Universidad de Valladolid

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

Control remoto de caldera mediante Arduino Yun y aplicación Android.

Autor:

Gallego Rico, Carlos

Tutor:

Plaza Pérez, Francisco
Departamento de tecnología Electrónica

Valladolid, Junio de 2017

RESUMEN

Se pretende realizar un programa en un módulo Arduino YUN, para que controle una caldera de Gas utilizada para la calefacción en una vivienda.

También se desarrollará una aplicación en Android que permita enviar/recibir información de configuración del funcionamiento de dicha caldera.

Palabras clave: Arduino, Android, Caldera, Control, Calefacción.

ÍNDICE GENERAL

Contenido

RESUMEN 3
ÍNDICE GENERAL 5
INDICE DE ILUSTRACIONES 7
INDICE DE TABLAS 9
GLOSARIO DE TÉRMINOS Y ABREVIATURAS 11
PRESENTACIÓN 13
ESTADO ACTUAL DE LA TECNOLOGÍA 15
1 Termostato DUCASA CONTROL 3G WIFI CALDERA 15
2 Termostato Inteligente - Kit de Inicio (v3) Tado 16
3 Junkers Easy Control CT 100 17
Conclusiones 18
OBJETIVO: 19
DESCRIPCIÓN DEL PROYECTO: 21
ELECCIÓN DEL HARDWARE 23
Elección del microprocesador 23
Elección del relé 26
Elección del sensor de temperatura. 27
CAPÍTULO 1: ARDUINO 29
1.1 Arduino YUN 30
1.2 Objetivos 33
1.3 Programa Arduino 35
CAPÍTULO 2: ANDROID 43
2.1 Arquitectura Android 44
2.2 Entorno de desarrollo 46
2.3 Función y diseño 47

CAPÍTULO 3: COMUNICACIÓN 63
3.1 Configuración Router 65
3.2 Configuración DNS 67
3.3 Protocolo 69
CAPÍTULO 4: MONTAJE FÍSICO 71
CAPÍTULO 5: OBJETIVOS Y FUTURAS MEJORAS 77
5.1 Objetivos 77
5.2 Futuras mejoras 79
ANEXO 1 INFORMACION CALDERAS. 81
Tipos de calefacción 81
Instalación caldera 87
ANEXO 2 ARDUINO 89
2.1 Instalación del IDE 89
2.2 Entorno de desarrollo 91
2.3 Configuración del IDE 92
ANEXO 3 ANDROID 93
3.1 Instalación de Java. 93
3.2 Instalación de Android Studio y el SDK de Android. 97
3.3 Conocimientos básicos Android 102
3.4 Creación primer proyecto. 104
3.5 Entorno de desarrollo 108
BIBLIOGRAFÍA 112
Publicaciones escritas 112
Publicaciones Web 112

INDICE DE ILUSTRACIONES

Ilustración 1. Termostato DUCASA. 15
Ilustración 2. Termostato inteligente Tado 16
Ilustración 3. Controladora Junkers. 17
Ilustración 4. Esquema general proyecto. 21
Ilustración 5. Rapsberry vs Arduino Yun 23
Ilustración 6. Relé doble 26
Ilustración 7. Sensor DS18B20 27
Ilustración 8. Arduino Yun 30
Ilustración 9. Esquema Arduino Yun. 31
Ilustración 10. Esquema general funcionamiento Arduino 33
Ilustración 11. Función Obtaindate. 35
Ilustración 12. Control temperatura 36
Ilustración 13. Obtener temperatura 36
Ilustración 14. Aceptar clientes 37
Ilustración 15. Selección de modo 37
Ilustración 16. Función leer temperatura 38
Ilustración 17. Función modo Manual 39
Ilustración 18. Función actualiza matriz base 39
Ilustración 19. Función modo programa 40
Ilustración 20. Función modo edición 41
Ilustración 21. Arquitectura Android 44
Ilustración 22. Pantalla principal app 48
Ilustración 23. Código pantalla principal 49
Ilustración 24. Tarea asíncrona 50
Ilustración 25. Modo manual 51
Ilustración 26. Código modo manual 52
Ilustración 27. Modo automático. 53
Ilustración 28. Código modo automático 1 54
Ilustración 29. Código modo automático 2 55
Ilustración 30. Selección temperaturas 56
Ilustración 31. Código selección temperaturas 57
Ilustración 32. Modo edición. 58
Ilustración 33. Código pestañas modo edición 59
Ilustración 34.Código Checkbox modo edición 60
Ilustración 35. Código formar comandos 61

Ilustración 36. Configuración router 1. 65
Ilustración 37. Configuración router 2 66
Ilustración 38. Configuración router 3, redirección de puertos 66
Ilustración 39. Caja integradora 71
Ilustración 40. Conexión Sensor de Temperatura y Arduino 72
Ilustración 41. Conexión sensor-relé-Arduino 73
Ilustración 42. Conexión sensores 74
Ilustración 43. Montaje final 74
Ilustración 44. Resultado final 75
Ilustración 45. Consumos energéticos 85
Ilustración 46. Distribución energía sector doméstico 85
Ilustración 47. Instalación hidráulica caldera. 87
Ilustración 48. Caja de conexiones caldera. 88
Ilustración 49. Descarga Arduino 89
Ilustración 50. Arduino instalación completada 90
Ilustración 51. IDE Arduino. 91
Ilustración 52. Descarga Java 93
Ilustración 53. Instalación de java 1 94
Ilustración 54. Instalación de java 2 94
Ilustración 55. Instalación de java 3 95
Ilustración 56. Instalación de java 4 95
Ilustración 57. Crear variable de entorno 96
Ilustración 58. Descarga Android Studio 97
Ilustración 59. Instalación Android Studio 1 98
Ilustración 60. Instalación Android Studio 2 98
Ilustración 61. Instalación Android Studio 3 99
Ilustración 62. Instalación Completada Android Studio 99
Ilustración 63. Configuración Android Studio 1 100
Ilustración 64. Configuración Android Studio 2 101
Ilustración 65. Inicio proyecto Android Studio. 104
Ilustración 66. Nuevo proyecto Android Studio 105
Ilustración 67. Selección API Android Studio 106
Ilustración 68. Selección activity Android Studio 107
Ilustración 69. Nombrar activity Android Studio 107
Ilustración 70. Entorno de desarrollo Android Studio 108
Ilustración 71. Barra de herramientas Android Studio 108
Ilustración 72. Entorno de programación Android Studio 109
Ilustración 73.Ficheros y carpetas Android Studio 111

INDICE DE TABLAS

Tabla 1. Microcontrolador Arduino. 31
Tabla 2. Microprocesador Arduino. 32

GLOSARIO DE TÉRMINOS Y ABREVIATURAS

ADT	*Android Develompent Tools*
API	*Application Programming Interface*
APP	*Aplicación informática diseñada para ser ejecutada en teléfonos inteligentes, tabletas y otros dispositivos móviles*
AVD	*Android Virtual Device*
BAUD	*Unidad de medida de transmisión.*
EPROM	*Erasable Programmable Read-Only Memory*
ETHERNET	*Estándar de redes de área local para computadores con acceso al medio por detección de la onda portadora y con detección de colisiones (CSMA/CD).*
HTML	*Hyper Text Markup Languaje*
IOT	*Internet Of Things*
IP	*Internet Protocol*
KERNEL	*Núcleo del Sistema Operativo.*
HTTP	*HyperText Transfer Protocol*
IDE	*Integrated Development Environment*
JDK	*Java Development Kit*
JVM	*Java Virtual Machine*
NAT	*Network Address Translation*
PHP	*Hypertext PreProcessor*

REST	*Representational State Transfer*
SD	*Secure Digital*
SDK	*Software Development Kit*
SO	*Sistema Operativo*
TCP	*Transmision Control Protocol*
URL	*Universal Resource Locator*
WIFI	*Mecanismo de conexión de dispositivos electrónicos de forma inalámbrica.*
WLAN	*Wide Local Area Network*
XML	*Extensible Markup Language*

PRESENTACIÓN

Actualmente el mundo de la tecnología está sufriendo un espectacular desarrollo en todos sus ámbitos.

Uno de los campos que permiten un mayor margen de mejora y que más afectan en nuestro modo de vida es la domótica.

La domótica consiste en la automatización de la vivienda, integrando la tecnología en todos los ámbitos posibles.

Con este proyecto se incide especialmente en el control remoto de la caldera, pero asienta las bases para integrar en el futuro, de manera sencilla todos los aspectos que se requieran, como puede ser el manejo de persianas, sistemas de seguridad, gestión de electrodomésticos, etc.

Basándonos en la versatilidad del SO Android y las capacidades de Arduino Yun, vamos a integrar el control remoto de una caldera.

ESTADO ACTUAL DE LA TECNOLOGÍA

A continuación vamos a realizar una comparativa de los diferentes productos similares que hay en el mercado:

1 Termostato DUCASA CONTROL 3G WIFI CALDERA

Ilustración 1. Termostato DUCASA.

199.99€

Termostato digital que permite apagar y encender tu caldera o estufa por internet mediante PC, tablet o smartphone.

Además te brinda la posibilidad de establecer una programación semanal y controla las horas de funcionamiento de la caldera. Ideal para sustituir antiguos termostatos de 2 hilos con muchas más funcionalidades.

2 Termostato Inteligente - Kit de Inicio (v3) Tado

Ilustración 2. Termostato inteligente Tado.

249.00€

El termostato inteligente Tado, presenta control multizona y un nuevo algoritmo de calefacción, tado° que nos permite ahorrar aún más en la factura energética y hacer tu vida más cómoda.

Con una interfaz mejorada en el propio termostato y la nueva Programación Inteligente en la app hacen el control de tu calefacción aún más fácil.

3 Junkers Easy Control CT 100

Ilustración 3. Controladora Junkers.

275.00 €

Este dispositivo de Junkers es un controlador que nos permite controlar la instalación de calefacción desde cualquier dispositivo móvil con conexión a internet.

A través de la tecnología Wifi y descargándonos la app en nuestro dispositivo móvil, permite poder programar y gestionar su sistema de calefacción remotamente.

Con dos modos manual y programación podemos gestionar ajustar el funcionamiento a nuestras necesidades.

Conclusiones

Entre los termostatos inteligentes hemos seleccionado estos tres debido a que son de marcas con cierto reconocimiento en el mundo de la domótica, descartando productos de menor calidad.

Podemos observar que el primer termostato, de la marca Ducasa permite realizar un control completo de la caldera de la vivienda. La principal pega que le podemos encontrar es la no utilidad para viviendas de nueva construcción, que en su mayoría constan de instalaciones bitubulares, contando con dos zonas (anillos) de calefacción con una misma caldera.

Con este tipo de instalaciones se puede conseguir un gran control en las temperaturas, permitiendo diferenciar por completo ambas zonas, logrando un nivel de confort y ahorro energético óptimo.

El segundo y tercer termostatos, de la marca Tado y Junkers respectivamente, si permiten este control multizona, contando con su correspondiente app para realizar el control remoto y permitiendo los modos manual y programación.

Esta mejora se ve en el gran incremento de precio 250 y 275 euros respectivamente, es un precio muy elevado que lo hace inaccesible para una gran cantidad de familias.

OBJETIVO:

Nuestro objetivo es conseguir un termostato inteligente, con capacidades completas, control multizona y control remoto mediante una app, a un precio reducido.

A su vez debe integrar varios modos, como el control manual y permitir la programación personalizada, siendo el más completo del mercado.

El proyecto es bastante extenso debido a que comprende todas las fases, desde la implementación hardware hasta el desarrollo software, tanto del microprocesador como de la aplicación Android.

A su vez el objetivo es dejar preparada la base para en un futuro, de manera sencilla y con un coste mínimo poder implementar todos los aspectos de la domótica de la vivienda.

DESCRIPCIÓN DEL PROYECTO:

A continuación vamos a realizar una descripción general del proyecto.

Dentro del mismo se pueden diferenciar dos partes claramente.

En primer lugar tenemos el microprocesador, conectado al mismo se encuentra el relé encargado de controlar las electroválvulas correspondientes a los circuitos, así como los sensores de temperatura. Este es el elemento central, que se encuentra situado en la vivienda y al cual van conectados todos los sensores que se quieran implementar.

Este elemento consta de parte hardware (el propio dispositivo, sensores, relé y alimentación) y de parte software (la programación del mismo)

En segundo lugar tenemos la aplicación Android para manejar el dispositivo.

Esta parte solo consta de Software ya que el hardware es nuestro propio terminal móvil.

Finalmente lo más complejo es la integración de ambas plataformas para realizar la comunicación entre ellas a través de internet.

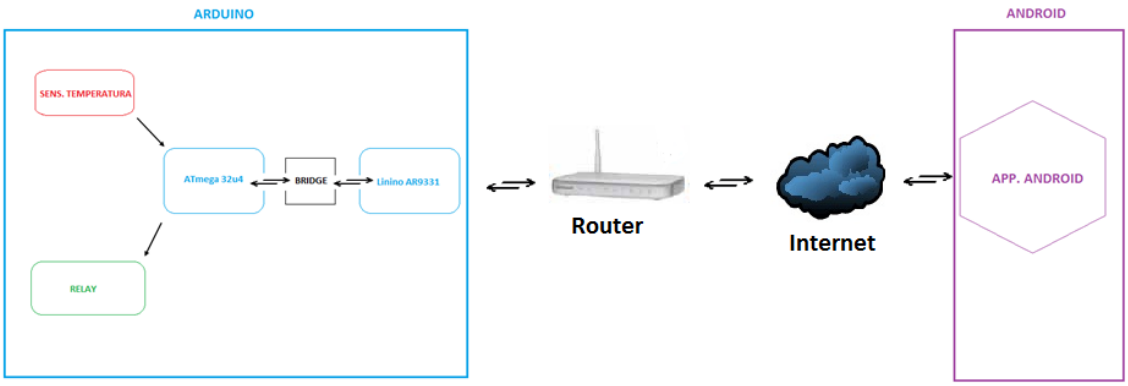


Ilustración 4. Esquema general proyecto.

ELECCIÓN DEL HARDWARE

Elección del microprocesador

A la hora de elegir el microprocesador nos encontramos con dos alternativas. Por un lado existe la posibilidad de utilizar una Raspberry 3, por otro lado existe la opción de utilizar la plataforma Arduino, a su vez dentro de Arduino existen varios modelos con diferentes características.

Ilustración 5. Rapsberry vs Arduino Yun

Raspberry:

La raspberry pi es un computador de de placa única, desarrollado por la Fundación Raspberry pi para facilitar la enseñanza de informática en el entorno educativo.

Se puede decir que una Raspberry es un ordenador completo de tamaño reducido. Con un potente procesador ARM de 700Mhz y 512 Mb de Ram, con este dispositivo disponemos de un gran potencial para realizar cualquier proyecto que se desee. Destacando especialmente por su conectividad y adaptación a los protocolos Ethernet, así como su flexibilidad para configurar una amplia gama de Software, sin embargo es más complejo de utilizar en el lado Hardware, por ejemplo la no inclusión de entradas analógicas, lo que dificulta en gran medida la utilización de este tipo de sensores, teniendo que utilizar hardware adicional para realizar la conversión analógico digital.

Arduino:

Arduino es una plataforma electrónica de código abierto basada en hardware y software fáciles de usar.

Históricamente Arduino ha estado enfocado al ámbito electrónico, dejando un poco al margen el área informática.

Con un lenguaje de programación propio, basado en C++ y enfocado al hardware, ofrece un entorno perfecto para el desarrollo de proyectos interactivos.

El punto fuerte de Arduino está en la facilidad de conectarse con el mundo, gracias a las entradas tanto analógicas como digitales con las que cuenta y de lo fácil que resulta activar o desactivar una de las entradas/salidas gracias a su software, sin embargo presentaba un déficit en lo que a la comunicación mediante Ethernet se refería, así como el manejo de bases de datos.

Esto se ha solucionado con la llegada de la nueva placa Arduino YUN, que combina conectividad Wi-Fi con el sistema operativo Linux, lo que ofrece infinidad de posibilidades.

Arduino Yun integra en la misma placa dos microprocesadores, el procesador ATmega32u4 que es el corazón del arduino y un Atheros AR9331 que maneja la distribución Linux. Por un lado contamos con toda la conectividad y capacidad de desarrollo de un Arduino y por el otro con una máquina Linux, lo que hace que podamos ejecutar comandos, aplicaciones o scripts en el lado Linux mientras que utilizamos su conectividad Ethernet y Wi-Fi. También cuenta con una ranura para poner una tarjeta micro-SD.

Para facilitar la programación web, YÚN incluye una librería puente que conecta las transacciones HTTP entre los dos lados (Linux y microcontrolador). Además soporta Temboo, un kit de desarrollo que permite la interacción con Facebook, Dropbox, FedEx y muchísimos servicios web más, lo que hace que YÚN sea ideal para el Internet de las Cosas y su integración con otros dispositivos o servicios en internet.

Decisión

Tras analizar en profundidad ambas opciones finalmente nos decidimos por el nuevo prototipo de Arduino, que con el Yun ha logrado un dispositivo potente y funcional, que combina la facilidad de arduino con las capacidades de Linux.

Pensamos que quizás Arduino Yun es más apropiado para los proyectos relacionados con el hardware de IoT mientras que Raspberry Pi es más potente y flexible, lo que permite configurar una amplia gama de software (incluyendo un sistema operativo Linux completo como Raspbian o Fedora), pero de hecho, más complejo de usar, especialmente en el lado Hardware.

Una vez decidido el elemento principal de nuestro proyecto, debemos realizar un estudio sobre los diferentes tipos de calderas existentes y su control.

Este pequeño resumen se desarrolla en el Anexo1.

Elección del relé

Tras conocer el funcionamiento de las calderas y el modo de controlarlas, nos decidimos por utilizar un relé especial para Arduino, que manejándolo con los 5V que proporciona la placa, puede controlar los 220V que necesitamos para el manejo de los circuitos de calefacción.

