Practical -10

Objective: Create program to develop Android App for the WiFi connection with Node MCU and monitor the sensor data using Node MCU.

Introduction: Nowadays, it has become common practice to use mobile phones as a remote control in IoT applications and for this, there are several development alternatives such as:

- Android Studio
- App Inventor
- Scratch
- Swift (Apple)
- Kotlin
- etc.

What we are going to show here is an alternative solution for developing native mobile applications using HTML5 and known technologies such as:

•Jquery: JavaScript library that facilitates the manipulation of elements and events in WEB pages.

•Onsen UI: Framework of responsive CSS components for developing mobile web applications with technologies such as HTML5, CSS, JavaScript.

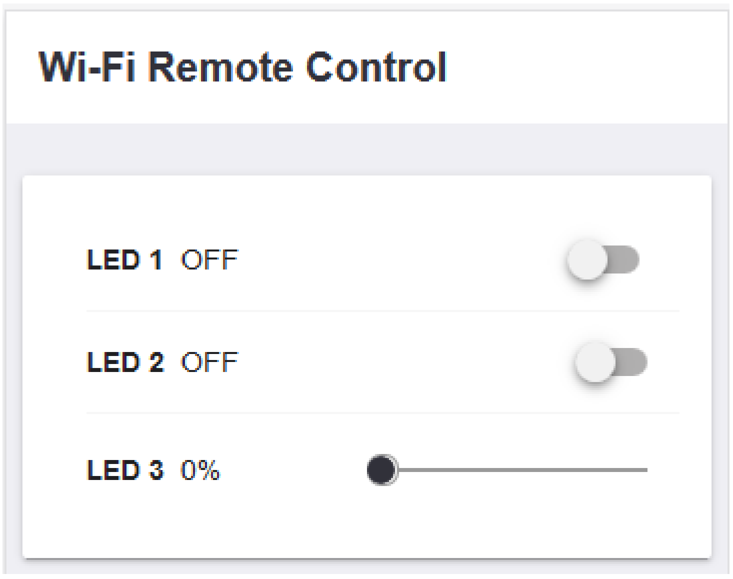
•PhoneGap: Framework for generating hybrid mobile applications from web applications,
Hybrid mobile applications are applications that combine native components and web components.

From the user's point of view, a hybrid application is identical to a native application. However, internally, a hybrid application uses a web view component that contains most of the content and logic of the application.

The Project:

To test the tools described above, we will develop a small project as follows:

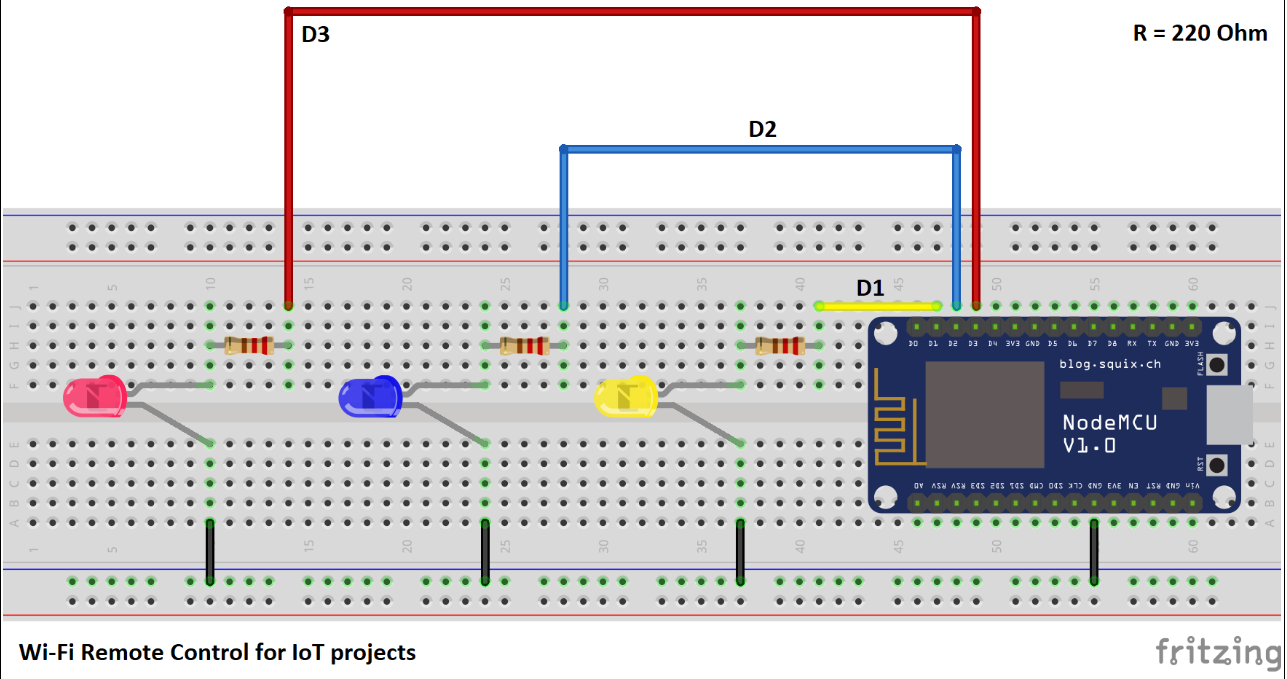
- A NodeMCU board connected to three LEDs
- This board will connect to the wi-fi network and run a web server that will wait for the requests of a client.
- The client will be our remote control app on the mobile phone that will display a screen to control the three LEDs. LED 1 and 2 can be turned on and off with a button. Already, the LED 3, we will control its luminosity through a slider.



