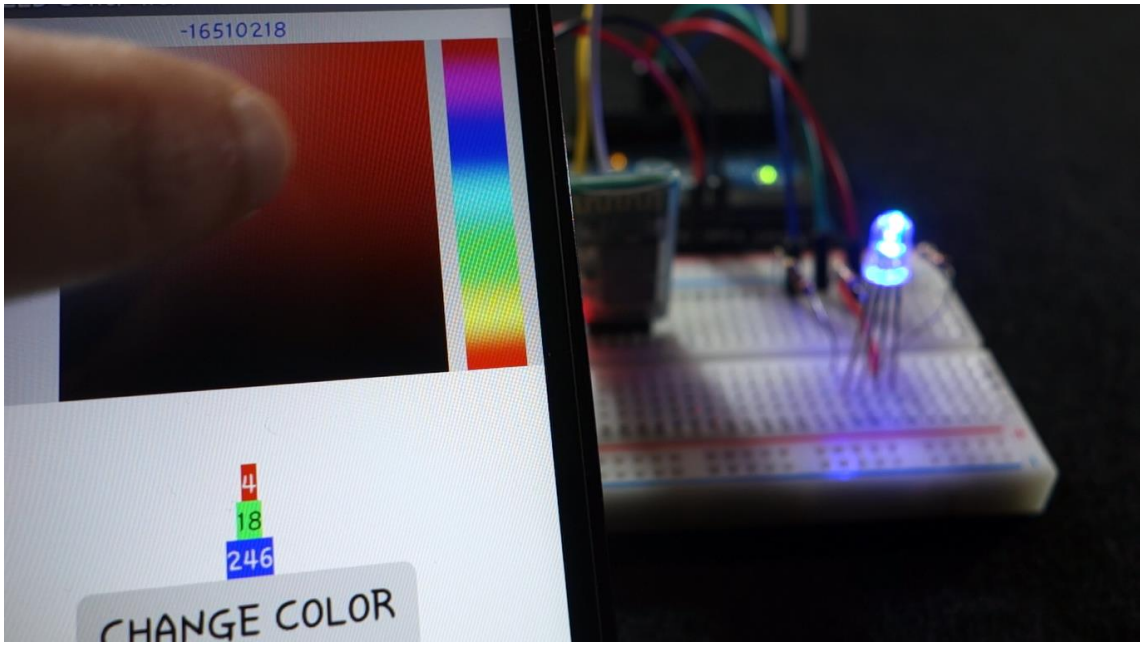


Demonstration

After establishing the bluetooth connection, when you move the slider, both the physical LED and the LED image change brightness accordingly to the current thumb position.



Project 4: RGB LED Controller



Project 4: RGB LED Controller

In this project you're going to control the color of an RGB LED. You'll build a color picker to set the color. The app you're going to build was inspired in a project called "A Better Color Picker" by Italo.

Parts Required

Here's a complete list of parts you need to do this project.

Figure	Name	eBay
	Arduino UNO	http://ebay.to/1SQda0R
	HC-04 or HC-05 or HC-06 Bluetooth module	http://ebay.to/2dt1b6M
	RGB LED (common anode)	http://ebay.to/2nIxaE6
	Breadboard	http://ebay.to/21bEojM
	3x 220Ohm Resistor	http://ebay.to/1KsMYFP
	Jumper Wires	http://ebay.to/1PXeaJz

Introducing RGB LEDs

With an RGB LED you can produce almost any color you want. How is this possible with just one single LED?

RGB LED

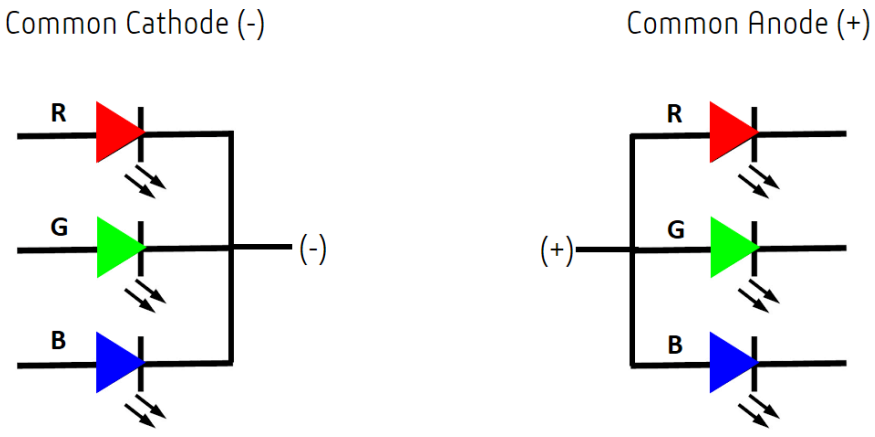
An RGB LED is a combination of 3 LEDs in one package:

- 1x Red LED
- 1x Green LED
- 1x Blue LED

The light that the RGB LED produces is made combining these three LED colors. An RGB LED is illustrated in the following figure:

Two types of RGB LEDs

There are common anode RGB LEDs and common cathode RGB LEDs. See figure below:



As you can see, the 3 LEDs can either share the cathode or the anode. This results in an RGB LED that has 4 pins, one for each LED, and one common cathode or one common anode. The common anode RGB LED is the most popular type.

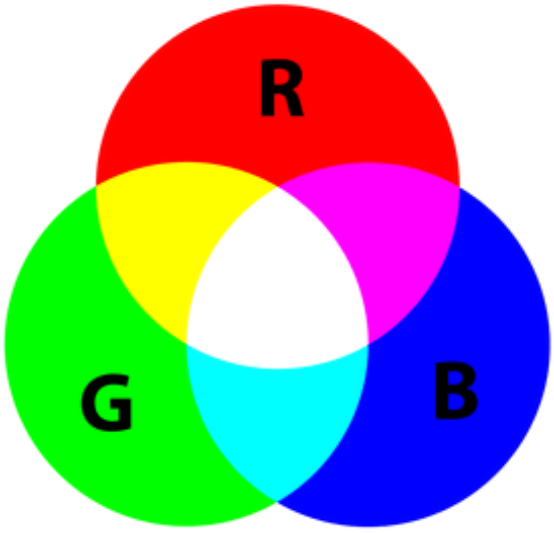
Colors

You can create one of those three colors – red, green or blue – by activating just one LED. For example, if you want to produce pure blue, you set the blue LED to its maximum brightness and turn off the other two.

Mixing colors

To produce other colors, you combine the three colors in different intensities. To generate different colors you can use PWM to adjust the brightness of each LED. As the LEDs are very close to each other, we can only see the final colors result rather than the three colors individually.

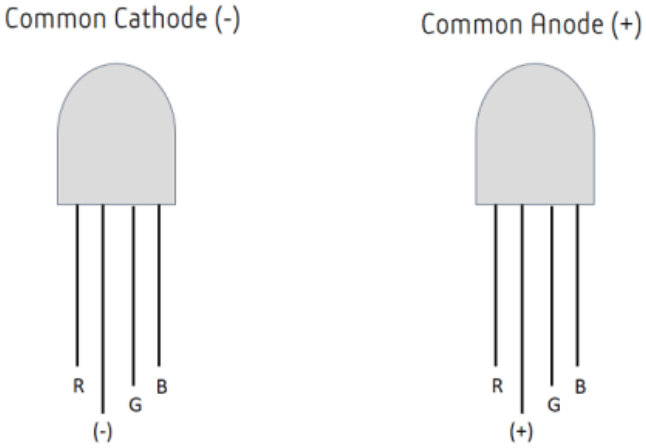
To have an idea on how to combine the colors, look at the following chart.



RGB LED Pins

RGB LEDs have 4 pins which can be distinguished by their length. The longest one is the ground (-) or voltage (+) depending if it is a common cathode or common anode LED, respectively.

The other three legs correspond to red, green, and blue, as you can see in the figure below:

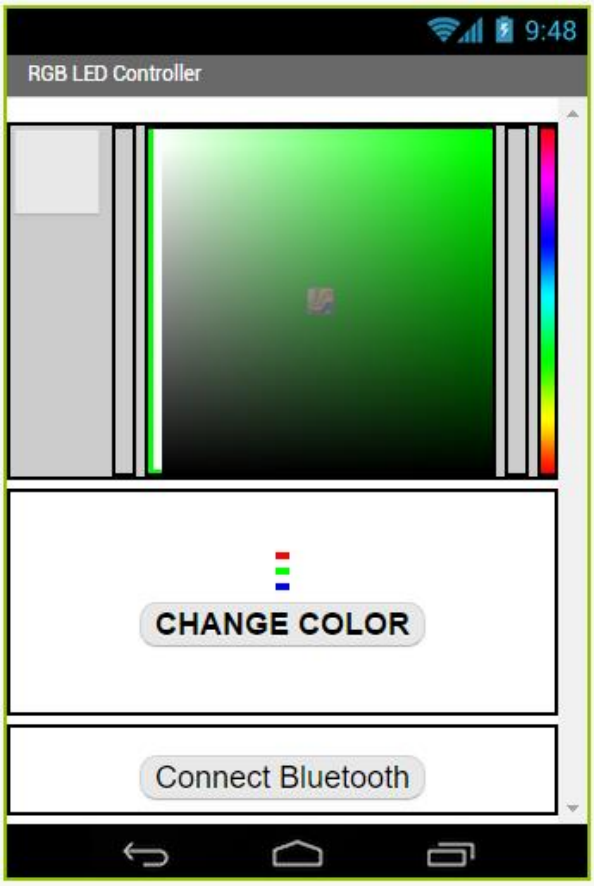


Building the App

Go to http://ai2.appinventor.mit.edu for opening the MIT App Inventor 2 software. Go to **Projects** ▸ **Start New Project** and call it *RGB_LED_Controller*.

Alternatively, if you prefer, you can upload the *.aia* file and edit the app the way you like. You just have to go to **Projects** ▸ **Import project (.aia) from my computer** and upload the *RGB_LED_Controller.aia* file that comes with the eBook's resources.

Here's how your app looks by the end of this project:



Here's how your app works:

- At the right, you choose the color – on that rainbow strip.
- Then, you choose the intensity at the center.
- The square at left, shows the selected color.
- You also have a label showing the RGB color.
- The **CHANGE COLOR** button, sends the selected color to the bluetooth module.

Design

You have all the materials available for this app on the eBook's resources folder.

Layout

Start by setting your app's layout. Go to **Palette ▸ Layout** and drag a **verticalArrangement** and two **horizontalArrangements** into the **Viewer**. Rename them to **colorPickerHorizontalArrangement**, **buttonVerticalArrangement** and **connectVerticalArrangement**.

Set the following properties for the **colorPickerHorizontalArrangement**:

- **AlignHorizontal:** Center : 3
- **AlignVertical:** Top : 1
- **BackgroundColor:** Light Gray
- **Height:** Automatic...
- **Width:** Fill parent...
- **Image:** none...
- **Visible:** ✓

Edit the **buttonVerticalArrangement** properties:

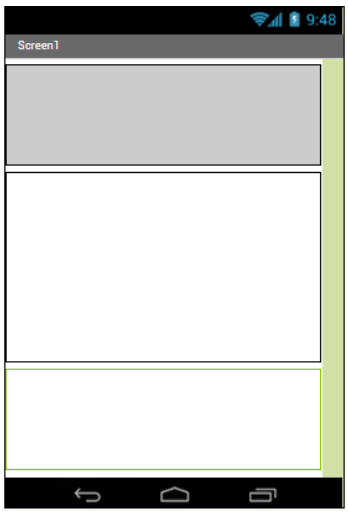
- **AlignHorizontal:** Center : 3
- **AlignVertical:** Center : 2
- **BackgroundColor:** white
- **Height:** fill parent...
- **Width:** fill parent...
- **Image:** none...
- **Visible:** ✓

Finally, the **connectVerticalArrangement:**

- **AlignHorizontal:** Center : 3
- **AlignVertical:** Center : 2
- **BackgroundColor:** white
- **Height:** automatic

- **Width:** fill parent...
- **Image:** none...
- **Visible:** ✓

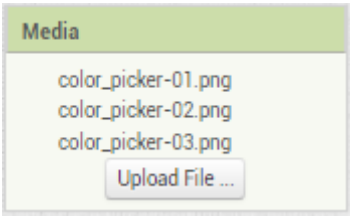
The figure below shows how your app layout looks after editing the arrangements.



Media

Next, upload the provided images. Go to **Media** ▸ **Upload File** and upload the following images:

- color_picker-01.png
- color_picker-02.png
- color_picker-03.png



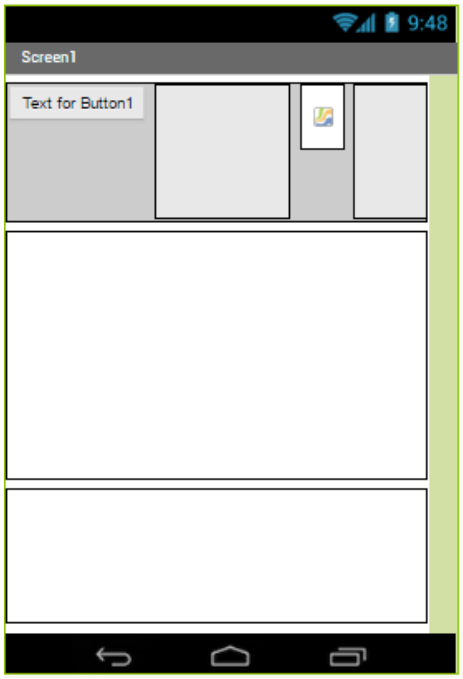
Color picker

Go to the **Palette,** drag a **Button** and two **verticalArrangement** into the **colorPickerHorizontalArrangement**. Rename the button to **displayColorButton.** This button changes color accordingly to the color chosen in the color picker.

Then, go to **Drawing and Animation** and drag two **Canvas** into the **colorPickerHorizontalArrangement.** Rename one of the **Canvas** to **colorPickerCanvas.**

Organize the components inside the **colorPickerHorizontalArrangement** so that they stay in this order: **displayColorButton**, **verticalArrangement1**, **colorPickerCanvas** and **verticalArrangement2**.

Your app will look weird. Like the figure below:



Edit the **Components** properties. Change the **displayColorButton** properties:

- **BackgroundColor**: default
- **Enables**: ✓
- **FontBold**: ☐
- **FontItalic**: ☐
- **Fontsize**: 14
- **FontTypeFace:** default
- **Height:** 50 pixels
- **Width:** 50 pixels
- **Image:** none
- **Shape:** default
- **ShowFeedBack:** ✓
- **Text:**
- **TextAlignment** center : 1
- **TextColor:** default
- **Visible:** ✓

Edit **verticalArrangement1** and **2** with the same properties:

- **AlignHorizontal:** Left : 1
- **AlignVertical:** Top : 1
- **BackgroundColor:** none
- **Height:** fill parent...
- **Width:** 10 pixels

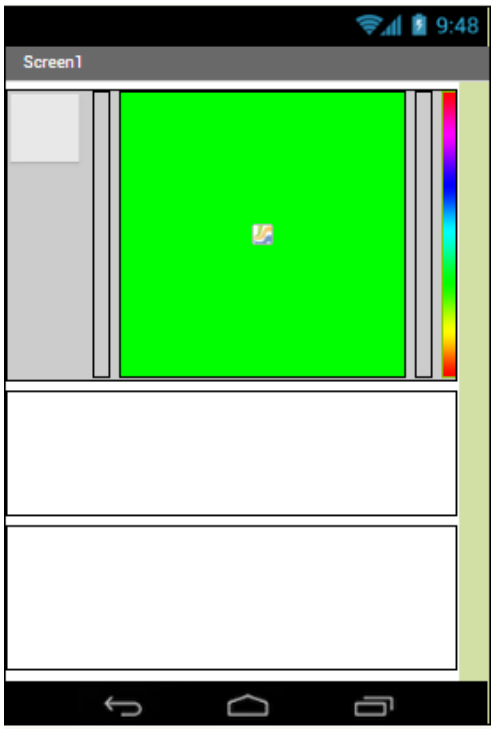
- **Image:** none...
- **Visible:** ✓

Edit the **colorPickerCanvas:**

- **BackgroundColor:** Green
- **BackgroundImage:** none
- **FontSize**: 14
- **Height**: 200 pixels
- **Width**: 200 pixels
- **LineWidth**: 20
- **PrintColor**: black
- **TextAlignment**: center : 1
- **Visible**: ✓

Edit **Canvas2:**

- **BackgroundColor:** none
- **BackgroundImage:** color_picker-03.png
- **FontSize**: 14
- **Height**: 200 pixels
- **Width**: 35 pixels
- **LineWidth**: 20
- **PrintColor**: black
- **TextAlignment**: center : 1
- **Visible**: ✓



Your app looks like the figure at the right.

Go to **Drawing and Animation** and drag two sprites into the **colorPickerCanvas.** Change **imageSprite1** properties to:

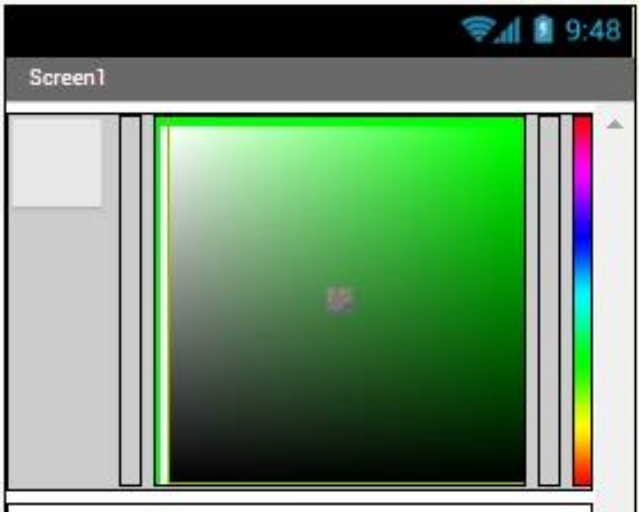
- **Enabled:** ✓
- **Heading:** 0
- **Height:** Automatic...
- **Width:** Automatic...
- **Interval:** 100
- **Picture:** color_picker-02.png
- **Rotates:** ✓
- **Speed:** 0
- **Visible:** ✓
- **X:** 1
- **Y:** 4

- **Z:** 1.0

Finally, the **imageSprite2** properties:

- **Enabled:** ✓
- **Heading:** 0
- **Height:** Automatic...
- **Width:** Automatic...
- **Interval:** 100
- **Picture:** color_picker-01.png
- **Rotates:** ✓
- **Speed:** 0
- **Visible:** ✓
- **X:** 6
- **Y:** -2
- **Z:** 1.0

Now, you have a gradient color background as shown below.



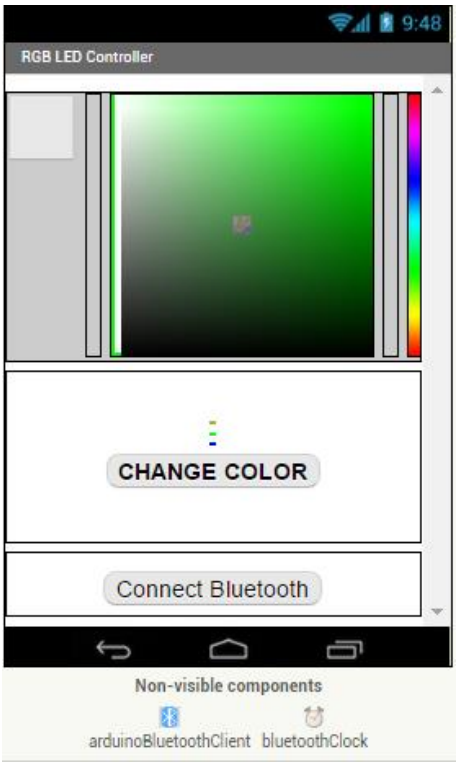
Buttons

Next, drag three labels into to **buttonVerticalArrangement** and rename them to **redLabel**, **greenLabel** and **blueLabel**.

These labels will show the RGB parameters for the chosen color. Add a button with the text "**CHANGE COLOR**". Rename the button to **changeColorButton.** Edit its properties the way you like.

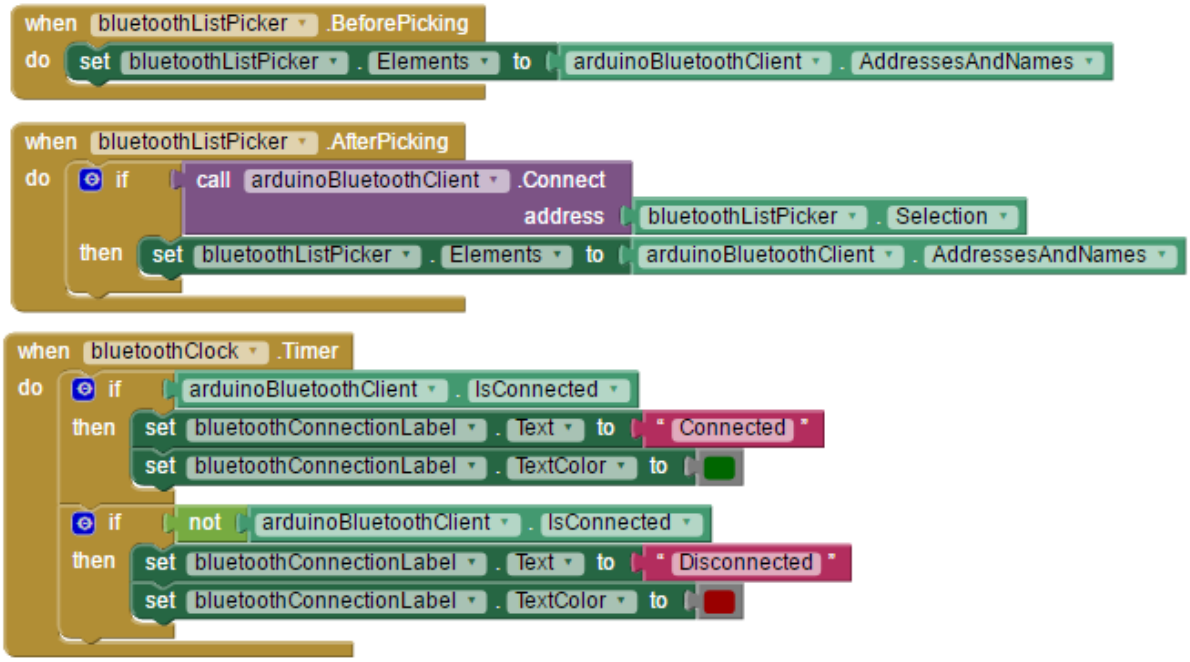
In the **connectVerticalArrangement** add the **bluetoothListpicker** and the information **Labels**.

Finally, add a **Bluetooth Client** and a **Clock**. Your design is ready.



Blocks Editor

As usual, you start by dragging the blocks for the bluetooth connection.



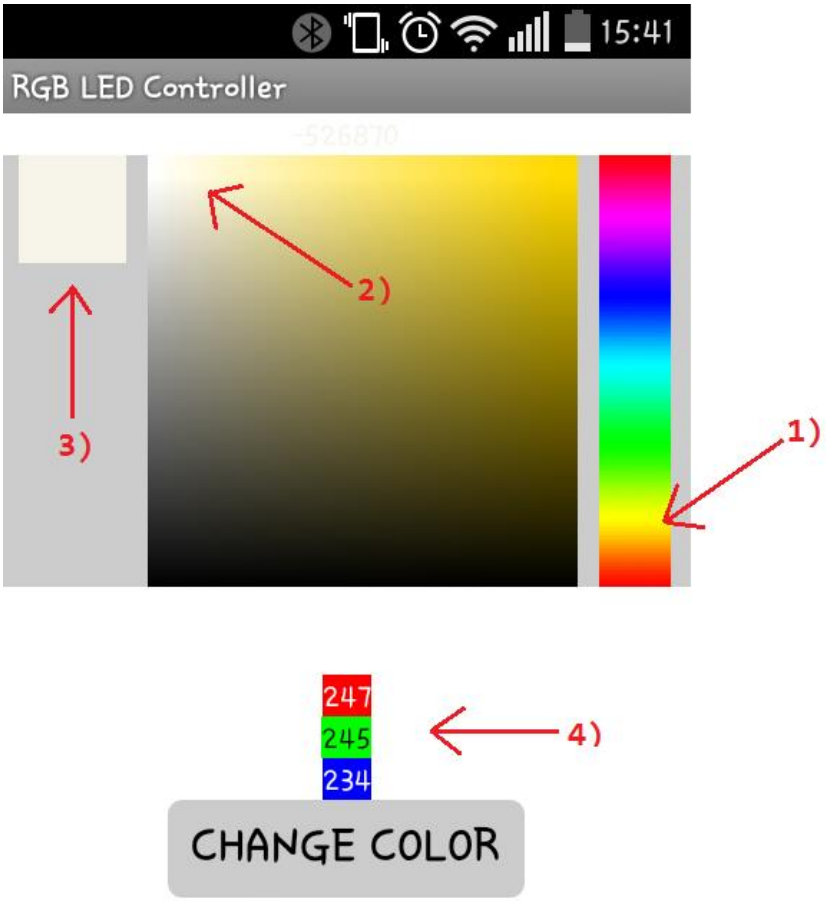
Then, initialize a global variable called **colorMessage,** as follows:

This variable saves the current color.

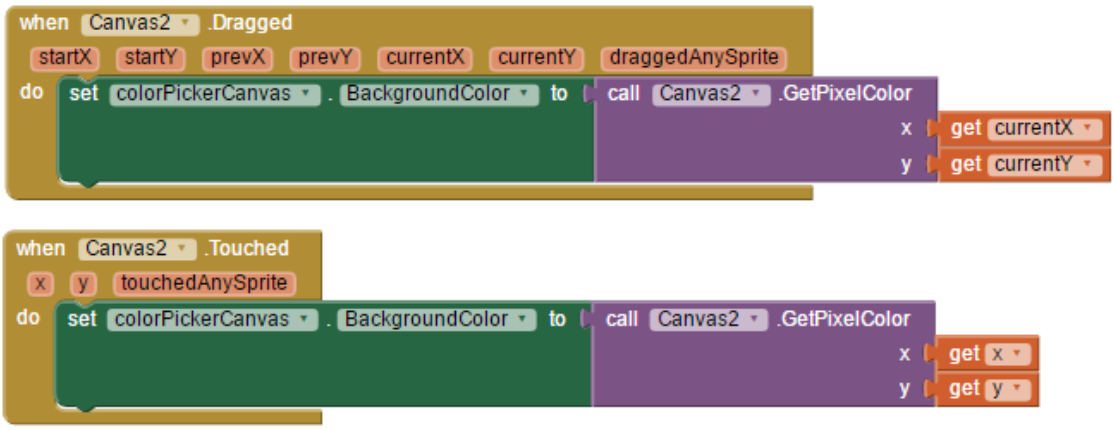
When the app starts, the **imageSprites** are set to the **colorPickerCanvas** Width and Height so that they are completely overlapped.

Selecting a color

The following figure shows the steps for setting a color and what happens once the color is selected.