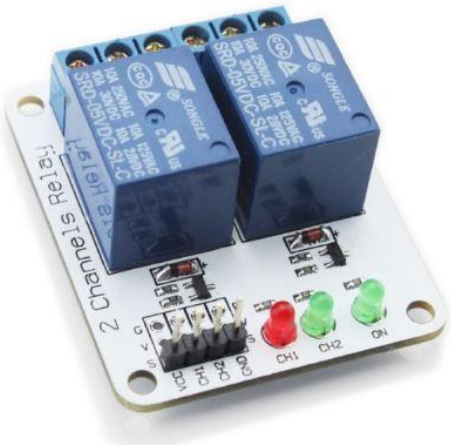


Ilustración 6. Relé doble

Con este dispositivo podemos realizar un adecuado manejo de señales de corriente alterna de alto voltaje, con un Arduino que funciona en corriente continua.

Elección del sensor de temperatura.

A la hora de realizar la medición de temperaturas existen diferentes alternativas que pueden ser adecuadas para nuestro proyecto.

Hemos optado por utilizar el sensor de temperatura DS18B20.

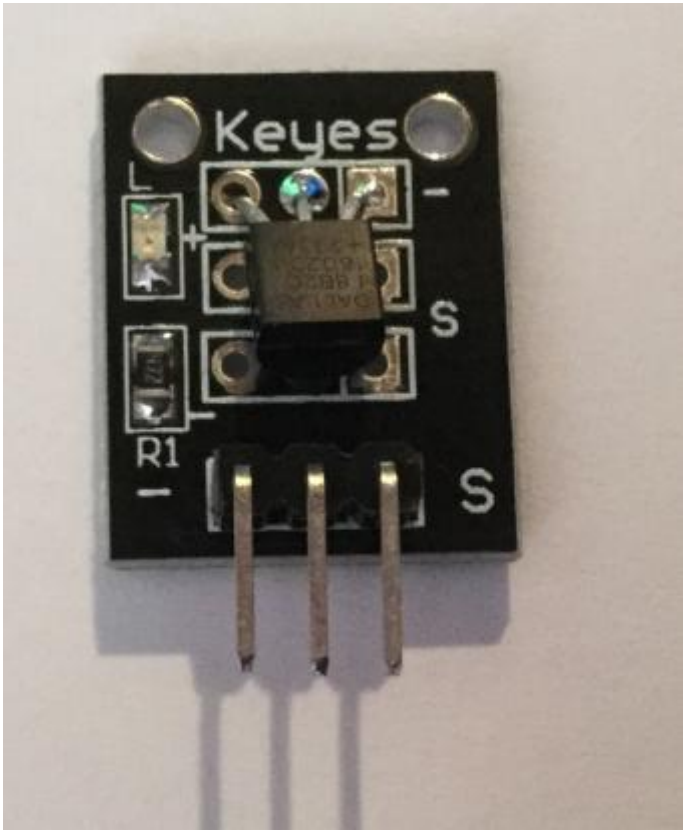


Ilustración 7. Sensor DS18B20

El sensor DS18B20 es un sensor barato y, sin embargo, bastante avanzado. Dispone de un rango amplio de medición de -55ºC a +125ºC y una precisión superior a ±0.5°C en el rango –10°C de +85°C, lo que es más que suficiente para nuestro proyecto.

Es un dispositivo que se comunica de forma digital. Cuenta con tres terminales, los dos de alimentación y el pin “data”.

El DS18B20 emplea un bus de comunicación denominado 1-Wire propietario de la empresa Maxim Integrated, aunque podemos usarlo sin tener que pagar por ninguna tasa (es parte del precio del dispositivo).

La principal ventaja del bus 1-Wire es que necesita un único conductor para realizar la comunicación (sin contar el conductor de tierra). Los dispositivos pueden ser alimentados directamente por la línea de datos, o mediante una línea adicional con una tensión de 3.0 a 5.5V.

Dentro del mismo bus 1-Wire podemos instalar tantos sensores como deseemos. Además, el bus 1-Wire permite emplear cables más largos que otros sistemas antes de que se deteriore la comunicación.

El DS18B20 también dispone de un sistema de alarma que permite grabar en la memoria no volátil del DS18B20 los límites inferiores y superiores. El bus 1-Wire permite consultar si algún dispositivo conectado ha activado una alarma.

Por todo esto nos decidimos por este sensor ya que además de ser muy robusto, tener gran precisión y un bajo precio, nos permite instalar tantos sensores como queramos en la misma red, pudiendo acceder a la información de cada uno de ellos desde el Arduino fácilmente.

En nuestro proyecto únicamente utilizaremos dos, uno para cada zona, pero si se quisiera aumentar el confort podría añadirse de forma sencilla tantos como fuera necesario.

Una vez decididos los elementos que vamos a utilizar pasamos a explicar el proyecto en sí, basándonos en la división realizada anteriormente entre la parte de Arduino, instalada en la vivienda y la parte de Android, instalada en nuestros dispositivos móviles.

CAPÍTULO 1 ARDUINO

Como hemos visto en el estudio preliminar, Arduino es una plataforma de realización de prototipos electrónicos compuesta de una placa (Hardware) y de un software para programar sus acciones.

A continuación vamos a hacer una explicación en detalle del dispositivo elegido (Arduino Yun)

1.1 Arduino YUN

Arduino Yun es uno de los modelos fabricados por Arduino y que utilizaremos este proyecto para realizar el control de los periféricos, así como establecer la conexión con Android.

Esta placa, como característica principal, combina dos procesadores.

El primero de ellos, el Atmel ATmega 32U4, que se encarga de ejecutar las aplicaciones de Arduino. El segundo, el Atheros AR9331, compatible con el servidor Linux, se encarga de la gestión de redes (Ethernet, WiFi), el USB Host (pendrive, teclados, ratones) y la microSD (almacenamiento de datos).

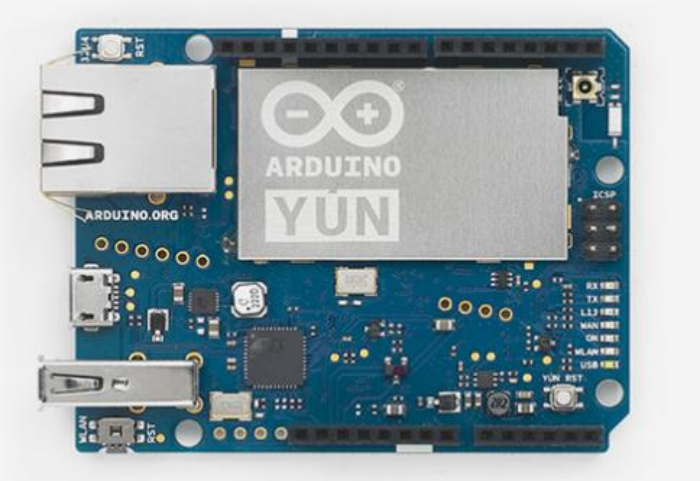


Ilustración 8. Arduino Yun

La placa incorpora soporte Ethernet y WiFi, puerto USB-A, ranura para tarjetas micro-SD, 20 pines de entrada / salida digitales (7 de ellos pueden utilizarse como salidas PWM y 12 entradas analógicas), oscilador de cristal de 16 MHz, Conexión micro USB, un encabezado ICSP y 3 botones de reinicio.

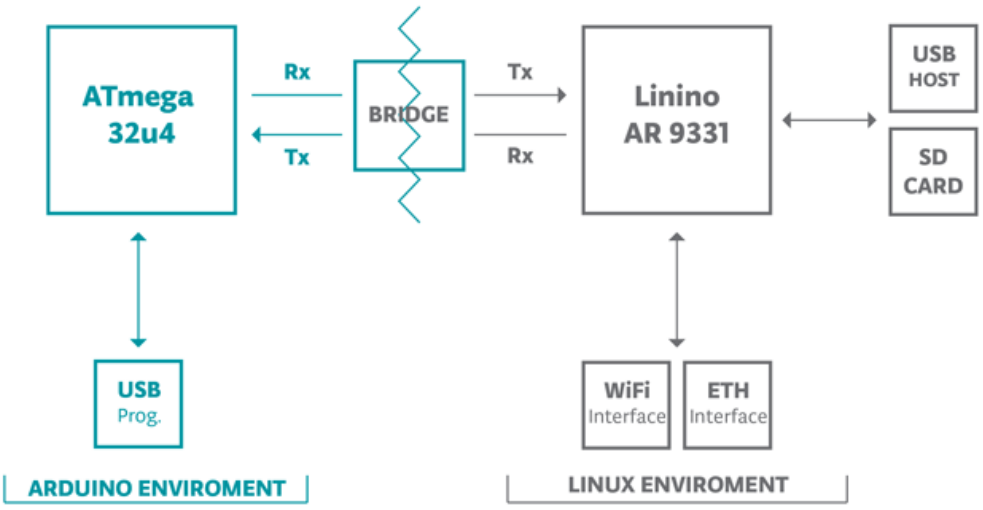


Ilustración 9. Esquema Arduino Yun.

Algunas de las características principales vienen resumidas en las siguientes tablas:

AVR MICROCONTROLADOR ARDUINO	
Microcontrolador	ATmega32U4
Voltaje de funcionamiento	5V
Voltaje de entrada	5 V
Pines I/O Digitales.	20
Salidas PWM	7
Pines I/O Analógicos	12
Corriente DC para pines I/O	40 mA on I/O Pins; 50 mA on 3,3 Pin
Memoria Flash	32 KB (of which 4 KB used by bootloader)
SRAM	2.5 KB
EEPROM	1 KB
Clock Speed	16 MHz

Tabla 1. Microcontrolador Arduino.

MICROPROCESADOR ARDUINO	
Procesador	Atheros AR9331
Arquitectura	MIPS
Voltaje de funcionamiento	3.3V
Ethernet	802.3 10/100Mbit/s
WiFi	802.11b/g/n 2.4 GHz
USB Type	2.0 Host
Card Reader	Micro-SD
RAM	64 MB DDR2
Memoria Flash	16 MB
SRAM	2.5 KB
EEPROM	1 KB
Clock Speed	400 MHz

Tabla 2. Microprocesador Arduino.

Una vez explicadas las características principales de nuestro elemento principal en el Anexo 2 se entra en profundidad sobre las características del hardware, instalación y configuración inicial.

1.2 Objetivos

En primer lugar vamos a definir los objetivos de esta parte.

Queremos conseguir el elemento físico del termostato.

Para ello disponemos de los sensores de temperatura, que realizan la monitorización de la misma (Entradas) y los relés que manejan la apertura de las electroválvulas (Salidas.)

Todo esto es controlado por el procesador Atmel ATmega 32U4 que se encarga de ejecutar las aplicaciones de Arduino.

A su vez disponemos de una librería Bridge que nos permite comunicarnos con el segundo procesador, el Atheros AR9331, compatible con el servidor Linux, el cual se encarga de la gestión de redes (Ethernet, WiFi) y de la comunicación con la aplicación.

Todos estos detalles sobre la configuración del Arduino vienen explicados en el Anexo 2.

En el siguiente esquema se observa la estructura general de esta parte:

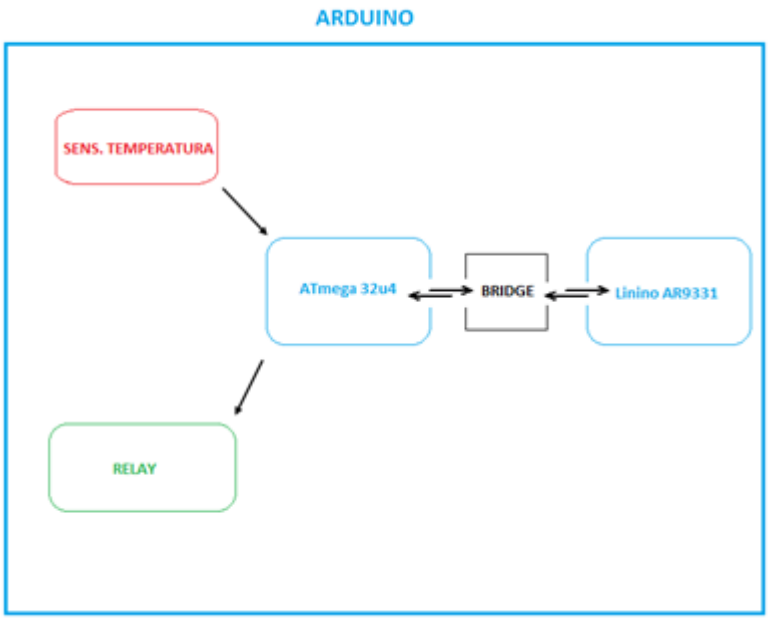


Ilustración 10. Esquema general funcionamiento Arduino

Una vez definida la estructura general del hardware, debemos definir los requisitos del sistema.

El objetivo del dispositivo es monitorizar continuamente la temperatura de dos estancias representativas de cada zona (Habitaciones y zonas comunes) las cuales constan de dos circuitos independientes de calefacción.

Por tanto la función del programa de Arduino será comparar cíclicamente la temperatura de la estancia con la medida deseada.

Cuando la temperatura de la zona sea inferior a la deseada, conducirá el relé, activando la electroválvula correspondiente y activando la calefacción en esa zona.

Adicionalmente a esta función básica queremos conseguir un completo control de la temperatura, permitiendo la programación por horas y días de la semana al usuario.

De este modo ofreceremos al usuario tres modos generales:

MODO MANUAL, en el cual puede fijar la temperatura deseada en las habitaciones y la temperatura deseada en las zonas comunes.

MODO AUTOMÁTICO, el cual le permite al usuario seleccionar entre 3 perfiles pre-configurados por horas.

MODO PERSONALIZADO en el cual se le permite al usuario seleccionar manualmente todas las temperaturas de cada hora de cada día de la semana.

Por tanto nuestro programa de Arduino, además de realizar la comparativa entre la temperatura deseada y la temperatura real, debe realizar la comunicación con la aplicación android, para saber el modo o temperatura que debe utilizar, así como enviar la temperatura actual a la aplicación si es requerido.

También debe saber en todo momento la hora para poder realizar un correcto seguimiento de la referencia.

1.3 Programa Arduino

A continuación analizaremos paso a paso la estructura básica que hemos utilizado para conseguir los objetivos anteriores.

En primer lugar se nos plantea la necesidad de saber la hora.

El programa de arduino debe conocer en todo momento la hora actual, para conocer la temperatura deseada en tiempo real.

Esto lo realizamos con la función "Obtaindate"

```
//Función con la que obtenemos el día de la semana y la hora actual
void obtaindate(int *day, int *hour) {
  date.begin("date");
  //Obtenemos el día de la semana (%u) y la hora (%H)
  date.addParameter("+%u:%H");
  date.run();
  while (date.available() > 0) {
    // Obtenemos el resultado del date process (dd:hh):
    String timeString = date.readString();
    // Buscamos la posición del siguiente carcter (:):
    int firstColon = timeString.indexOf(":");
    // Obtenemos las subcadenas del día y la hora
    String dayString = timeString.substring(0, firstColon);
    String hourString = timeString.substring(firstColon + 1);
    // Lo convertimos a entero.
    *hour = hourString.toInt();
    *day = dayString.toInt();
  }
```

Ilustración 11. Función Obtaindate.

Con esta función, mantenemos continuamente actualizado el día de la semana y la hora actual.

Una vez resuelto este primer problema debemos plantear como organizar los datos.

Para ello decidimos organizar dos matrices de enteros, una para las habitaciones y otra para las zonas comunes.

En ellas se guardarán los valores deseados en función del modo de funcionamiento elegido y en todo momento se compara la temperatura actual con la temperatura deseada.

Hay que tener en cuenta la inclusión de una histéresis, para evitar que el sistema entre en continua fluctuación.

```
sensores.requestTemperatures();//Enviamos el comando para obtener los datos de los sensores
 float Temp_actual_common = Mostrar_Temperatura(S1);
 float Temp_actual_rooms = Mostrar_Temperatura(S2);

 if (Temp_actual_common < programaactualcommon[hour][day] - 0.5) {

   digitalWrite(controlrelecommon, HIGH);
 }
 if (Temp_actual_common > programaactualcommon[hour][day] + 0.5) {

   digitalWrite(controlrelecommon, LOW);
 }

 if (Temp_actual_rooms < programaactualrooms[hour][day] - 0.5) {

   digitalWrite(controlrelerooms, HIGH);
 }
 if (Temp_actual_rooms > programaactualrooms[hour][day] + 0.5) {

   digitalWrite(controlrelerooms, LOW);
 }

 delay(5000);
}
```

Ilustración 12. Control temperatura

En la imagen se observa cómo se llama a la función encargada de obtener la temperatura de los sensores.

```
//Funcion que muestra la temperatura en grados centigrados del sensor
float Mostrar_Temperatura(DeviceAddress direccion)
{
  float tempC = sensores.getTempC(direccion);


  return (tempC);
}
```

Ilustración 13. Obtener temperatura

Una vez obtenida la temperatura, se compara con los valores deseados y se abre o cierra el contacto del relé encargado de activar o desactivar la caldera.

Una vez realizado esto, ya tenemos el funcionamiento general del termostato.

Solo nos queda integrar la comunicación y el protocolo.

La comunicación, como hemos hablado anteriormente, la realizamos mediante la librería Bridge, que nos permite establecer comunicación a través de Internet.

Solicitamos el proceso y aceptamos los clientes.

```
// Arduino Yun acepta clientes
BridgeClient client = server.accept();

//¿Hay nuevos clientes?
if (client) {
  // Solicita el proceso
  process(client);

  // Cierra la conexión y libera los recursos
  client.stop();
}
```

Ilustración 14. Aceptar clientes

Una vez recibimos el comando debemos interpretarlo en función del protocolo y llamar a la función deseada en cada caso.