Step 1: The Prototype

The following components will be needed to assemble our experiment:

- Protoboard
- NodeMCU or compatible card (Wemos, MKR1000, ESP8266 Standalone)
- Three colored LEDs
- Three 220 Ohms resistors
- Wires jumpers
- The assembly will look like this:



Step 2: NodeMCU Sketch

The function of our sketch will basically be to connect to the Wi-Fi network and create a web server that will be waiting for and responding to requests with LED control commands. Let's use Arduino's own IDE for development. If you have not already done so, you will need to configure the environment with the ESP8266 software. To do this, follow the steps in this article: NodeMCU with the Arduino IDE.

Let's now look at the source code with the comments:

Arduino's programming: Fine name wifi_rc.ino

```
*/
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WiFiMulti.h>
#include <ESP8266mDNS.h>
#include <ESP8266WebServer.h>

ESP8266WiFiMulti wifiMulti;     // For multiple wi-fi configuratiosn
ESP8266WebServer server(80);    // Create a webserver object that listens for HTTP request on port
80

// function prototypes for HTTP handlers
void handleRoot();
void handleRequest();
void handleNotFound();

void setup(void){
  delay(1000);
  pinMode(5, OUTPUT); //Led 1
  pinMode(4, OUTPUT); //Led 2
  digitalWrite(5, 0);
  digitalWrite(4, 0);

  analogWriteRange(100); //Led 3 (PWM)
  analogWrite(0, 0);

  Serial.begin(9600);         // Start the Serial communication to send messages to the computer
  delay(10);
  Serial.println('\n');

  wifiMulti.addAP("ssid1", "password1");   // add Wi-Fi networks you want to connect to
  wifiMulti.addAP("ssid2", "password2");

  Serial.println("Connecting...");
  int i = 0;
  while (wifiMulti.run() != WL_CONNECTED) { // Wait for the Wi-Fi to connect: scan for Wi-Fi
networks, and connect to the strongest of the networks above
    delay(250);
    Serial.print('.');
  }
  Serial.println('\n');
  Serial.print("Connected to ");
  Serial.println(WiF.SSID()); // Tell us what network we're connected to
  Serial.print("IP address:\t");
  Serial.println(WiFi.localIP());         // Send the IP address of the ESP8266 to the computer

  if (MDNS.begin("esp8266")) {    // Start the mDNS responder for esp8266.local
    Serial.println("mDNS responder started");
  } else {
    Serial.println("Error setting up MDNS responder!");
  }
  server.on("/", HTTP_GET, handleRoot);        // Call the 'handleRoot' function when a client
requests URI "/"
  server.on("/command", HTTP_POST, handleRequest); // Call the 'handRequest' function when a
POST request is made to URI "/command"
  server.onNotFound(handleNotFound);           // When a client requests an unknown URI (i.e.
something other than "/"), call function "handleNotFound"

  server.begin();                            // Actually start the server
  Serial.println("HTTP server started");
  return;
}

void loop(void){
  server.handleClient();       // Listen for HTTP requests from clients
  return;
}

void handleRoot() {  // When URI / is requested, send a standard web page
  server.send(200, "text/html", "Wi-fi Remote Control Example");
  return;
}

void handleNotFound(){
  server.send(404, "text/plain", "404: Not found"); // Send HTTP status 404 (Not Found) when
there's no handler for the URI in the request
  return;
}



void handleRequest() { // If a POST request is made to URI /command
  // Validate parameters
  if(!server.hasArg("pin") || !server.hasArg("value") || server.arg("pin") == NULL ||
server.arg("value") == NULL) {
    server.send(400, "text/plain", "400: Invalid Request");         // The request is invalid, so send
HTTP status 400
    return;
  }
```

```
 // Get de parameters: pin and value
 String temp = "";
 temp = server.arg("pin");
 int pin = temp.toInt();
 temp = server.arg("value");
 int value = temp.toInt();

 Serial.println(pin);
 Serial.println(value);
 if (pin >= 0 && pin < 17 && value >= 0 && value <= 100) {
   if (pin == 0) {
     analogWrite(pin, value);
   } else {
     digitalWrite(pin, value);
   }
 }
 server.send(200, "text/html", "Wi-fi Remote Control Example");
 return;
}
```