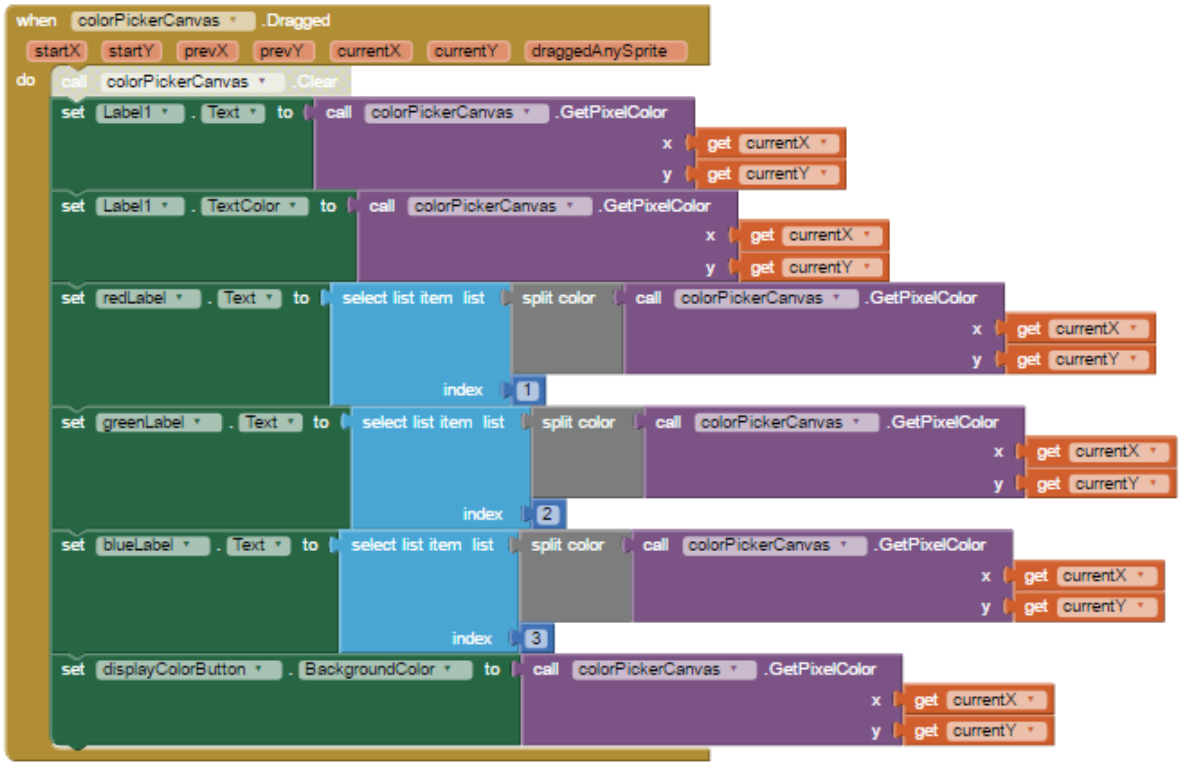


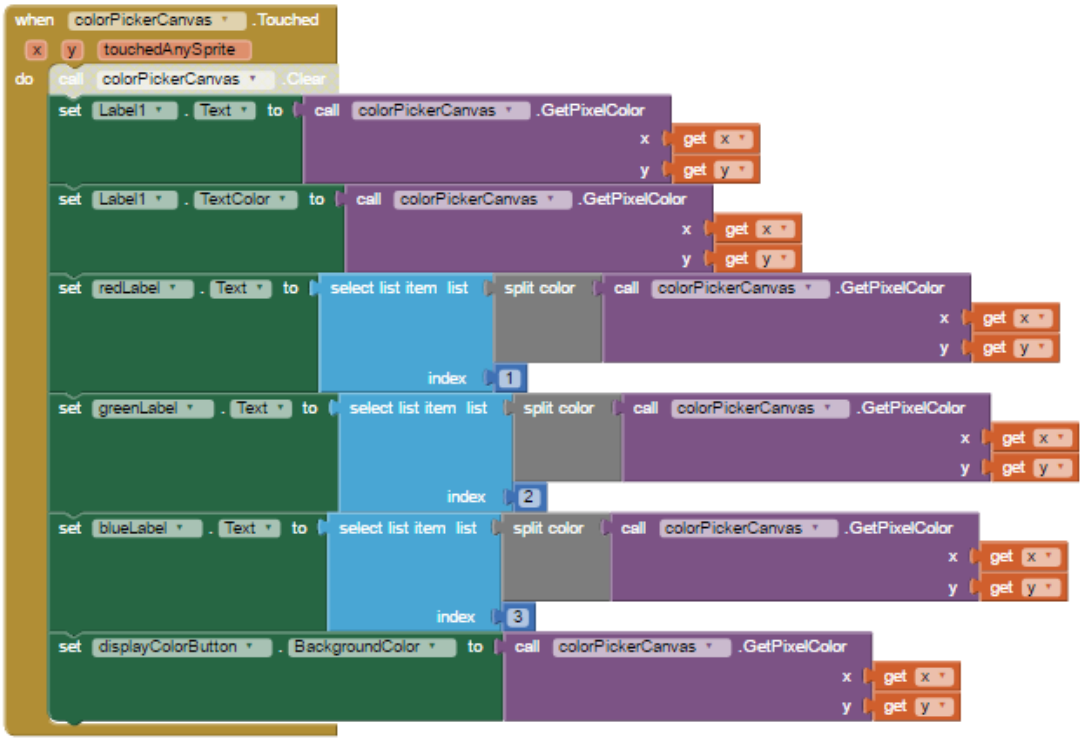
First, you select a color at **1)**, yellow. When you select the color, the canvas at the middle **2)** changes its background color to the color you've selected at **1)**. The following two blocks do precisely that. You can choose a color either by dragging or touching the **1)**.



After choosing the color at **1)**, you select the color intensity at **2)**.

When you select the color **2)**, the square **3)** changes its color to the selected color and the labels **4)** show the RGB parameters for that color. This is done with these following blocks.

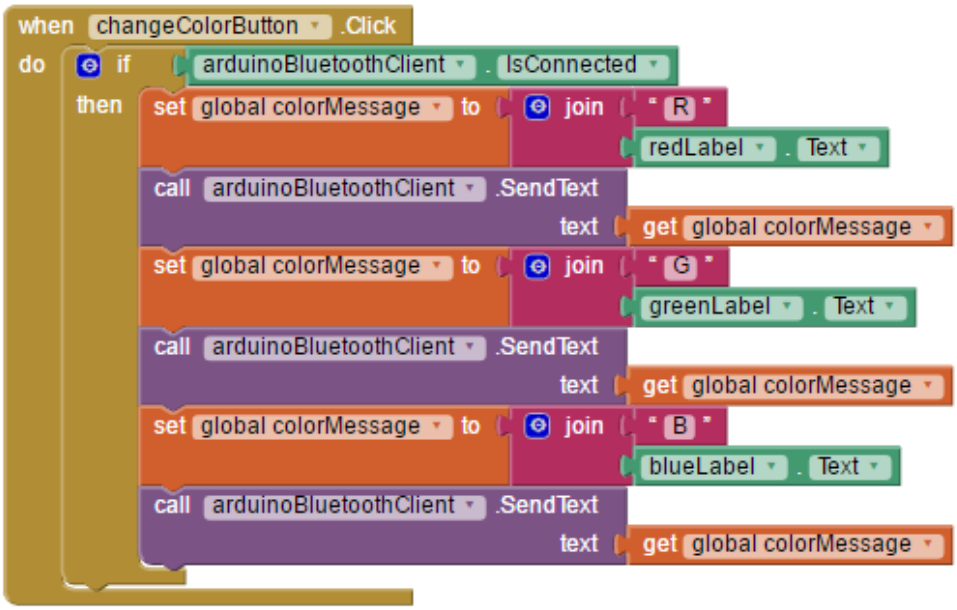




There are two useful blocks for these tasks:

- **call colorPickerCanvas.GetPixelColor x, y:** that gets the color from the current selected pixel;
- **split color** that splits a color into its RGB parameters.

Finally, the following block sends the color to the bluetooth module when you click the **CHANGE COLOR** button:



The message the Arduino receives has up to 12 characters and is something like this: “RXXXGXXXBXXX”. In which the X’s represent the numbers for the RGB color.

To create the Android app, go to the **Build** tab and generate a new *.apk* file. Alternatively, you can use the *.apk* file that comes with the eBook's resources folder. Move the *.apk* file to your Android phone and follow the on-screen instructions to install it.

Writing the Sketch

Upload the following sketch to your Arduino board. Make sure you have the right board and COM port selected.

Warning: When you're uploading code to your Arduino you should disconnect the TX and RX cables from the bluetooth module. These pins are needed for serial communication between the Arduino and your computer.

Tip: People often make the wrong connections between the Arduino and bluetooth module. **TX** from the bluetooth module connects to **RX** pin of the Arduino. **RX** from the bluetooth module connects to **TX** pin of the Arduino.

```
/*
 * Rui Santos
 * Complete Project Details http://randomnerdtutorials.com
 */

#define max_char 12
char message[max_char];     // stores you message
char r_char;                // reads each character
byte index = 0;             // defines the position into your array
int i;

int redPin = 11;      // Red RGB pin -> D11
int greenPin = 10;    // Green RGB pin -> D10
int bluePin = 9;      // Blue RGB pin -> D9

int redValue = 255;      // Red RGB pin -> D11
int greenValue = 255;    // Green RGB pin -> D10
int blueValue = 255;      // Blue RGB pin -> D9

String redTempValue;      // Red RGB pin -> D11
String greenTempValue;    // Green RGB pin -> D10
String blueTempValue;      // Blue RGB pin -> D9

int flag = 0;
char currentColor;

void setup() {
  pinMode(redPin,OUTPUT);
  pinMode(bluePin,OUTPUT);
  pinMode(greenPin, OUTPUT);
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
}

void loop() {
```

```
  //while is reading the message
  while(Serial.available() > 0){
    flag = 0;
    //the message can have up to 12 characters
    if(index < (max_char-1)){
      r_char = Serial.read();      // Reads a character
      message[index] = r_char;  // Stores the character in message array
      if(r_char=='R'){
         currentColor = 'R';
         redTempValue = "";
      }
      else if(r_char=='G'){
         currentColor = 'G';
         greenTempValue = "";
      }
      else if(r_char=='B'){
         currentColor = 'B';
         blueTempValue = "";
      }
      if(currentColor == 'R' && r_char!='R'){
         redTempValue += r_char;
      }
      else if(currentColor == 'G' && r_char!='G'){
         greenTempValue += r_char;
      }
      else if(currentColor == 'B' && r_char!='B'){
         blueTempValue += r_char;
      }
      index++;                      // Increment position
      message[index] = '\0';        // Delete the last position
   }

 }
 if(flag == 0){
   // RGB LED Common ANODE
   analogWrite(redPin, 255-redTempValue.toInt());
   analogWrite(greenPin, 255-greenTempValue.toInt());
   analogWrite(bluePin, 255-blueTempValue.toInt());

   // RGB LED Common CATHODE
   // analogWrite(redPin, redTempValue.toInt());
   // analogWrite(greenPin, greenTempValue.toInt());
   // analogWrite(bluePin, blueTempValue.toInt());

   /*Serial.print('R');
   Serial.println(redTempValue);
   Serial.print('G');
   Serial.println(greenTempValue);
   Serial.print('B');
   Serial.println(blueTempValue);
   Serial.print("MESSAGE ");*/
   Serial.println(message);
   flag=1;
       for(i=0; i<12; i++){
      message[i] = '\0';
    }
    //resets the index
    index=0;
 }
}
```

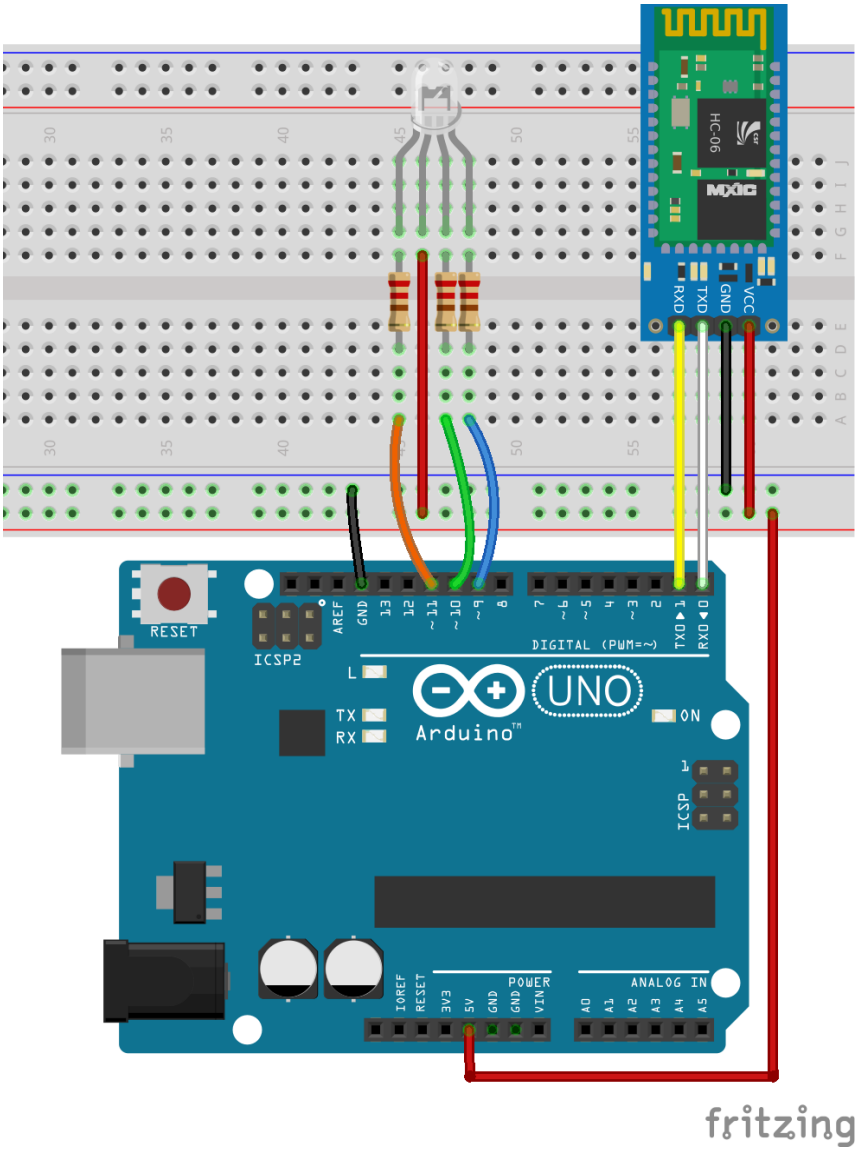
This code receives an array message that has up to 12 characters. It has the following structure: “RXXXGXXXBXXX”, in which the X’s represent the numbers for the RGB color. Then, the code splits the message to get the red, green and blue values.

Finally, it sets each of the RGB LED leads to the corresponding value, generating the chosen color.

Wiring the Circuit

The following schematics shows the circuit for the RGB LED circuit. We are using a common anode RGB LED.

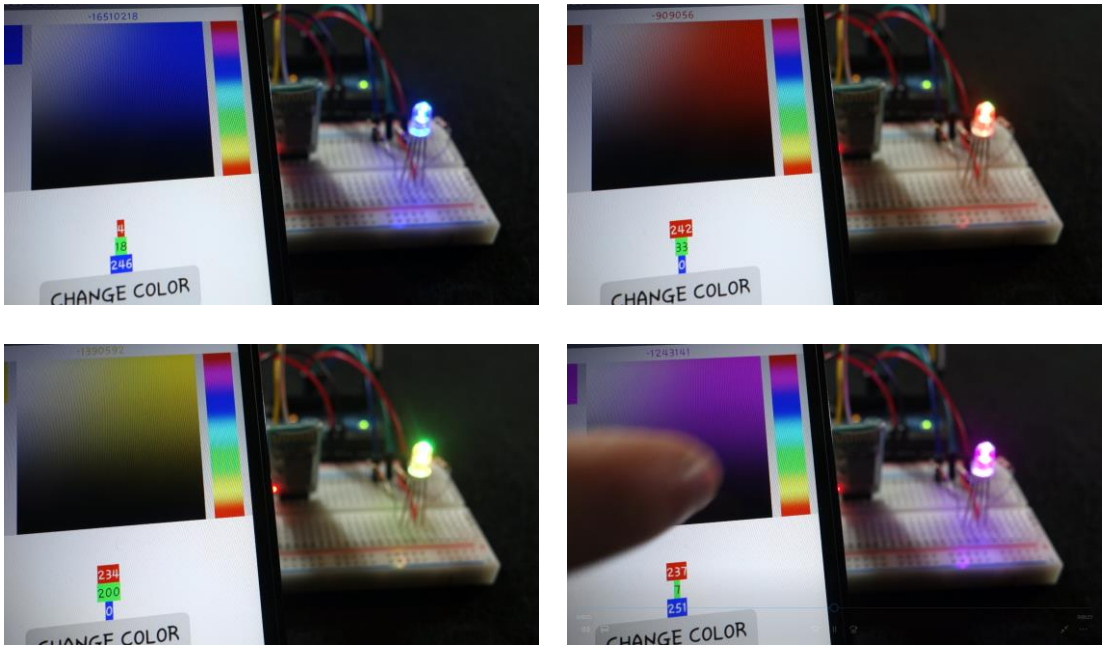
If you have a common cathode RGB LED you can also use it. You connect the longest lead to GND and comment/uncomment the RGB LED common cathode code lines.



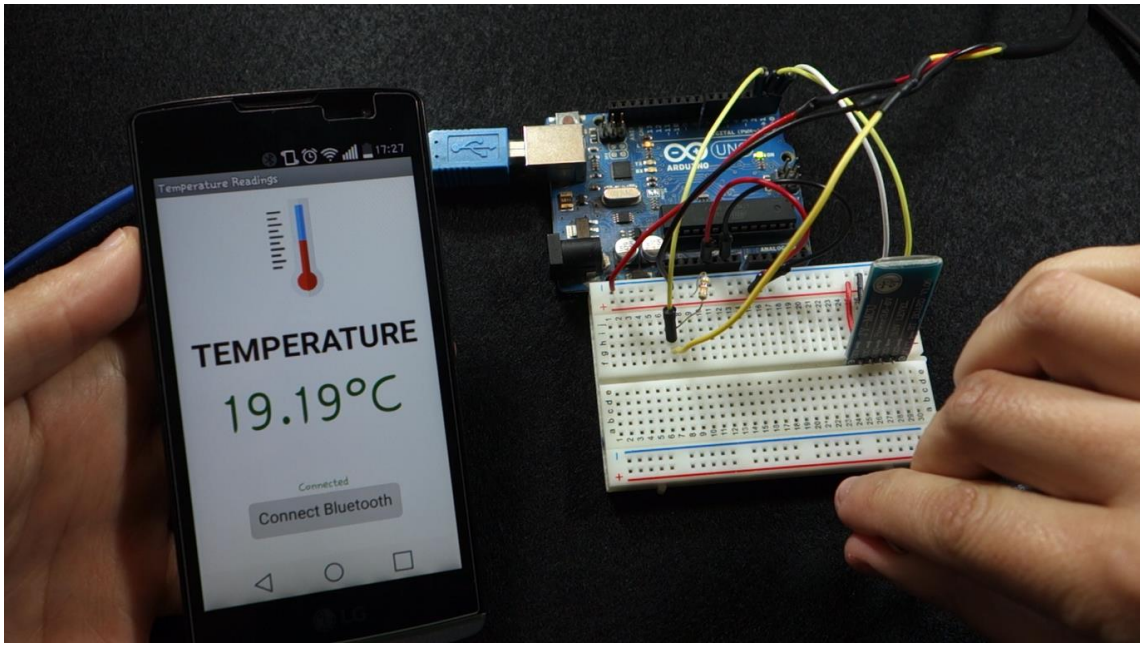
Demonstration

To choose a color for the RGB LED:

1. Connect the bluetooth by tapping the **Connect Bluetooth** button
2. Choose a color at the rainbow stripe at the right
3. Then, choose the color intensity at the square in the middle
4. Click the **Change Color** button to change the RGB LED color



Project 5: Temperature Readings



Project 5: Temperature Readings

In this project you're going to read the temperature with the DS18B20 temperature sensor and display the values on an Android app.

Parts Required

Here's a complete list of parts you need to do this project.

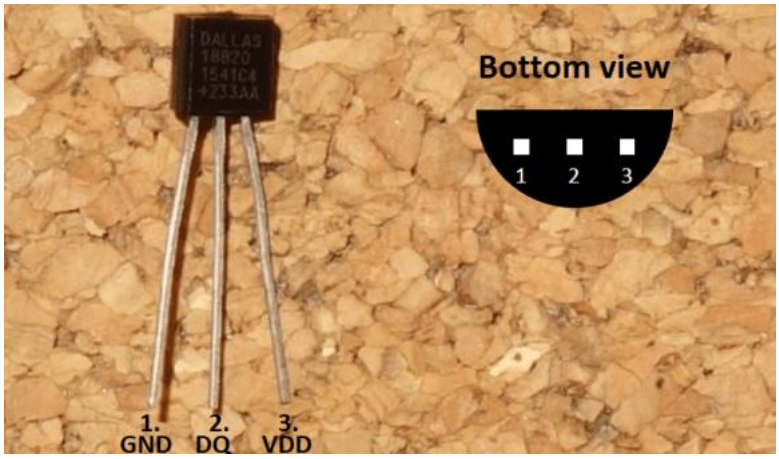
Figure	Name	eBay
	Arduino UNO	http://ebay.to/1SQda0R
	HC-04 or HC-05 or HC-06 Bluetooth module	http://ebay.to/2dt1b6M
	DS18B20 Temperature Sensor	http://ebay.to/2aRm6DC
	Breadboard	http://ebay.to/21bEojM
	4.7kOhm Resistor	http://ebay.to/1KsMYFP
	Jumper Wires	http://ebay.to/1PXealz

Introducing DS18B20 Temperature Sensor

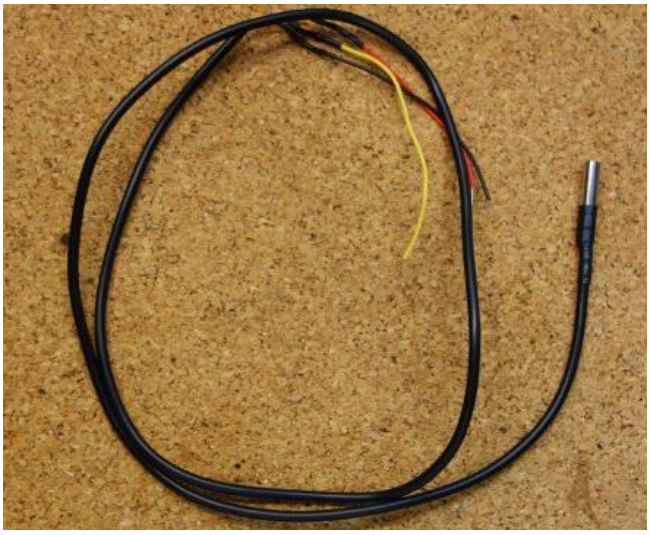
The DS18B20 temperature sensor is a 1-wire digital temperature sensor. This means that you can read the temperature with a simple circuit setup.

It communicates on common bus, which means that you can connect several devices and read their values using just one digital pin of the Arduino.

The sensor has three pins as shown in the following figure:



The DS18B20 is also available in waterproof version – that's the one we're going to use. Both versions work with the same circuit wiring and code.



Features

Here's the main features of the DS18B20 temperature sensor:

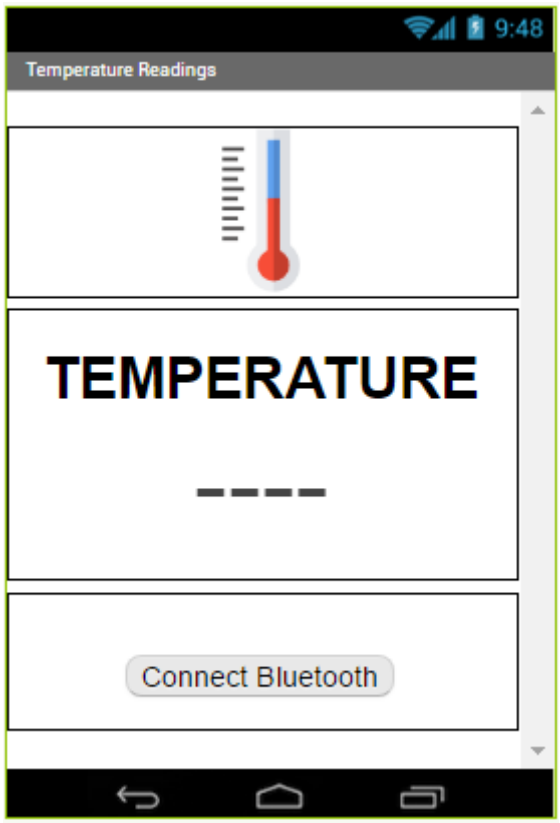
- Communicates over 1-wire bus communication;
- Operating range temperature: -55ºC to 125ºC;
- Accuracy +/-0.5 ºC (between the range -10ºC to 85ºC).

Building the App

Go to http://ai2.appinventor.mit.edu for opening the MIT App Inventor 2 software.
Go to **Projects ▸ Start New Project** and call it *Temperature_Readings*.

Alternatively, if you prefer, you can upload the *.aia* file and edit the app the way you like. You just have to go to **Projects ▸ Import project (.aia) from my computer** and upload the *Temperature_Readings.aia* file that comes with the eBook's resources.

Here's the app you're going to build in this project.



The app has the following features:

- A thermometer image.
- A **label** saying **TEMPERATURE.**
- A **label** that displays the temperature - at the moment is showing ----.
- The color of the temperature label changes accordingly to the temperature measured by the sensor.
- A **Connect Bluetooth** button that opens a **Listpicker**.

Design

In the **Pallete** go to **Layout** and drag an **HorizontalArrangement** and two **VerticalArrangements,** in this order. Rename them to **temperatureImageArrangement**, **labelVerticalArrnagement**, and **connectVerticalArrangement**, respectively.

The **temperatureImageArrangement** displays the thermometer image. Make sure this horizontal arrangement has the following properties:

- **AlignHorizontal:** Center : 3
- **AlignVertical:** Center : 2
- **BackgroundColor:** none
- **Height:** 25 percent...
- **Width:** fill parent...
- **Image:** none...
- **Visible:** ✓

The **labelHorizontalArrangement** is where you're going to place the label saying **TEMPERATURE** and the temperature value. This horizontal arrangement should have these properties:

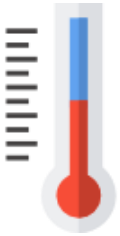
- **AlignHorizontal:** Center : 3
- **AlignVertical:** Center : 2
- **BackgroundColor:** none
- **Height:** 40 percent...
- **Width:** fill parent...
- **Image:** none...
- **Visible:** ✓

The **connectVerticalArrangement** has the **Connect Bluetooth listpicker** and the state **Label**. The vertical arrangement has these properties:

- **AlignHorizontal:** Center : 3
- **AlignVertical:** Center : 2
- **BackgroundColor:** none
- **Height:** 20 percent...
- **Width:** fill parent...
- **Image:** none...
- **Visible:** ✓

The figure at the right shows how your layout looks.

Go to media and upload the **thermometer.png** file.



Go to **Palette** ▸ **User Interface** and drag an **Image** into the **TemperatureImageArrangement**. Edit the **Image** with the following properties:

- **Height**: Fill parent
- **Width**: Automatic
- **Picture**: thermometer.png
- **RotationAngle**: 0.0
- **ScalePictureToFit**: ☐
- **Visible:** ✓

Drag two **labels** into the **labelVerticalArrangement**. Rename them to **temperatureLabel** and **temperatureValueLabel**.

The **temperatureLabel** should have these properties:

- **Background:** none
- **FontBold:** ✓
- **FontItalic:** ☐
- **FontSize:** 40
- **FontTypeface:** sans serif
- **HTMLFormat:** ☐
- **HasMargins:** ✓
- **Height:** Automatic
- **Width:** Fill parent
- **Text:** TEMPERATURE
- **TextAlignment:** center : 1
- **TextColor:** black
- **Visible:**✓

The **temperatureValueLabel** has the following properties:

- **Background:** none
- **FontBold:** ☐
- **FontItalic:** ☐
- **FontSize:** 70
- **FontTypeface:** default

- **HTMLFormat:** ☐
- **HasMargins:** ✓
- **Height:** Automatic
- **Width:** Fill parent
- **Text:** ----
- **TextAlignment:** center : 1
- **TextColor:** **Dark Gray**
- **Visible:**✓

Into the **connectVerticalArrangement** drag a **label** and a **listpicker.** Rename the label to **bluetoothConnectionLabel** and **bluetoothListpicker**.