En la siguiente imagen se puede observar como extraemos los primeros dígitos del comando, que nos permiten interpretar el modo deseado.

```
void process(BridgeClient client) {
  int Personalizado[24][7];
  String Mode = client.readStringUntil('/');
  // Elegir entre los 4 modos
  if (Mode == "00") {
   ReadTemperature(client);
  }
   else if (Mode == "01") {
    ManualMode(client);
  }
  else if (Mode == "10") {
    ProgramMode(client, Personalizado);
  }
  else if (Mode == "11") {
    EditionMode(client, Personalizado);
  }
}
```

Ilustración 15. Selección de modo

Si el modo elegido es "ReadTemperature", Arduino debe obtener la temperatura media de las dos zonas y enviar la información a través del Bridge para que lo interprete la aplicación Android.

En la siguiente imagen observamos cómo realizarlo.

```
void ReadTemperature (BridgeClient client) {

  float Temp_actual_common = Mostrar_Temperatura(S1);
  float Temp_actual_rooms = Mostrar_Temperatura(S2);
  float Temp_media=(Temp_actual_common+Temp_actual_rooms)/2;

  //Enviamos mediante el Bridge la temperatura media a la aplicación Android.

  client.print(Temp_actual);

}
```

Ilustración 16. Función leer temperatura

Si el modo elegido es “ManualMode”, debemos recibir la temperatura, ver en que circuito queremos implementarlo y actualizar la matriz de temperaturas.

En la siguiente imagen lo podemos observar.

```
void ManualMode(BridgeClient client) {

  int Circuit = client.parseInt();
  int Temperature = client.parseInt();

  if (Circuit == 01) {

    for (int i = 0; i < 7; i++) {

       for (int j = 0; j < 23; j++) {

        int manual[j][i] = Temperature;
       }
    }

    Copiarmatrizrooms(manual);
  }

  else if (Circuit == 10) {

    for (int i = 0; i < 7; i++) {

       for (int j = 0; j < 23; j++) {

        int manual[j][i] = Temperature;
       }
    }

    Copiarmatizcommon(manual);
  }
}
```

Ilustración 17. Función modo Manual

Una vez completada la matriz de temperaturas, la enviamos a la función encargada de actualizar la matriz base, la cual comparamos con la temperatura actual que hemos obtenido de los sensores.

```
void Copiarmatrizrooms(int origen1 [24][7]) {
  for (int i = 0; i < 7; i++) {

    for (int j = 0; j < 23; j++) {

     programaactualrooms[j][i] = origen1[j][i];
    }
  }

}

void CopiarmatrizCommon(int origen [24][7]) {
  for (int i = 0; i < 7; i++) {

    for (int j = 0; j < 23; j++) {

      programaactualcommon[j][i] = origen[j][i];
    }
  }

}
```

Ilustración 18. Función actualiza matriz base

Si el modo elegido es “ProgramMode”, iniciamos la función encargada de seleccionar modo.

En ella aparecen los modos preconfigurados, Mantenimiento, Confort y Modo Noche.

Adicionalmente enviamos a la función la matriz con los datos del Modo Personalizado.

Leemos el comando, obteniendo el modo deseado y el circuito en el cual queremos implementarlo.

Una vez realizado esto, enviamos a la función encargada de actualizar las matrices base, vista anteriormente, el modo deseado.

```
void ProgramMode(BridgeClient client, int Personalizado[24][7]) {

  int Mantenimiento [24][7] = {{4,4,4,4,4,4,4}, {4,4,4,4,4,4,4}, {4,4,4,4,4,4,4}, {4,4,4,4,4,4,4}
{4,4,4,4,4,4,4}, {4,4,4,4,4,4,4}, {4,4,4,4,4,4,4}, {4,4,4,4,4,4,4}, {4,4,4,4,4,4,4}, {4,4,4,4,4,4,4
{4,4,4,4,4,4,4}, {4,4,4,4,4,4,4},{4,4,4,4,4,4,4},{4,4,4,4,4,4,4}};

 int Confort [24][7]={{17,17,17,17,17,17,17},{17,17,17,17,17,17,17},{17,17,17,17,17,17,17},{17,17
{19,19,19,19,19,19,19},  {20,20,20,20,20,20,20}, {20,20,20,20,20,20,20}, {19,19,19,19,19,19,19},
{22,22,22,22,22,22,22}, {21,21,21,21,21,21,21},{21,21,21,21,21,21,21},{22,22,22,22,22,22,22}, {22
{21,21,21,21,21,21,21}};

  int ModoNoche[24][7]= {{17,17,17,17,17,17,17}, {17,17,17,17,17,17,17}, {17,17,17,17,17,17,17},
{17,17,17,17,17,17,17}, {18,18,18,18,18,18,18}, {19,19,19,19,19,19,19}, {20,20,20,20,20,20,20}, {
{19,19,19,19,19,19,19}, {19,19,19,19,19,19,19},{19,19,19,19,19,19,19},{19,19,19,19,19,19,19},{19,
{22,22,22,22,22,22,22}, {22,22,22,22,22,22,22}, {23,23,23,23,23,23,23}, {23,23,23,23,23,23,23},{2

  String Circuit = client.readStringUntil('/');
  String Program = client.readStringUntil('/');
| if (Circuit == "01") {

    if (Program == "00") {
        Copiarmatrizrooms(Mantenimiento);
    }

    if (Program == "01") {
       Copiarmatrizrooms(Confort);
    }

    if (Program == "10") {
      Copiarmatrizrooms(ModoNoche);
    }

    if (Program == "11") {
      Copiarmatrizrooms(Personalizado);
    }
  }

  else if (Circuit == "10") {

    if (Program == "00") {
      CopiarmatrizCommon(Mantenimiento);
    }

    if (Program == "01") {
      CopiarmatrizCommon(Confort);
    }

    if (Program == "10") {
       CopiarmatrizCommon(ModoNoche);
    }

    if (Program == "11") {
      CopiarmatrizCommon(Personalizado);
    }
  }
 }
```

Ilustración 19. Función modo programa

Finalmente, si el modo elegido es "EditionMode", editamos la matriz Personalizada.

Dado que desde la aplicación Android, enviamos los datos uno a uno, en esta función solamente tenemos que interpretar, en base al protocolo, la casilla de la matriz donde debemos incluir el dato de la temperatura.

En la siguiente imagen se puede observar la función.

```
void EditionMode(BridgeClient client, int Personalizado[24][7]) {

  int day = client.parseInt();
  int hour = client.parseInt();
  int Temp = client.parseInt();

  Personalizado[hour][day - 1] = Temp;
}
```

Ilustración 20. Función modo edición

CAPÍTULO 2 ANDROID

A continuación vamos a realizar una pequeña introducción al sistema operativo Android.

Android es un sistema operativo basado en el núcleo Linux al cual se le han hecho ciertas modificaciones para que pueda ejecutarse en teléfonos y terminales móviles.

Fue diseñado principalmente para dispositivos móviles con pantalla táctil, como teléfonos inteligentes y tablets.

Inicialmente fue desarrollado por Android Inc., empresa que Google respaldó económicamente y más tarde, en 2005, compró.

Lo que ha hecho diferente Android respecto de otros sistemas operativos como iOS, Symbian y Blackberry OS, es que está basado en Linux, un núcleo de sistema operativo libre, gratuito y multiplataforma.

El sistema permite programar aplicaciones en una variación de Java llamada Dalvik. El sistema operativo proporciona todas las interfaces necesarias para desarrollar aplicaciones que accedan a las funciones del teléfono (como el GPS, las llamadas, la agenda, etc.) de una forma muy sencilla en un lenguaje de programación muy conocido como es Java.

Esta sencillez, junto a la existencia de herramientas de programación gratuitas, hacen que una de las cosas más importantes de este sistema operativo sea la cantidad de aplicaciones disponibles, que extienden casi sin límites la experiencia del usuario.

2.1 Arquitectura Android

En la siguiente figura se visualizan los aspectos más importantes de la arquitectura del sistema operativo:

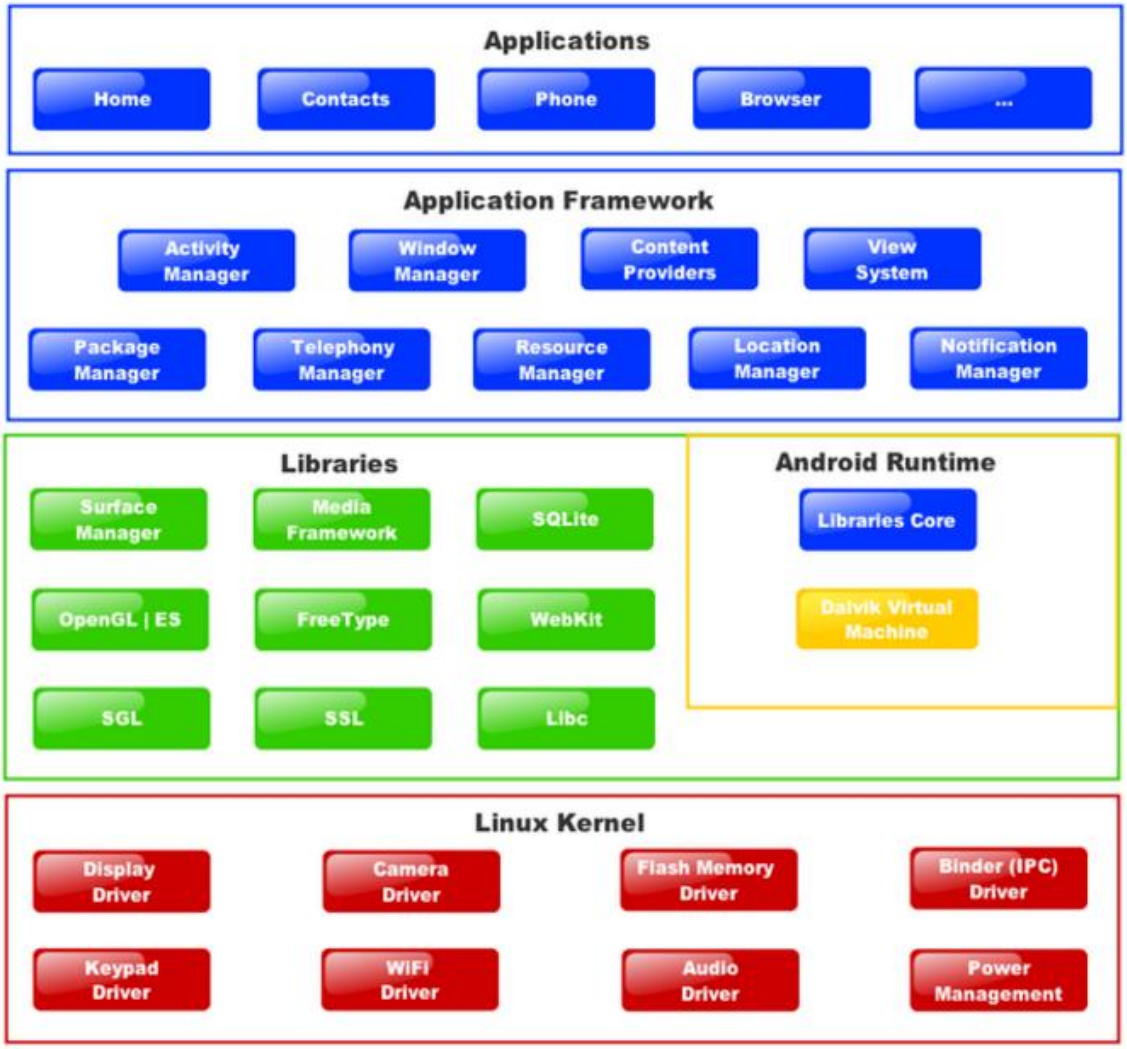


Ilustración 21. Arquitectura Android

APLICACIONES: Este nivel contiene, tanto las aplicaciones base, incluidas por defecto de Android: cliente de correo electrónico, programa de SMS, calendario, mapas, navegador, contactos y otros, como aquellas que el usuario ha ido añadiendo posteriormente.

Todas estas aplicaciones utilizan los servicios, las API y librerías de los niveles inferiores.

FRAMEWORK DE APLICACIONES: Representa fundamentalmente el conjunto de herramientas de desarrollo de cualquier aplicación. El objetivo de este diseño de arquitectura es simplificar la reutilización de componentes, cualquier aplicación puede publicar sus capacidades y el resto puede hacer uso de estas capacidades.

LIBRERÍAS: La siguiente capa se corresponde con las librerías utilizadas por Android.

Están escritas en C/C++ y dotan a Android de gran parte de sus capacidades más características. Están compiladas en código nativo del procesador.

RUNTIME DE ANDROID: Al mismo nivel que las librerías de Android se sitúa el entorno de ejecución, Android incluye una serie de bibliotecas base, que proporcionan la mayor parte de las funciones disponibles en las bibliotecas base del lenguaje Java.

KERNEL DE LINUX: Android utiliza el núcleo de Linux 2.6 como una capa de abstracción para el hardware disponible en los dispositivos móviles. Esta capa contiene los drivers necesarios para que cualquier componente hardware pueda ser utilizado mediante las llamadas correspondientes. Siempre que un fabricante incluye un nuevo elemento de hardware, lo primero que se debe realizar para que pueda ser utilizado desde Android es crear las librerías de control o drivers necesarios dentro de este kernel de Linux embebido en el propio Android.

2.2 Entorno de desarrollo

Para comenzar a desarrollar una aplicación de forma nativa, debemos configurar en nuestro PC, el entorno y las herramientas de trabajo para implementar nuestras aplicaciones.

Llegados a este punto existía la posibilidad de realizar la aplicación con diversos software.

Actualmente hay diversos software como appinventor que nos permiten desarrollar aplicaciones de forma sencilla y gráfica, pero nos decantamos por utilizar el entorno recomendado por Android, ya que nos permitía obtener un conocimiento mayor.

Por esta razón optamos por utilizar AndroidStudio, entorno de desarrollo que ha desbancado completamente a Eclipse, convirtiéndose en el IDE oficial para el desarrollo de aplicaciones para Android.

Está basado en el software IntelliJ IDEA de JetBrains, y es publicado de forma gratuita a través de la Licencia Apache 2.0.

En el anexo 3 se entra en detalle sobre la configuración del entorno de desarrollo, su instalación y nociones principales.

2.3 Función y diseño

Una vez instalado y configurado el entorno de desarrollo, el siguiente paso es analizar de forma clara las funciones que queremos que realice nuestra app, definir las pantallas y los elementos que queremos en cada una de ellas.

Como explicamos en las funciones del elemento hardware, nuestro termostato tiene dos Modos de funcionamiento generales, MANUAL y AUTOMÁTICO. A su vez consta de un tercer modo que es el modo EDICIÓN, en el cual se configura un programa personalizado.

Por este motivo decidimos incluir en la pantalla principal un selector de modo en el cual podamos elegir entre estos 3 modos.

Al pulsar en cada uno de ellos, nos redireccionará a la pantalla deseada.

A su vez, en la pantalla principal, disponemos de un pulsador, mediante el cual requerimos una media de la temperatura actual de la vivienda en tiempo real.

Cuando pulsamos este botón, se realiza una petición al Arduino que nos responde de manera inmediata con la temperatura actual.

La pantalla obtenida es la siguiente:

Ilustración 22. Pantalla principal app

Para logarlo, definimos un Linear Layout, que nos distribuye linealmente todos los controles que vamos a incluir en su interior.

En primer lugar tenemos el Check Box, utilizado para seleccionar el modo deseado.

Este control se utiliza para marcar o desmarcar opciones, y conseguimos mediante la clase OnClickListener (control que se activa cuando se pulsa una de las opciones), disparar el Intent que nos lanza una nueva actividad. De este modo conseguimos mediante el Check Box, leer las pulsaciones y lanzar la pantalla deseada en cada caso.

En la siguiente imagen se puede observar el modo de implementarlo.