Points of Interest:

- The `WiFiMulti` lib allows you to configure the authentication of multiple wi-fi networks, making connection easy. Replace `ssid` and `password` according to your network.
- The Lib `mDNS` allows naming a `DNS` for the Esp Local Network. In this case, we're not using this feature because Android does not support it.
- The `handleClient` method starts a loop that acts as a listener for web requests.
- There are three functions that are responsible for dealing with requests coming from the cell phone:
- The `handleNotFound` function is fired when the URI was not found.
- The `handleRoot function` fires when it receives a standard `GET` request. In our case, such respond is despised.
- The `handleRequest` function triggered by the `server.on` condition is responsible for handling the `POST` requests triggered by the remote control in the mobile phone. At this point, we will validate the parameters sent by the remote control and take the appropriate action.

Step 3: WEB Application:

For the development of the remote control application, we will have to download the libraries JQuery and OnsenUI and unpack them in the respective folders, according to the structure described below:

- app
  config.xml← Phonegap Configuration file

        - www
          index.html← Main HTML file
            - assets
                - img ← Here are the App Icons
                  icon-128.png
                  icon-256.png
                - js
                  wifi_rc.js ← Js file with App Logic
                - lib
                    - Jquery ← Unzip Jquery here
                    - Onsenui ← Unzip the Onsenui here

- At this moment, we will worry about the files **index.html** and **wifi_rc.js** responsible for the presentation and programming of our remote control, as below:

index.html

```
<!--
 Wi-fi Remote Control with JQuery and Onsen UI
 Demo by José Cintra
 www.josecintra.com/blog
-->
<!DOCTYPE html>
<html lang="pt-br">
 <head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width,
       initial-scale=1, shrink-to-fit=no">
  <meta name="Author" content="José Cintra"
  <link rel="stylesheet" href="assets/lib/OnsenUI/css/onsenui.min.css">
  <link rel="stylesheet" href="assets/lib/OnsenUI/css/onsen-css-components.min.css">
 </head>
 <body>

 <ons-page modifier="material">
  <ons-toolbar modifier="material">
   <div class="center"><strong>Wi-Fi Remote Control</strong></div>
  </ons-toolbar>
  <br/>
  <ons-card modifier="material">
    <div class="content">
      <ons-list modifier="material">
        <ons-list-item modifier="material">
          <div class="center" >
            <strong>LED 1 &nbsp;</strong> <label id="led1v">OFF</label>
          </div>
          <div class="right">
            <ons-switch id="led1" class="sw" modifier="material"></ons-switch>&nbsp;
          </div>
        </ons-list-item >
        <ons-list-item modifier="material">
          <div class="center" >
            <strong>LED 2 &nbsp;</strong> <label id="led2v">OFF</label>
          </div>
          <div class="right">
            <ons-switch id="led2" class="sw" modifier="material"></ons-switch>
          </div>
        </ons-list-item>
        <ons-list-item modifier="material">
          <div class="center">
            <strong>LED 3 &nbsp;</strong> <label id="led3v">0</label>%
          </div>
          <div class="right">
            <ons-range id="led3" class="rg" modifier="material"
             style="width: 100%;" value="0" max = "100" ></ons-range>
          </div>
        </ons-list-item>
      </ons-list>
    </div>
  </ons-card>

</ons-page>
<!-- Javascript -->
<script src="assets/lib/OnsenUI/js/onsenui.min.js"></script>
<script src="assets/lib/jquery/jquery-3.3.1.min.js"></script>
```

```
  <script src="assets/js/wifi_rc.js"></script>

</body>
</html>
```

wifi_rc.js

```
$(function () {

  // Server address and pin numbers of the board (ESP8266/32 and compatibles)
  let addr = "http://192.168.0.33/command";
  let pins = new Map([
    [ '#led1', '05' ],
    [ '#led2', '04' ],
    [ '#led3', '00' ], ]);

  // Click Event on switch Class
  $('.sw').on('click', function (e) {
    let onoff = ['OFF','ON'];
    let id = "#" + $(this).attr("id");          // Get the id of the control
    let pin = pins.get(id);                      // Pin number
    let value = String(+$(id).prop('checked')); // On or Off
    $(id + 'v').html(onoff[value]);
    sendAjax(addr, pin, value);
  });




  // Input event on range class
  $('.rg').on('input', function (e) {
    let id = "#" + $(this).attr("id");          // Get the id of the control
    let value = String($(id).val());            // Input range
    $(id + 'v').html(value);                    // Notification
  });

$('.rg').on('change', function (e) {
    let id = "#" + $(this).attr("id");          // Get the id of the control
    let pin = pins.get(id);                     // Pin number
    let value = String($(id).val());            // Input range
    sendAjax(addr, pin, value);
  });
});

function sendAjax(addr, p, v) {
  $.ajax({
    method: "POST",
    url: addr,
    data: {pin: p, value: v}
  });
}
```