The **bluetoothConnectionLabel** should have the following properties:

- **BackgroundColor**: none
- **FontBold:** ☐
- **FontItalic:** ☐
- **FontSize:** 14
- **FontType:** default
- **HTMLFormat:** ☐
- **HasMargins:** ✓
- **Height:** Automatic
- **Width:** Automatic
- **Text:** Disconnected
- **TextAlignment:** center : 1
- **TextColor:** none
- **Visible:** ✓

The **bluetoothListPicker** should have these next properties**:**

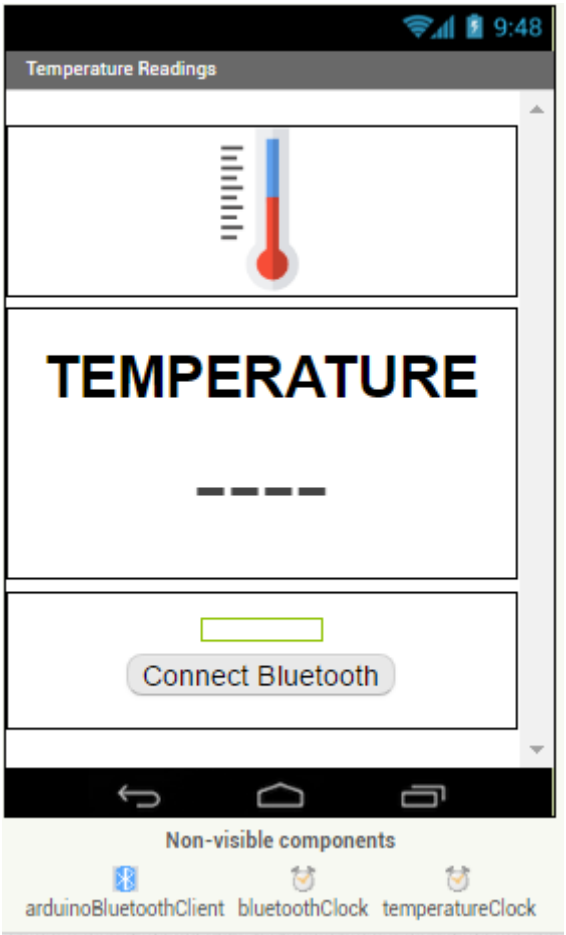
- **BackgroundColor:** Default
- **ElementsFromString:**
- **Enabled:** ✓
- **FontBold:** ☐
- **FontItalic:** ☐
- **FontSize:** 20
- **FontTypeface:** sans serif
- **Height:** Automatic
- **Width:** Automatic
- **Image:** none
- **ItemBackgroundColor:** Black
- **ItemTextColor:** White

- **Selection:**
- **Shape:** rounded
- **ShowFeedBack:** ✓
- **ShowFitterBar:** ☐
- **Text:** Connect Bluetooth
- **TextAlignment:** center : 1
- **TextColor:** Default
- **Title:**
- **Visible:** ✓

Finally, drag a **BluetoothClient** and two **Clocks**. Rename them to **arduinoBluetoothClient**, **bluetoothClock** and **temperatureClock.**

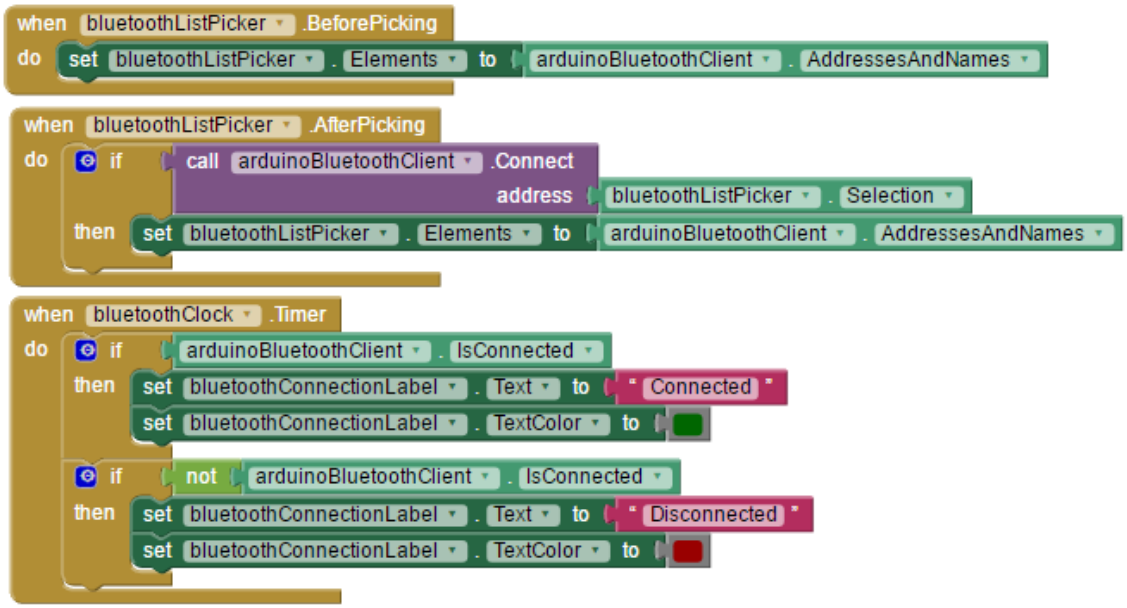
The **temperatureClock** controls how often the temperature sensor takes readings.

The following figure shows how your app should look like.



Blocks Editor

As usual, start by dragging the blocks to establish the bluetooth connection.

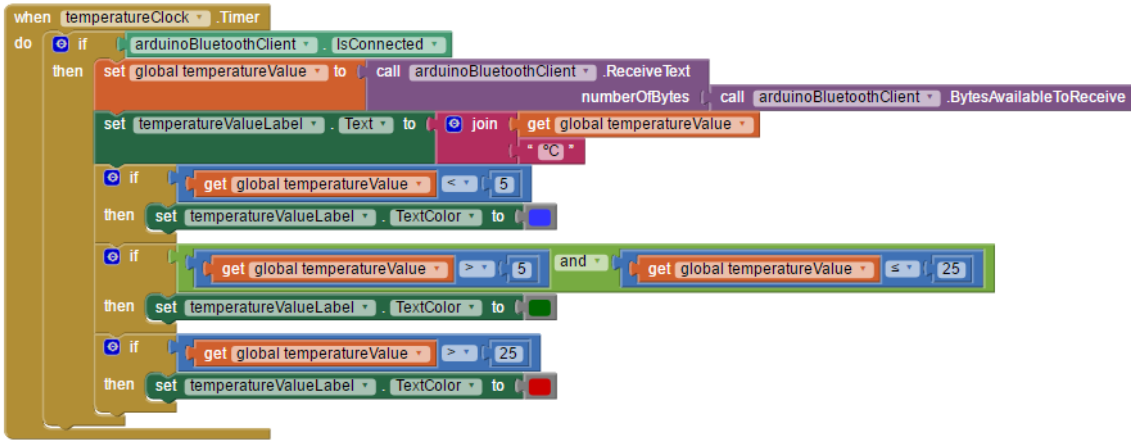


Initialize a global value to store the temperature values.

Next, if your bluetooth smartphone is connected, the **temperatureValue** is set to the value received from the bluetooth module.

After having the temperature value stored in the **temperatureValue** variable, you need to set the **temperatureValueLabel** text to the value received by the sensor. We also change the label color accordingly to temperature reading:

- **blue** if the temperature is lower than 5 °C
- **green** if the temperature is between 5 °C and 25 °C
- **red** if the temperature is higher than 25 °C.



To create the Android app, go to the **Build** tab and generate a new *.apk* file. Alternatively, you can use the *.apk* file that comes with the eBook's resources folder. Move the *.apk* file to your Android phone and follow the on-screen instructions to install it.

Writing the Sketch

For this project you need to install the **OneWire** library and the **DallasTemperature** library in your Arduino IDE.

Installing the OneWire Library

1. Click here to download the OneWire library. You should have a *.zip* folder in your Downloads
2. Unzip the *.zip* folder and you should get **OneWire-master** folder
3. Rename your folder from ~~**OneWire-master**~~ to **OneWire**
4. Move the **OneWire** folder to your Arduino IDE installation libraries folder
5. Finally, re-open your Arduino IDE

Installing the DallasTemperature Library

1. Click here to download the DallasTemperature library. You should have a *.zip* folder in your Downloads
2. Unzip the *.zip* folder and you should get **Arduino-Temperature-Control-Library-master** folder
3. Rename your folder from ~~**Arduino-Temperature-Control-Library-master**~~ to **DallasTemperature**
4. Move the **DallasTemperature** folder to your Arduino IDE installation libraries folder
5. Finally, re-open your Arduino IDE

Uploading the Sketch

The following sketch reads the temperature from the DS18B20 temperature sensor and sends the readings via serial to the bluetooth module. The app gathers the temperature data from the bluetooth module.

Upload the following code to your Arduino board – make sure you have the right board and COM port selected.

Warning: When you're uploading code to your Arduino you should disconnect the TX and RX cables from the bluetooth module. These pins are needed for serial communication between the Arduino and your computer.

Tip: People often make the wrong connections between the Arduino and bluetooth module. **TX** from the bluetooth module connects to **RX** pin of the Arduino. **RX** from the bluetooth module connects to **TX** pin of the Arduino.

```
/*
 * Rui Santos
 * Complete Project Details http://randomnerdtutorials.com
 */

#include <OneWire.h>
#include <DallasTemperature.h>

// Data wire is connected to the Arduino digital pin 2
#define ONE_WIRE_BUS 2

// Setup a oneWire instance to communicate with any OneWire devices
OneWire oneWire(ONE_WIRE_BUS);

// Pass our oneWire reference to Dallas Temperature sensor
DallasTemperature sensors(&oneWire);

unsigned long previousMillis = 0;
const long interval = 5000;


void setup() {
  // Start up the library
  sensors.begin();

  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
}

void loop() {
  unsigned long currentMillis = millis();
  if (currentMillis - previousMillis >= interval) {
    // save the last time you blinked the LED
    previousMillis = currentMillis;
    // Call sensors.requestTemperatures() to issue a
    // global temperature and Requests to all devices on the bus
    sensors.requestTemperatures();

    // Celsius temperature
    // Why "byIndex"? You can have more than one IC
    // on the same bus. 0 refers to the first IC on the wire
    Serial.print(sensors.getTempCByIndex(0));

    //Fahrenheit temperature
    //Serial.println(sensors.getTempFByIndex(0));
  }
}
```

This code reads the temperature from the DS18B20 temperature sensor every five seconds and sends the readings to the Android app.

To send information from the Arduino to the app, you use the `Serial.print(sensors.getTempCByIndex(0))` command - between parentheses you pass the variable you want to send.

The code is well commented on how to request the temperature from the sensor.

You can either send the temperature in Celsius or Fahrenheit. To get temperature in Fahrenheit uncomment line:

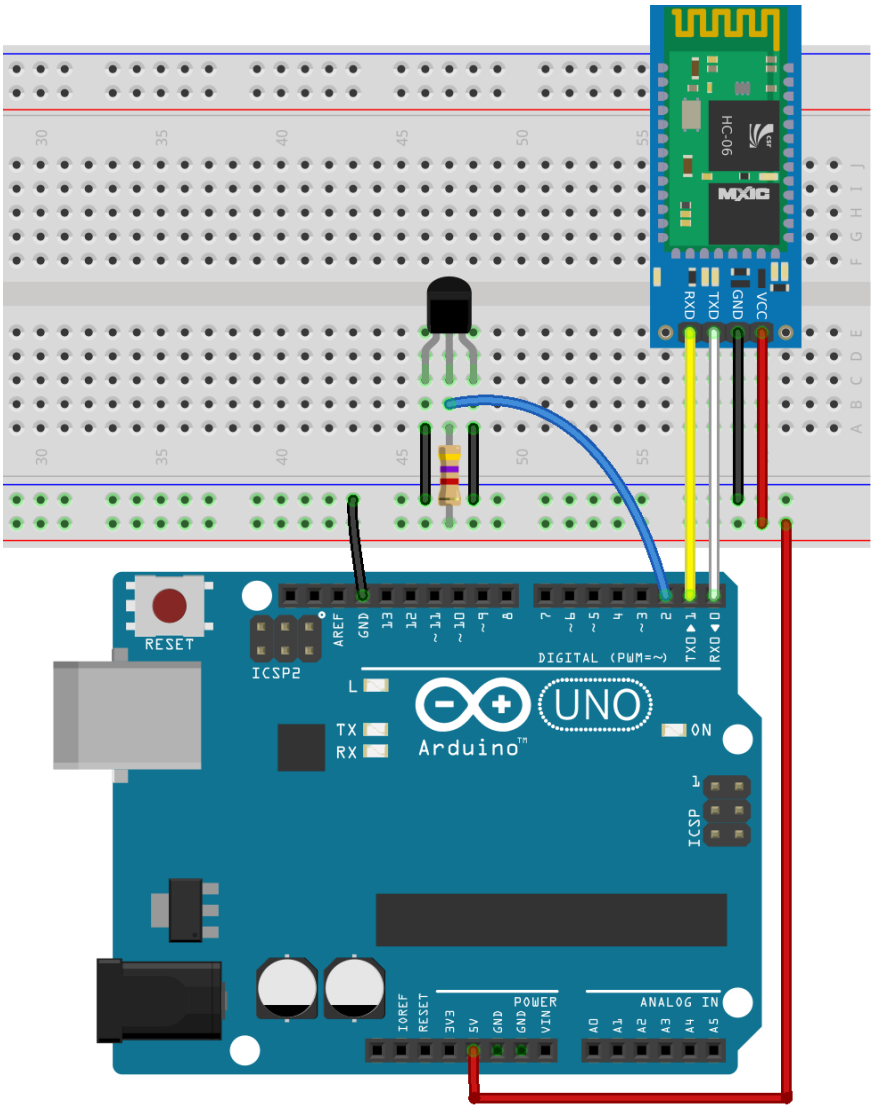
```
Serial.println(sensors.getTempFByIndex(0)).
```

And comment this one:

```
//Serial.print(sensors.getTempCByIndex(0));
```

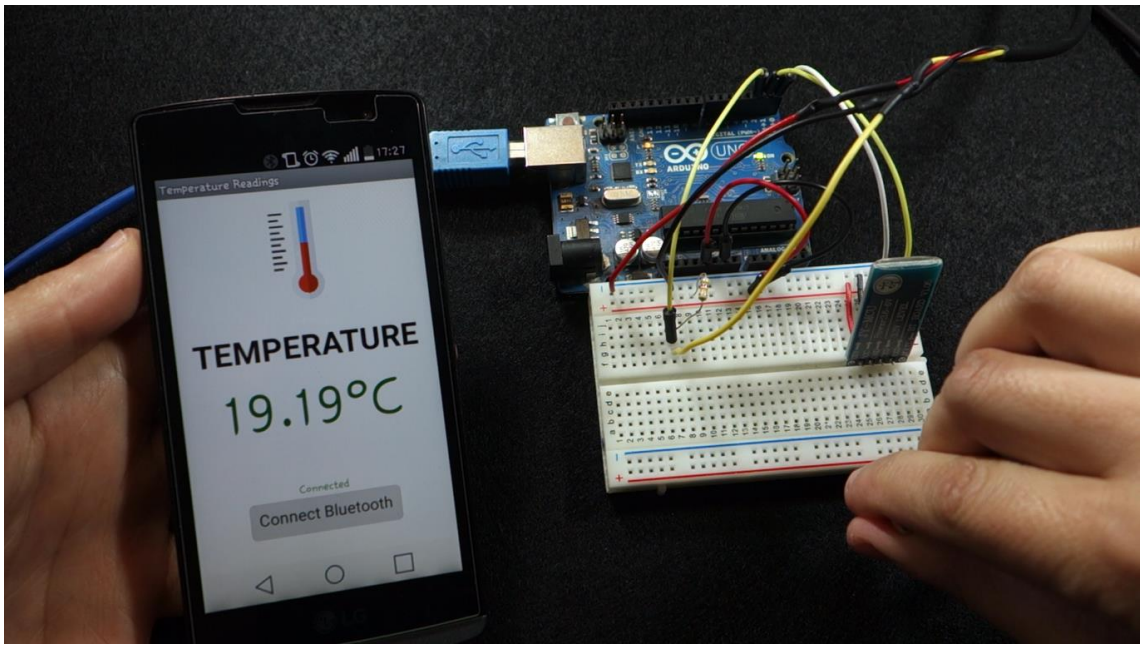
Wiring the Circuit

Wire the DS18B20 temperature sensor and the bluetooth module to the Arduino board as shown in the schematic below.

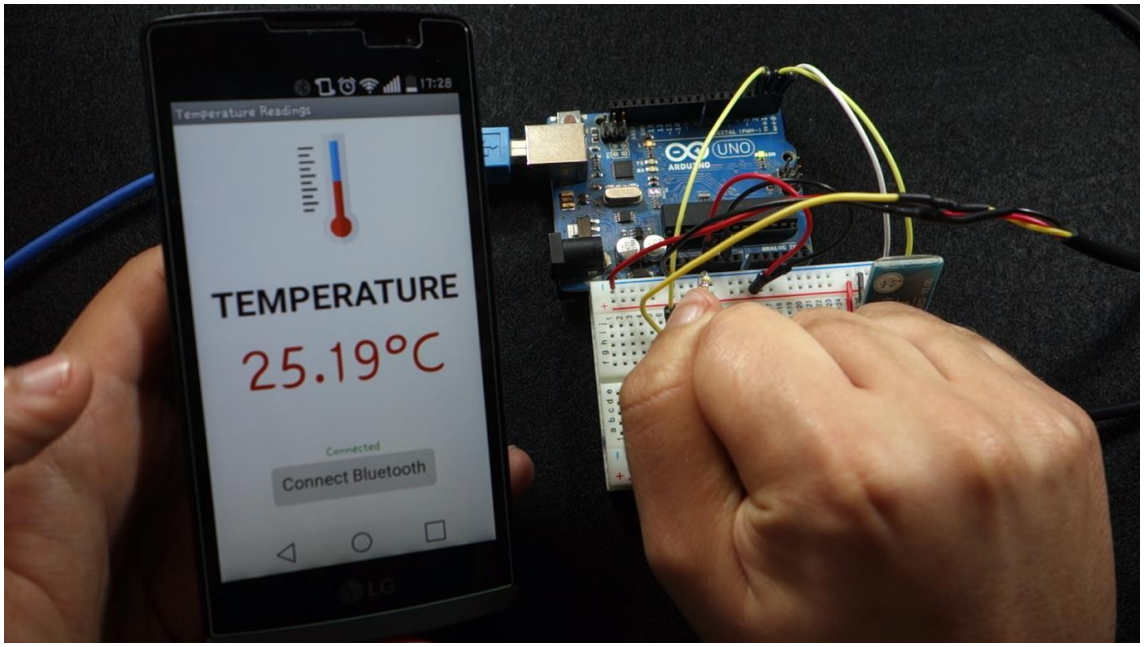


Demonstration

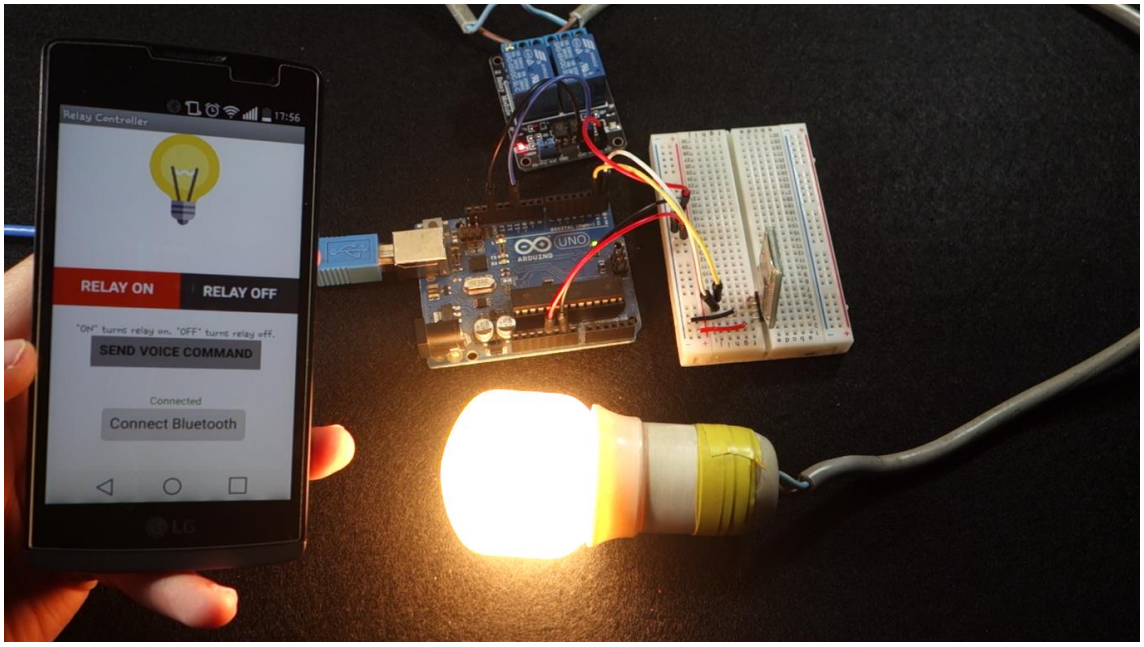
Your app updates the temperature every five seconds.



And the text color changes accordingly to the temperature.



Project 6: Relay Controller



Project 6: Relay Controller

In this project you're going to build an Android app that controls a lamp connected to a relay using three different methods: with **ON** and **OFF** buttons, with **voice commands** and by **shaking your smartphone**.

Before proceeding with the project make sure you read the safety warning below.

SAFETY WARNING!

When you are making projects that are connected to mains voltage, you really need to know what you are doing, otherwise **you may shock yourself.** This is a serious topic and I want you to be safe. If you are not 100% sure what you are doing, do yourself a favor and don't touch anything. Ask someone who knows!

Parts Required

Here's a complete list of parts you need to do this project.

Figure	Name	eBay
	Arduino UNO	http://ebay.to/1SQda0R
	HC-04 or HC-05 or HC-06 Bluetooth module	http://ebay.to/2dt1b6M
	Relay Module	http://ebay.to/2dRv8lh
	Lamp Cord Set	http://ebay.to/2feHELc
	Breadboard	http://ebay.to/21bEojM
	Jumper Wires	http://ebay.to/1PXeaJz

Introducing the Accelerometer

Recent smartphones have an accelerometer sensor. An accelerometer measures the linear acceleration of the device allowing you to measure changes in velocity and changes in position. It measures the acceleration across an XYZ axis.

The accelerometer has microscopic crystals that go under stress when vibration occurs. From that stress, results a voltage that allows you to measure the acceleration.

For this project, the accelerometer detects if your smartphone was shaken or not.

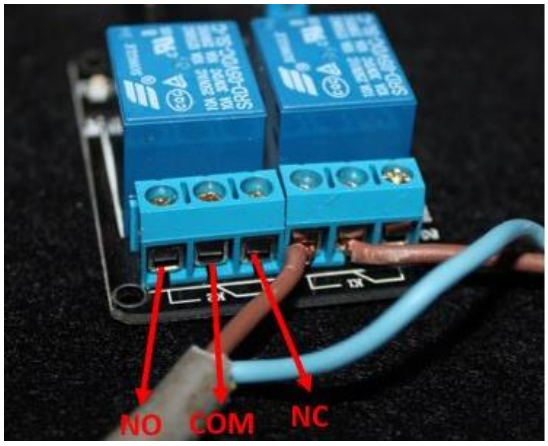
Introducing the Relay Module

A relay is an electrically operated switch. It means that it can be turned on or off, letting the current going through or not. The relay module is the one in the figure below.

This relay module board comes with two relays (those blue cubes).

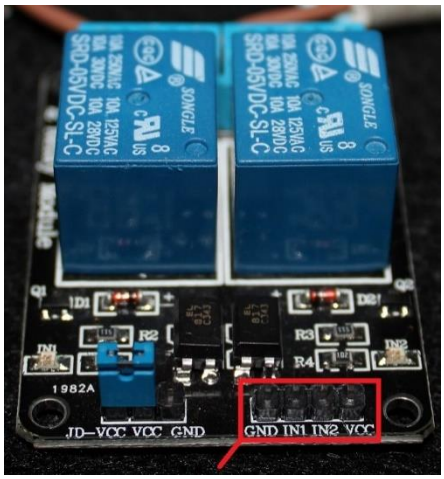
Relating mains voltage, relays have 3 possible connections:

- **COM**: common pin
- **NO**: normally open – there is no contact between the common pin and the normally open pin. So, when you trigger the relay, it connects to the COM pin and power is provided to the load (a lamp, in our case).
- **NC**: normally closed – there is contact between the common pin and the normally closed pin. There is always contact between the COM and NC pins, even when the relay is turned off. When you trigger the relay, the circuit is opened and there is no power provided to the load.



Relating this project, it is better to use a normally open circuit, because we want to light up the lamp occasionally.

The connections between the relay and the Arduino are simple:



- **GND**: goes to ground
- **IN1**: controls the first relay. Should be connected to an Arduino digital pin
- **IN2**: controls the second relay. Should be connected to an Arduino digital pin
- **VCC**: goes to 5V

Building the App

Go to http://ai2.appinventor.mit.edu for opening the MIT App Inventor 2 software.
Go to **Projects** ▸ **Start New Project** and call it *Relay_Controller.*

Alternatively, if you prefer, you can upload the *.aia* file and edit the app the way you like. You just have to go to **Projects** ▸ **Import project (.aia) from my computer** and upload the *Relay_Controller.aia* file that comes with the eBook's resources.

Here's how the *Relay_Controller* app looks.

The app allows you to control the relay with:

- ON and OFF buttons
- Voice commands
- Phone shaking

Design

Drag a **HorizontalArrangement** and place two **images** inside. Upload the images: **light_bulb_off.png** and **light_bulb_on.png** from the eBook's resources. Set the light_bulb_on.png image to **invisible** by removing the ✓ at the **Visible** option in the properties section.

Then, drag a **VerticalArrangement** and let it stay empty. This empty arrangement is useful to keep the app in place when you call the voice recognizer. The **VerticalArrangement** should have **Automatic Height** and a **Fill parent Width** – make sure you select that in the properties section.

Next, drag a **HorizontalArrangement** to add the **RELAY ON** and **RELAY OFF buttons** – you should be familiar with adding and editing buttons.

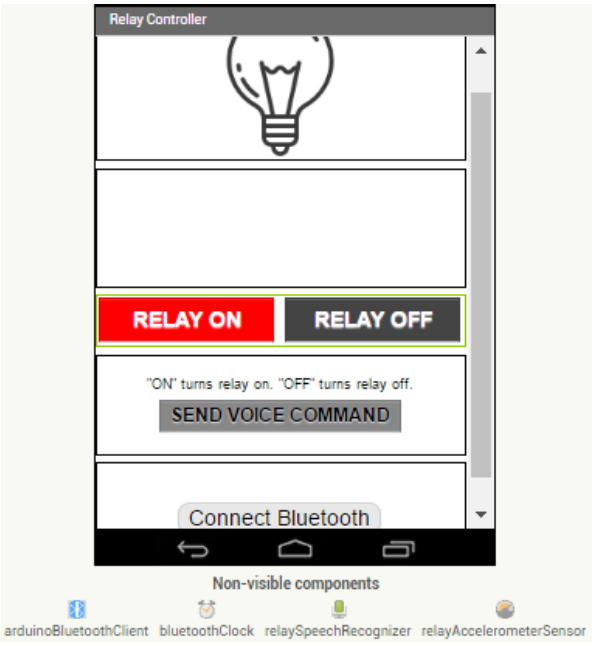
Drag a **VerticalArrangement** for the voice command button. This section should contain a label and a button**.**

Finally, add the **bluetooth listpicker** and **Label** as you usually do in the other apps.

You need these non-visible components:

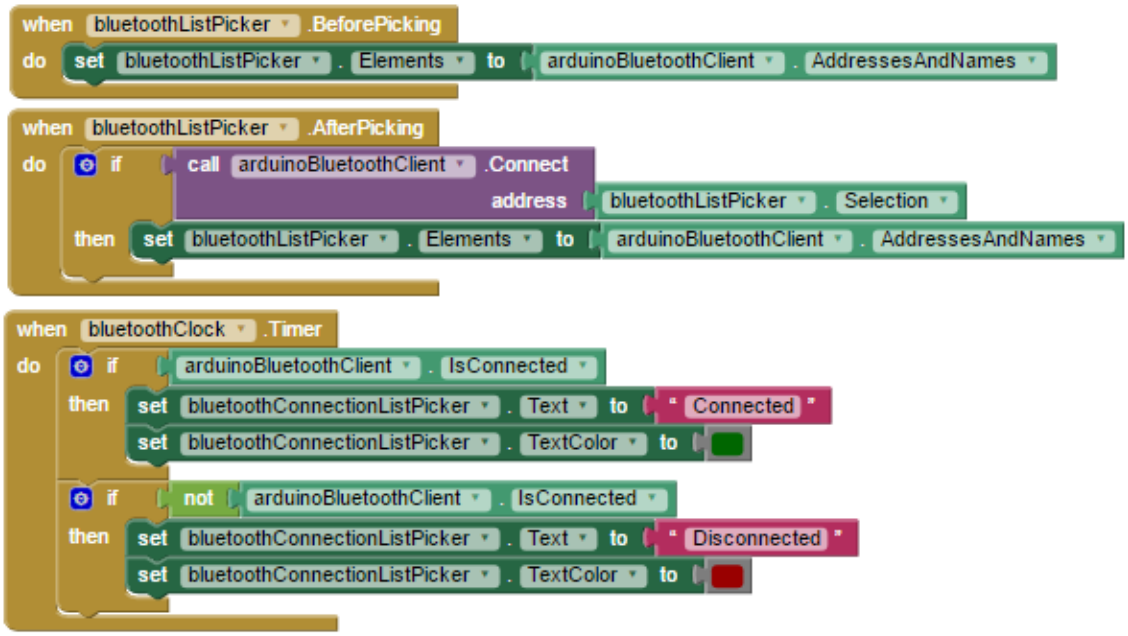
- **BluetoothClient**: can be found in the **Connectivity** tab within the **Palette**
- **Clock:** is in the **Sensors** tab
- **SpeechRecognizer**: is used to recognize the speech – is in the **Media** tab
- **AccelerometerSensor**: detects if your phone was shaken or not – can be found in the **Sensors** tab.

Rename all your components with elucidative names.