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

       /* RadioButton*/
        rbOpcion1 = (RadioButton) findViewById(R.id.RbOpcion1);
       /* RadioButton*/
        rbOpcion2 = (RadioButton) findViewById(R.id.RbOpcion2);
       /* RadioButton*/
        rbOpcion3 = (RadioButton) findViewById(R.id.RbOpcion3);

        View.OnClickListener list = new View.OnClickListener() {
            @Override
            public void onClick(View view) {

                switch (view.getId()) {
                    case R.id.RbOpcion1:
                        // Creamos el Intent para llamar a Manual
                        Intent manual =
                                new Intent(MenuActivity.this, Manual.class);
                        startActivity(manual);
                        break;
                    case R.id.RbOpcion2:
                        // Creamos el Intent para llamar a Automatico
                        Intent automatico =
                                new Intent(MenuActivity.this, Auto.class);
                        startActivity(automatico);
                        break;
                    case R.id.RbOpcion3:
                        // Creamos el Intent para llamar a Edición
                        Intent edition =
                                new Intent(MenuActivity.this, Edition.class);
                        startActivity(edition);
                        break;
                }
            }

        };
```

Ilustración 23. Código pantalla principal

Una vez logrado el selector, debemos realizar una de las partes más complejas de la aplicación.

Definimos un botón, el cual detecta la pulsación mediante la clase OnClickListener, igual que el CheckBox anterior, una vez pulsado, realiza una petición http a través de internet a nuestro Arduino, el cual lee la temperatura media de los sensores de la vivienda y responde.

La aplicación debe leer esta respuesta e imprimir por pantalla en el cuadro de texto la temperatura recibida.

El principal problema de este tipo de comunicaciones es que requieren muchos recursos, por lo cual si son realizadas en el hilo principal conllevan la ralentización e incluso cierre de la aplicación.

Para resolver este problema creamos una TAREA ASÍNCRONA desde la cual realizamos esta petición http.

En la siguiente imagen podemos observar la manera de implementarlo:

```
//Obtenemos una referencia a los controles de la interfaz
        txtTemperatura = (TextView) findViewById(R.id.txtTemperatura);
        btnActualizar = (Button) findViewById(R.id.btnActualizar);
        //Implementamos el evento click del botón
        btnActualizar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                //Creamos la tarea asíncrona para comunicarnos con el arduino.

                new AsyncTask<Void, Void, String>() {
                    @Override
                    protected String doInBackground(Void... params) {
                        HttpClient httpclient = new DefaultHttpClient();
                        String aux = web_service + "00/0";
                        String respuesta="";
                        try {
                            HttpResponse response = httpclient.execute(new HttpGet(aux));
                            ByteArrayOutputStream out = new ByteArrayOutputStream();
                            response.getEntity().writeTo(out);
                            out.close();

                            respuesta = out.toString();



                        } catch (Exception e) {
                            e.printStackTrace();
                        }
                        return respuesta;
                    }

                    @Override
                    protected void onPostExecute(String result) {

                        txtTemperatura.setText(result+" ºC");
                    }

                }.execute();
            }

        });
```

Ilustración 24. Tarea asíncrona

Además de esta pantalla principal, como hemos hablado anteriormente, disponemos 4 pantallas adicionales, una para cada modo, excepto el modo EDICIÓN, en el cual necesitamos dos pantallas debido a la complejidad y extensión requeridas.

Cuando pulsamos en el selector principal el modo manual, nos redirecciona a una pantalla en la cual simplemente nos pide introducir la temperatura deseada en la zona de habitaciones y la temperatura deseada en las zonas comunes.

Si pulsamos el botón guardar cambios, esta información es enviada al Arduino e inmediatamente empieza a funcionar en este modo.

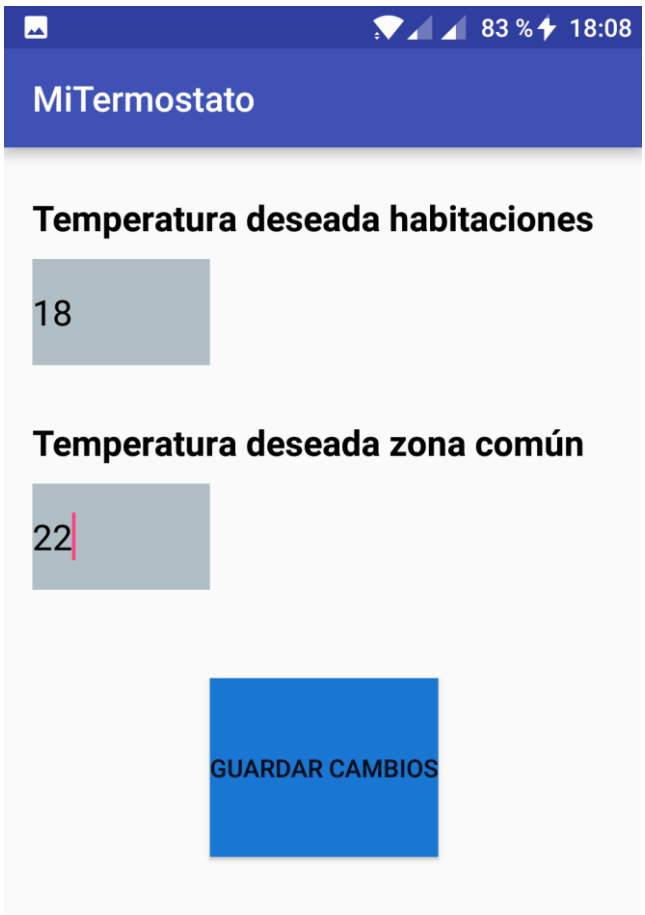


Ilustración 25. Modo manual

El modo de implementar esta pantalla es muy sencillo, en un Linear Layout introducimos todos los controles necesarios, tanto las etiquetas de texto, como las ediciones de texto como el botón.

Una vez introducidos los valores deseados por pantalla en nuestro dispositivo, pulsamos el botón.

Este, con un funcionamiento análogo al botón explicado anteriormente envía una petición http a nuestro Arduino, informando de que se ha configurado el modo manual, y las temperaturas deseadas.

Al igual que en el caso anterior debemos realizarlo en una tarea asíncrona para no bloquear el hilo principal de la aplicación.

Una vez realizado esto, lanzamos de nuevo la pantalla del Menú.

En la siguiente imagen se puede ver el modo de implementarlo.

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

    //Obtenemos una referencia a los controles de la interfaz
    txtTemphabitaciones = (TextView) findViewById(R.id.txtmanualhabitaciones);
    final String tmphabitaciones=txtTemphabitaciones.getText().toString();
    txtTempZC = (TextView) findViewById(R.id.txtmanualzonacomun);
    final String tmpZC=txtTempZC.getText().toString();
    btnActualizar = (Button) findViewById(R.id.btnGuardarCambios);

    //Implementamos el evento click del botón
    btnActualizar.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            //Creamos la tarea asíncrona para comunicarnos con el arduino.

            new AsyncTask<Void, Void, Void>() {
                @Override
                protected Void doInBackground(Void... params) {
                    HttpClient httpClient = new DefaultHttpClient();
                    String aux ="";
                    String aux2="";
                    aux=web_service + "01/01/"+tmphabitaciones;
                    aux2=web_service+"01/01/"+tmpZC;
                    HttpGet httpget = new HttpGet(aux);
                    Log.i(TAG, aux);
                    HttpGet httpget2 = new HttpGet(aux2);
                    Log.i(TAG, aux2);
                    try {
                        httpClient.execute(httpget);
                        httpClient.execute(httpget2);
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                    return null;
                }
            }.execute();

                //Creamos el Intent para volver a la pantalla principal
                Intent intent =
                        new Intent(Manual.this, MenuActivity.class);

                //Iniciamos la nueva actividad
                startActivity(intent);
        }
    });
```

Ilustración 26. Código modo manual

Si en el menú principal seleccionamos el modo automático, nos redirecciona a la pantalla del Modo Automático, aquí observamos dos selectores análogos a los del menú principal, en los cuales nos deja seleccionar entre los 4 perfiles disponibles: Mantenimiento, Confort, Modo Noche y Personalizado.

Cuando pulsamos el botón guardar cambios esta información es enviada al Arduino mediante una petición http al igual que en el modo manual e inmediatamente se actualiza el perfil deseado.

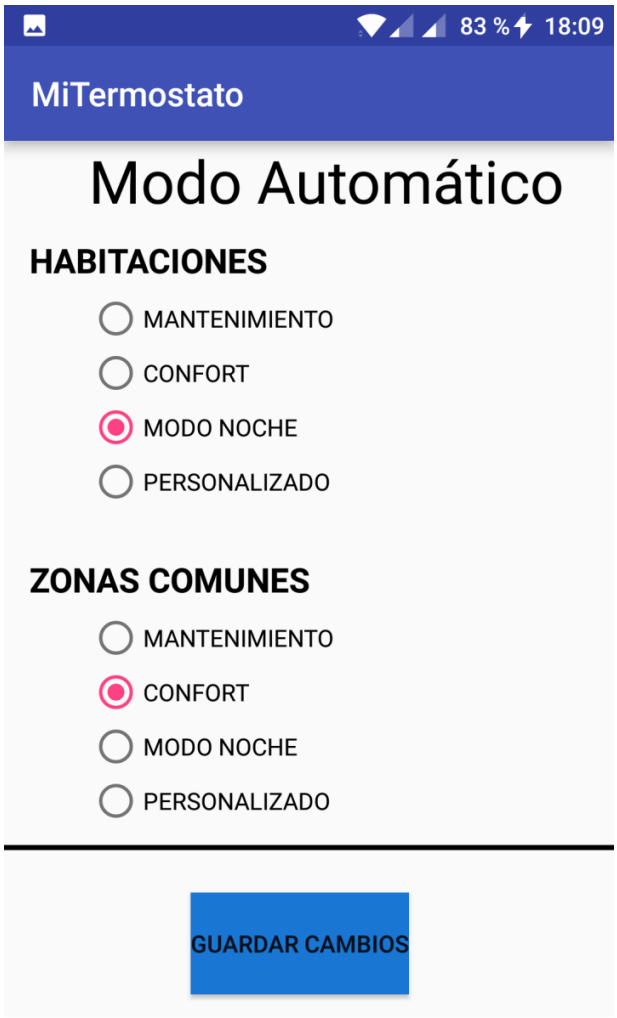


Ilustración 27. Modo automático.

El modo de implementarlo es análogo al de las pantallas anteriores, mediante el CheckBox seleccionamos el modo deseado para cada circuito y al pulsar el botón, se crea una tarea asíncrona, que lanza la petición http hacia el Arduino como se puede observar en las siguientes imágenes:

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

       //RadioGroup Circuito 1
       rbPrograma1 = (RadioButton) findViewById(R.id.RbPrograma1);
       rbPrograma2 = (RadioButton) findViewById(R.id.RbPrograma2);
     | rbPrograma3 = (RadioButton) findViewById(R.id.RbPrograma3);
       rbPrograma4 = (RadioButton) findViewById(R.id.RbPrograma4);

       //RadioGroup Circuito 2
       rbPrograma12 = (RadioButton) findViewById(R.id.RbPrograma12);
       rbPrograma22 = (RadioButton) findViewById(R.id.RbPrograma22);
       rbPrograma32 = (RadioButton) findViewById(R.id.RbPrograma32);
       rbPrograma42 = (RadioButton) findViewById(R.id.RbPrograma42);

       //Button
       btnGuardarmodo = (Button) findViewById(R.id.btnGuardarModo);

       View.OnClickListener list = new View.OnClickListener() {
           @Override
           public void onClick(View view) {

               switch (view.getId()) {
                   case R.id.RbPrograma1:
                       aux=web_service+"10/01/00";
                       break;
                   case R.id.RbPrograma2:
                       aux=web_service+"10/01/01";
                       break;
                   case R.id.RbPrograma3:
                       aux=web_service+"10/01/10";
                       break;
                   case R.id.RbPrograma4:
                       aux=web_service+"10/01/11";
                       break;
               }
           }
       };

       View.OnClickListener list2 = new View.OnClickListener() {
           @Override
           public void onClick(View view) {

               switch (view.getId()) {
                   case R.id.RbPrograma12:
                       aux2=web_service+"10/10/00";
                       break;
                   case R.id.RbPrograma22:
                       aux2=web_service+"10/10/01";
                       break;
                   case R.id.RbPrograma32:
                       aux2=web_service+"10/10/10";
                       break;
                   case R.id.RbPrograma42:
                       aux2=web_service+"10/10/11";
                       break;
               }
           }
       };
```

Ilustración 28. Código modo automático 1

```
//Implementamos el evento click del botón
btnGuardarmodo.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        //Creamos la tarea asíncrona para comunicarnos con el arduino.

        new AsyncTask<Void, Void, Void>() {
            @Override
            protected Void doInBackground(Void... params) {
                HttpClient httpClient = new DefaultHttpClient();

                HttpGet httpget = new HttpGet(aux);
                Log.i(TAG, aux);
                HttpGet httpget2 = new HttpGet(aux2);
                Log.i(TAG, aux2);
                try {
                    httpClient.execute(httpget);
                    httpClient.execute(httpget2);
                } catch (Exception e) {
                    e.printStackTrace();
                }
                return null;
            }
        }.execute();


        //Creamos el Intent para volver a la pantalla principal
        Intent intent =
                new Intent(Auto.this, MenuActivity.class);

        //Iniciamos la nueva actividad
        startActivity(intent);


    }

});
```

Ilustración 29. Código modo automático 2

Con el modo Edición, hemos decidido que la mejor opción era optar entre 3 temperaturas, que el usuario pudiera elegir, y en base a estas temperaturas luego seleccionar hora a hora el nivel deseado.

Por esta razón, cuando seleccionamos en el selector principal el MODO EDICIÖN, nos redirecciona a la siguiente pantalla, en la cual podemos introducir 3 temperaturas:

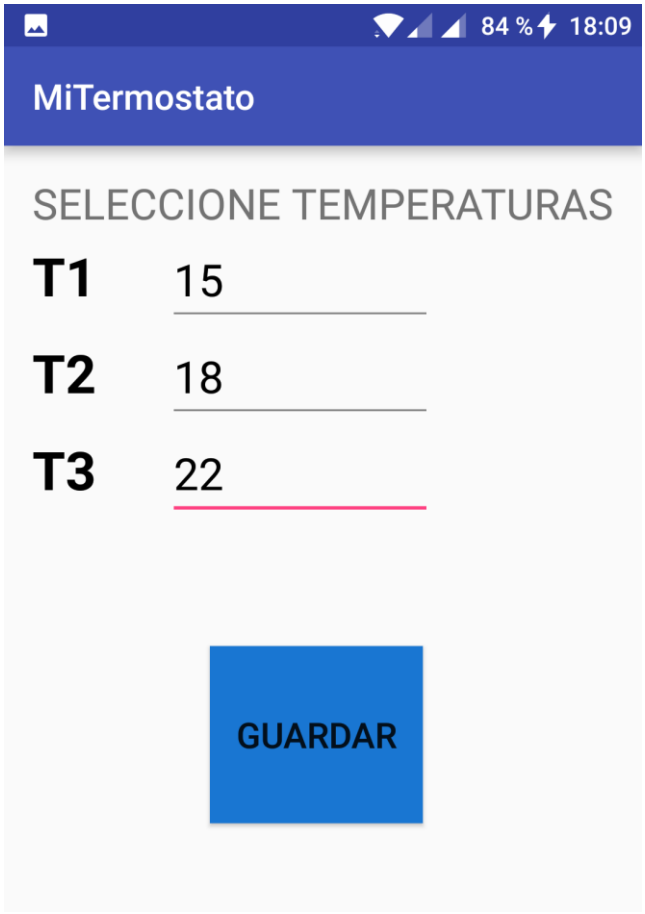


Ilustración 30. Selección temperaturas

El modo de realizar esta pantalla es bastante sencillo, implementamos tres editores de texto, en los cuales podemos introducir las temperaturas deseadas.

A su vez de modo similar a las anteriores pantallas, incluimos un botón, que nos lanzará la siguiente actividad.

Cuando el programa detecta que el botón ha sido pulsado, recoge los datos introducidos en los campos de texto y los envía a la siguiente pantalla, como se puede observar en la siguiente imagen.

```
public class Edition extends AppCompatActivity {

    private EditText txtT1;
    private EditText txtT2;
    private EditText txtT3;
    private Button btnGuardarCambios;
    private Button btnPasstoedition;
    private Button btnReturnmenu;

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

        //Obtenemos una referencia a los controles de la interfaz
        txtT1 = (EditText)findViewById(R.id.txtT1);
        txtT2 = (EditText)findViewById(R.id.txtT2);
        txtT3 = (EditText)findViewById(R.id.txtT3);

        btnPasstoedition = (Button)findViewById(R.id.btnPasstoedition);


        //Implementamos el evento click del botón
        btnPasstoedition.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                //Creamos el Intent
                Intent intent =
                        new Intent(Edition.this, Edition_temperatures.class);

                //Creamos la información a pasar entre actividades
                Bundle b = new Bundle();
                b.putString("Temp1", txtT1.getText().toString());
                b.putString("Temp2", txtT2.getText().toString());
                b.putString("Temp3", txtT3.getText().toString());

                //Añadimos la información al intent
                intent.putExtras(b);

                //Iniciamos la nueva actividad
                startActivity(intent);
            }
        });
    }
}
```

Ilustración 31. Código selección temperaturas

Una vez pulsamos el botón guardar, la aplicación nos lleva a la pantalla siguiente, en la cual podemos elegir de forma rápida e intuitiva el nivel de temperatura que queremos en función del rango de horas.

Esta es una pantalla bastante compleja, en la cual tenemos 7 pestañas, una por cada día de la semana.

A su vez en cada día de la semana hay la posibilidad de elegir las 24 horas del día el nivel de temperatura que queremos.

Una vez seleccionado el día, pulsamos si queremos guardar esa configuración personalizada, en el circuito de las habitaciones o en el circuito de las zonas comunes.

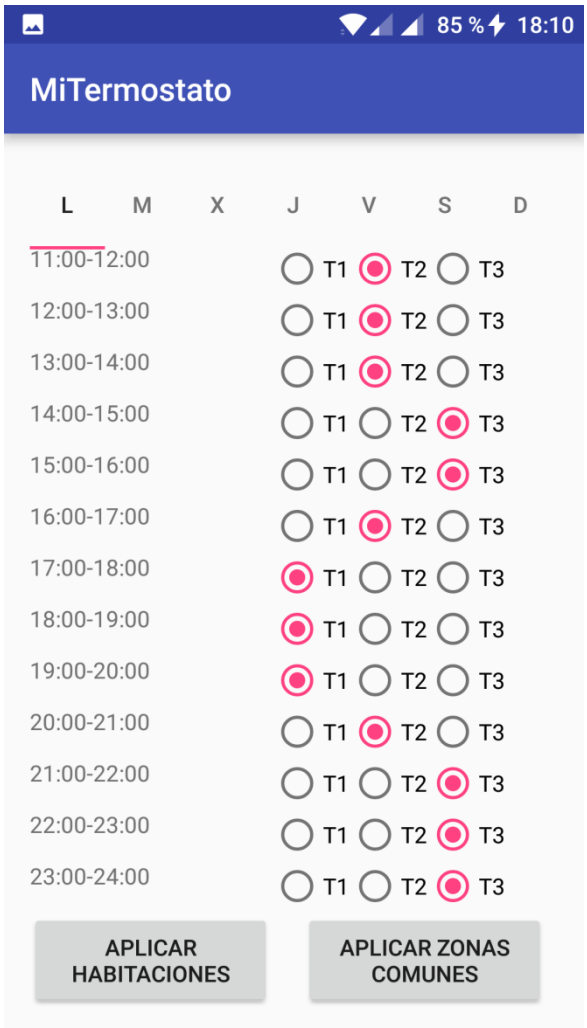


Ilustración 32. Modo edición.

Esta pantalla es sin duda la más compleja de todas, por esta razón iremos paso a paso analizando la construcción.

En primer lugar debemos crear las 7 pestañas para los 7 días de la semana.

En la siguiente imagen se puede ver el modo de realizarlo:

```
TabHost tabs = (TabHost) findViewById(android.R.id.tabhost);
tabs.setup();

TabHost.TabSpec spec = tabs.newTabSpec("mitab1");
spec.setContent(R.id.l);
spec.setIndicator("L");
tabs.addTab(spec);


spec = tabs.newTabSpec("mitab2");
spec.setContent(R.id.m);
spec.setIndicator("M");
tabs.addTab(spec);


spec = tabs.newTabSpec("mitab3");
spec.setContent(R.id.x);
spec.setIndicator("X");
tabs.addTab(spec);


spec = tabs.newTabSpec("mitab4");
spec.setContent(R.id.j);
spec.setIndicator("J");
tabs.addTab(spec);


spec = tabs.newTabSpec("mitab5");
spec.setContent(R.id.v);
spec.setIndicator("V");
tabs.addTab(spec);


spec = tabs.newTabSpec("mitab6");
spec.setContent(R.id.s);
spec.setIndicator("S");
tabs.addTab(spec);


spec = tabs.newTabSpec("mitab7");
spec.setContent(R.id.d);
spec.setIndicator("D");
tabs.addTab(spec);

tabs.setCurrentTab(0);
```

Ilustración 33. Código pestañas modo edición

Mediante el control TabHosts, conseguimos realizar esta organización de la pantalla, y obtener información sobre la pantalla en la que se encuentra.

Una vez conseguida esta primera estructura, en cada una de ellas, incluimos 24 CheckBox de tres opciones, de manera similar a los CheckBox utilizados en pantallas anteriores.

En la siguiente imagen se puede observar la implementación para el Lunes.

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


    //Lunes
    rb1LOpcion1 = (RadioButton) findViewById(R.id.Rb1LOpcion1);
    rb1LOpcion2 = (RadioButton) findViewById(R.id.Rb1LOpcion2);
    rb1LOpcion3 = (RadioButton) findViewById(R.id.Rb1LOpcion3);
    rb2LOpcion1 = (RadioButton) findViewById(R.id.Rb2LOpcion1);
    rb2LOpcion2 = (RadioButton) findViewById(R.id.Rb2LOpcion2);
    rb2LOpcion3 = (RadioButton) findViewById(R.id.Rb2LOpcion3);
    rb3LOpcion1 = (RadioButton) findViewById(R.id.Rb3LOpcion1);
    rb3LOpcion2 = (RadioButton) findViewById(R.id.Rb3LOpcion2);
    rb3LOpcion3 = (RadioButton) findViewById(R.id.Rb3LOpcion3);
    rb4LOpcion1 = (RadioButton) findViewById(R.id.Rb4LOpcion1);
    rb4LOpcion2 = (RadioButton) findViewById(R.id.Rb4LOpcion2);
    rb4LOpcion3 = (RadioButton) findViewById(R.id.Rb4LOpcion3);
    rb5LOpcion1 = (RadioButton) findViewById(R.id.Rb5LOpcion1);
    rb5LOpcion2 = (RadioButton) findViewById(R.id.Rb5LOpcion2);
    rb5LOpcion3 = (RadioButton) findViewById(R.id.Rb5LOpcion3);
    rb6LOpcion1 = (RadioButton) findViewById(R.id.Rb6LOpcion1);
    rb6LOpcion2 = (RadioButton) findViewById(R.id.Rb6LOpcion2);
    rb6LOpcion3 = (RadioButton) findViewById(R.id.Rb6LOpcion3);
    rb7LOpcion1 = (RadioButton) findViewById(R.id.Rb7LOpcion1);
    rb7LOpcion2 = (RadioButton) findViewById(R.id.Rb7LOpcion2);
    rb7LOpcion3 = (RadioButton) findViewById(R.id.Rb7LOpcion3);
    rb8LOpcion1 = (RadioButton) findViewById(R.id.Rb8LOpcion1);
    rb8LOpcion2 = (RadioButton) findViewById(R.id.Rb8LOpcion2);
    rb8LOpcion3 = (RadioButton) findViewById(R.id.Rb8LOpcion3);
    rb9LOpcion1 = (RadioButton) findViewById(R.id.Rb9LOpcion1);
    rb9LOpcion2 = (RadioButton) findViewById(R.id.Rb9LOpcion2);
    rb9LOpcion3 = (RadioButton) findViewById(R.id.Rb9LOpcion3);
    rb10LOpcion1 = (RadioButton) findViewById(R.id.Rb10LOpcion1);
    rb10LOpcion2 = (RadioButton) findViewById(R.id.Rb10LOpcion2);
    rb10LOpcion3 = (RadioButton) findViewById(R.id.Rb10LOpcion3);
    rb11LOpcion1 = (RadioButton) findViewById(R.id.Rb11LOpcion1);
    rb11LOpcion2 = (RadioButton) findViewById(R.id.Rb11LOpcion2);
    rb11LOpcion3 = (RadioButton) findViewById(R.id.Rb11LOpcion3);
    rb12LOpcion1 = (RadioButton) findViewById(R.id.Rb12LOpcion1);
    rb12LOpcion2 = (RadioButton) findViewById(R.id.Rb12LOpcion2);
    rb12LOpcion3 = (RadioButton) findViewById(R.id.Rb12LOpcion3);
    rb13LOpcion1 = (RadioButton) findViewById(R.id.Rb13LOpcion1);
    rb13LOpcion2 = (RadioButton) findViewById(R.id.Rb13LOpcion2);
    rb13LOpcion3 = (RadioButton) findViewById(R.id.Rb13LOpcion3);
    rb14LOpcion1 = (RadioButton) findViewById(R.id.Rb14LOpcion1);
    rb14LOpcion2 = (RadioButton) findViewById(R.id.Rb14LOpcion2);
    rb14LOpcion3 = (RadioButton) findViewById(R.id.Rb14LOpcion3);
    rb15LOpcion1 = (RadioButton) findViewById(R.id.Rb15LOpcion1);
    rb15LOpcion2 = (RadioButton) findViewById(R.id.Rb15LOpcion2);
    rb15LOpcion3 = (RadioButton) findViewById(R.id.Rb15LOpcion3);
    rb16LOpcion1 = (RadioButton) findViewById(R.id.Rb16LOpcion1);
    rb16LOpcion2 = (RadioButton) findViewById(R.id.Rb16LOpcion2);
    rb16LOpcion3 = (RadioButton) findViewById(R.id.Rb16LOpcion3);
    rb17LOpcion1 = (RadioButton) findViewById(R.id.Rb17LOpcion1);
    rb17LOpcion2 = (RadioButton) findViewById(R.id.Rb17LOpcion2);
    rb17LOpcion3 = (RadioButton) findViewById(R.id.Rb17LOpcion3);
    rb18LOpcion1 = (RadioButton) findViewById(R.id.Rb18LOpcion1);
    rb18LOpcion2 = (RadioButton) findViewById(R.id.Rb18LOpcion2);
```

Ilustración 34.Código Checkbox modo edición

Una vez conseguido implementar este control en cada una de las pestañas, tenemos toda la estructura de la pantalla.

A continuación debemos establecer la lógica de la misma.

En primer lugar debemos recuperar la información proveniente de la pantalla anterior, en la cual le enviábamos las temperaturas fijadas por el usuario.

Una vez recuperadas las 3 temperaturas, según el usuario va marcando las opciones deseadas en el CheckBox, se van formando las cadenas de texto que se enviarán con posterioridad al arduino.

Se puede observar en la siguiente imagen el modo de implementarlo.

```
View.OnClickListener list = new View.OnClickListener() {
    @Override

    public void onClick(View view) {

        //Recuperamos la información pasada en el intent
        Bundle bundle = getIntent().getExtras();
        Resources res = getResources();

        String temp1=bundle.getString("Temp1");;
        String temp2=bundle.getString("Temp2");
        String temp3=bundle.getString("Temp3");


        switch (view.getId()) {
            case R.id.Rb1LOpcion1:
                aux1 = "11/00/00/" + temp1;
                break;

            case R.id.Rb1LOpcion2:
                aux1 = "11/00/00/" + temp2;
                break;

            case R.id.Rb1LOpcion3:
                aux1 = "11/00/00/" +temp3;
                break;

            case R.id.Rb2LOpcion1:
                aux2 = "11/00/01/" + temp1;
                break;

            case R.id.Rb2LOpcion2:
                aux2 = "11/00/01/" + temp2;
                break;

            case R.id.Rb2LOpcion3:
                aux2 = "11/00/01/" +temp3;
                break;

            case R.id.Rb3LOpcion1:
                aux3 = "11/00/02/" + temp1;
                break;

            case R.id.Rb3LOpcion2:
                aux3 = "11/00/02/" + temp2;
                break;

            case R.id.Rb3LOpcion3:
                aux3 = "11/00/02/" +temp3;
                break;

            case R.id.Rb4LOpcion1:
                aux4 = "11/00/03/" + temp1;
                break;

            case R.id.Rb4LOpcion2:
                aux4 = "11/00/03/" + temp2;
```

Ilustración 35. Código formar comandos

Una vez formadas las cadenas en función del protocolo de comunicación deseado ya tenemos hecho lo más complicado y el resto es similar a lo realizado en las pantallas anteriores.

Disponemos de dos botones en cada pestaña. Cuando cualquiera de ellos es pulsado, se crea una tarea asíncrona, que manda mediante 24 peticiones http las 24 cadenas de texto, con la información del día completo. En función del botón pulsado, esta información se aplicará en el circuito de las zonas comunes o en el circuito de las habitaciones.

CAPÍTULO 3 COMUNICACIÓN

Una vez definidas las dos áreas principales de nuestro proyecto debemos definir la integración entre ambas y como lograr la comunicación.

Como hablamos en el capítulo dedicado a Arduino, la nueva placa Arduino Yun presenta la ventaja de combinar conectividad Wi-Fi con el sistema operativo Linux, lo que ofrece infinidad de posibilidades.

Para ello, se integran en la misma placa dos microprocesadores, el procesador ATmega32u4 que es el corazón del arduino y un Atheros AR9331 que maneja la distribución Linux. Por un lado contamos con toda la conectividad y capacidad de desarrollo de un Arduino y por el otro con una máquina Linux, lo que hace que podamos ejecutar comandos, aplicaciones o scripts en el lado Linux mientras que utilizamos su conectividad ethernet y Wi-Fi

Para facilitar la programación web, YÚN incluye una librería puente "Bridge" que conecta las transacciones HTTP entre los dos lados (Linux y microcontrolador).

La biblioteca Bridge simplifica la comunicación entre el ATmega32U4 y el AR9331.

Los comandos Bridge del microcontrolador son interpretados por Python en el AR9331.

Su función es ejecutar programas en el lado de GNU / Linux cuando se lo pida Arduino, proporcionar un espacio de almacenamiento compartido para compartir datos como lecturas de sensores entre el Arduino e Internet, y recibir comandos de Internet y pasarlos directamente al Arduino.

Esta librería permite la comunicación en ambas direcciones, actuando como una interfaz a la línea de comandos de Linux.

De este modo conseguimos mediante peticiones Rest, poder leer la información de los sensores de temperatura, y enviárselos a la app, así como

mandar a la parte de Arduino la información del modo o temperatura elegidos.

Llegados a este punto nos encontramos con un problema, en primer lugar, como explicamos en el Anexo 2 Configuración del Arduino Yun, debemos asignar una dirección fija a nuestro Arduino. Como vimos asignamos la dirección 192.168.1.168.

En segundo lugar, la aplicación Android que controlará al YUN, puede conectarse fácilmente al mismo utilizando su ip si estamos conectados a nuestra red doméstica. Sin embargo cuando no estemos conectados a la misma red que YUN, tendremos que utilizar otro mecanismo para poder acceder a él.

Para ello hay que realizar dos acciones:

Primero, configurar nuestro router doméstico para redirigir las peticiones al puerto 80 al YUN.

En segundo lugar, para no tener que recordar nuestra ip externa utilizaremos el servicio NO_IP para mapear nuestra ip a un nombre de dominio que podamos recordar fácilmente.

Tenemos dos opciones, el servicio por un coste de 24.95€ al año, que puede ser accesible, se encarga de mantener actualizada la IP en caso de que nuestro proveedor de servicios la cambie. Sin embargo hemos conseguido que el propio YUN se encargue mediante una tarea de cron, de mantener actualizada la IP externa en NO-IP.

3.1 Configuración Router

Las acciones a realizar para configurar nuestro router para que redireccione todas las peticiones realizadas al puerto 80 de nuestra IP externa, al puerto 80 del Arduino, son similares en todos los routers, por lo que pondré el ejemplo del mío en cuestión, siendo fácilmente extrapolable al de cualquier marca.

En primer lugar accedemos al panel de configuración, poniendo para ello en un navegador la ip de nuestro router e introduciendo usuario y contraseña.

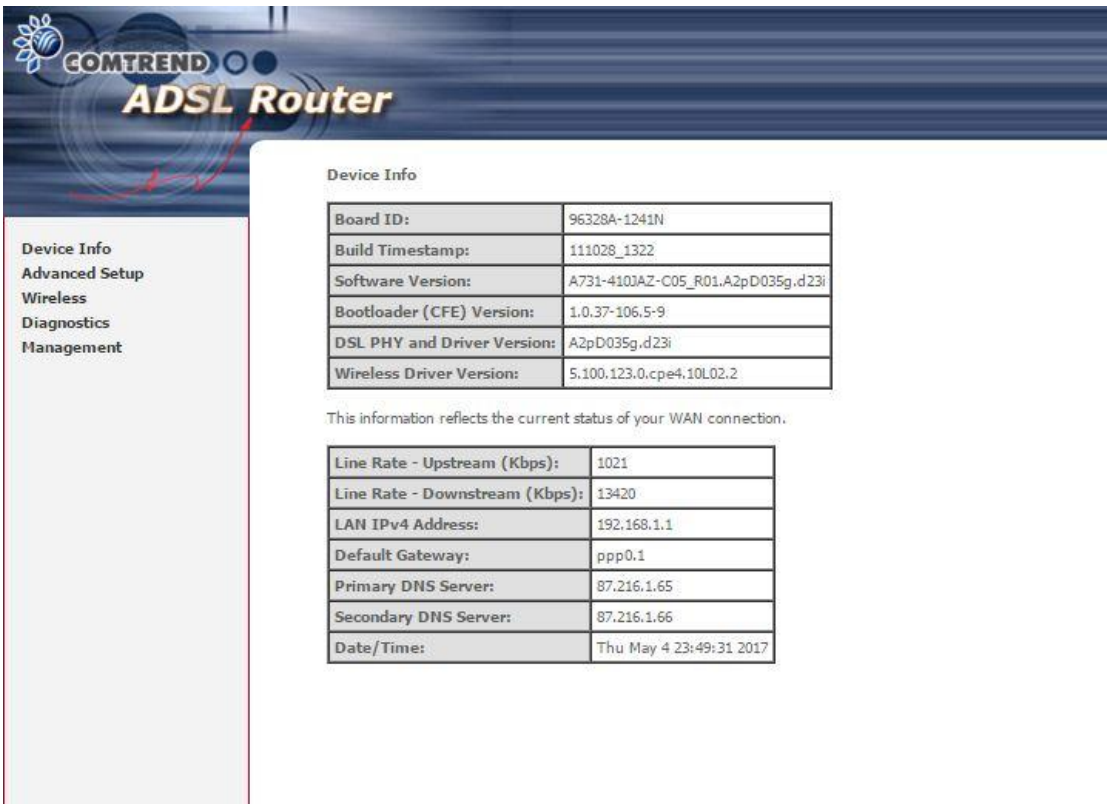


Ilustración 36. Configuración router 1.

A continuación, en el menú lateral, accedemos a *Advanced Setup,* a *NAT,* y a *Virtual Servers.*

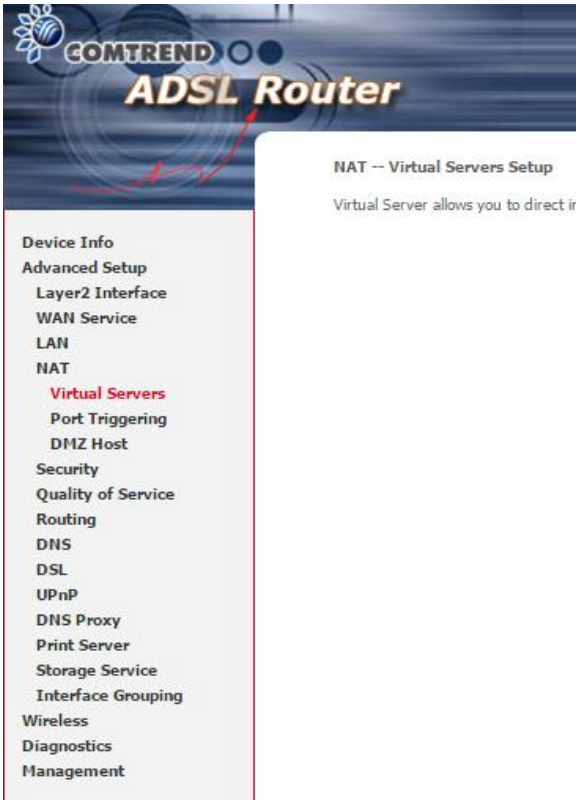
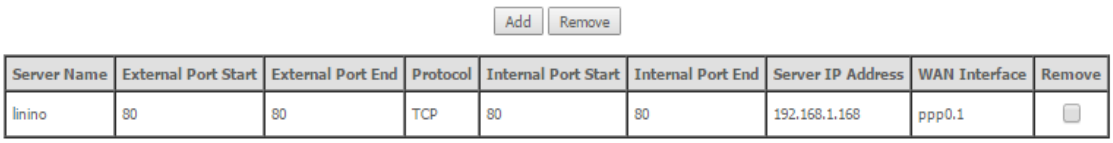


Ilustración 37. Configuración router 2

Una vez aquí nos sale la opción para configurar la redirección del puerto 80 de nuestra IP externa, al puerto 80 de la IP estática que hemos asignado a nuestro Arduino.