Points of Interest:

1.In the index.html file is the whole presentation part of the application with CSS elements made available by the OnsenUI framework.

2.LEDs 1 and 2 will be controlled by a switch button and LED 3 by a `range` control. It is important to note the `id` and `class` attributes that will be used in the JS script.

3.The JavaScript code of the **wifi_rc.js** file adopts the ES6 standard.

4.The **wifi_rc.js** file highlights the `function()` event that occurs only once after page loading. This is where we program the actions that respond to the events of the controls.

5.To represent the pin numbers in the `NodeMCU`, we use a `MAP` structure. This information will be passed, along with the values of the controls, to the `sendAjax` method and sent to the web server of the board.

Step 4: Mobile Application

Finally, at this point, we will generate our mobile application to be installed on the mobile phone. In order for our HTML5 application to be installed as a native app, you will need to use the Phonegap build tools. The simplest way to do this is to use the cloud compilation services of Phonegap Build.

The steps are as follows:

- Create an account in the Phonegap Build, service, choosing a paid or free plan.
- Create the application icons that will be displayed on your phone.
- Create a **config.xml** file with the required settings.

- Compress all files and upload to compilation.
- Request compilation of the application.
- Choose the desired platform (Android, iPhone or Microsoft) and download the compiled app for this platform.

Note: The icon files must be in `png` format, in sizes 128 × 128 and 256 × 256.
The following is an example config file:

config.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<widget id="com.josecintra.wifi_rc" version="1.0.0"
 xmlns="http://www.w3.org/ns/widgets" xmlns:cdv="http://cordova.apache.org/ns/1.0">
   <name>Wi-fi Remote Control</name>
   <description>Example of building a wi-fi remote control with
    HTML5/JQuery/Onsen UI and Phonegap. It will be used to control IoT devices,
    such as NodeMCU and compatibles</description>
   <author href="http://www.josecintra.com/blog"
    email="josecintra@josecintra.com">Jose Cintra</author>
   <icon src="www/assets/img/icon-256.png" width="256"
        height="256" density="xxxhdpi" />
   <icon src="www/assets/img/icon-128.png" width="128"
        height="128" density="xhdpi" />



   <preference name="android-targetSdkVersion" value="26" />
   <preference name="orientation" value="portrait" />
   <preference name="fullscreen" value="true" />
   <preference name="DisallowOverscroll" value="true" />
   <config-file parent="UIStatusBarHidden" platform="ios"
            target="*-Info.plist"><true/></config-file>
   <config-file parent="UIViewControllerBasedStatusBarAppearance"
               platform="ios" target="*-Info.plist"><false/></config-file>
   <preference name="deployment-target" value="10.0" />
   <preference name="android-minSdkVersion" value="21" />
   <access origin="*" />
   <plugin name="cordova-custom-config" />
   <plugin name="cordova-plugin-file" />
   <plugin name="cordova-plugin-media" />
   <plugin name="cordova-plugin-statusbar" />
   <plugin name="cordova-plugin-whitelist" />
   <engine name="ios" />
   <engine name="android" />
</widget>
```

Points of Interest

The **config.xml** file contains all the information needed to generate the app. Among them, we highlight:

- **name:** Name of the app
- **description:** App description
- **author:** Info about the author of the app
- **icon:** Name of the icons that will identify the app on the mobile phone

- **Orientation**: The app can be displayed in `portrait`, `landscape` mode. If this parameter are not informed, the app will adapt to the inclination of the cell phone.
- **access origin:** Defines which urls the application can access

Apart from these, there are dozens of other settings. An important point is the choice of plugins that allow you to add additional functionality to the application, such as access to the camera, gps, and other native features of the phone. For more details on the build process, see this link.

Step 5: Installing the App: In the previous step, we've generated an app that can be installed on your phone or made available for download on app stores. In our case, we generated an Android app with the apk extension. To test this app on mobile without going through the app store, we need to configure Android to allow the installation of "Unknown sources" apps. Usually, this option is in the security section.

After that, just copy the app to your phone. This can be done in various ways such as USB, Kies, Wi-Fi or via an Internet link. Android itself will take care of the installation.