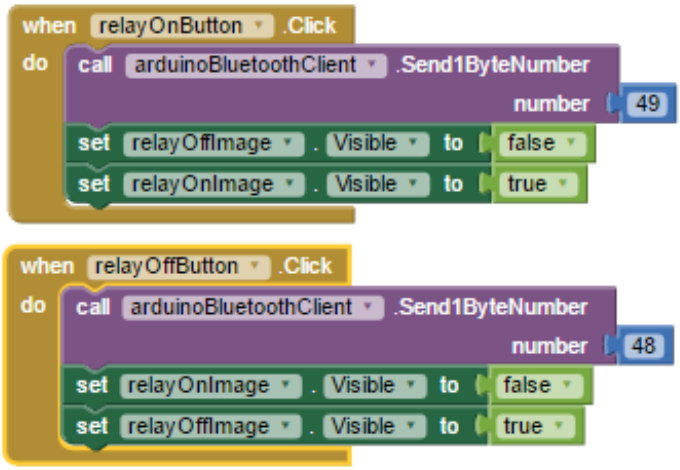
Blocks Editor

Start by dragging the blocks that establish the bluetooth connection:



Buttons

Turning the relay on and off is as simple as follows:

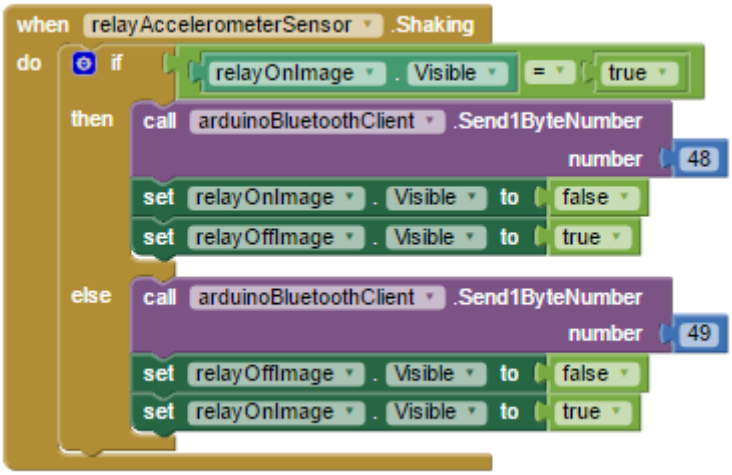


When you click the **RELAY ON** button you send 49 decimal (1 in char) to the bluetooth module. You also set the light bulb on image to visible and the light bulb off image to invisible.

When you tap the **RELAY OFF** button you send 48, which is 0 in char, and you set light bulb off image to visible.

Accelerometer

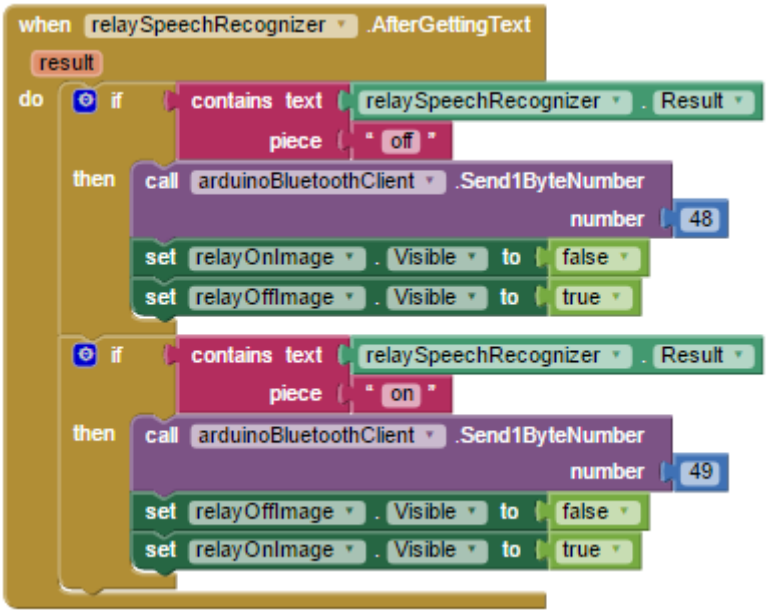
You want to send 49 or 48 when you shake the phone depending on the current relay state. You also need to set the right image to visible accordingly to the relay state. For that, you use the following block.



Voice commands

For the voice command, you start by calling the **SpeechRecognizer** when you press the **Voice Command** button. Then, the voice command is translated into text.

The next block checks if the text from the **SpeechRecognizer** contains the words “on” or “off” and sets the realy state accordingly. It also sets the right image to visible.



To create the Android app, go to the **Build** tab and generate a new *.apk* file. Alternatively, you can use the *.apk* file that comes with the eBook's resources folder. Move the *.apk* file to your Android phone and follow the on-screen instructions to install it.

Writing the Sketch

Upload the following sketch to your Arduino board. Make sure you have the right board and COM port selected.

Warning: When you're uploading code to your Arduino you should disconnect the TX and RX cables from the bluetooth module. These pins are needed for serial communication between the Arduino and your computer.

Tip: People often make the wrong connections between the Arduino and bluetooth module. **TX** from the bluetooth module connects to **RX** pin of the Arduino. **RX** from the bluetooth module connects to **TX** pin of the Arduino.

```
/*
 * Rui Santos
 * Complete Project Details http://randomnerdtutorials.com
 */

int relay = 11;      // pin Digital 11
int state;           // saves the state
int flag=0;    // makes sure that the serial only prints once the state

void setup() {
    // sets the Relay as output:
    pinMode(relay, OUTPUT);
    digitalWrite(relay, HIGH);

    // initialize serial communication at 9600 bits per second:
    Serial.begin(9600);
}

void loop() {
    //if some date is sent, reads it and saves in state
    if(Serial.available() > 0){
      state = Serial.read();
      flag=0;
    }
    // if the state is '0' the relay will turn OFF
    if (state == '0') {
        digitalWrite(relay, HIGH);
        if(flag == 0){
          Serial.println("Relay Off!");
          flag=1;
        }
    }
```

```
    // if the state is '1' the relay will turn ON
    else if (state == '1') {
        digitalWrite(relay, LOW);
        if(flag == 0){
          Serial.println("Relay On!");
          flag=1;
        }
    }
    //Uncomment For debugging purpose
    //Serial.println(state);
}
```

The sketch for this project is simple. It is similar to the sketch in Project 1.

You need to read the information sent from the Android app and check whether it was 1 or 0 to set the relay state to LOW or HIGH.

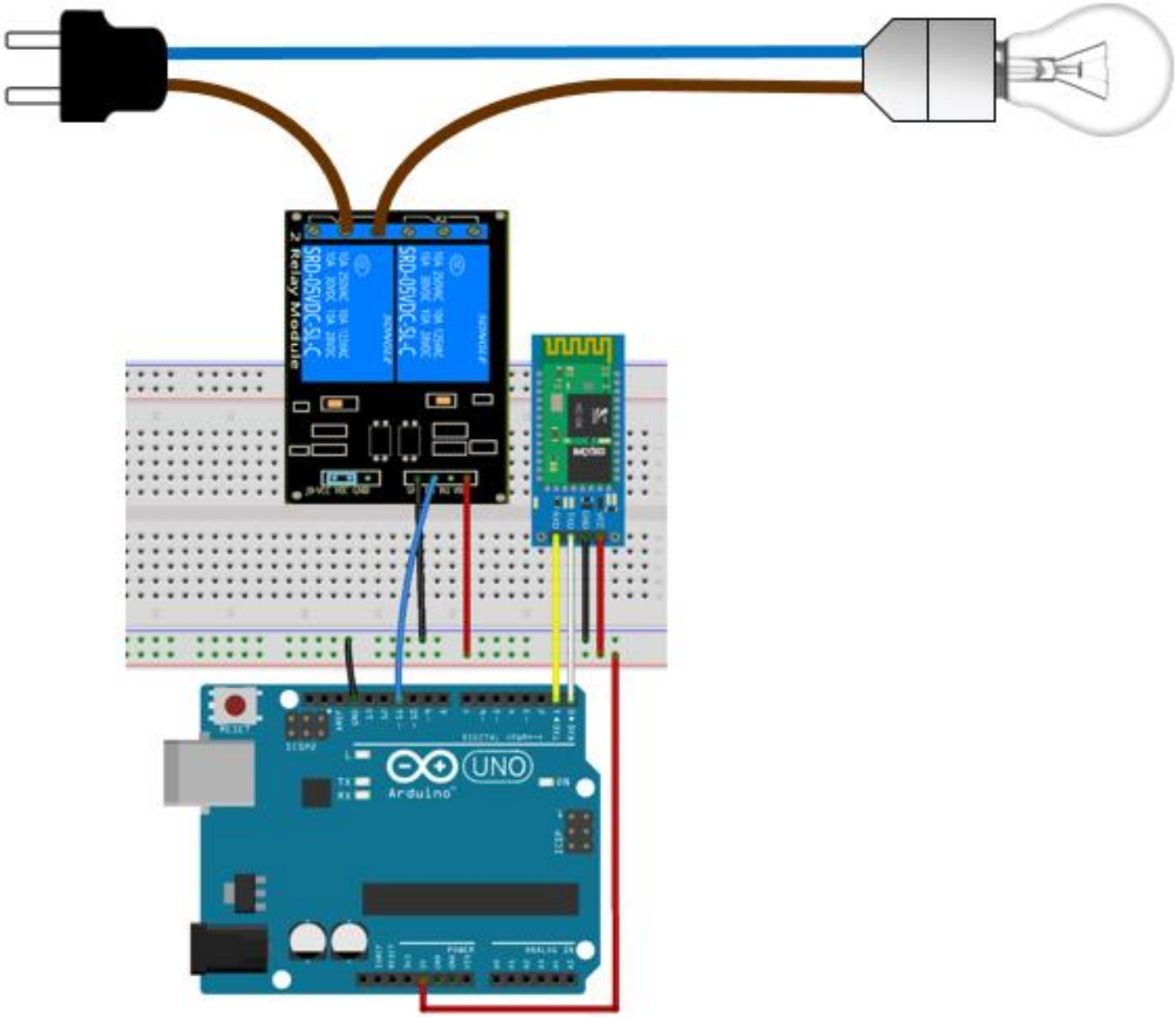
In the code, when you write `digitalWrite(relay, HIGH)` you are turning the relay off. When you have `digitalWrite(relay, LOW)` you are turning the relay on.

Wiring the Circuit

Follow the next schematic for building the circuit. If you're not comfortable dealing with mains voltage, you can also do the Project. Just replace the relay with an LED and use the sketch in Project 1.

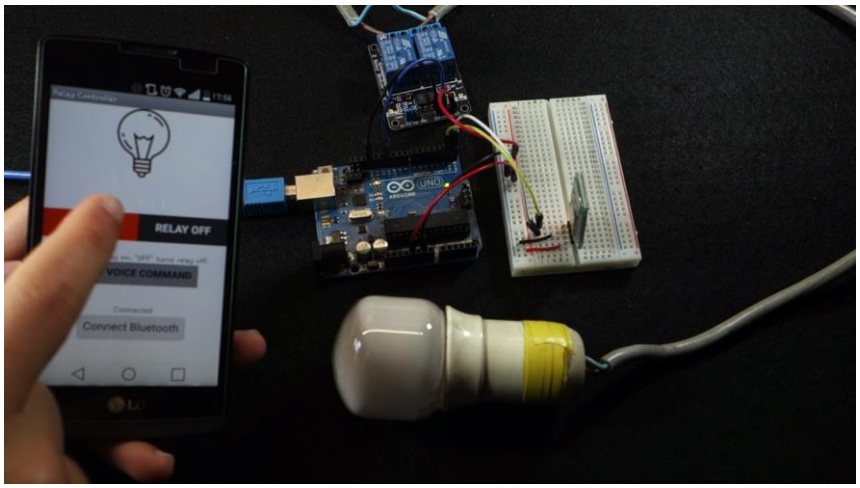
Don't touch the circuit when it is connected to mains voltage!

Warning: do not upload a new code to your Arduino board while your lamp is connected to the mains voltage. You should unplug the lamp from mains voltage, before uploading a new sketch to your Arduino.

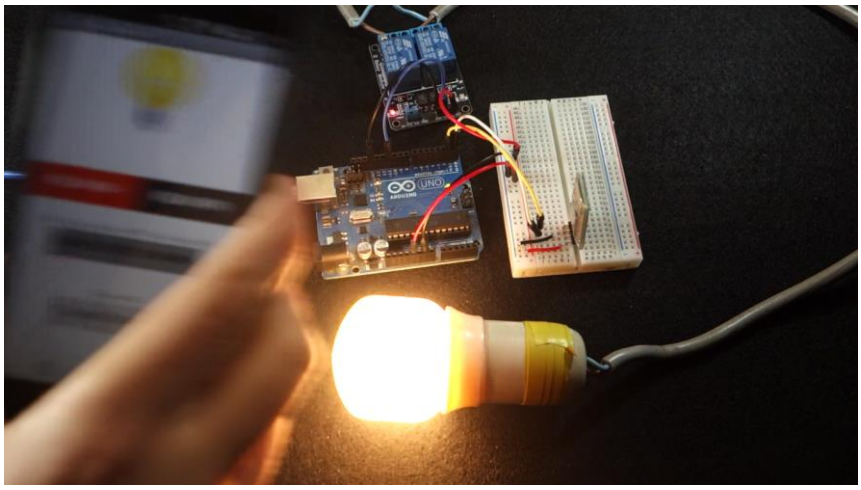


Demonstration

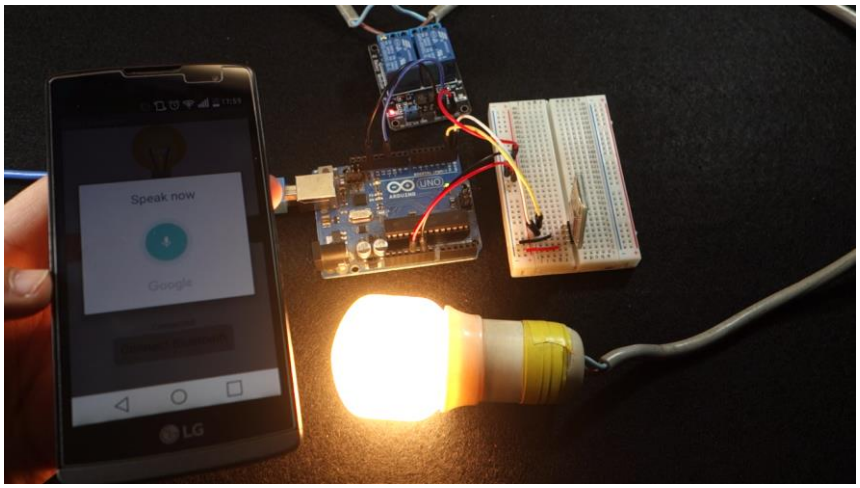
After connecting the bluetooth, you can control the relay with the **RELAY ON** and **RELAY OFF** buttons:



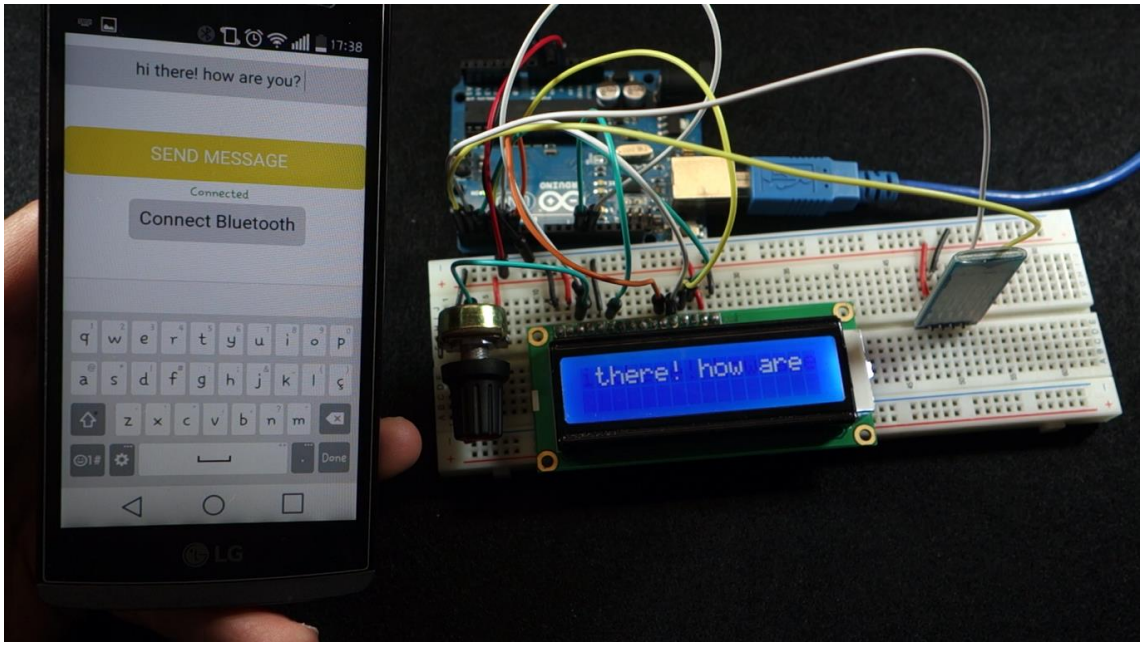
By shaking your smartphone:



And with voice commands.



Project 7: Display Messages

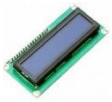


Project 7: Display Messages

In this project you're going to build an Android app that allows you to write messages and send them to the Arduino to be displayed on a 16x2 LCD display.

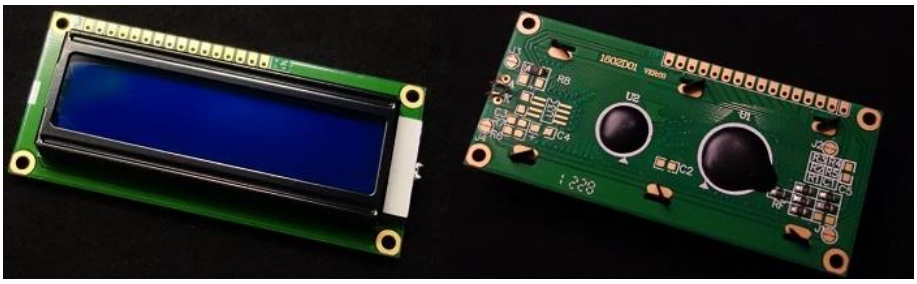
Parts Required

Here's a complete list of parts you need to do this project.

Figure	Name	eBay
	Arduino UNO	http://ebay.to/1SQda0R
	HC-04 or HC-05 or HC-06 Bluetooth module	http://ebay.to/2dt1b6M
	16x2 LCD display (Hitachi HD44780 driver)	http://ebay.to/2cgnKeW
	Breadboard	http://ebay.to/21bEojM
	10kOhm Potentiometer	http://ebay.to/1PUefOb
	Jumper Wires	http://ebay.to/1PXeaJz

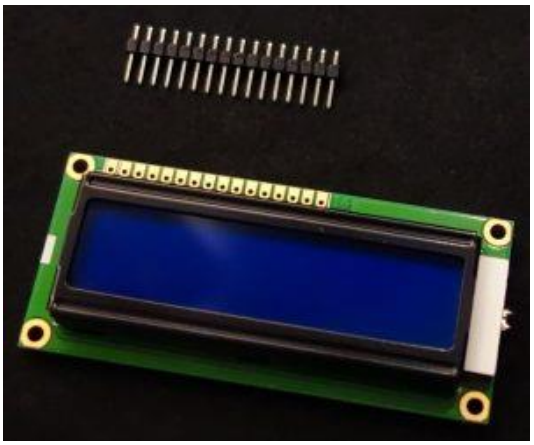
Introducing the Liquid Crystal Display (LCD)

The 16×2 LCD is the one in the following figure (front and the back view).

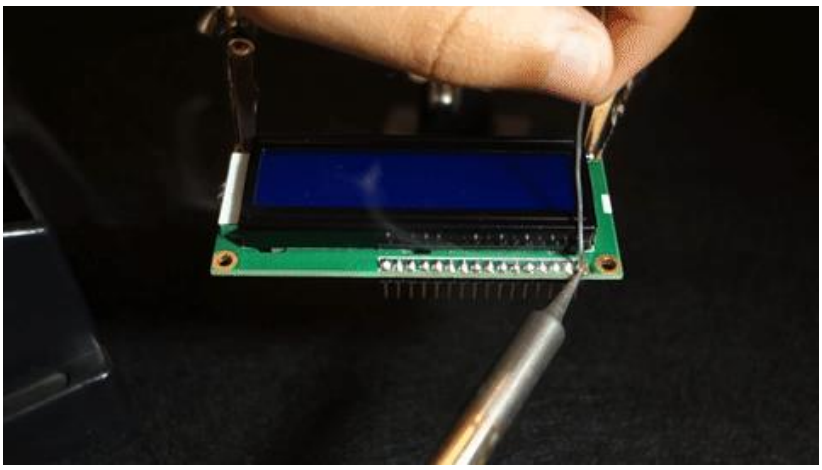


The 16×2 LCD is commonly used in several circuits and devices. This LCD has 2 rows, and each row can display 16 characters. It also has LED backlight that can be adjusted with a potentiometer.

When you buy a 16×2 LCD, usually it doesn't come with breadboard friendly pins. So, you may need some headers.



Solder the headers to your LCD, and it is ready to use.

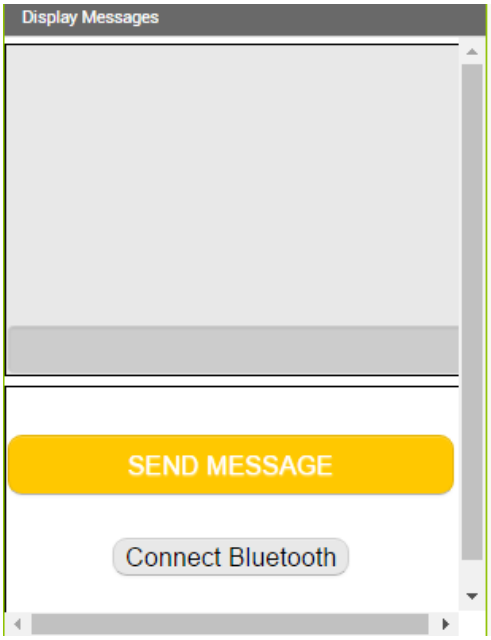


Building the App

Go to http://ai2.appinventor.mit.edu for opening the MIT App Inventor 2 software. Go to **Projects ▸ Start New Project** and call it *Display_Messages*.

> Alternatively, if you prefer, you can upload the *.aia* file and edit the app the way you like. You just have to go to **Projects ▸ Import project (.aia) from my computer** and upload the *Display_Messages.aia* file that comes with the eBook's resources.

Here's how your app looks in the end of this project:



Design

Add a **Connect Bluetooth** listpicker and a **SEND MESSAGE** button. You also need two non-visible components: the **BluetoothClient** and the **Clock**.

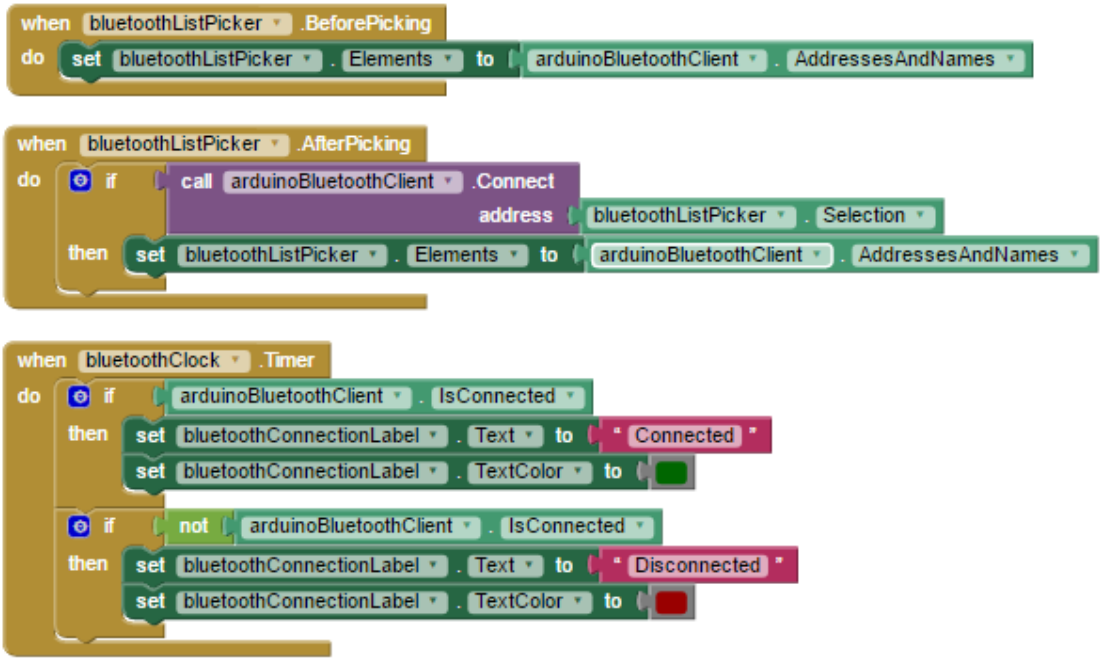
For this project, you need a field where you can write text in and there's a component called **TextBox** for that task. Go to **Palette ▸ User Interface** and drag a **TextBox** into the **Viewer**. Rename the **TextBox** to **messageTextBox** and edit its properties:

- **Background color:** Light Gray
- **Enabled:** ✓
- **FontBold:**
- **FontItalic:**
- **FontSize:** 18
- **FontTypeface:** sans serif
- **Height:** Automatic...
- **Width:** Fill parent...

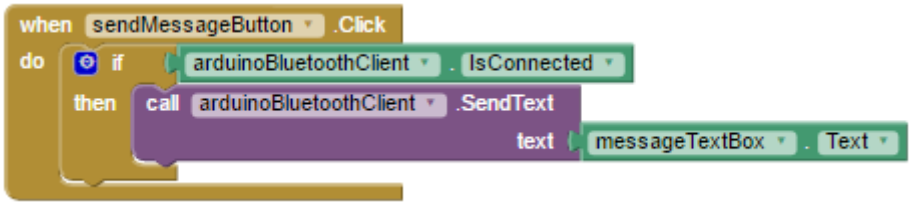
- **Hint:** Type your message…
- **Multiline:** ☐
- **NumbersOnly**: ☐
- **Text**:
- **TextAlignment**: center: 1
- **TextColor**: Black
- **Visible**: ✓

Blocks Editor

Start by dragging the blocks that establish the bluetooth connection.



Then, you want to send the text in the **messageTextBox** to the Arduino when you click the **SEND MESSAGE** button. For that, you use the following blocks:



To create the Android app, go to the **Build** tab and generate a new *.apk* file. Alternatively, you can use the *.apk* file that comes with the eBook's resources folder. Move the *.apk* file to your Android phone and follow the on-screen instructions to install it.

Writing the Sketch

Copy the following code to the Arduino IDE and upload it to your board. Make sure you have the right board and COM port selected.

Warning: When you're uploading code to your Arduino you should disconnect the TX and RX cables from the bluetooth module. These pins are needed for serial communication between the Arduino and your computer.