Quedaría de la siguiente manera:



Server Name	External Port Start	External Port End	Protocol	Internal Port Start	Internal Port End	Server IP Address	WAN Interface	Remove
linino	80	80	TCP	80	80	192.168.1.168	ppp0.1	

Ilustración 38. Configuración router 3, redirección de puertos

Una vez realizados estos pasos, nuestro Arduino es accesible desde Internet, tecleando en un navegador cualquiera nuestra IP externa.

3.2 Configuración DNS

Debido a que los proveedores de internet suelen modificar la IP externa, vamos a recurrir a un proveedor de dns dinámica gratuito como NO-IP.

Este proveedor nos proporciona un hostname, que quedará asociado a nuestra IP externa.

El alta en esta página es muy sencilla, simplemente debemos proporcionar un correo electrónico y una contraseña, elegir el hostname que queremos e introducir la IP externa de nuestro router.

En este caso he optado por la siguiente:

Proyectoarduino.hopto.org

La cuenta libre de NO-IP, caduca a los 30 días si no actualizamos nuestra IP en este tiempo, para evitar tener que actualizar y mantener nuestra IP manualmente, configuraremos al Arduino para que lo haga por nosotros.

Para ello añadiremos un comando al inicio del sistema, que actualice la IP en NO-IP cuando encendamos el Arduino.

En segundo lugar, vamos a programar una tarea en el cron de Linino que cada 24 horas actualice la IP en NO-IP, para mantener la cuenta activa.

Lo primero que necesitamos es averiguar nuestra IP externa, hay gran cantidad de servicios gratuitos que nos la proporcionan, pero he optado por IPEcho.net, que nos ofrece la posibilidad de devolvernos nuestra IP externa, haciendo una petición HTTP.

De este modo en PHP podemos averiguarlo fácilmente.

Quedaría de la siguiente manera:

```
function retrieve_external_ip() {

$i = do_Curl("http://ipecho.net/plain");

return $i;

}
```

Si la IP es diferente de la que tiene NO-IP guardada, la actualizamos de la siguiente manera:

```
function update_both_ips($ip, $link) {

$aux = "http://" . USER_NO_IP . ":" . PASSWORD_NO_IP . "@dynupdate.no-ip.com/nic/update?hostname=" . HOST_NAME_NO_IP . "&amp;amp;myip=" . $ip;

 $r = do_Curl($aux);logi("Result: " . $r);
```

Por tanto ya hemos resuelto todos los problemas que nos habían surgido en lo referente a la comunicación, ahora nuestro router redirecciona las peticiones externas a nuestra red al puerto 80.

Una vez dados de alta en NO-IP y mediante la utilización de IPECHO, una tarea de CRON, utilizando PHP y peticiones HTTP, podemos mantener nuestra IP actualizada en NO-IP de forma automática.

3.3 Protocolo

Otra de las cosas que debemos definir es el protocolo de comunicación entre nuestra APP y Arduino.

En informática, se entiende como protocolo de comunicación al sistema de reglas que permiten que dos o más entidades de un sistema de comunicación se comuniquen entre ellas.

En nuestro caso, utilizamos una cadena de números que el Arduino conoce y va interpretando.

En primer lugar, indicamos el modo elegido:

00: Recibir temperatura

01: Modo manual

10: Modo automático

11: Modo edición.

En segundo lugar, separados por el siguiente carácter "/", indicamos el circuito al que queremos aplicar el modo anterior si procede.

En caso de ser el modo Recibir temperatura no procede añadir más campos.

Elegimos el circuito con los siguientes números:

01: Habitaciones

10: Zonas comunes

A partir de aquí en función del modo inicial elegido tendremos diferentes opciones.

Por ejemplo si el modo elegido es MANUAL, solo quedaría elegir la temperatura deseada.

Por lo que la cadena a enviar sería la siguiente:

proyectoarduino.hopto.org/arduino/01/01/19

Para seleccionar una temperatura de 19 ºC en las habitaciones.

Si el modo elegido es modo AUTOMÁTICO, debemos seleccionar el programa entre las 4 opciones:

00: Programa Mantenimiento

01: Programa Confort

10: Programa Modo Noche

11: Programa personalizado

De este modo un ejemplo sería el siguiente:

proyectoarduino.hopto.org/arduino/10/01/01

Para aplicar el programa Confort en las habitaciones.

Si el modo elegido es modo PERSONALIZADO, en la cadena a enviar, va incluido el circuito, el día de la semana (1-7), la hora (0-23), y la temperatura deseada para esa franja horaria.

De este modo un ejemplo sería el siguiente:

proyectoarduino.hopto.org/arduino/11/01/3/15/22

Para modificar el programa personalizado y colocar una temperatura de 22 ºC el Miércoles de 15:00 a 16:00 en las habitaciones.

CAPÍTULO 4 MONTAJE FÍSICO

En este capítulo vamos a proceder a explicar el montaje físico de los componentes para conseguir el termostato.

Para conseguir un proyecto mas visual, vamos a a utilizar el relé con sus dos circuitos, para controlar dos bombillas alimentadas a 220 VAC, que simulan los dos circuitos de la caldera, de esa tensión.

De este modo se puede observar perfectamente el conexionado o desconexión en función de la temperatura.

Por otra parte integraremos todos los dispositivos en una caja, la cual se alimenta mediante dos cables, uno de 5V para alimentar el Arduino y otro de 220 VAC para alimentar las bombillas.

De este modo conseguimos un dispositivo modular y portable.

Ilustración 39. Caja integradora

En primer lugar montamos el sensor de temperatura DS18B20 en una protoboard, este sensor, como hemos comentado anteriormente es un sensor muy sencillo que presenta únicamente 3 pines.

Conectamos la alimentación a 5V y el pin de datos a la entrada digital que hemos definido en nuestro Arduino Yun.

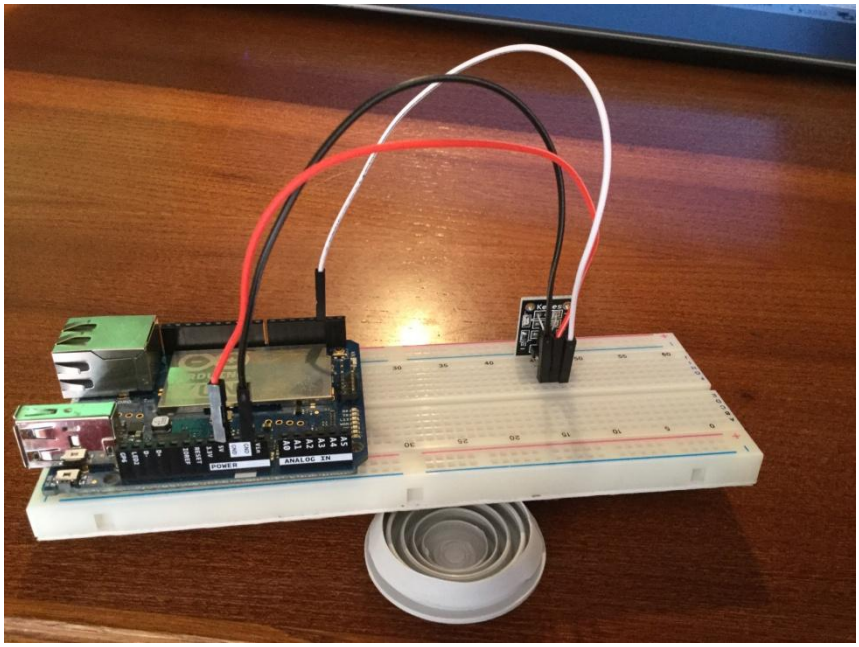


Ilustración 40. Conexión Sensor de Temperatura y Arduino

Una vez comprobado el funcionamiento de estas conexiones, conectamos el relé a alimentación y tierra (de la protoboard) y los dos cables de datos que se encargan de activar y desactivar los circuitos de alterna, a dos salidas del Arduino, como se puede observar en la siguiente imagen.

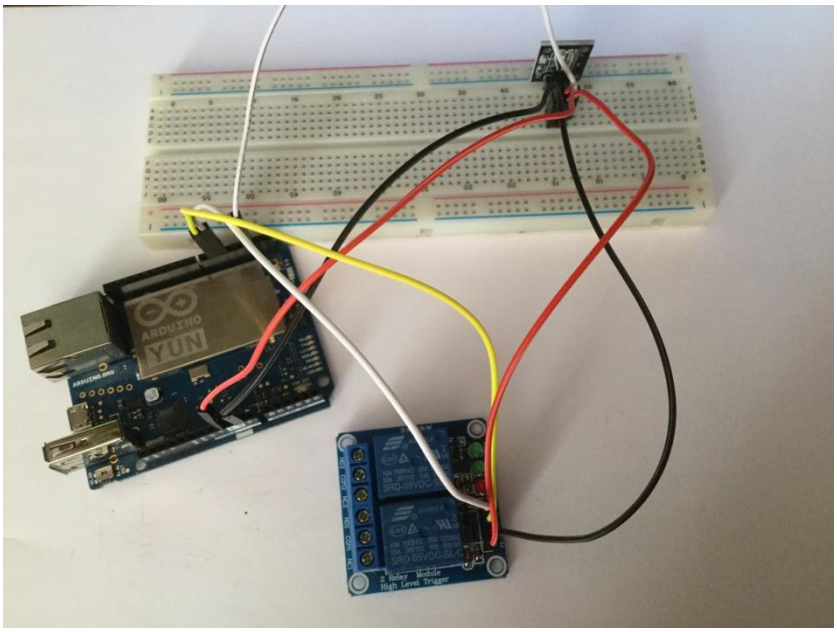


Ilustración 41. Conexión sensor-relé-Arduino

Realizado este circuito, añadimos el otro sensor de temperatura, la conexión de datos es común a ambos sensores ya que la tecnología mediante la que se comunican, el bus 1-Wire, permite instalar tantos sensores como deseemos debido a que cada uno tiene su propia dirección.

A continuación procedemos a integrarlo en nuestra caja exterior.

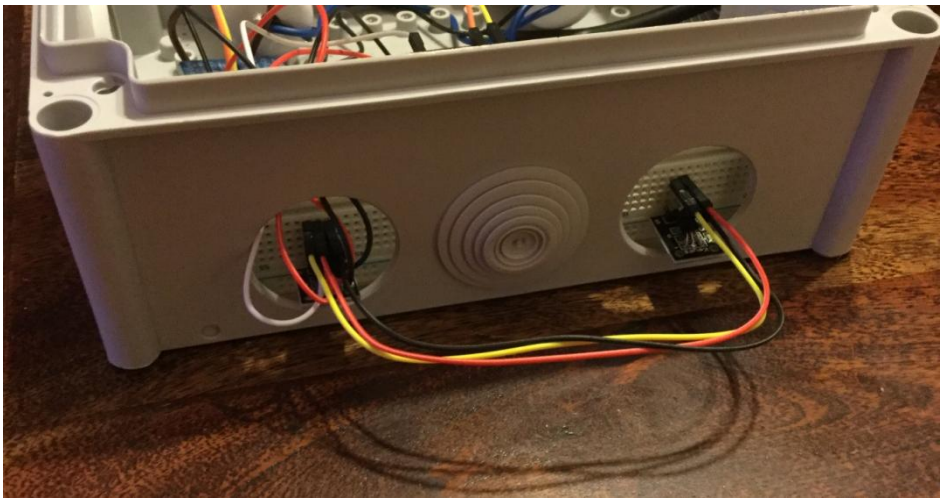


Ilustración 42. Conexión sensores

Por último solo nos queda conectar los casquillos de las bombillas en las aberturas de la caja y realizar el conexionado en alterna.

Este cableado es muy sencillo, únicamente alimentamos el común del relé con Línea del cable enchufado a la corriente y cableamos el contacto normalmente abierto con la bombilla.

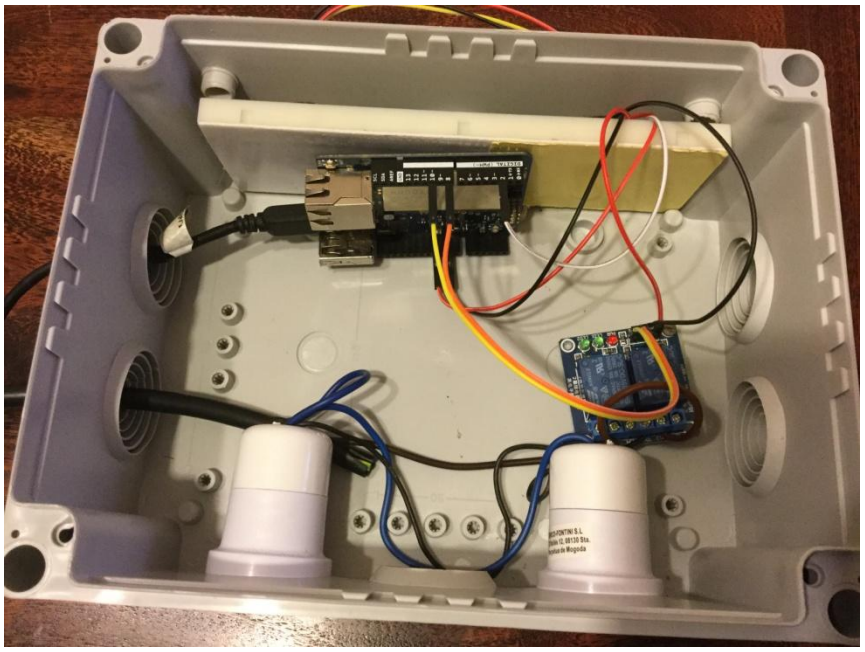


Ilustración 43. Montaje final

Con esto conseguimos el termostato final que se instalará en la vivienda.

El único cambio que se ha de realizar es eliminar las bombillas y conectar en los dos contactos del relé, los 4 cables provenientes de las bornas de la caldera. De este modo al cerrar y abrir el circuito, se conecta o desconecta.

El aspecto final es el siguiente.

Ilustración 44. Resultado final

CAPÍTULO 5 OBJETIVOS Y FUTURAS MEJORAS

En este capítulo vamos a resumir el trabajo realizado a lo largo del proyecto, con la revisión de los objetivos a cumplir y estableceremos las mejoras que se podrían llevar a cabo para complementar el trabajo o llevar a cabo una evolución.

5.1 Objetivos

Durante todo el proyecto hemos estado guiados por una serie de objetivos que eran nuestra meta a la hora de dar por finalizado el proyecto.

En primer lugar el objetivo primordial de un proyecto formativo como este, y así lo hemos enfocado, es el de aprender.

Obtener un conocimiento amplio de los lenguajes de programación C y Java, que nos permitan trabajar con fluidez en ambos, pudiéndonos desenvolver en un futuro ante los problemas que nos puedan surgir en un entorno laboral.

A su vez dentro del objetivo formativo, otra de las misiones era aumentar los conocimientos dentro de la plataforma Arduino, ya conocida pero no trabajada en profundidad.

También hemos podido formarnos con gran detalle en todo lo que rodea al S.O. Android, desde su estructura mas interna, hasta el modo de comunicarse con el exterior, adquiriendo competencias en el desarrollo de aplicaciones complejas mediante el IDE oficial Android Studio.

En segundo lugar, en lo que a objetivos físicos se trataba, se debía lograr un termostato con funcionalidades completas, que permitiera la lectura y tratamiento de los datos adquiridos por sensores de temperatura instalados en diferentes partes de la vivienda y a su vez tomar decisiones de manera inteligente y en tiempo real en función de estos datos.

También se ha logrado el desarrollo de la aplicación Android, con completas funcionalidades y con los modos requeridos, para obtener información y control, así como permitir la programación del modo personalizado.

Por tanto consideramos que los resultados han sido satisfactorios, hemos podido adquirir los conocimientos teóricos y aplicarlos de forma práctica, consiguiendo hacer funcionar el dispositivo.

5.2 Futuras mejoras

Una vez finalizado el proyecto se va a proceder a la instalación del prototipo en una vivienda con el objetivo de testear el mismo en una aplicación real, permitiéndonos obtener un feedback representativo para poder realizar mejoras en el futuro.

Aun así, como explicamos inicialmente, la intención de este proyecto no es darlo por finalizado aquí, se ha conseguido la base, mediante la programación de Arduino, la programación de la APP y la comunicación entre ambas, pero a raíz de este trabajo pueden surgir grandes posibilidades de expansión.

Una vez logrados los objetivos propuestos se nos ocurre la posibilidad de utilizar todo lo logrado hasta ahora para conseguir muchas mas funcionalidades.

¿Por qué no transformar el termostato en una central domótica que controle todos los ámbitos de la vivienda?

Con la estructura generada, tenemos la base para realizar cualquier tarea que se nos ocurra.

Gran cantidad de casos se podrían tratar con la misma lógica, lectura de datos mediante los sensores correspondientes, tratamiento en el Arduino, información a la APP y control mediante esta aplicación de las salidas deseadas.

Se nos ocurren varias como pueden ser el control de los motores encargados de subir y bajar las persianas en función de los sensores de luminosidad exteriores y mediante el control manual con la app o el riego del jardín, controlando la humedad del terreno mediante higrómetros, sensores que miden la humedad de la tierra y nos permitirían realizar un riego sectorizando las zonas en las que es necesario permitiendo un ahorro de agua y energía.

Todas estas funcionalidades se pueden añadir de forma sencilla mediante la copia y ligera modificación de ciertas funciones de programación.

ANEXO 1 INFORMACION CALDERAS.

A continuación vamos a realizar un pequeño estudio sobre los tipos de caldera que se pueden implementar en la vivienda, así como sus métodos de control.

Los tipos de calefacción se pueden dividir según la fuente de energía (biomasa, geotérmica, solar, eléctrica y gas) o según el aparato o sistema a partir del cual se obtiene el calor (suelo radiante, bomba de aire, eléctrica por acumuladores, eléctrica por convectores, emisores termoeléctricos y calderas con radiadores de agua).

Tipos de calefacción

Calefacción de gas

Es una de las fuentes de energías más empleadas en los hogares españoles. Se trata de una energía limpia, eficaz, que no contamina. Tanto para la calefacción, cocina y la producción de agua caliente se puede elegir entre 3 tipos de combustible: gas natural, gasóleo C o gas propano.

El primero, resulta perfecto porque no hay que preocuparnos por su almacenamiento ni distribución, sin embargo, su suministro no suele llegar lejos de las ciudades. El segundo, el gasóleo C, resulta algo más peligroso, ya que se debe almacenar en tanques dentro de casa; también es más contaminante y sucio, aunque es una buena elección para calentar hogares grandes. El gas propano es perfecto para casas grandes o viviendas en pequeñas poblaciones, ya que tiene una potencia calorífica superior al gas natural y similar al gasóleo; se puede almacenar en el exterior de la casa, en recipientes pequeños o en depósitos, lo que lo hace menos seguro que el gas natural.

Cualquiera de estos tres combustibles requiere el uso de radiadores, que permiten un calor homogéneo en toda la casa.

Calefacción eléctrica por acumulación

Se trata de uno de los sistemas de calefacción más habituales, debido a su instalación sencilla, su mantenimiento y seguridad. La electricidad se convierte en calor gracias a las resistencias eléctricas que hay dentro de cada calefactor eléctrico, a través de las que pasa la corriente, convirtiendo la electricidad en calor.

La calefacción eléctrica se puede encontrar en diversos sistemas, según las necesidades, pero sea cual sea el sistema, es una energía que no consume oxígeno, ni emite gases contaminantes. Tampoco necesita un mantenimiento continuo, excepto limpieza periódica del filtro de aire, aunque para su soporte es necesario contar con una instalación eléctrica apropiada.

A la hora de decantarse por este sistema se debe pensar que, aunque su instalación es mucho más barata que un sistema de calefacción a gas, ya que no se requiere de obra alguna para su colocación, la tarifa eléctrica resulta más costosa que el gas natural corriente.

La calefacción eléctrica es buena opción en regiones cálidas, en las que no se necesita mucha potencia ni tiempo para calentar la casa. También para segundas viviendas, donde el uso se limita a cortos periodos de tiempo, y en hogares pequeños, que se calientan rápidamente.

Calefacción eléctrica por convectores

Este tipo de calefacción funciona mediante una resistencia que calienta el aire que circula por el interior de los convectores. En este sistema, perfecto para hogares ubicados en zonas cálidas, el agua caliente se obtiene mediante un termo.

Entre sus ventajas: una instalación barata, sin obras, y un suministro cómodo del agua caliente. Sus inconvenientes: el coste de su funcionamiento suele ser caro y el termo para el agua caliente consume bastante mientras está encendido, aunque no se use.

Emisores termoeléctricos

Los emisores termoeléctricos son radiadores de aceite. Este sistema logra la transmisión de calor a través de un aceite térmico que se calienta mediante una resistencia eléctrica blindada de un acero especial.

Cada radiador es independiente y se puede enchufar en cualquier lugar sin obras, ya que no tiene ni caldera ni tuberías. Para lograr una temperatura constante y homogénea, los aparatos llevan incorporado un termostato y un programador, que ayudan a ahorrar energía.

Otra de las ventajas de este sistema es que, tras apagar los radiadores, estos siguen irradiando calor durante horas. También son más seguros que los radiadores de agua, ya que el aceite no produce ninguna presión interna. La desventaja es que, si se necesitan muchos radiadores, resulta un sistema caro y se puede necesitar contratar más potencia de luz.

Caldera con radiadores de agua

Es el sistema más utilizado en España. El calor se produce, mediante la quema de combustibles como el gas natural, en una caldera situada en un local específico o en el interior de la vivienda y se distribuye a unos elementos terminales (radiadores) mediante el agua, emitiendo el calor a aquellos espacios que lo requieren.

La elección del agua como caloportador se debe a que es una sustancia barata, común en todas las edificaciones y su calor específico es mayor que el de otras sustancias, por lo que requiere un caudal menor para transportar la misma cantidad de calor.

Al estar la caldera situada en otro espacio, puede airearse libremente sin problemas. Esta puede servir a un solo usuario (calefacción centralizada individual), a todo un edificio (calefacción centralizada colectiva), a una barriada e incluso a una ciudad (calefacción urbana).

Suelo radiante

El suelo radiante es uno de los sistemas de calefacción más confortables para los climas fríos. Consiste en una instalación cables eléctricos o tuberías por las que circula agua a temperatura elevada, ocultos bajo el suelo de la vivienda. Estos desprenden calor, que se propaga hacia arriba, calentando el suelo y el ambiente de la vivienda.

Entre las ventajas de este sistema, decir que permite ahorrar entre el 10% y el 30% el consumo de calefacción, proporciona un calor agradable y uniforme sin resecar el ambiente y permite una imagen más estética al no haber aparatos de calefacción en las paredes. Se trata de un sistema seguro, muy recomendable si hay niños en casa. La instalación aporta un aislamiento acústico y térmico adicional y necesita poco mantenimiento.

Entre sus desventajas destacan la elevada inversión inicial y las obras que comporta (debe levantarse el pavimento de la vivienda). Además, decir que hasta que alcanza la temperatura deseada tarda cierto tiempo, por lo que se recomienda para residencias habituales.

Bomba de calor

La bomba de calor permite tener calefacción en invierno y aire acondicionado en verano en un solo aparato. El proporcionar estas dos opciones en un único sistema, abarata la inversión y simplifica la instalación. La gran variedad de marcas y modelos hacen posible su instalación en distintos lugares.

Es un sistema eficiente, ya que consume menos energía hasta alcanzar la temperatura deseada, si bien también el calor se dispersa antes. Por eso, es recomendable en climas cálidos o templados con inviernos suaves. Entre sus desventajas, el ruido del ventilador puede resultar algo molesto y el elevado precio de la instalación de la bomba de calor por conductos.

La bomba de calor requiere de pocos cuidados, excepto la limpieza periódica del filtro de aire. Cualquier modificación en la instalación debe realizarla siempre un instalador especializado, previo estudio.

Una vez conocidos las principales opciones vamos a observar cuales son las opciones más utilizadas en España.

Como se puede observar en el siguiente informe elaborado por el ministerio de Industria y Energía en España Predomina en gran medida la utilización de combustibles petrolíferos y Gases como combustible para la obtención de Calefacción y Agua caliente.

IDAE

7ª Edición. Junio 2016
SECRETARÍA GENERAL
Departamento de Planificación y Estudios

INFORME ANUAL DE CONSUMOS ENERGÉTICOS. UNIDADES COMERCIALES. AÑO 2014

Consumo de Energía Final: Sector Residencial/Hogares.

Tipo de Uso	Carbón	Productos Petrolíferos			Gases [3]	Renovables					Energía Eléctrica	TOTAL	Consumo Total según Usos	
		GLP	Combustibles Líquidos [2]	TOTAL Productos Petrolíferos		Biomasa	Solar	Geotermia	Biocarburantes	TOTAL Renovables			Térmicos	Eléctricos
Unidad de medida:	ktep	ktep	ktep	ktep	ktep	ktep	ktep	ktep	ktep	ktep	ktep	ktep	ktep	ktep
Calefacción	75	401	1.476	1.876	1.433	2.459	15	5,36	--	2.479	448	**6.311**	5.863	448
ACS	6	474	133	607	1.324	52	188,3	3,03	--	243	454	**2.634**	2.179	454
Cocina	11	191	--	191	337	27	--	--	--	27	565	**1.131**	566	565
Iluminación	--	--	--	--	--	--	--	--	--	--	714	**714**	--	714
Aire Acondicionado	--	--	--	--	--	--	--	2,26	--	2	142	**144**	2	142
Electrodomésticos	--	--	--	--	--	--	--	--	--	--	3.758	**3.758**	--	3.758
Otros Usos	--	--	--	--	--	--	--	--	0,88	1	--	**1**	1	--
CONSUMO TOTAL DE LOS HOG	**92**	**1.066**	**1.608**	**2.674**	**3.094**	**2.537**	**203**	**10,65**	**0,88**	**2.752,20**	**6.081**	**14.692**	**8.611**	**6.081**
CONSUMO TOTAL RESIDENCIA	**92**	**1.066**	**1.629**	**2.695**	**3.094**	**2.537**	**203**	**10,65**	**10,65**	**10,65**	**6.081**	**14.713**	**5.891**	**6.081**

Ilustración 45. Consumos energéticos

Por otra parte se puede observar en este gráfico más detallado que el gasto de energía destinado a la obtención de calefacción es un 47% del total.

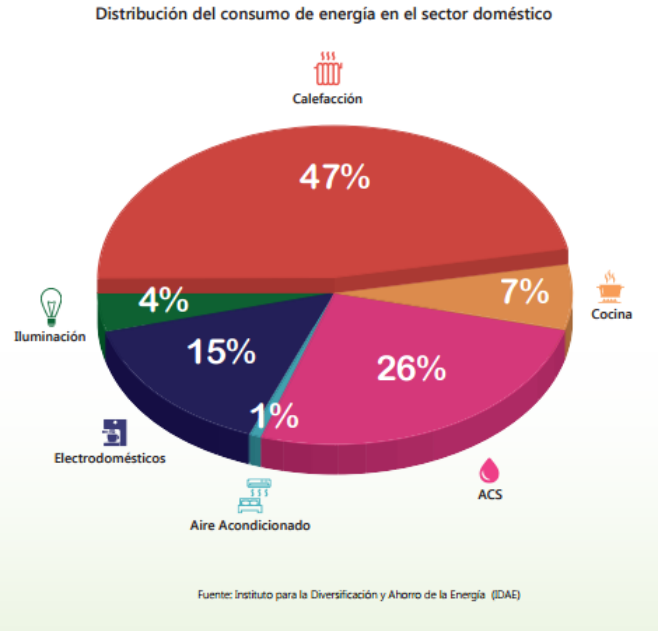


Ilustración 46. Distribución energía sector doméstico

Por esta razón tiene un gran sentido intentar realizar un control completo de la calefacción, con un termostato de gran precisión y una separación por zonas, que permita no malgastar energía innecesariamente, calentando estancias que no se utilizan, evitar cambios bruscos de temperatura mediante un encendido programado que consiga un calentamiento progresivo hasta llegar a obtener el máximo confort a la hora de utilización.

Por tanto vamos a basar nuestro proyecto en la instalación del termostato en una caldera individual que transmite el calor a la vivienda mediante radiadores ya que un 41% de los sistemas de calefacción en España son individuales con radiadores.

Instalación caldera

En concreto nos vamos a basar en una caldera de la marca Domusa Teknik, marca de reconocido prestigio en el mundo de la climatización.

En concreto tomaremos como ejemplo el modelo Avanttia H pero este funcionamiento es extrapolable a la gran mayoría de calderas de cualquier marca.

El montaje que se debe realizar en la instalación hidráulica es el siguiente:

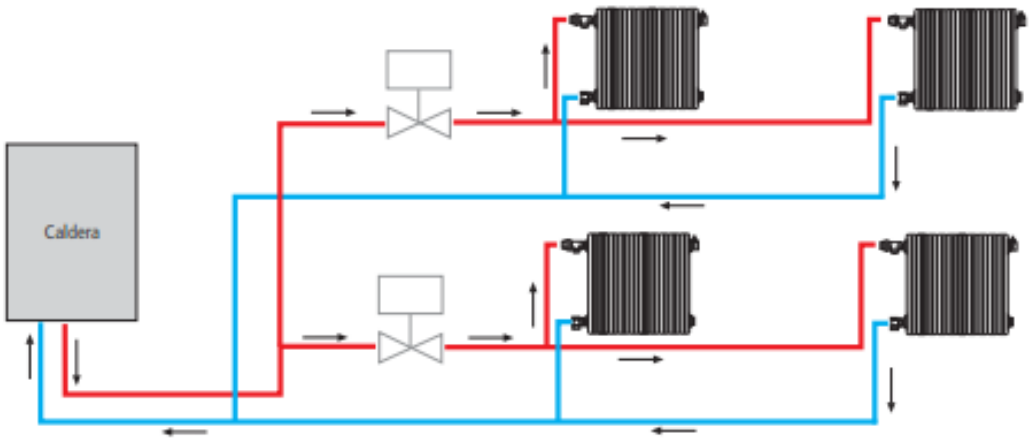
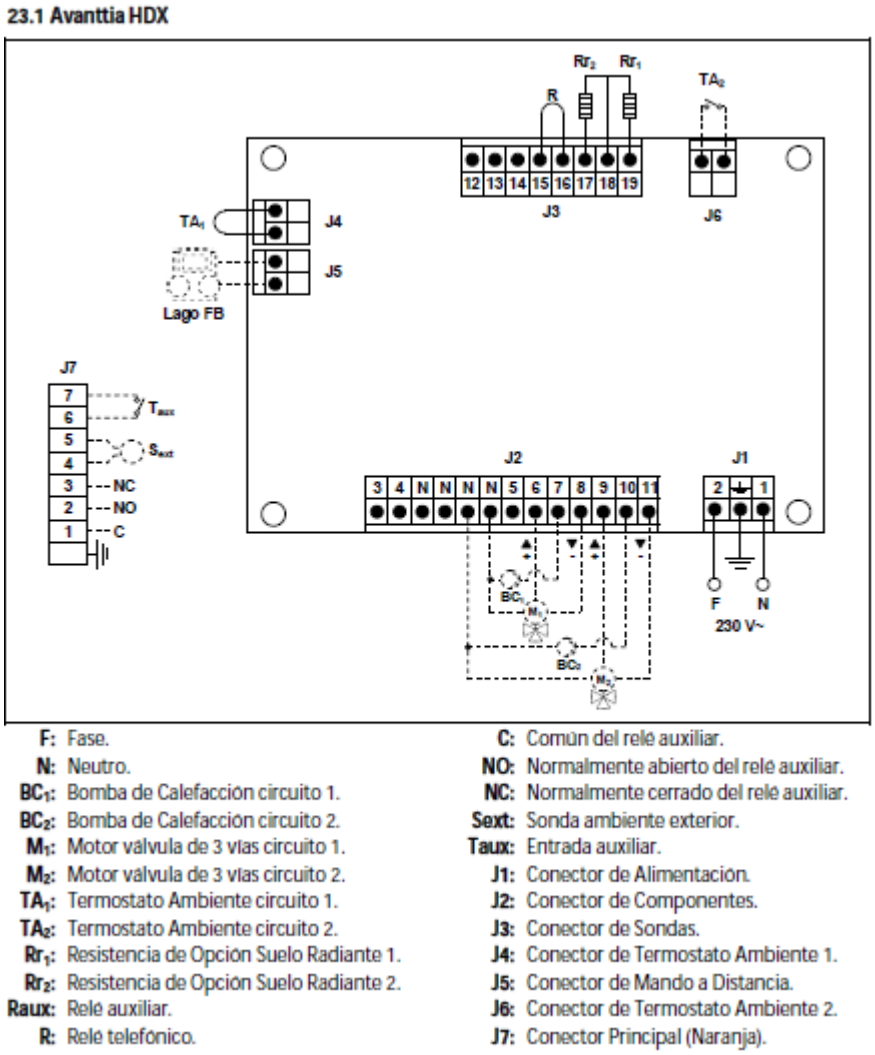


Ilustración 47. Instalación hidráulica caldera.

Se trata de una instalación sencilla, en cada circuito de calefacción se deberá instalar una válvula motorizada.

La válvula motorizada instalada en el circuito de calefacción Nº 1, deberá ser conectada eléctricamente entre las bornas N y 7 de la regleta de conexiones J2, mientras que la del circuito de calefacción Nº 2, deberá ser conectada entre las bornas N y 10.

Esto se observa con más claridad en la siguiente imagen donde se muestra la caja de conexiones de la caldera.



F: Fase.
N: Neutro.
BC1: Bomba de Calefacción circuito 1.
BC2: Bomba de Calefacción circuito 2.
M1: Motor válvula de 3 vías circuito 1.
M2: Motor válvula de 3 vías circuito 2.
TA1: Termostato Ambiente circuito 1.
TA2: Termostato Ambiente circuito 2.
Rr1: Resistencia de Opción Suelo Radiante 1.
Rr2: Resistencia de Opción Suelo Radiante 2.
Raux: Relé auxiliar.
R: Relé telefónico.
C: Común del relé auxiliar.
NO: Normalmente abierto del relé auxiliar.
NC: Normalmente cerrado del relé auxiliar.
Sext: Sonda ambiente exterior.
Taux: Entrada auxiliar.
J1: Conector de Alimentación.
J2: Conector de Componentes.
J3: Conector de Sondas.
J4: Conector de Termostato Ambiente 1.
J5: Conector de Mando a Distancia.
J6: Conector de Termostato Ambiente 2.
J7: Conector Principal (Naranja).

Ilustración 48. Caja de conexiones caldera.

De este modo, la gran mayoría de calderas se encargan de gestionar la apertura y cierre de las válvulas motorizadas, en función de la información que les llega de los termostatos.

En nuestro caso únicamente deberemos colocar los relés entre las regletas J6 y J4 de la caja de conexiones para conseguir un manejo completo de los circuitos, teniendo en cuenta que tienen que soportar una tensión de 230 V que es a la que trabaja la caldera.

ANEXO 2 ARDUINO

Como hemos hablado en la memoria, Arduino es una plataforma que nos permite la realización de prototipos electrónicos.

Hemos hablado en profundidad sobre el hardware del sistema, pero en este Anexo nos centraremos en el Software.

2.1 Instalación del IDE.

En primer lugar vamos a descargar e instalar el entorno de desarrollo de Arduino (IDE), que nos facilita en gran manera la programación del mismo.