Tip: People often make the wrong connections between the Arduino and bluetooth module. **TX** from the bluetooth module connects to **RX** pin of the Arduino. **RX** from the bluetooth module connects to **TX** pin of the Arduino.

```
/*
 * Rui Santos
 * Complete Project Details http://randomnerdtutorials.com
 */

// include the library code:
#include <LiquidCrystal.h>

#define max_char 32
char message[max_char];     // stores you message
char r_char;                // reads each character
byte index = 0;             // defines the position into your array
int i;

// initialize the library with the numbers of the interface pins
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup() {
  // set up the LCD's number of columns and rows:
  lcd.begin(16, 2);
  delay(1000);
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
}

void loop() {
  //check if serial is available before reading
  //a new message delete's the old message
  if(Serial.available()){
    for(i=0; i<31; i++){
      message[i] = '\0';
    }
    //resets the index
    index=0;
  }
  //while is reading the message
  while(Serial.available() > 0){
    //the message can have up to 100 characters
    if(index < (max_char-1)){
      r_char = Serial.read();       // Reads a character
```

```
      message[index] = r_char; // Stores the character in message array
      index++;                 // Increment position
      message[index] = '\0';   // Delete the last position
    }
 }
 lcd.clear();
 lcd.print(message);

 // scroll 16 positions to the left
 // to move it offscreen left
 for(int positionCounter=0;positionCounter<index/2; positionCounter++){
  delay(750);
  // scroll one position left:
  lcd.scrollDisplayLeft();
 }

}
```

You start the sketch importing the needed library to control the LCD display:

```
#include <LiquidCrystal.h>
```

In the `setup()` you initialize the LCD and start the serial communication for the bluetooth module.

```
void setup() {
  // set up the LCD's number of columns and rows:
  lcd.begin(16, 2);
  delay(1000);
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
}
```

In the `loop()` you start cleaning your `message` variable. This ensures that old messages are deleted before receiving newer ones.

```
  if(Serial.available()){
    for(i=0; i<31; i++){
      message[i] = '\0';
    }
    //resets the index
    index=0;
  }
```

Then, the `message` variable stores the characters coming from the app.

```
//while is reading the message
  while(Serial.available() > 0){
    //the message can have up to 100 characters
    if(index < (max_char-1)){
      r_char = Serial.read();      // Reads a character
      message[index] = r_char;      // Stores the character in message
array
      index++;                     // Increment position
      message[index] = '\0';       // Delete the last position
    }
```

To write the message in the display you use:

```
lcd.print(message);
```

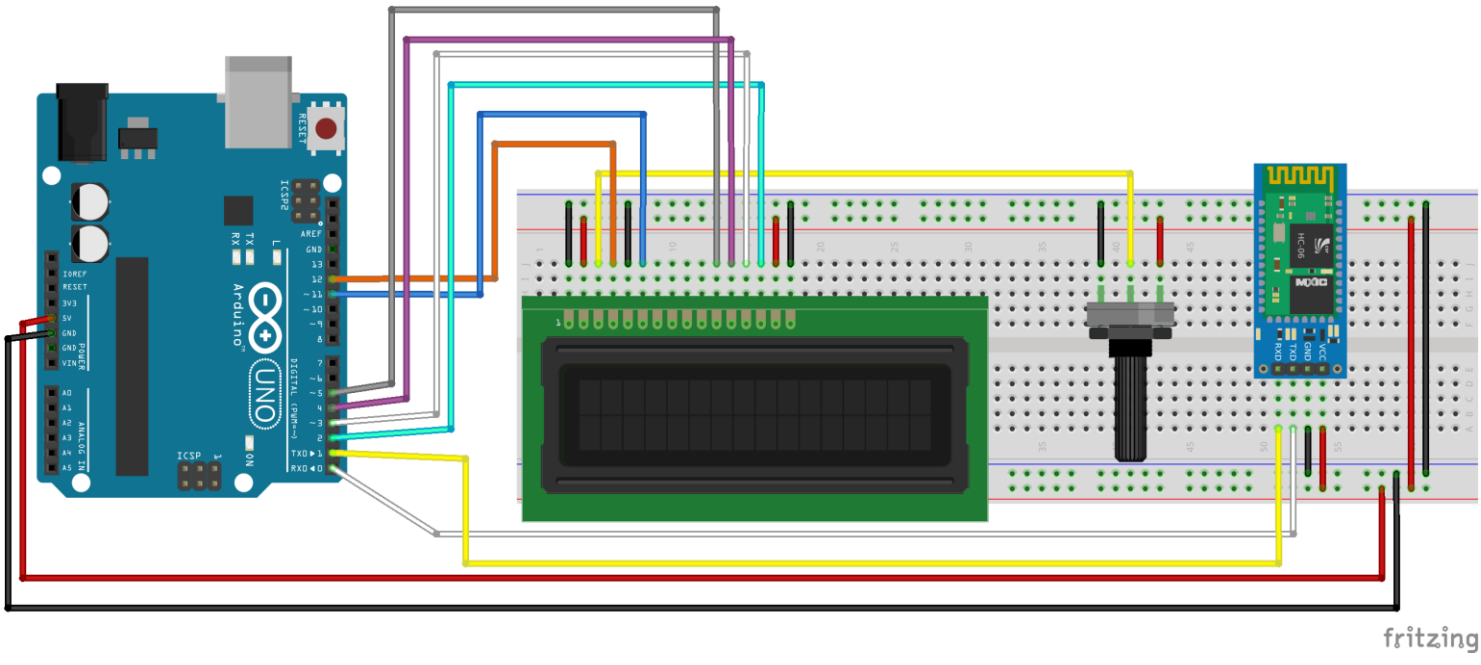
To scroll the message leftward you use the following lines of the code.

```
for(int positionCounter=0; positionCounter<index/2; positionCounter++){
  delay(750);
  // scroll one position left:
  lcd.scrollDisplayLeft();
}
```

Wiring the Circuit

Here's the circuit for this project. You just need to wire an LCD and the bluetooth module.

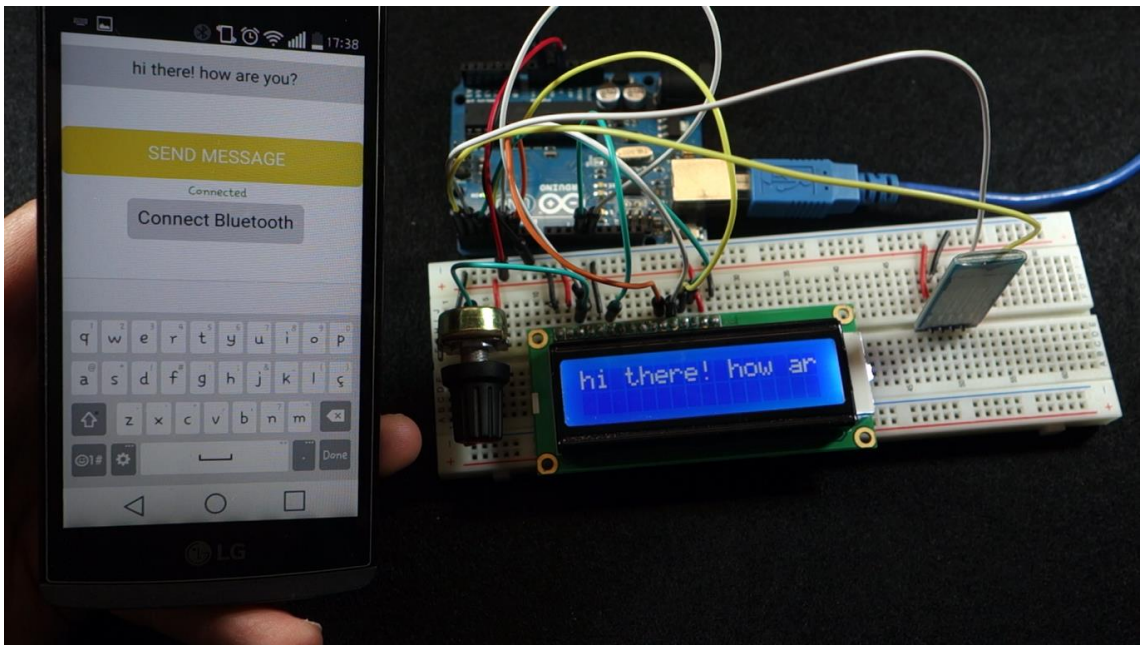
LCD	Wiring to
Pin 1	GND
Pin 2	5V
Pin 3	Potentiometer
Pin 4	Digital pin 12
Pin 5	GND
Pin 6	Digital pin 11
Pin 7	-
Pin 8	-
Pin 9	-
Pin 10	-
Pin 11	Digital pin 5
Pin 12	Digital pin 4
Pin 13	Digital pin 3
Pin 14	Digital pin 2
Pin 15	5V
Pin 16	GND



Demonstration

After establishing the bluetooth communication, write a message on the **TextBox**.

Then, click **SEND MESSAGE** button to send the message to the LCD.



Project 8: Arduino Bluetooth Robot

Project 8: Arduino Bluetooth Robot

In this project you're going to build a bluetooth remote controlled robot. We'll also show you how to assemble the robot car chassis kit.

Your app will have five controls: forward, reverse, left, right and stop.

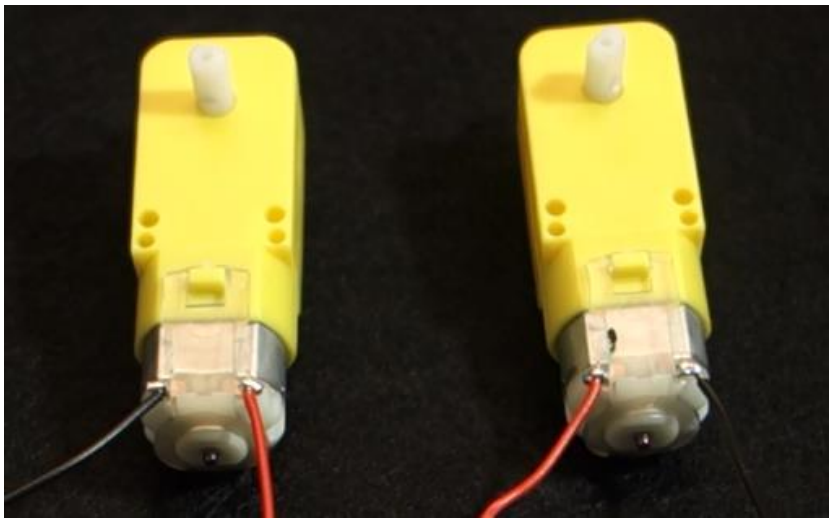
Parts Required

Here's a complete list of parts you need to do this project.

Figure	Name	eBay
	Arduino UNO/Nano	http://ebay.to/1SQda0R
	HC-04 or HC-05 or HC-06 Bluetooth module	http://ebay.to/2dt1b6M
	L293D IC	http://ebay.to/2e4AXt4
	Robot Chassis (includes 2DC motors)	http://ebay.to/2exBEKO
	Power Bank/Portable Charger	http://ebay.to/2dDhmRo
	Breadboard	http://ebay.to/21bEojM
	Jumper Wires	http://ebay.to/1PXeaJz

Introducing DC Motors

To get your robot car moving, you'll use two DC motors as shown in the figure below.



These two yellow DC motors come with the robot chassis kit.

Connecting the DC motors

You have two wires connected to your DC motors: one red and one black.

Note: the motors in these kits have no polarity (unless they are specifically marked with a + or -). So, changing the way you wire the motor changes the direction of rotation.

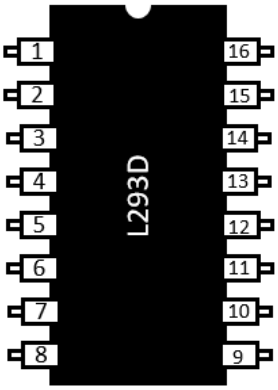
To get you familiar with the DC motor, connect the red wire to 5V and the black wire to the GND (or the other way around). Once they are connected, the shaft starts spinning continuously in one direction.

As you don't want your robot to just move forward or backward, you need to use an H-bridge.

Introducing L293D IC (H-bridge)

An H-bridge is a circuit that enables a voltage to be applied across a load (the motors) in either direction. For that, you need an L293D IC.

The next figure shows the L293D IC pinout.



Pin	Description
1	Enable 1
2	Input 1
3	Output 1
4	GND
5	GND
6	Output 2
7	Input 2
8	VCC 1
9	Enable 2
10	Input 3
11	Output 3
12	GND
13	GND
14	Output 4
15	Input 4
16	VCC 2

- The output pins are used to connect the DC motor. These pins output the signal that tells the motor what to do;
- The VCC pins power up the IC;
- The input pins are controlled by the Arduino. They determine in which direction the DC motor should rotate;
- The enable pins turn the motor on or off.

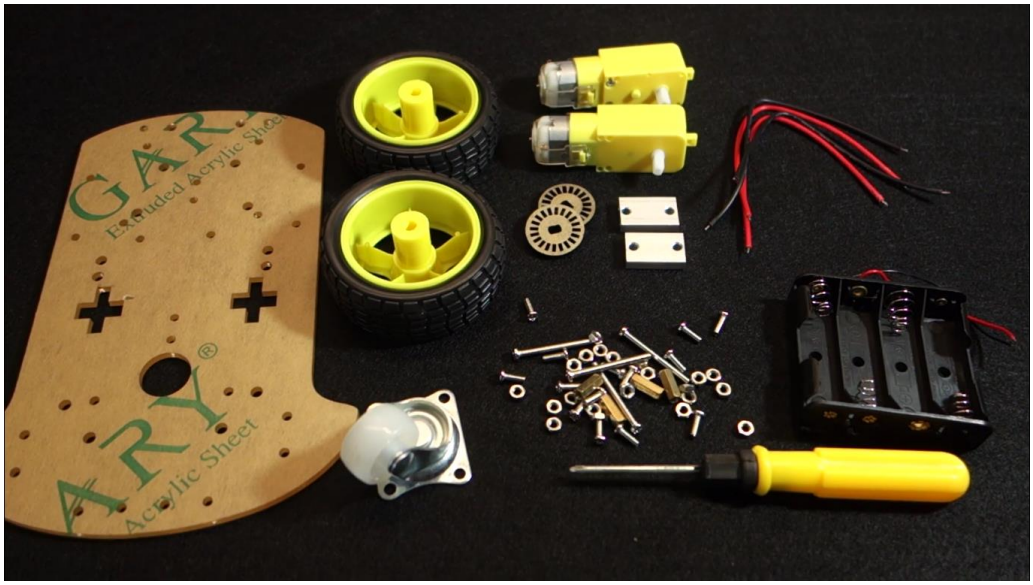
Assembling the Robot Car Chassis

You can use any robot chassis you want, or you can build one with materials that you might have laying around.

However, I decided to buy one at eBay, it is affordable and it comes with 2 DC motors. Here's how I assembled my robot chassis.

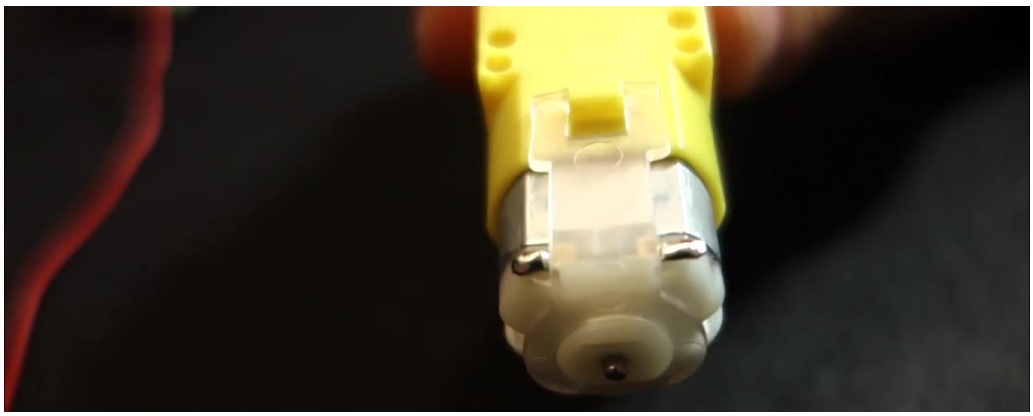
1. This is the package that you get when you buy the kit.

2. Open the package - it comes with one screwdriver, two DC motors, two wheels and one acrylic car chassis. There is also a small plastic bag that comes with a battery holder that I don't use need for this project, a small wheel for the front, some bolts and screws, four wires and other required components to assemble the robot.

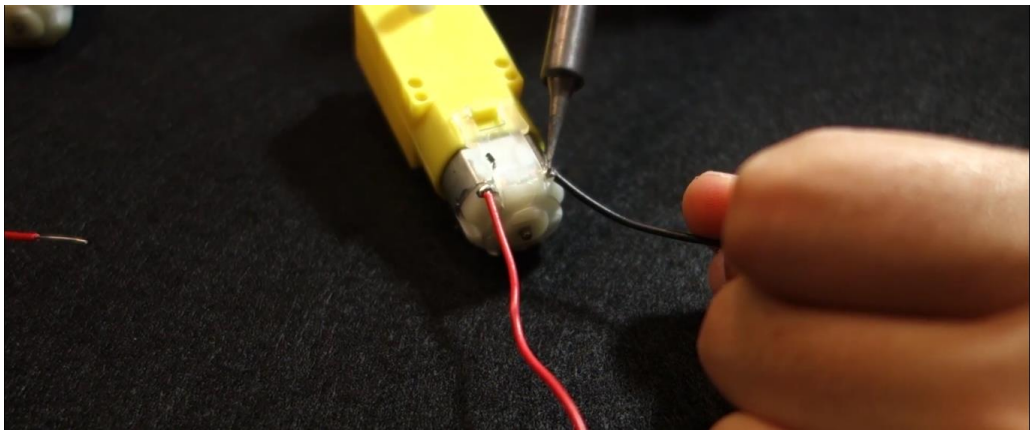


3. Although this kit comes with the needed wires, since they are a bit short and not very flexible, we've decided to replace them with other wires, because we can easily adjust their length. You'll need one red wire and one black wire for each motor. You can use a wire cutter to cut the wires to your desired length.

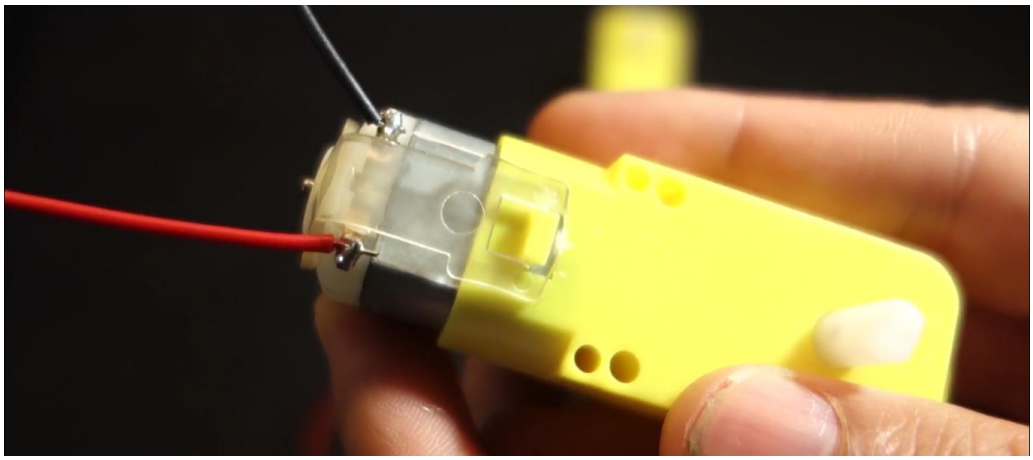
4. After preparing all four wires, you need to solder them to the DC motors. Tin the DC motor pins.



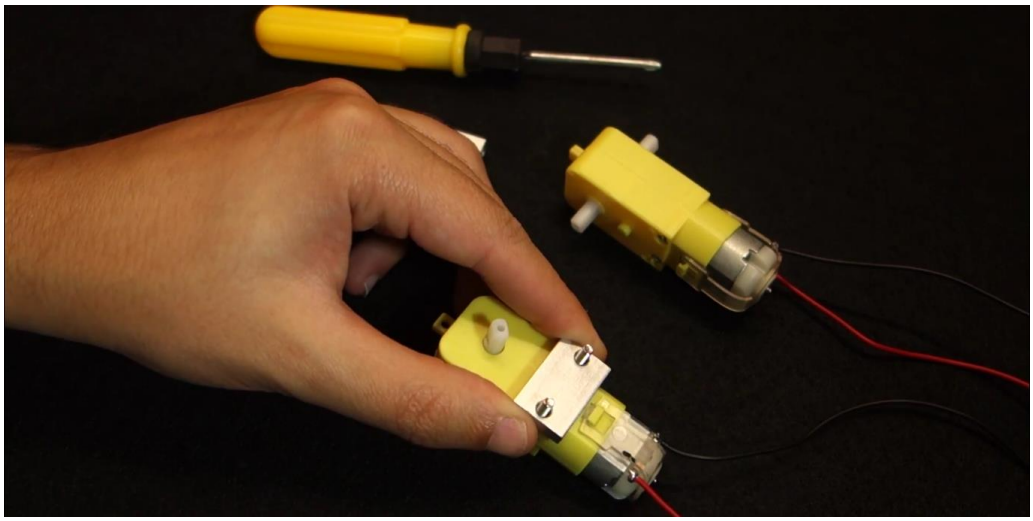
5. Then, grab the wire and solder it to the DC motor pin. Repeat that process to all the other wires.



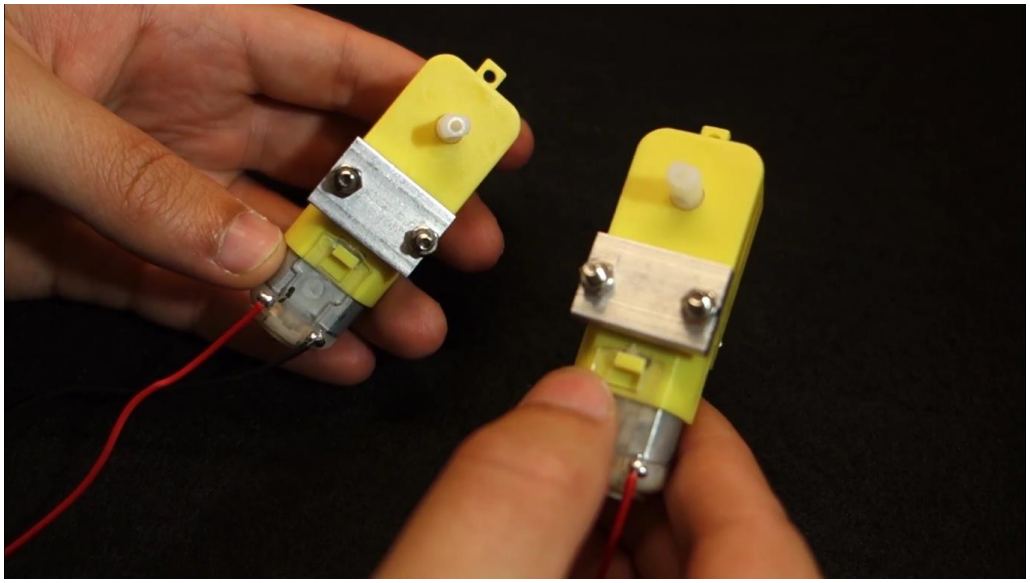
This is how the motors look like after soldering the wires.



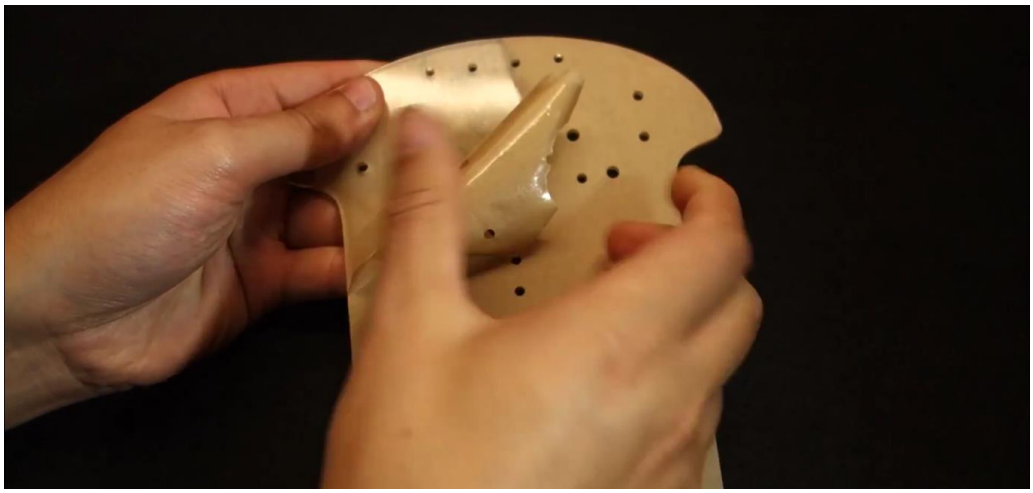
6. Now, you need the screwdriver, bolts and screws, and those metal pieces. Start by attaching the metal pieces to the DC motors.



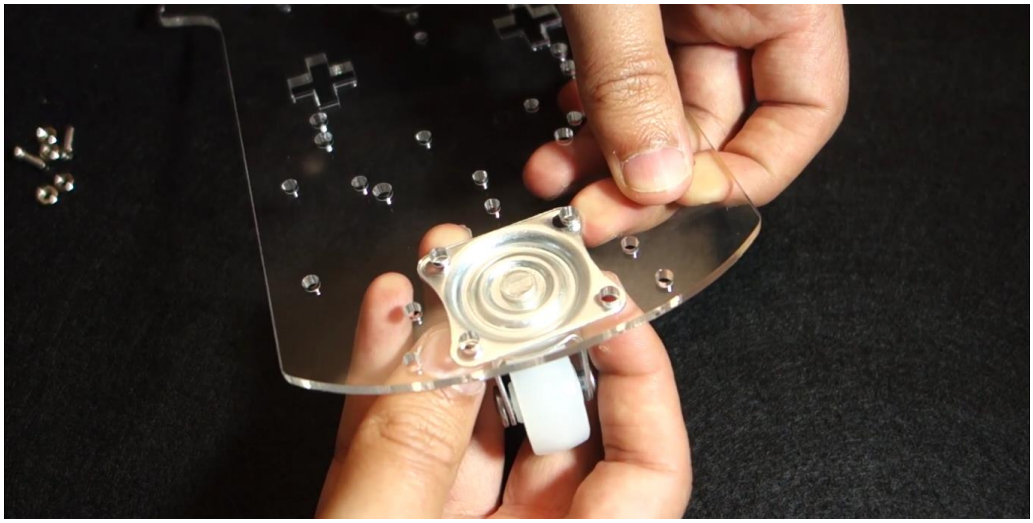
7. Here's how they look like after this step.



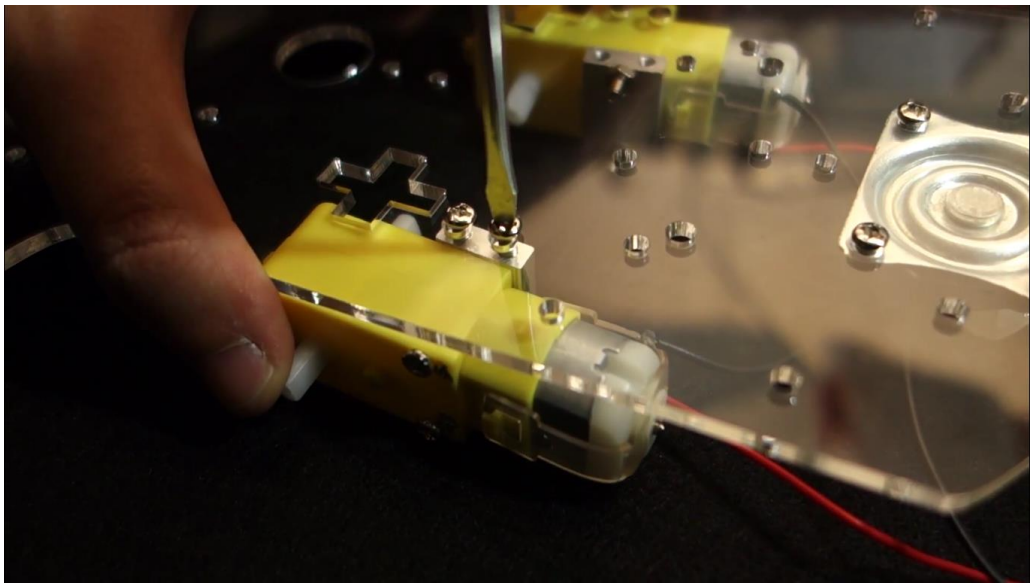
8. Remove the protective adhesive from the acrylic chassis in both sides.



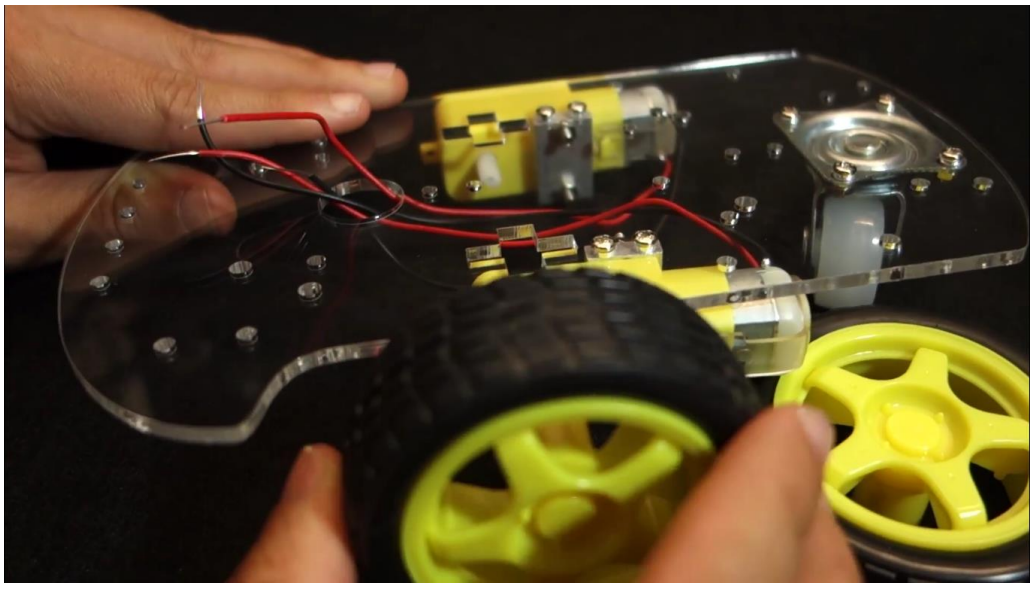
9. Grab the small wheel and attach it to the front part of the chassis.



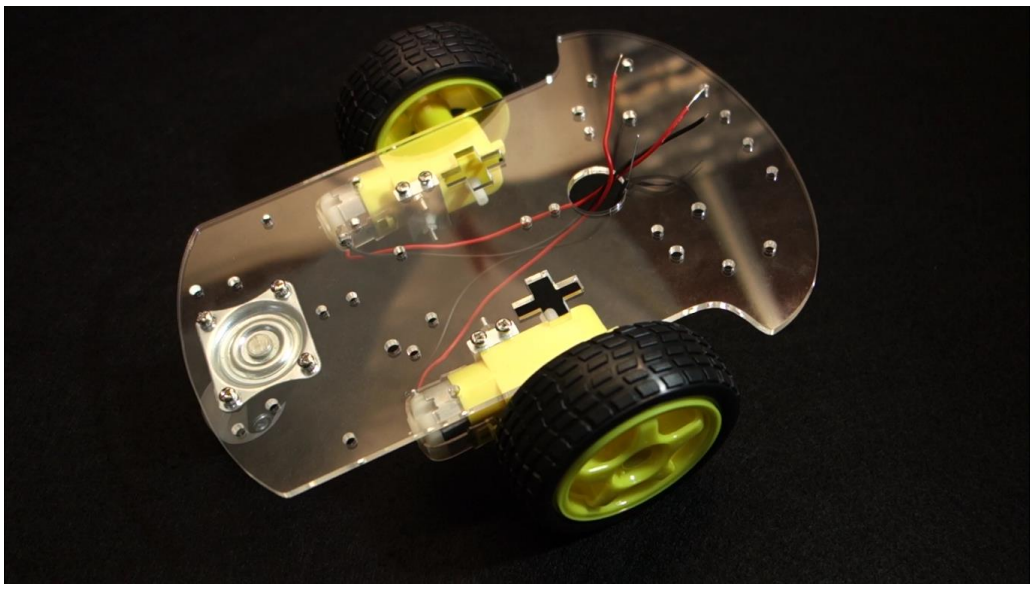
10. Finally, it is time to attach the DC motors to the chassis.



11. Lastly, connect the wheels to the DC motors.



Now, your robot car chassis is ready. Here's how it looks after assembling:



Tips:

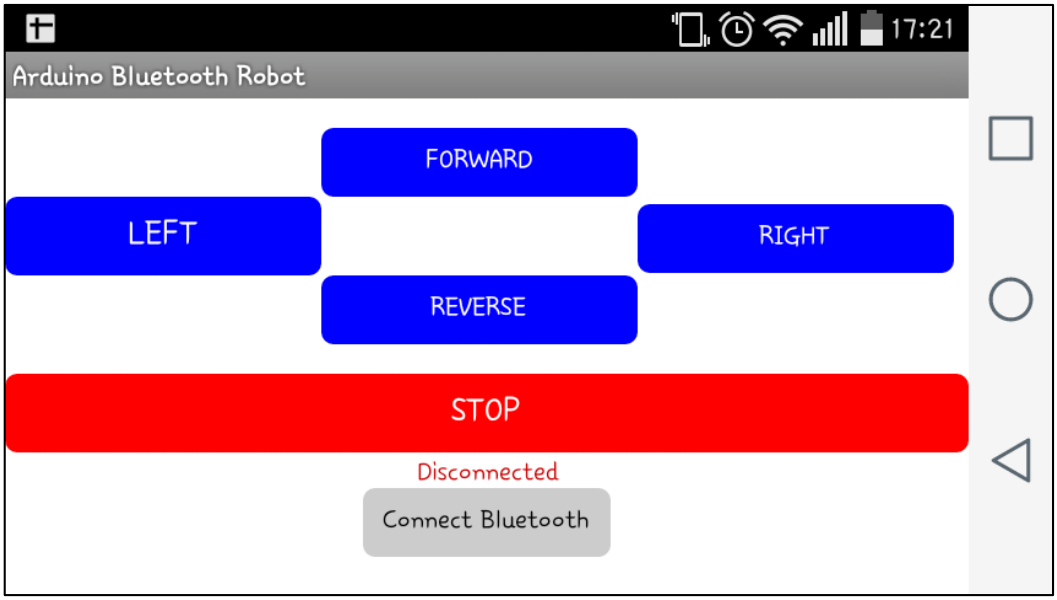
- When it comes to building robots with Arduino, we often use an Arduino Nano board, because it's lighter and fits well on the breadboard;
- We use an half breadboard that fits nicer on the robot chassis;
- We usually don't use the battery holder. Instead we recommend using a portable charger (power bank), it's rechargeable, occupies less space and is more practical.
- We use Velcro to attach the portable charger and the breadboard to the chassis. It's really practical as you can easily attach and detach the components without any damage.

Building the App

Go to http://ai2.appinventor.mit.edu for opening the MIT App Inventor 2 software. Go to **Projects** ▸ **Start New Project** and call it *Arduino_Bluetooth_Robot.*

> Alternatively, if you prefer, you can upload the *.aia* file and edit the app the way you like. You just have to go to **Projects** ▸ **Import project (.aia) from my computer** and upload the *Arduino_Bluetooth_Robot.aia* file that comes with the eBook's resources.

Here's how your app looks in the end of this project:

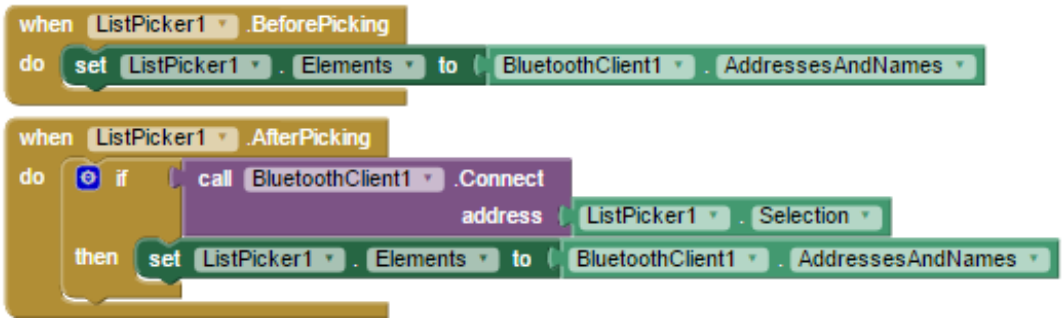


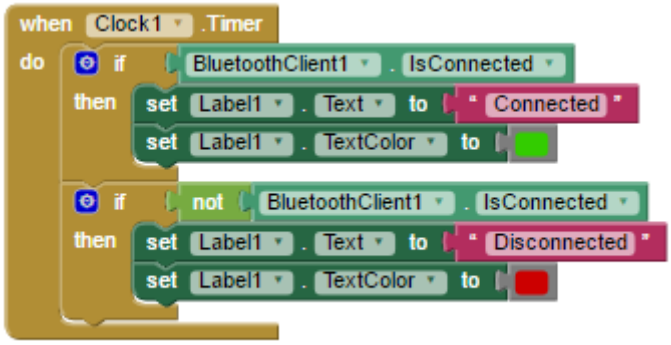
Design

The design for the app is simple. You need five robot controller buttons and one listpicker to connect bluetooth. Edit the buttons properties the way you like most. Rename them with elucidative names to simplify the code blocks section.

Blocks Editor

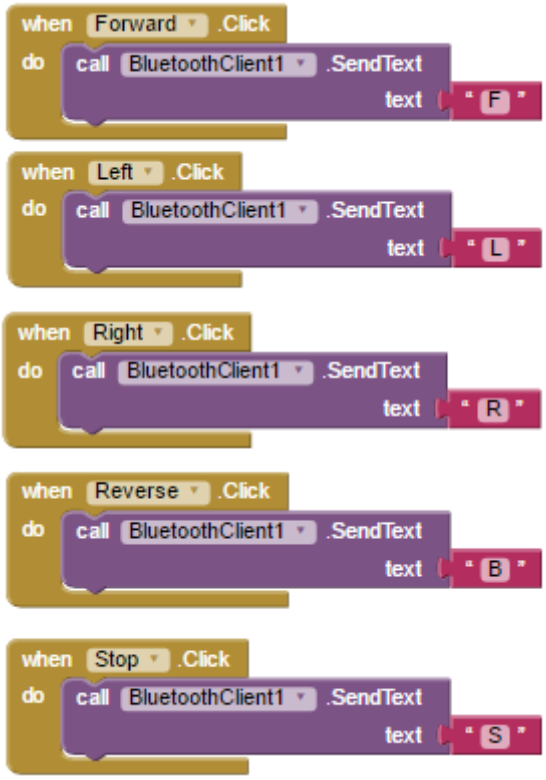
Drag the blocks to establish the bluetooth connection.





To control the robot you're going to send different characters depending on the button you're tapping:

- **Forward button**: "F"
- **Left button**: "L"
- **Right button**: "R"
- **Reverse button**: "B"
- **Stop button**: "S"



To create the Android app, go to the **Build** tab and generate a new *.apk* file. Alternatively, you can use the *.apk* file that comes with the eBook's resources folder. Move the *.apk* file to your Android phone and follow the on-screen instructions to install it.

Writing the Sketch

Upload the following code to your Arduino board. Make sure you have the right board and COM port selected.

Warning: When you're uploading code to your Arduino you should disconnect the TX and RX cables from the bluetooth module. These pins are needed for serial communication between the Arduino and your computer.

Tip: People often make the wrong connections between the Arduino and bluetooth module. **TX** from the bluetooth module connects to **RX** pin of the Arduino. **RX** from the bluetooth module connects to **TX** pin of the Arduino.

```
/*
 * Rui Santos
 * Complete Project Details http://randomnerdtutorials.com
 */

int motor1Pin1 = 3; // pin 2 on L293D IC
int motor1Pin2 = 4; // pin 7 on L293D IC
int enable1Pin = 6; // pin 1 on L293D IC
int motor2Pin1 = 8; // pin 10 on L293D IC
int motor2Pin2 = 9; // pin 15 on L293D IC
int enable2Pin = 11; // pin 9 on L293D IC
int state;
int flag=0;   //makes sure that the serial only prints once the state
int stateStop=0;
void setup() {
    // sets the pins as outputs:
    pinMode(motor1Pin1, OUTPUT);
    pinMode(motor1Pin2, OUTPUT);
    pinMode(enable1Pin, OUTPUT);
    pinMode(motor2Pin1, OUTPUT);
    pinMode(motor2Pin2, OUTPUT);
    pinMode(enable2Pin, OUTPUT);
    // sets enable1Pin and enable2Pin high so that motor can turn on:
    digitalWrite(enable1Pin, HIGH);
    digitalWrite(enable2Pin, HIGH);
    // initialize serial communication at 9600 bits per second:
    Serial.begin(9600);
}

void loop() {
    //if some date is sent, reads it and saves in state
    if(Serial.available() > 0){
      state = Serial.read();
      flag=0;
    }
    // if the state is 'F' the DC motor will go forward
    if (state == 'F') {
        digitalWrite(motor1Pin1, LOW);
        digitalWrite(motor1Pin2, HIGH);
        digitalWrite(motor2Pin1, HIGH);
```

```
        digitalWrite(motor2Pin2, LOW);
        if(flag == 0){
          Serial.println("Go Forward!");
          flag=1;
        }
    }

    // if the state is 'R' the motor will turn left
    else if (state == 'R') {
        digitalWrite(motor1Pin1, LOW);
        digitalWrite(motor1Pin2, LOW);
        digitalWrite(motor2Pin1, HIGH);
        digitalWrite(motor2Pin2, LOW);
        if(flag == 0){
          Serial.println("Turn Right");
          flag=1;
        }
        delay(500);
        state=3;
        stateStop=1;
    }
    // if the state is 'S' the motor will Stop
    else if (state == 'S' || stateStop == 1) {
        digitalWrite(motor1Pin1, LOW);
        digitalWrite(motor1Pin2, LOW);
        digitalWrite(motor2Pin1, LOW);
        digitalWrite(motor2Pin2, LOW);
        if(flag == 0){
          Serial.println("STOP!");
          flag=1;
        }
        stateStop=0;
    }
    // if the state is 'L' the motor will turn right
    else if (state == 'L') {
        digitalWrite(motor1Pin1, LOW);
        digitalWrite(motor1Pin2, HIGH);
        digitalWrite(motor2Pin1, LOW);
        digitalWrite(motor2Pin2, LOW);
        if(flag == 0){
          Serial.println("Turn Left");
          flag=1;
        }
        delay(500);
        state=3;
        stateStop=1;
    }
    // if the state is 'B' the motor will Reverse
    else if (state == 'B') {
        digitalWrite(motor1Pin1, HIGH);
        digitalWrite(motor1Pin2, LOW);
        digitalWrite(motor2Pin1, LOW);
        digitalWrite(motor2Pin2, HIGH);
        if(flag == 0){
          Serial.println("Reverse!");
          flag=1;
        }
    }
    //For debugging purpose
    //Serial.println(state);
}
```

The code is well commented and it is fairly easy to understand what is does.

You need to check the received character and send the right signal to the right motor pins.

To move forward:

- motor1Pin1, LOW
- motor1Pin2, HIGH
- motor2Pin1, HIGH
- motor2Pin2, LOW

To move backward:

- motor1Pin1, HIGH
- motor1Pin2, LOW
- motor2Pin1, LOW
- motor2Pin2, HIGH

To move right:

- motor1Pin1, LOW
- motor1Pin2, LOW
- motor2Pin1, HIGH
- motor2Pin2, LOW

To move left:

- motor1Pin1, LOW
- motor1Pin2, HIGH
- motor2Pin1, LOW
- motor2Pin2, LOW

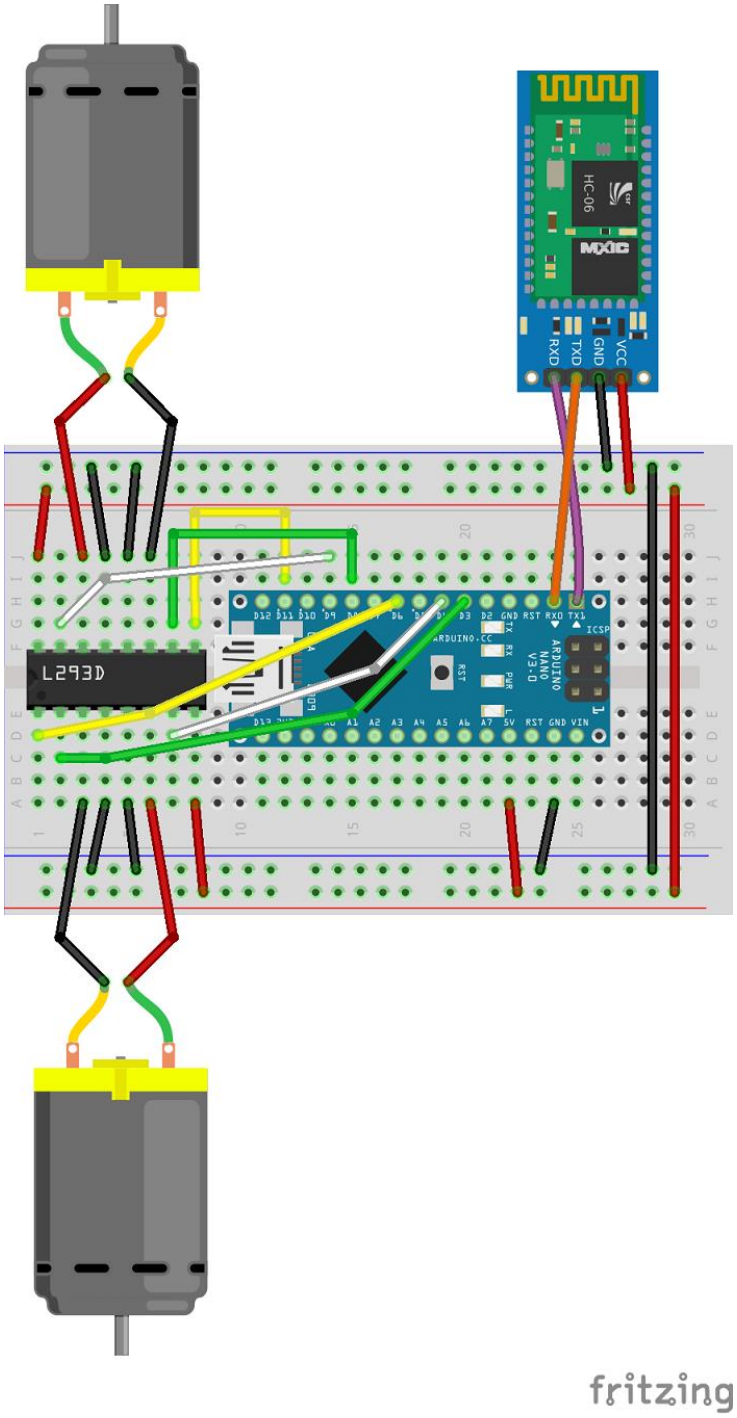
To stop:

- motor1Pin1, LOW
- motor1Pin2, LOW
- motor2Pin1, LOW
- motor2Pin2, LOW

Wiring the Circuit

Wire the circuit as shown in the schematic below. We're using an Arduino Nano because it fits better on the robot chassis. If you prefer, you can use an Arduino UNO, the connections are the same.

The Arduino Nano is being powered using a portable power bank - you can use batteries instead.

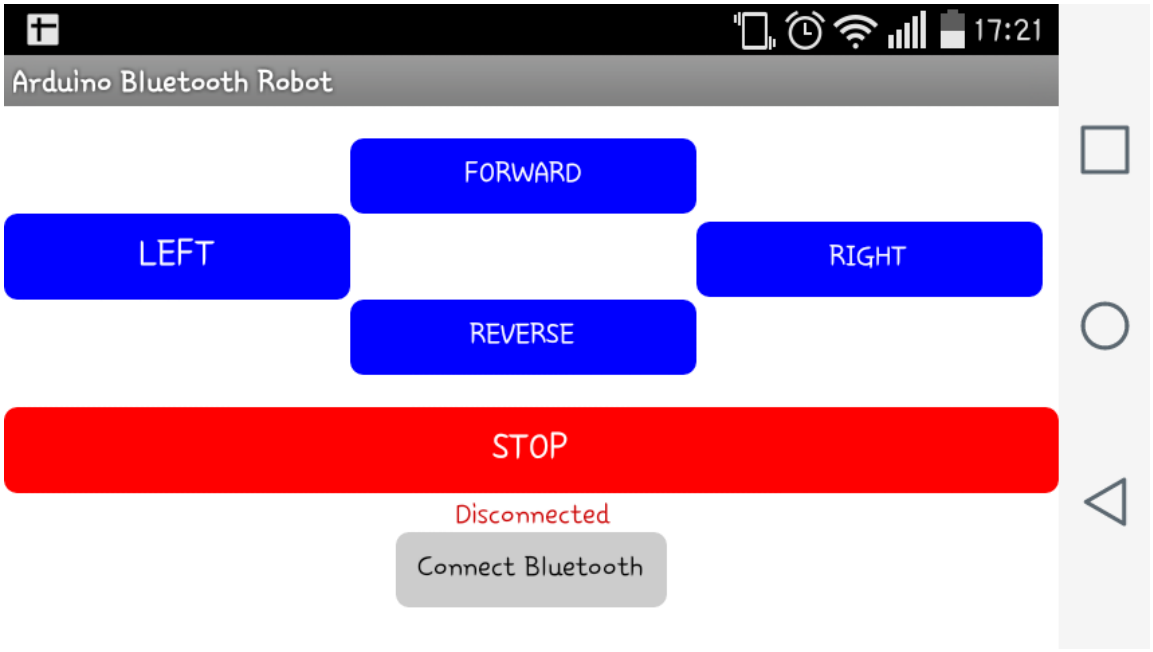


Here's a table showing the connections to L293D:

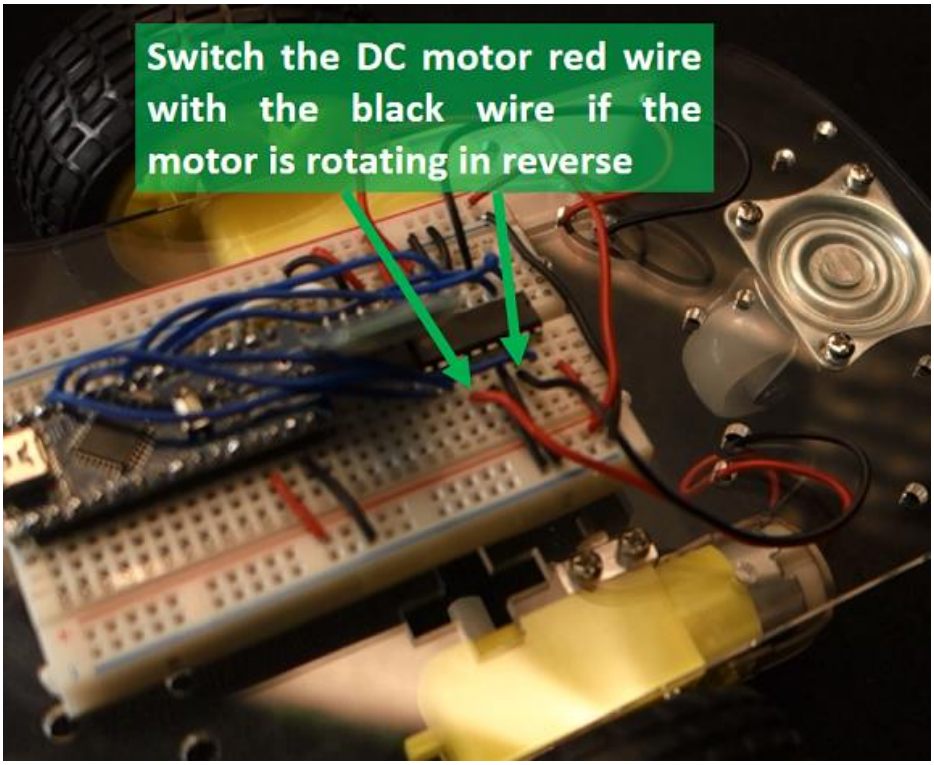
L293D	Wiring to
Pin 1	Digital Pin 6
Pin 2	Digital Pin 3
Pin 3	DC motor
Pin 4	GND
Pin 5	GND
Pin 6	DC motor
Pin 7	Digital Pin 4
Pin 8	5V
Pin 9	Digital Pin 11
Pin 10	Digital Pin 8
Pin 11	DC motor
Pin 12	GND
Pin 13	GND
Pin 14	DC motor
Pin 15	Digital Pin 9
Pin 16	5V

Troubleshooting

Test your app and control your robot.



If one or both motors are rotating in the wrong direction, you should switch the DC motor red wire with the black wire.



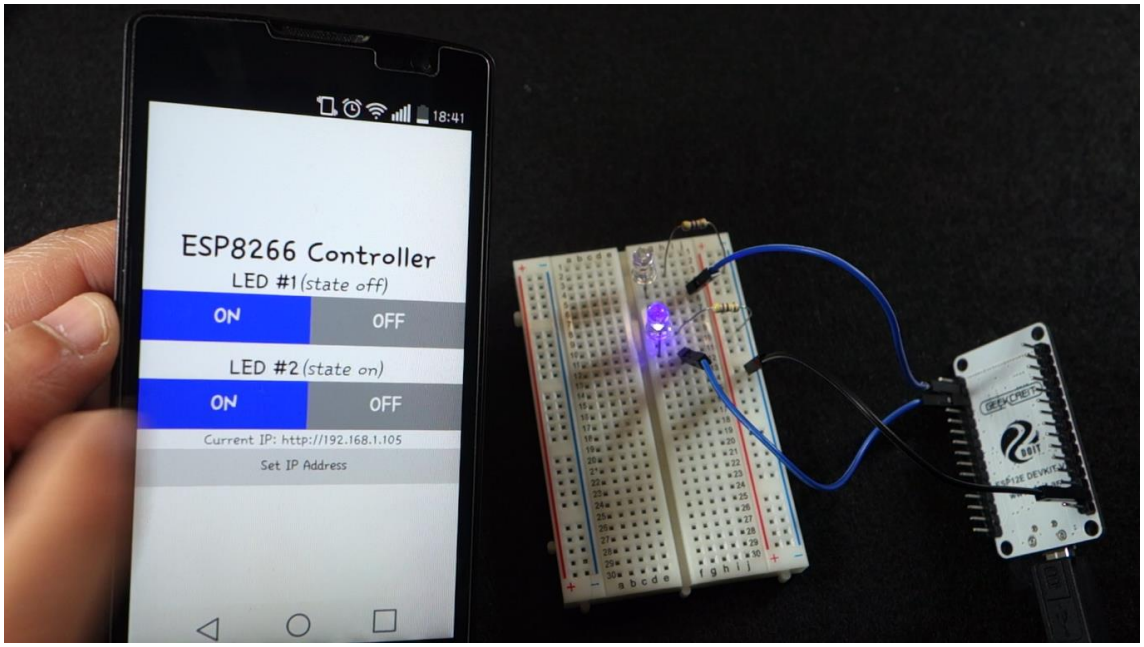
Demonstration

Now you can control your robot using your smartphone.

Have fun!

My dogs just freaked out with this robot.

Project 9: Control ESP8266 Outputs



Project 9: Control ESP8266 Outputs

This is an extra project to show you how to build an Android app to control the ESP8266 outputs. We'll start with an introduction to the ESP8266 WiFi module, followed by a blinking an LED project to get you familiar with the ESP8266. Then, we'll build an Android app to control two LEDS connected to the ESP8266.

Parts Required

Here's a complete list of parts you need to do this project.

Figure	Name	eBay
	1x ESP8266-12E NodeMCU Kit	http://ebay.to/1jU3piA
	2x LEDs	http://ebay.to/20H2Oyy
	Breadboard	http://ebay.to/21bEojM
	2× 470Ohm Resistors	http://ebay.to/1KsMYFP
	Jumper Wires	http://ebay.to/1PXeaJz

Introducing the ESP8266

The ESP8266 is a low cost WiFi module, which is a great platform for any home automation project.

Comparing the ESP with other WiFi modules in the market, this is definitely a great option for most "Internet of Things" projects! It's easy to see why it's so popular: it costs approximately $5 and can be integrated in advanced projects.

So what can you do with this module?

You can create a web server, send HTTP requests, establish an MQTT communication, control outputs, read inputs and interrupts, send emails, post tweets, etc.

Here's a list of tutorials I've created that you might find useful:

- ESP8266 Web Server with Arduino IDE
- DIY WiFi RGB LED Mood Light with ESP8266
- ESP8266 DHT11/DHT22 Temperature and Humidity Web Server
- Posting a Tweet with the ESP8266
- Retrieving Bitcoin Price Using ESP8266 WiFi Module
- Door Status Monitor using the ESP8266

ESP8266 specifications:

- 11 b/g/n protocol
- WiFi Direct (P2P), soft-AP
- Integrated TCP/IP protocol stack
- Built-in low-power 32-bit CPU
- SDIO 2.0, SPI, UART

Finding Your ESP8266

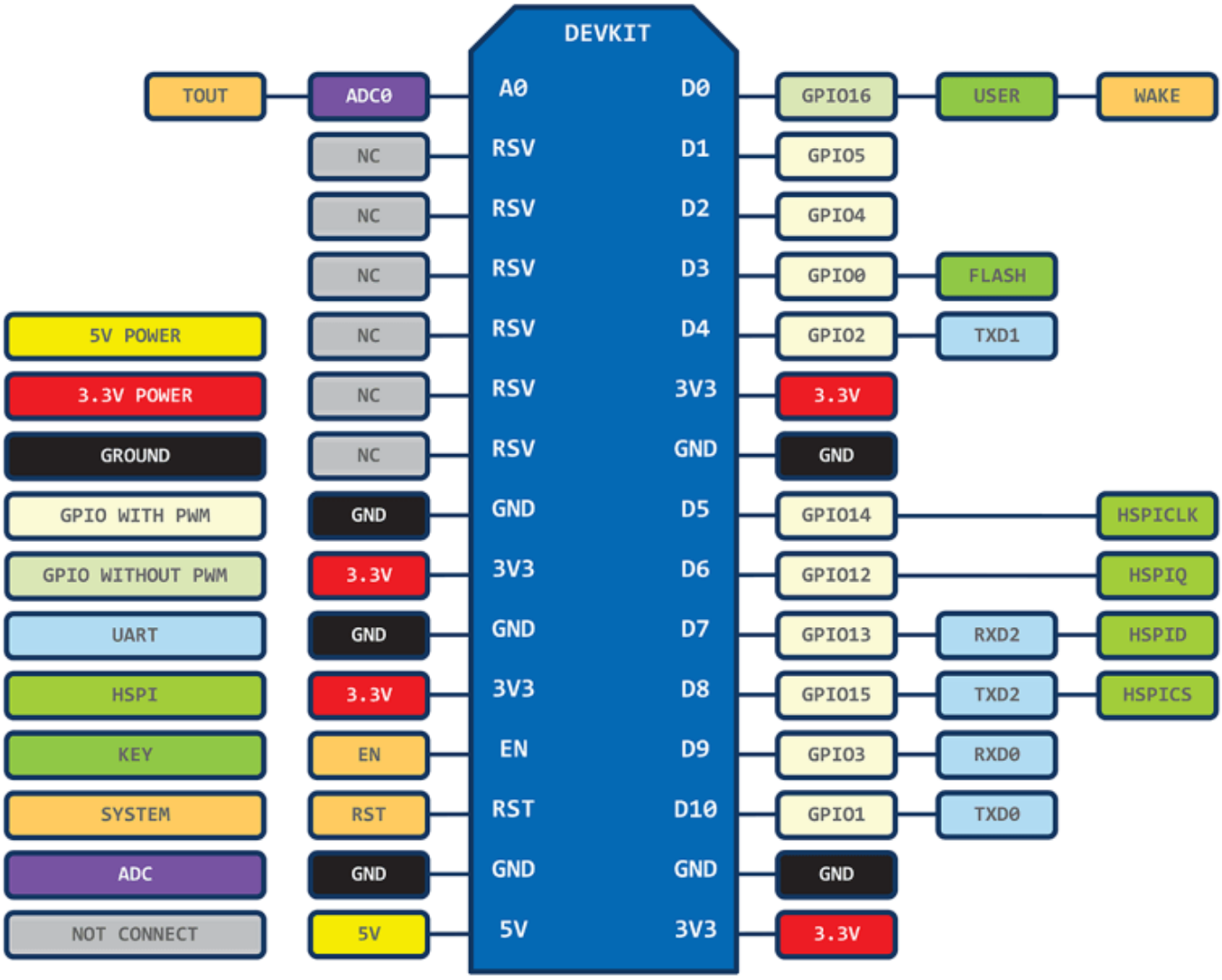
The ESP8266 comes in a wide variety of versions (see the following figure).

Throughout this project we'll be using the ESP-12E NodeMCU kit.

The ESP826-12E that has built-in programmer is the best ESP8266 board at the moment. The built-in programmer makes it easy to prototype and upload your programs.

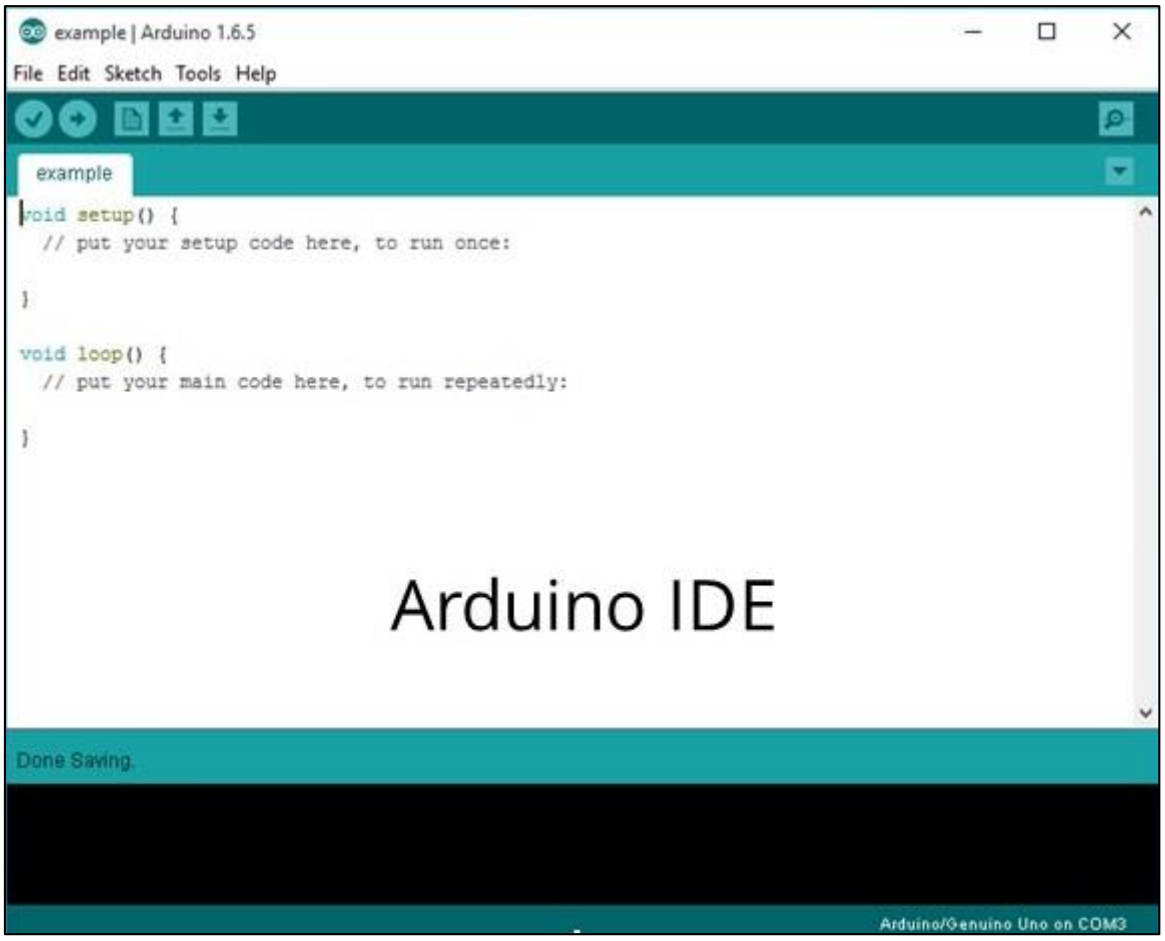
ESP-12E Pinout

Here's a quick overview of the ESP-12E pinout:



Programming the ESP8266 with Arduino IDE

There are a few ways of programming the ESP8266. In this project you'll program the ESP using the Arduino IDE software.



If you are already familiar with the ESP you probably can skip the next two steps, but it's still important that you at least that a look at them to ensure that you don't missing anything.

Install the ESP8266 Board in Arduino IDE

The ESP8266 community created an add-on for the Arduino IDE that allows you to program the ESP8266 using the Arduino IDE and its programming language.

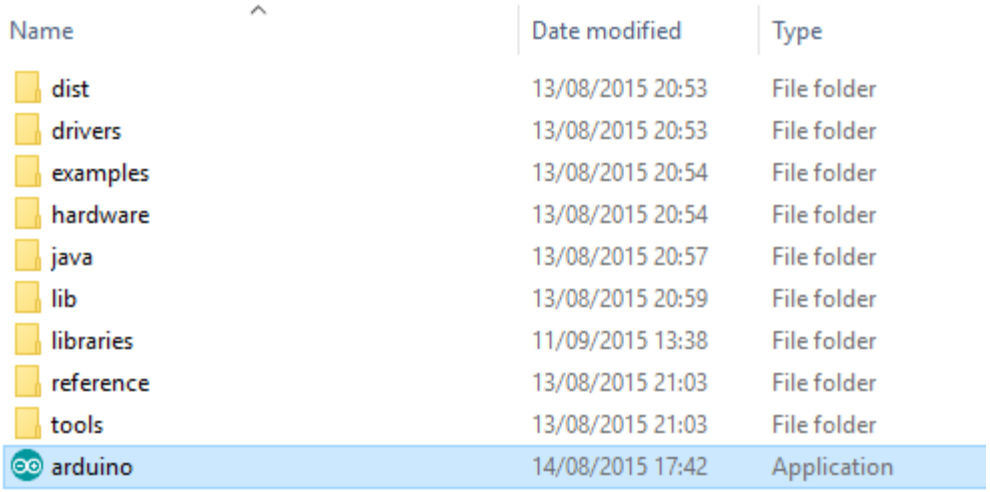
Downloading Arduino IDE

First, download the Arduino IDE to ensure that you have the latest software version (some older versions won't work), visit the following URL: https://www.arduino.cc/en/Main/Software.

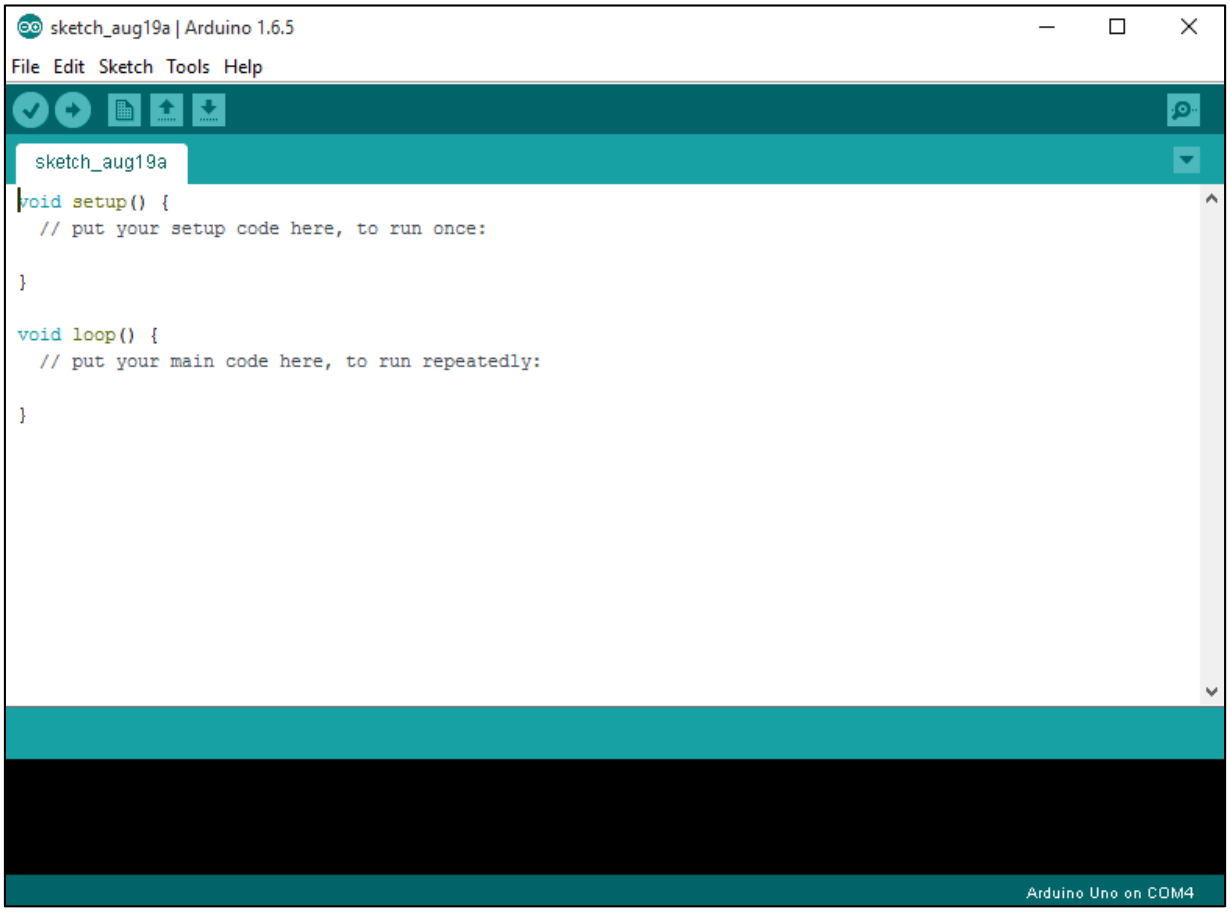
Then, select your operating system and download the latest software release of the Arduino IDE.

Installing Arduino IDE

Grab the file that you have just downloaded and open the Arduino IDE application file (see figure below).



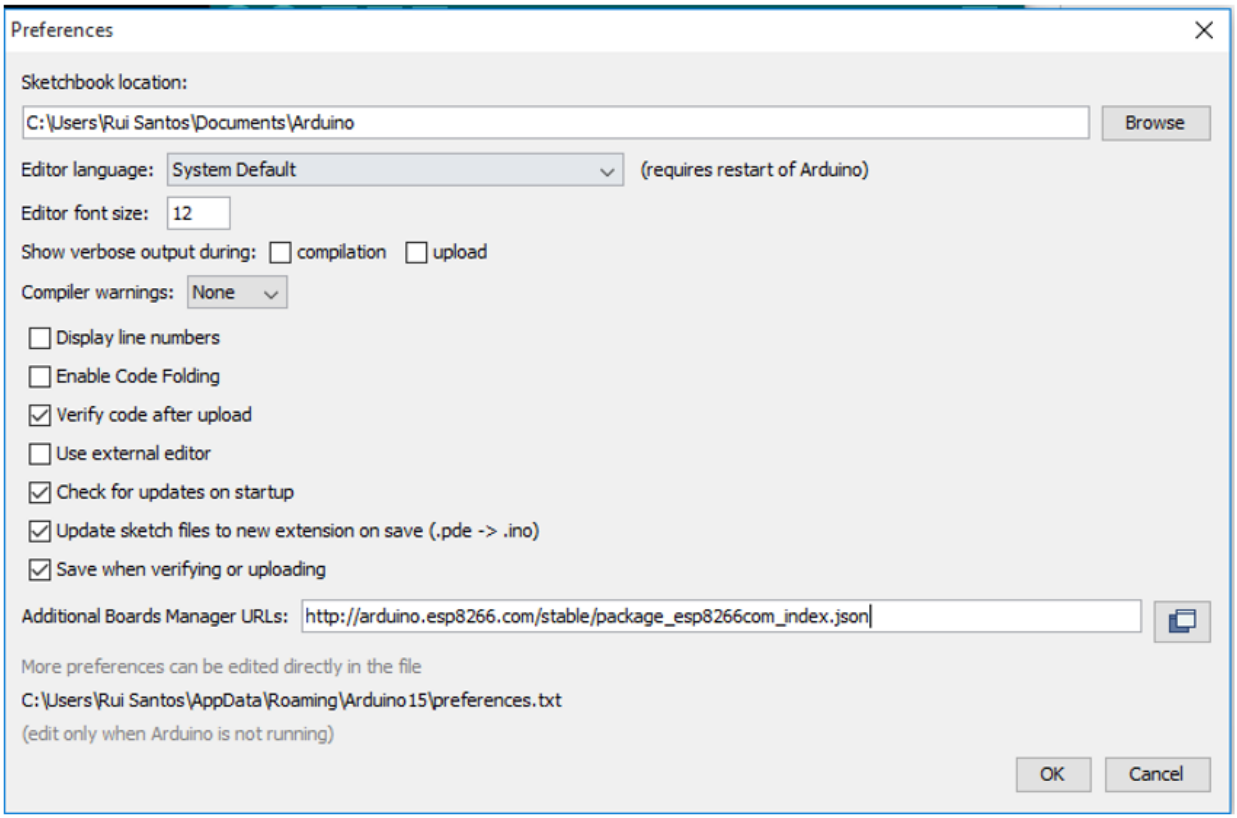
When the Arduino IDE first opens, this is what you should see:



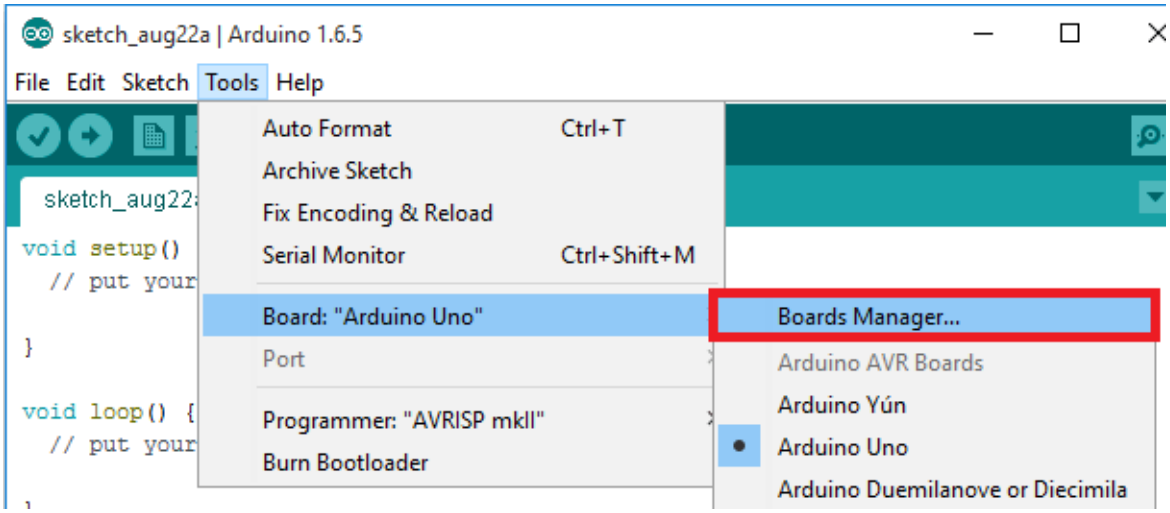
Installing the ESP8266 Board

To install the ESP8266 board in your Arduino IDE, follow these next instructions:

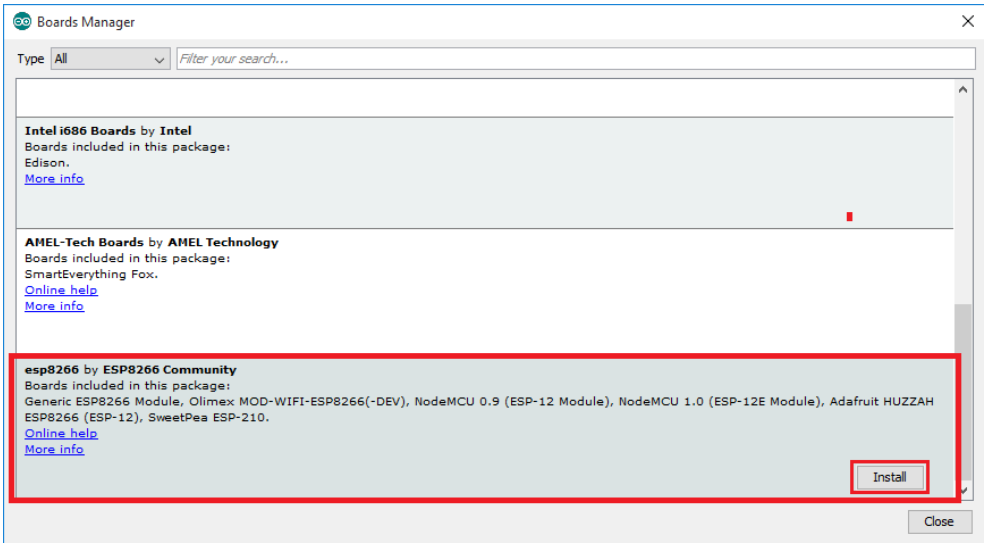
1. Open the preferences window from the Arduino IDE. Go to **File ▸ Preferences**.
2. Enter `http://arduino.esp8266.com/stable/package_esp8266com_index.json` into **Additional Board Manager URLs** field and click the "**OK**" button.



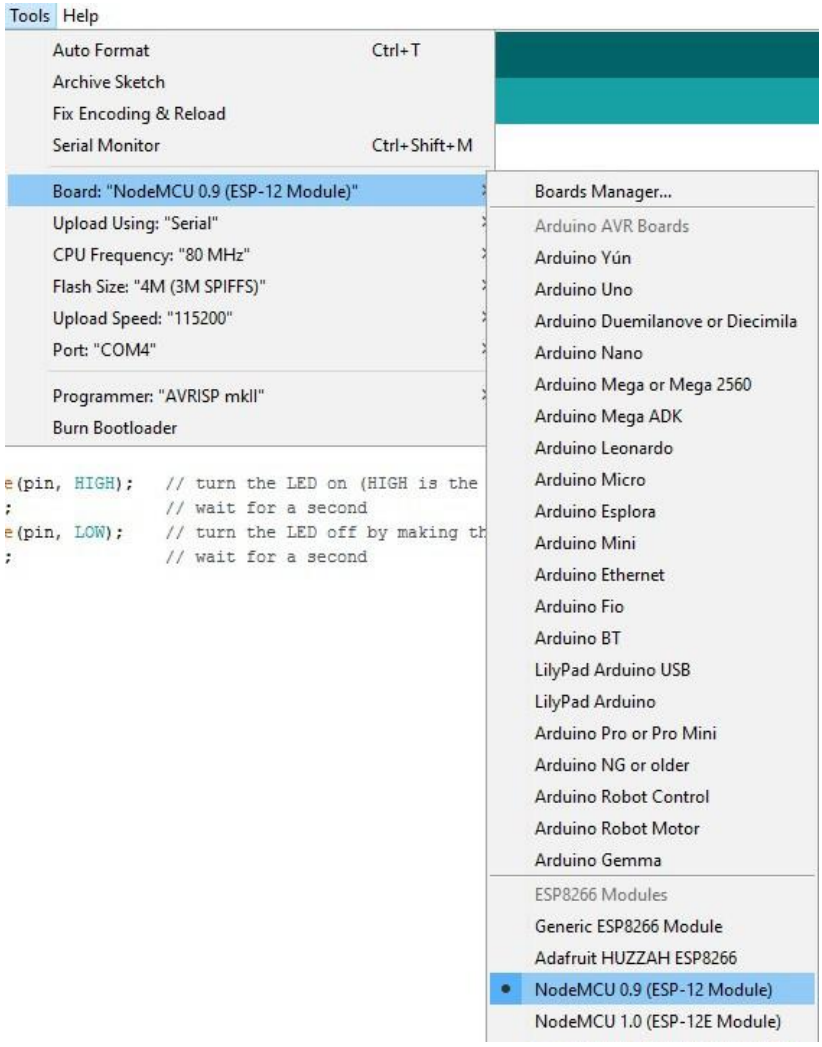
3. Open boards manager. Go to **Tools ▸ Board ▸ Boards Manager...**



4. Scroll down, select the ESP8266 board menu and install “esp8266 platform”.



5. Choose your ESP8266 board from **Tools** ▸ **Board** ▸ **NodeMCU 0.9** (in my case).



6. Finally, close and re-open your Arduino IDE

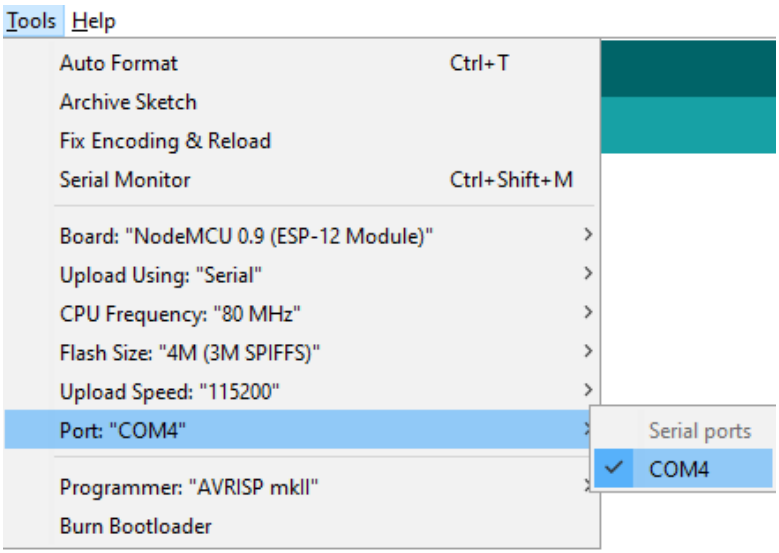
Testing the Installation

To test the ESP8266 add-on installation, let's see if we can blink an LED with the ESP8266 using the Arduino programming language.

Uploading Code

Connect a USB cable from your ESP to your computer to upload code. Having your Arduino IDE open and the right board selected follow these next instructions:

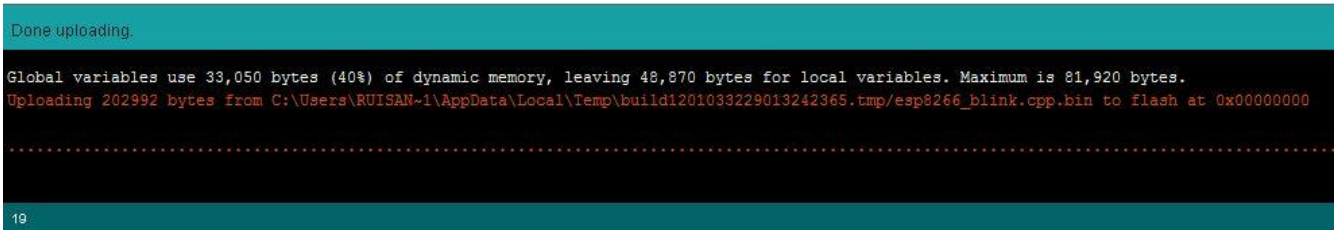
- Select your ESP port number under the **Tools** ▸ **Port** ▸ **COM4** (in my case)



- Copy and paste the sketch below to your Arduino IDE:

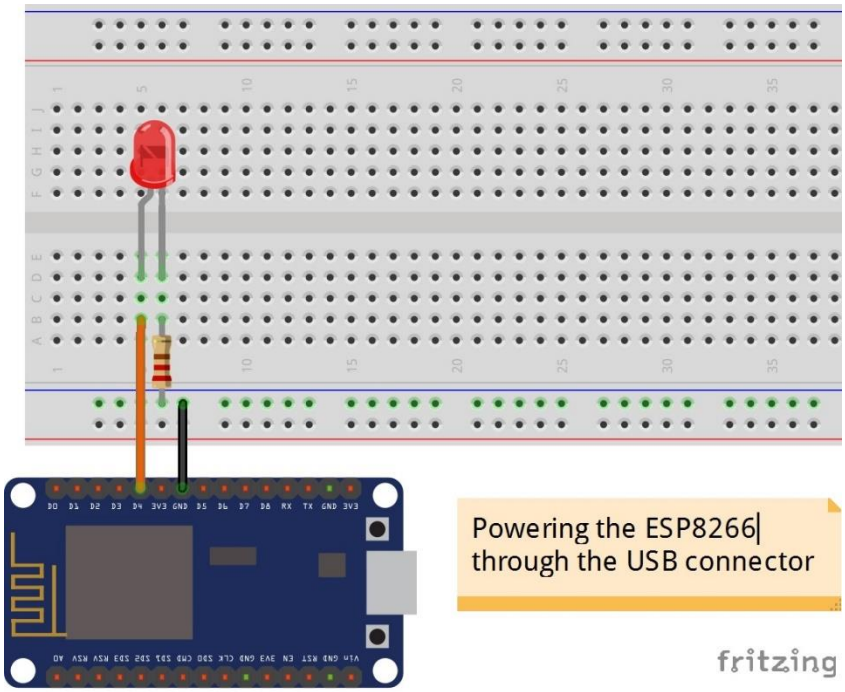
```
/*********
  Rui Santos
  Complete project details at http://randomnerdtutorials.com
*********/

int pin = 2;

void setup() {
  // initialize GPIO 2 as an output.
  pinMode(pin, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(pin, HIGH); // turn the LED on
  delay(1000);             // wait for a second
  digitalWrite(pin, LOW);  // turn the LED off
  delay(1000);              // wait for a second
}
```

- Click the “Upload” button. You should see “Done Uploading” after a few seconds

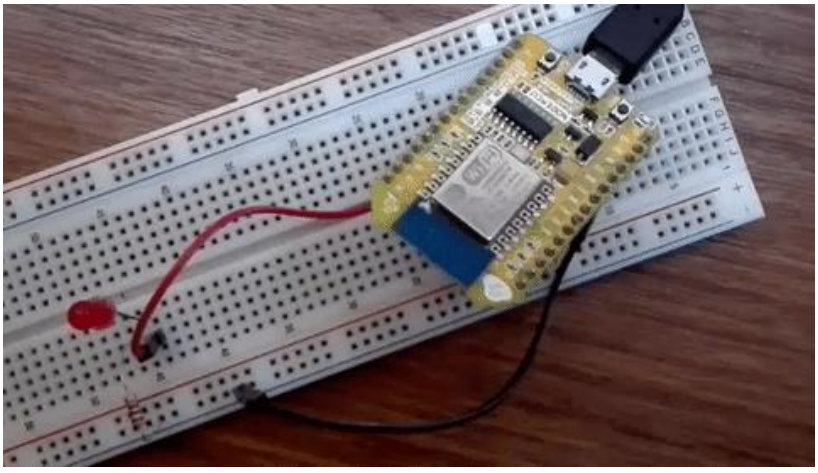


Demonstration

Connect an LED and a 220 Ohm resistor to your ESP D4 (GPIO 2).



Your LED should be blinking every 1 second.

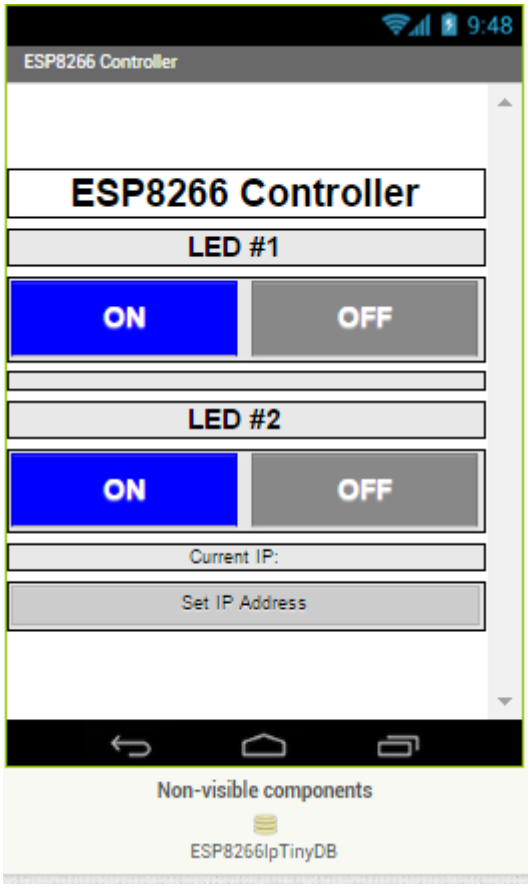


Building the App

Go to http://ai2.appinventor.mit.edu for opening the MIT App Inventor 2 software. Go to **Projects ▸ Start New Project** and call it *Control_ESP8266_Outputs.*

Alternatively, if you prefer, you can upload the *.aia* file and edit the app the way you like. You just have to go to **Projects ▸ Import project (.aia) from my computer** and upload the *Control_ESP8266_Outputs.aia* file that comes with the eBook's resources.

Here's how your app looks in the end of this project:



Here's how the app works:

- You need an internet connection;
- You need to be on the same network that your ESP is;
- You tap the **Set IP Address** button to enter your ESP IP address;
- Tap the **ON** and **OFF** buttons to control your LEDs;
- Your app stores your ESP IP address, so that you don't have to type it every time you use the app.

Design

You should be familiar with the app design for this project. We'll briefly show you how to build this layout:

1. Drag a **HorizontalArrangement** into the **Viewer**. Then, add a **label** inside this arrangement and set its text to **ESP8266 Controller.** Edit the other properties the way you like most.
2. Drag another **HorizontalArrangement** with a **label** inside that contains the text **LED #1.**
3. Drag a third **HorizontalArrangement** and add two **buttons** to this arrangement. Set the text of one button to **ON** and the other to **OFF.** Rename the buttons to **led1OnButton** and **led1OffButton.**
4. Drag a **HorizontalArrangement** with 10pixels height to separate the LED 1 section from the LED 2 section.
5. Repeat steps **2.** and **3.** to build the second LED layout, renaming the components appropriately.
6. Then, drag a **HorizontalArrangement** into the **Viewer**. Add two **labels** to this arrangement. The first label should be renamed to **ESP8266IPLabel** and its text should be set to **Current IP:**. The second label should be renamed to **ESP8266IpInfoLabel**. This label should have no text. We'll set it's text in the blocks section.
7. Drag a **HorizontalArrangement** and name it **saveHorizontalArrangement**. This arrangement is only visible when you tap the **Set IP Address** button. So, set it to not visible.
8. Drag a **TextBox** and a **Button** side by side into the **saveHorizontalArrangement.** Rename the **Button** to **SAVE.**
9. Drag another **HorizontalArrangement** and add a **Button**. Rename them to **setIPHorizontalArrangement** and **setIPButton**. Set its text to **Set IP Address.**
10. In the **Palette** go to **Storage** and drag a **TinyDB** to your screen. This is a non-visible component.
11. Finally, add a **WebViewer** to the app. The web viewer component is in the **Palette** inside the **User Interface** section.

Non-visible components

This app contains two new components: the **TinyDB** and the **WebViewer**.

Apps created with App Inventor are initialized each time they run. If an app sets the value of a variable and the user then quits the app, the value of that variable will not

be remembered the next time the app is run. The **TinyDB** TinyDB is a non-visible component that stores app data. This is useful to save data you want your app to remember. In this case we want our app to remember the ESP IP address.

The other component is the **WebViewer**.

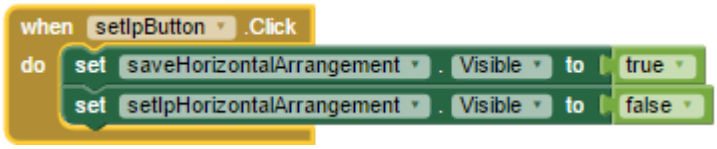
The **WebViewer** is a component for viewing web pages. In this case, we use the **WebViewer** to access the ESP8266 web server in the background to toggle the LEDs on and off when you tap the Android app **ON** and **OFF** buttons.

Blocks Editor

IP address

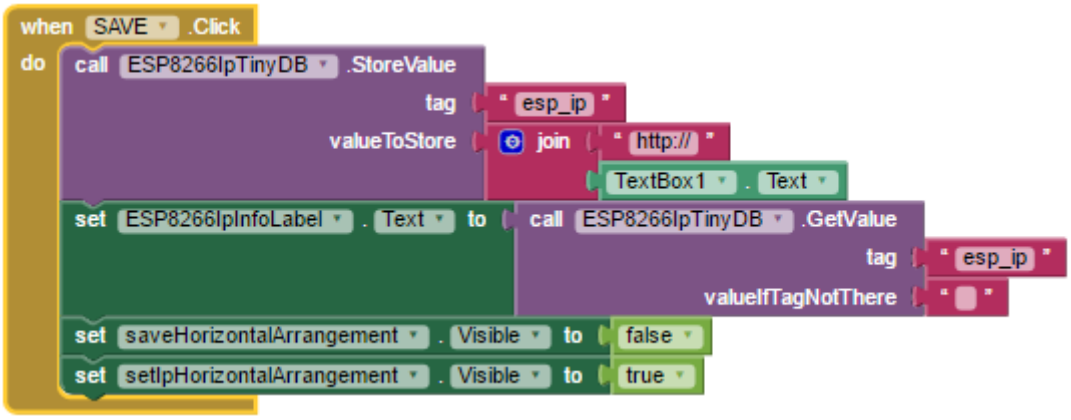
The following blocks set what happens when you tap the **setIPButton**:

- the **saveHorizontalArrangement** is set to visible, so that the textbox shows up and you can enter your ESP IP address;
- the **setIPHorizontalArrangement** disappears by setting its **visibility** to **False.**



Save

After entering the ESP IP address in the text box, you click the **SAVE** button. The following blocks define what happens when you click it.

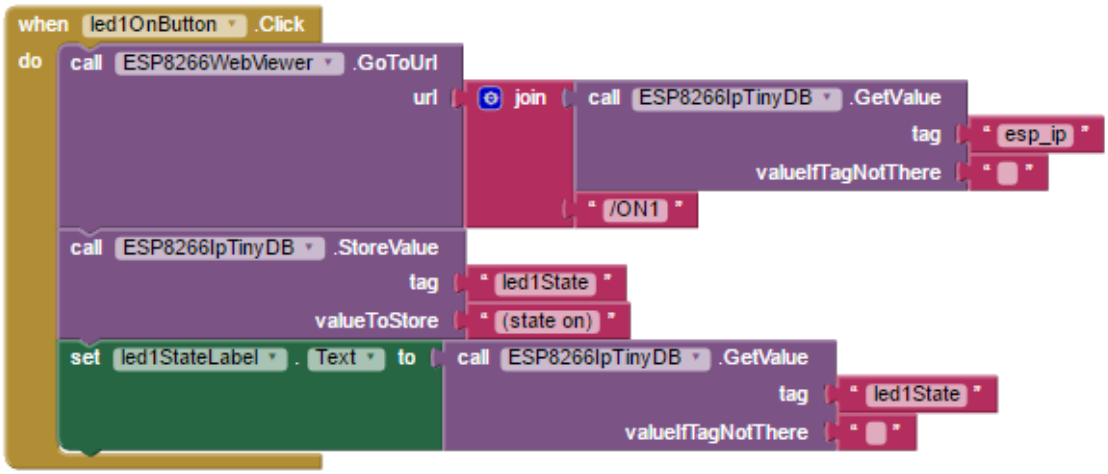


- You store the ESP IP address into your **TinyDB** under the tag name **esp_ip**.
- You set your **ESP8266InfoLabel** to the ESP value saved under the **esp_ip** tag. If you haven't save the ESP IP address yet, your label displays an empty string.

- Your **saveHorizontalArrangement** disappears by setting its visibility to **false.**
- Your **setIPHorizontalArrangement** appears again by setting its visibility to **true.**

Control the LEDs

The following step sets what happens when you click the buttons that control the LEDs. Let's see the **led1OnButton** blocks as an example.



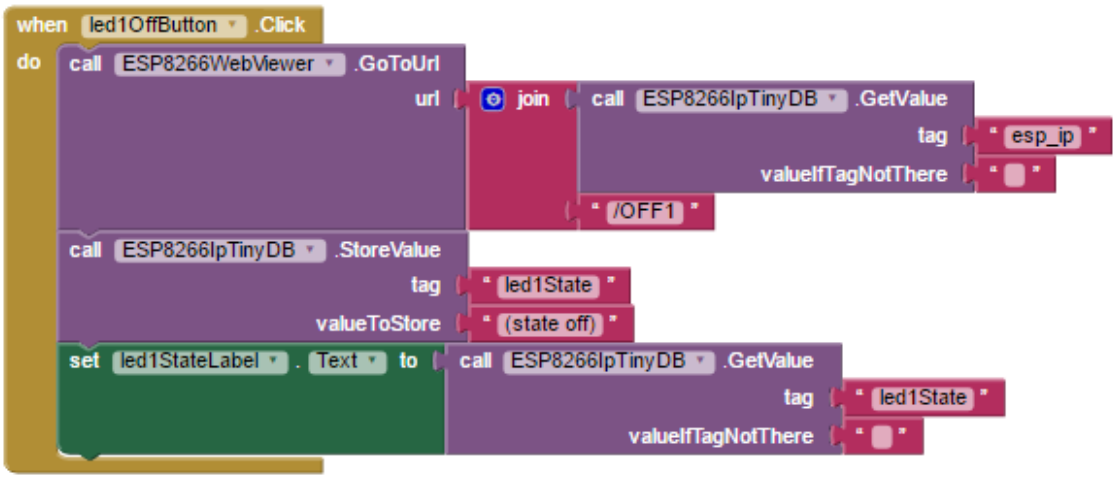
When you tap the **led1OnButton** you tell your **WebViewer** to go to the ESP IP address. The ESP IP address is saved inside your **TinyDB** under the tag name **esp_ip**. You also join the **/ON1** to the ESP IP address. This means that when you tap the **ON** button, the Android app accesses the following URL: http://192.168.1.105/ON1 to turn the LED on (your IP address is likely to be different).

Then, you store the LED state on your **TinyDB** with the tag **led1State**. Since the LED 1 is light up, we save (**state on)** on the **TinyDB.**

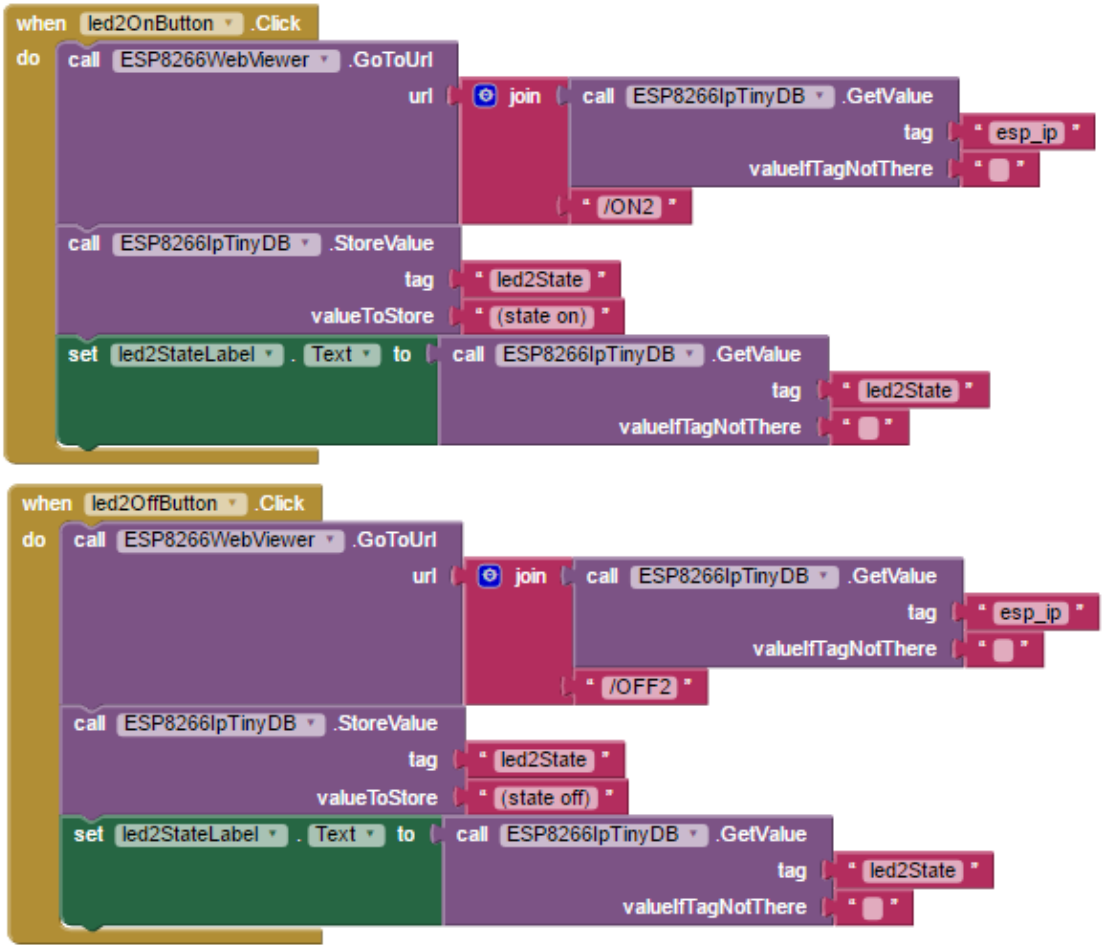
Finally, we set the **led1StateLabel** text to the LED state, which is saved on the **TinyDB** under the tag **led1State**. If nothing is saved on the **TinyDB**, the label should display an empty string.

You should follow this same logic for the **led1OffButton**, **led2OnButton** and **led1OffButton** modifying the labels and text.

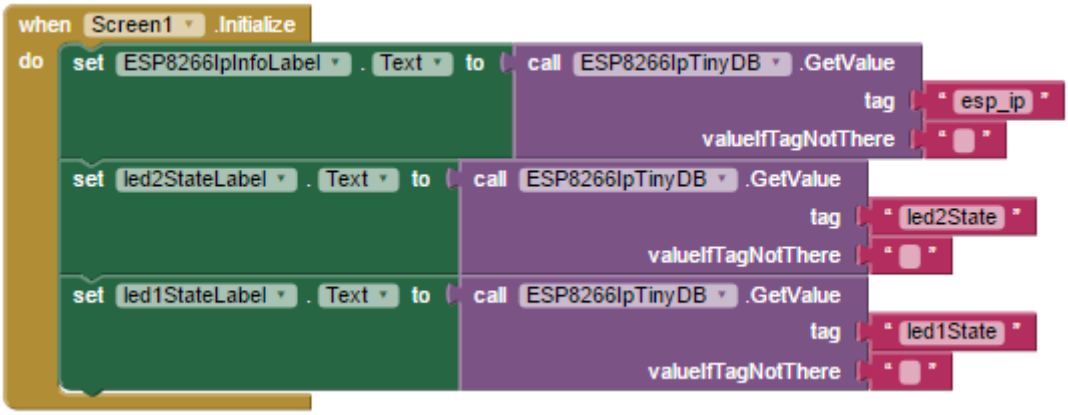
Thus, you'll have the following for the **led1OffButton**:



And for the LED 2, you should have:



Lastly, you display the latest saved ESP IP address and LED states, every time you initialize the app. For that:



These blocks, set the **ESP8266IPInfoLabel**, the **led2StateLabel** and **led1StateLabel** to the string values saved on the **TinyDB.**

To create the Android app, go to the **Build** tab and generate a new *.apk* file. Alternatively, you can use the *.apk* file that comes with the eBook's resources folder. Move the *.apk* file to your Android phone and follow the on-screen instructions to install it.

Writing the Sketch

This is the code you should upload to your ESP. But first, you need to edit the code with your own network credentials.

```
/*********
  Rui Santos
  Complete project details at http://randomnerdtutorials.com
*********/

#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include <ESP8266mDNS.h>

MDNSResponder mdns;

// Replace with your network credentials
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";

ESP8266WebServer server(80);

String webPage = "";

// LED GPIOs
int led1 = 5;
int led2 = 4;

void setup(void){
  // Creates a web page, so that you can control the ESP8266
  // using your web browser
```

```
  webPage += "<h1>ESP8266 Web Server</h1><p>LED #1 <a
href=\"ON1\"><button>ON</button></a> <a
href=\"OFF1\"><button>OFF</button></a></p>";
  webPage += "<p>LED #2 <a href=\"ON2\"><button>ON</button></a> <a
href=\"OFF2\"><button>OFF</button></a></p>";

  // Preparing GPIOs
  pinMode(led1, OUTPUT);
  digitalWrite(led1, LOW);
  pinMode(led2, OUTPUT);
  digitalWrite(led2, LOW);

  // Connects to your router via WiFi
  delay(1000);
  Serial.begin(115200);
  WiFi.begin(ssid, password);
  Serial.println("");

  // Wait for connection
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.print("Connected to ");
  Serial.println(ssid);
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());

  if (mdns.begin("esp8266", WiFi.localIP())) {
    Serial.println("MDNS responder started");
  }

  server.on("/", [](){
    server.send(200, "text/html", webPage);
  });
  // Turns the led1 ON
  server.on("/ON1", [](){
    server.send(200, "text/html", webPage);
    digitalWrite(led1, HIGH);
    delay(1000);
  });
  // Turns the led1 OFF
  server.on("/OFF1", [](){
    server.send(200, "text/html", webPage);
    digitalWrite(led1, LOW);
    delay(1000);
  });
  // Turns the led2 ON
  server.on("/ON2", [](){
    server.send(200, "text/html", webPage);
    digitalWrite(led2, HIGH);
    delay(1000);
  });
  // Turns the led2 OFF
  server.on("/OFF2", [](){
    server.send(200, "text/html", webPage);
    digitalWrite(led2, LOW);
    delay(1000);
  });
  server.begin();
```

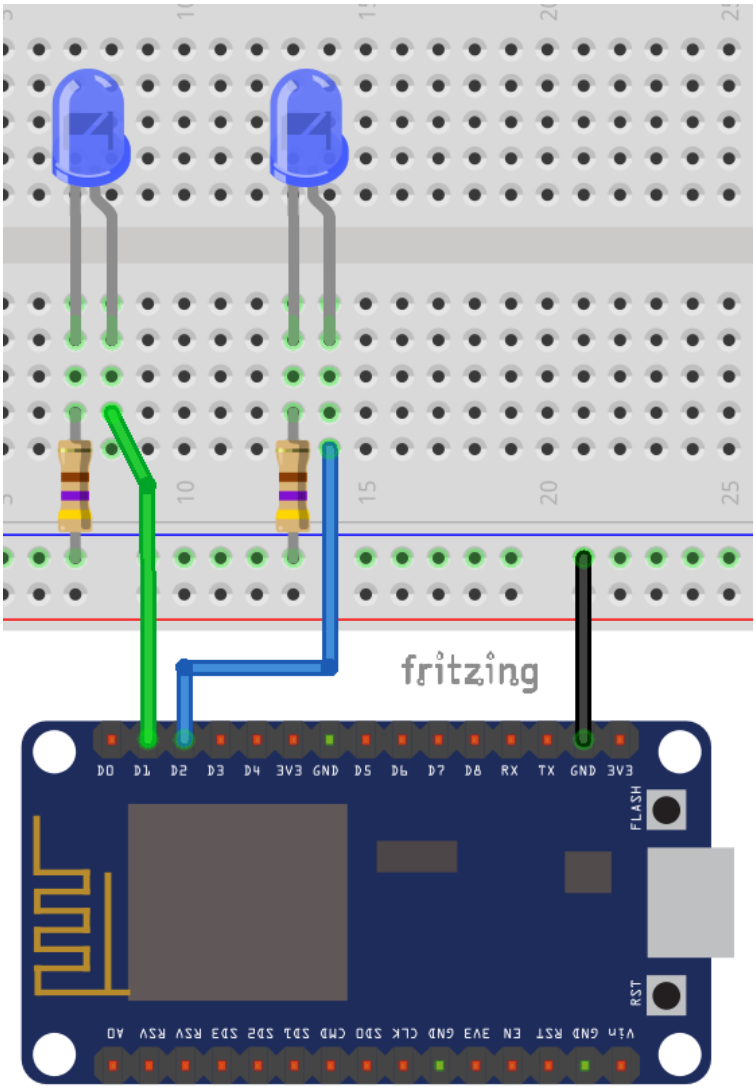
```
  Serial.println("HTTP server started");
}

void loop(void){
  // Handles all the client connections
  server.handleClient();
}
```

Replace these next lines with your own network credentials:

```
// Replace with your network credentials
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";
```

Wiring the Circuit

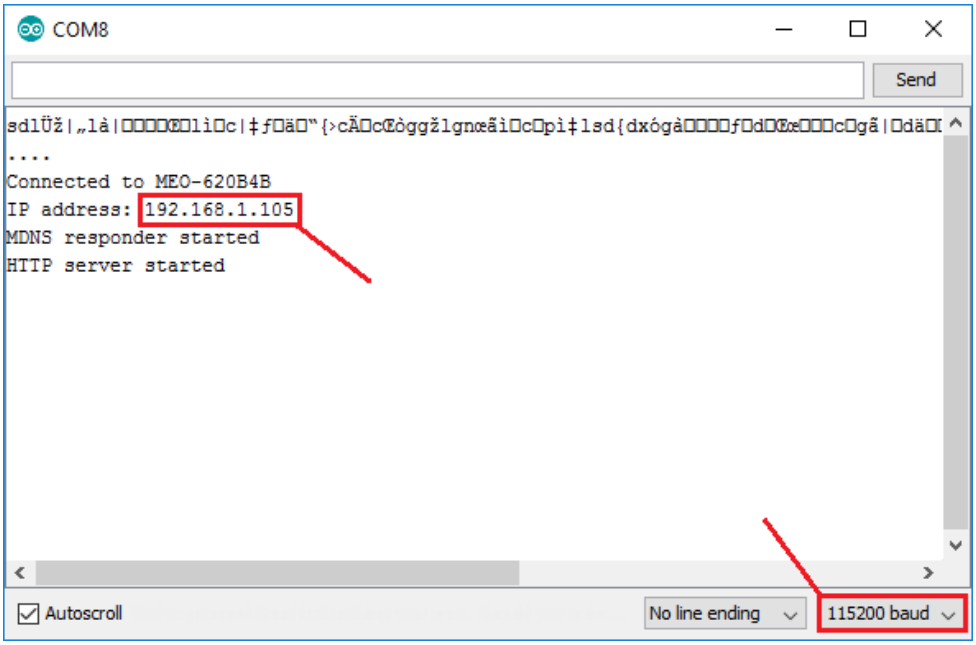
Wire your circuit as the schematic below.



ESP8266 IP Address

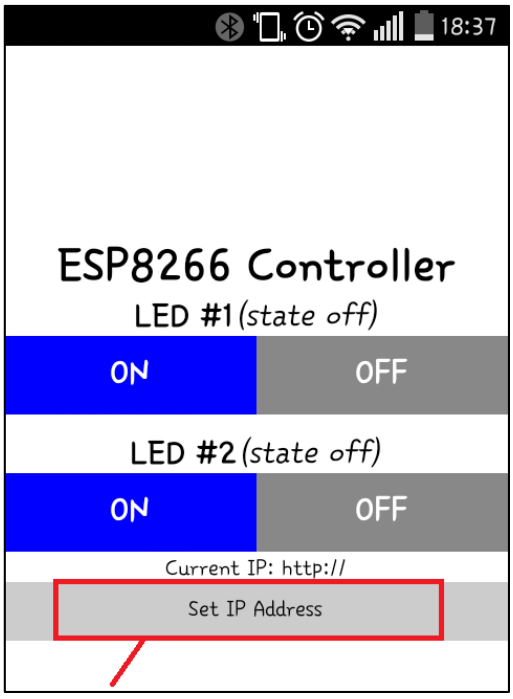
Open the Arduino IDE serial monitor at a baud rate of **115200** and reset your ESP8266 board.

After a few seconds your IP address should appear. In my case it's **192.168.1.105**.

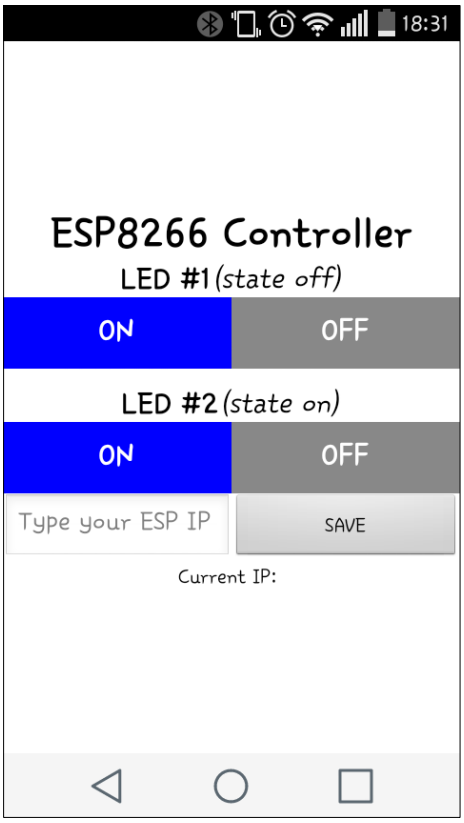


Demonstration

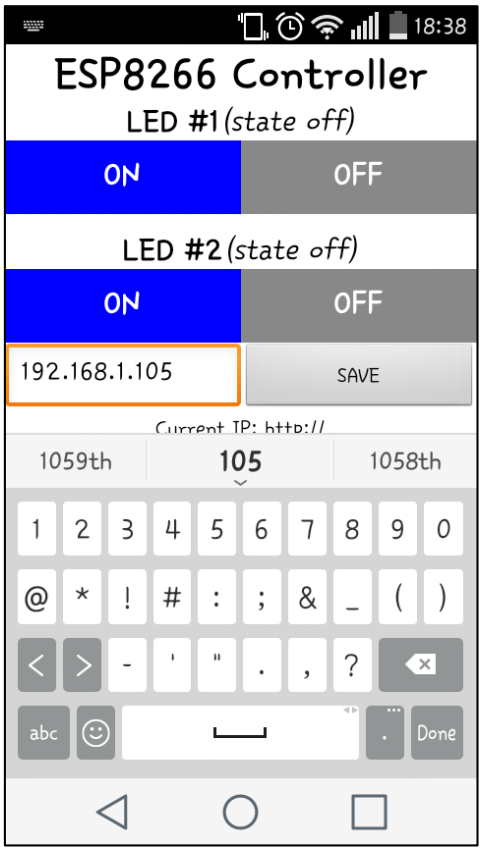
Open the app on your smartphone. Make sure you are on the same network that your ESP8266 is. Tap the **Set IP Address** button.



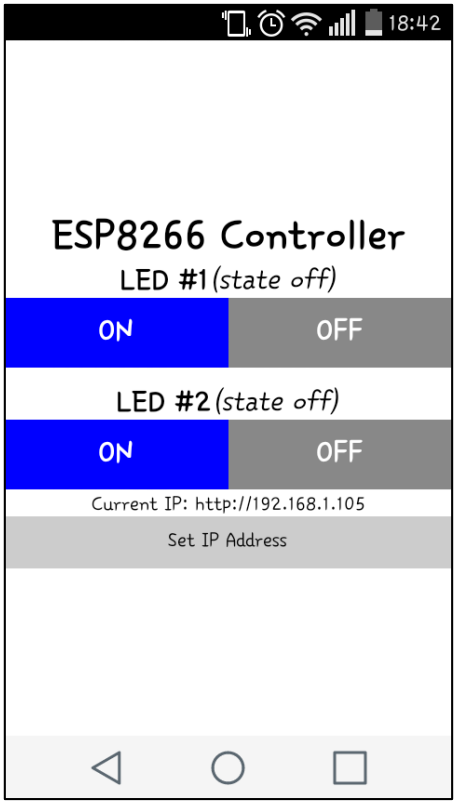
A text box and a **SAVE** button should appear.



Type your ESP IP address – the one you've found in the previous section. Then, press the **SAVE** button.



A label with the current IP shows up.



You can control the ESP8266 GPIOs with an Android app.

Final Thoughts

Congratulations for completing this course!

Here's what you've learned:

- How to use MIT App Inventor 2 and Arduino;
- Design an app with MIT App Inventor 2;
- Use code blocks to make your app do something;
- Establish a bluetooth communication between the Arduino and your Android app;
- Control Arduino pins;
- Read data from the Arduino and send it to the Android app;
- Send and display text messages on an LCD screen;
- Build and control a robot;
- etc...

Now, you can build your own Android applications to interact with the Arduino that can be applied to automate and control all sorts of tasks.

Good luck with your future projects!

Thanks for reading,

Rui and Sara

Download other RNT products

Random Nerd Tutorials is an online resource with electronics projects, tutorials and reviews.

Creating and posting new projects takes a lot of time. At this moment, Random Nerd Tutorials has nearly 100 free blog posts with complete tutorials using open-source hardware that anyone can read, remix and apply to their own projects: http://randomnerdtutorials.com

To keep free tutorials coming, there's also paid content or as I like to call "Premium Content".

To support Random Nerd Tutorials you can download Premium content here. If you enjoyed this eBook make sure you check all the others.

Arduino Step-by-step Projects

My step-by-step course to get you building cool Arduino projects even with no prior experience! This Arduino Course is a compilation of 23 projects divided in 5 Modules that you can build by following clear step-by-step instructions with schematics and downloadable code.

Home Automation Using ESP8266 (2nd Edition)

This eBook is my step-by-step guide designed to help you get started with this amazing $4 WiFi module called ESP8266.

If you're new to the world of ESP8266, this eBook is perfect for you. If you already used the ESP8266 before, I'm sure you'll also learn something new.

This eBook contains the information you need to get up to speed quickly and start your own venture with the ESP8266 applied to Home Automation! Read full product description.

Build a Home Automation System for $100

Learn **Raspberry Pi**, **ESP8266**, **Arduino** and **Node-RED**.

This is a premium step-by-step course to get you building a real world home automation system using open-source hardware and software. Read full product description.