Para ello debemos ir a la página de descarga de Arduino.

Una vez allí bajamos la versión más reciente del IDE, que actualmente es la versión 1.8.2 para el sistema operativo que deseemos utilizar.

En mi caso trabajaré con Windows.

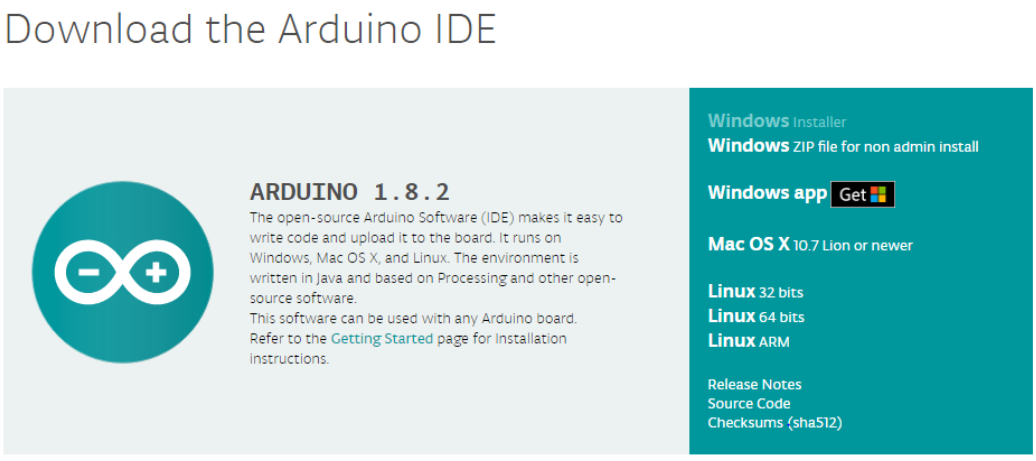


Ilustración 49. Descarga Arduino

Una vez descargado, procedemos a instalar el IDE.

Abrimos el ejecutable descargado y vamos realizando la instalación, siguiendo los pasos indicados por el instalador hasta que finalice.

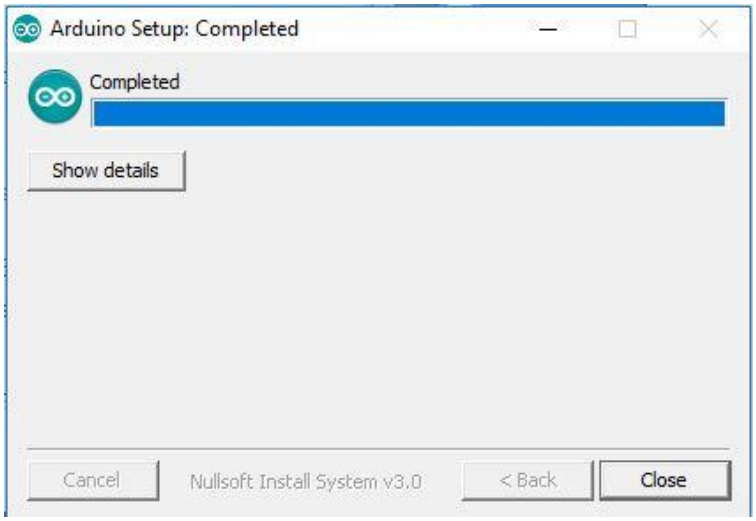


Ilustración 50. Arduino instalación completada

2.2 Entorno de desarrollo

Una vez instalado adecuadamente, se nos muestra la pantalla del entorno de desarrollo.

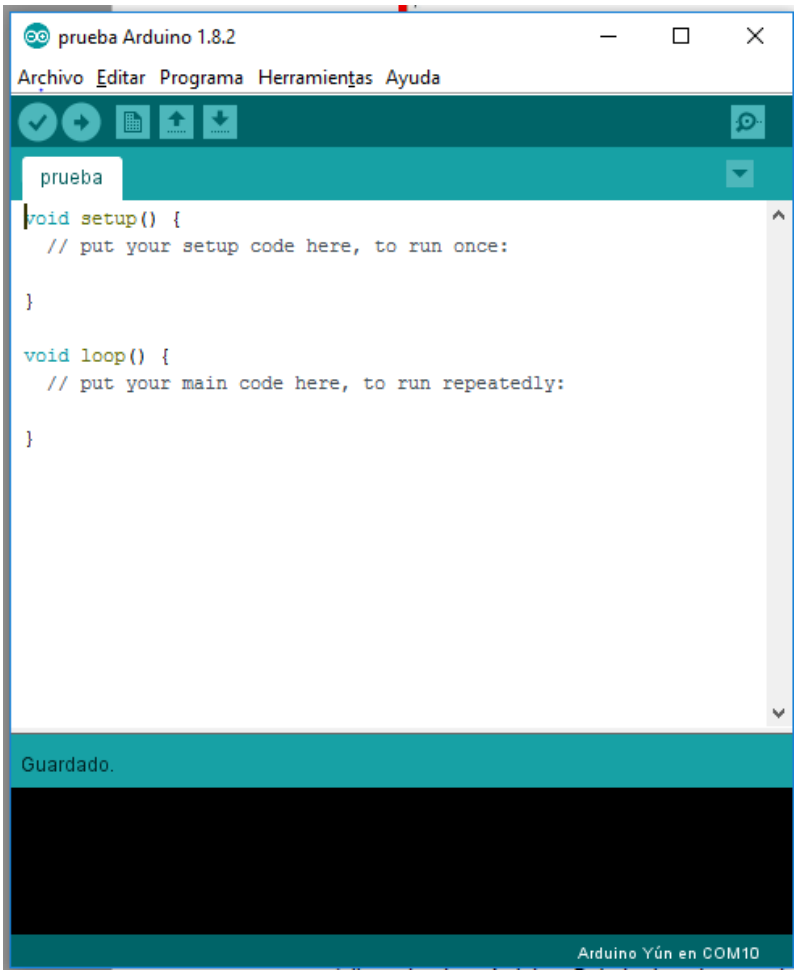


Ilustración 51. IDE Arduino.

En la ilustración 51 podemos observar la pantalla principal. Esta pantalla se puede dividir en varias partes.

En la parte superior viene una barra de estado con algunas de las acciones más comunes a realizar.

La parte central es el entorno de programación. El editor de texto donde escribiremos nuestro programa.

Por último en la parte inferior vemos una consola en la cual nos aparece información sobre la compilación, errores, compilación exitosa, etc.

2.3 Configuración del IDE

Una vez instalado, debemos realizar una serie de configuraciones para poder transferir los sketch a nuestro microprocesador.

Tenemos dos opciones, al ser un Arduino YUN, lo podemos conectar mediante la red wifi o mediante USB.

Procederemos a conectarlo mediante la red WIFI ya que será necesario para el funcionamiento posterior como termostato.

Una vez conectado el Arduino a través de nuestra red wifi procedemos a crear programa requerido.

ANEXO 3 ANDROID

Como hemos hablado en la memoria, vamos a trabajar con el entorno de desarrollo Android Studio, ya que es el IDE oficial designado por Android.

En este anexo vamos a describir los pasos a realizar para disponer en nuestro PC del entorno y las herramientas necesarias para comenzar a programar aplicaciones en la plataforma Android.

3.1 Instalación de Java.

En primer lugar debemos tener instalado el JDK (Java Development Kit), un software que provee herramientas de desarrollo para la creación de programas en Java.

Este software puede se puede obtener de manera gratuita desde la web de ORACLE.

Java SE Development Kit 8u131

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

Thank you for accepting the Oracle Binary Code License Agreement for Java SE; you may now download this software.

Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	77.87 MB	jdk-8u131-linux-arm32-vfp-hflt.tar.gz
Linux ARM 64 Hard Float ABI	74.81 MB	jdk-8u131-linux-arm64-vfp-hflt.tar.gz
Linux x86	164.66 MB	jdk-8u131-linux-i586.rpm
Linux x86	179.39 MB	jdk-8u131-linux-i586.tar.gz
Linux x64	162.11 MB	jdk-8u131-linux-x64.rpm
Linux x64	176.95 MB	jdk-8u131-linux-x64.tar.gz
Mac OS X	226.57 MB	jdk-8u131-macosx-x64.dmg
Solaris SPARC 64-bit	139.79 MB	jdk-8u131-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	99.13 MB	jdk-8u131-solaris-sparcv9.tar.gz
Solaris x64	140.51 MB	jdk-8u131-solaris-x64.tar.Z
Solaris x64	96.96 MB	jdk-8u131-solaris-x64.tar.gz
Windows x86	191.22 MB	jdk-8u131-windows-i586.exe
Windows x64	198.03 MB	jdk-8u131-windows-x64.exe

Ilustración 52. Descarga Java

Como se puede observar en la Ilustración 52 la última versión es Java 8. Por lo que descargamos la versión del Sistema Operativo con el que estemos trabajando. En mi caso Windows.

A continuación procedemos a instalar el software, siguiendo los pasos que nos indica el instalador.

Ilustración 53. Instalación de java 1

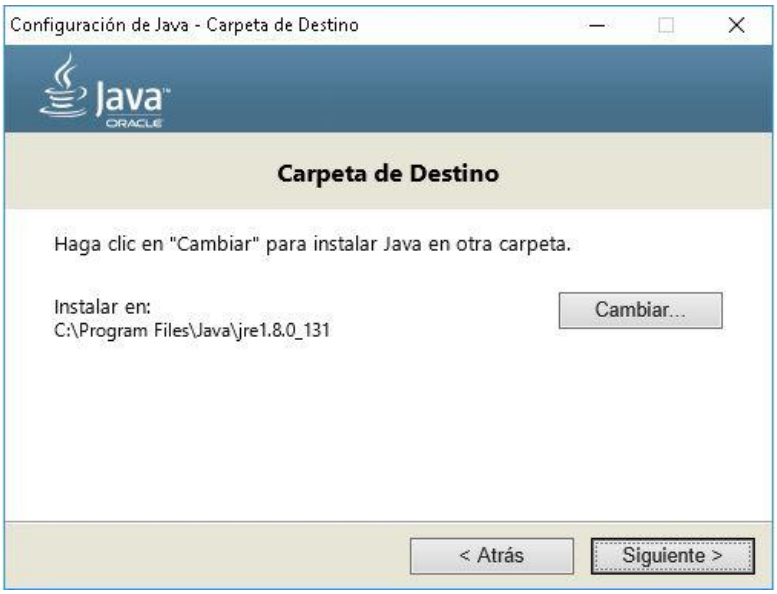


Ilustración 54. Instalación de java 2

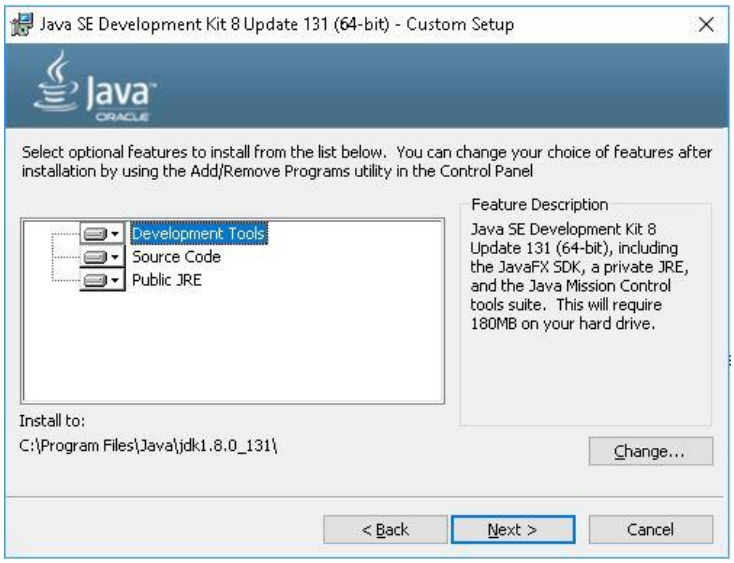


Ilustración 55. Instalación de java 3

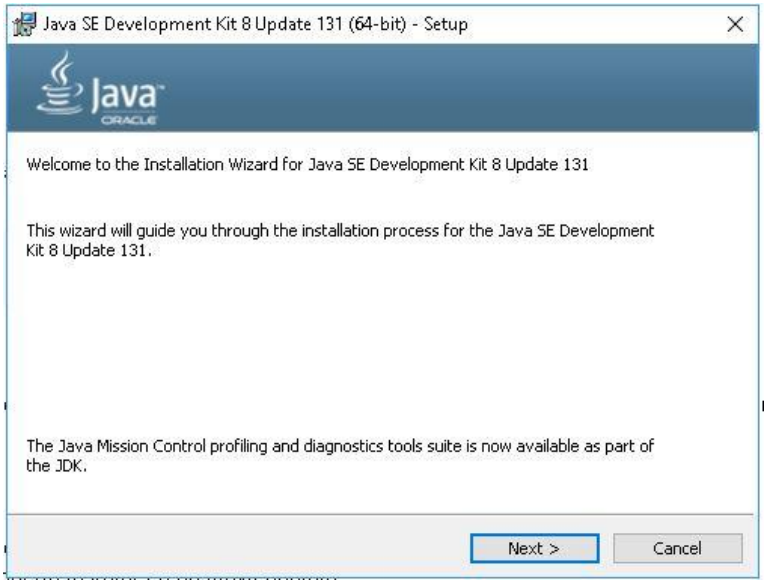


Ilustración 56. Instalación de java 4

Una vez instalado este software debemos crear una variable de entorno para asegurarnos que se sincroniza correctamente con Android Studio.

Crearemos una nueva variable de entorno llamada JAVA_HOME y cuyo valor sea la ruta donde hemos instalado el JDK, en nuestro caso C:\Program Files\Java\jdk1.8.0_131

Para añadir una variable de entorno del sistema en Windows debemos acceder al Panel de Control / Sistema y Seguridad / Sistema / Configuración avanzada del sistema / Opciones Avanzadas / Variables de entorno.

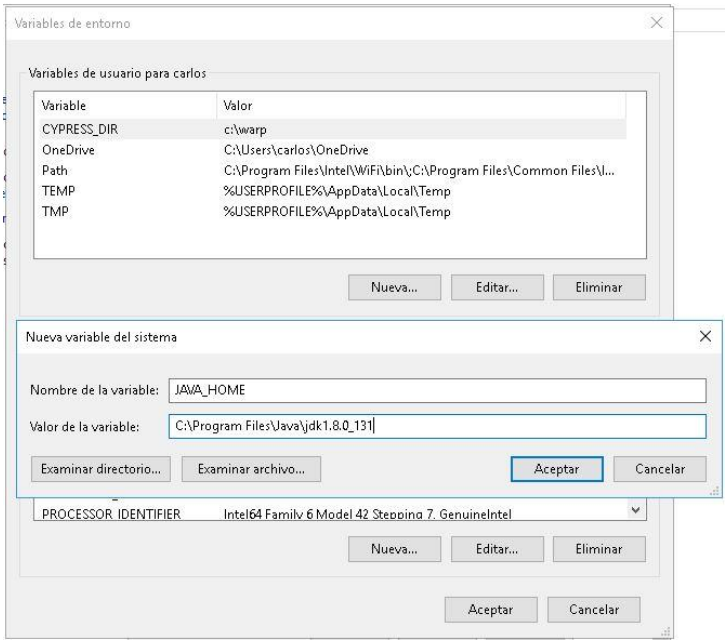


Ilustración 57. Crear variable de entorno

3.2 Instalación de Android Studio y el SDK de Android.

Una vez instalado el entorno Java, debemos instalar Android Studio. Es un software gratuito, por lo que accedemos a la página oficial y procedemos a descargarlo.

Ilustración 58. Descarga Android Studio

Para instalar la aplicación ejecutamos el instalador y seguimos el asistente aceptando todas las opciones seleccionadas por defecto como se pueden observar en las siguientes imágenes.

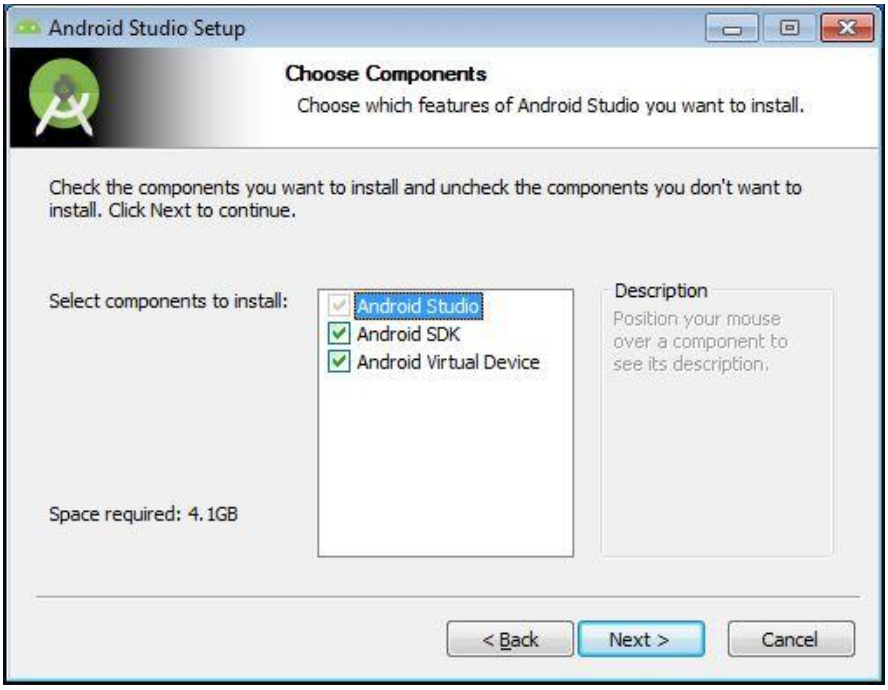


Ilustración 59. Instalación Android Studio 1

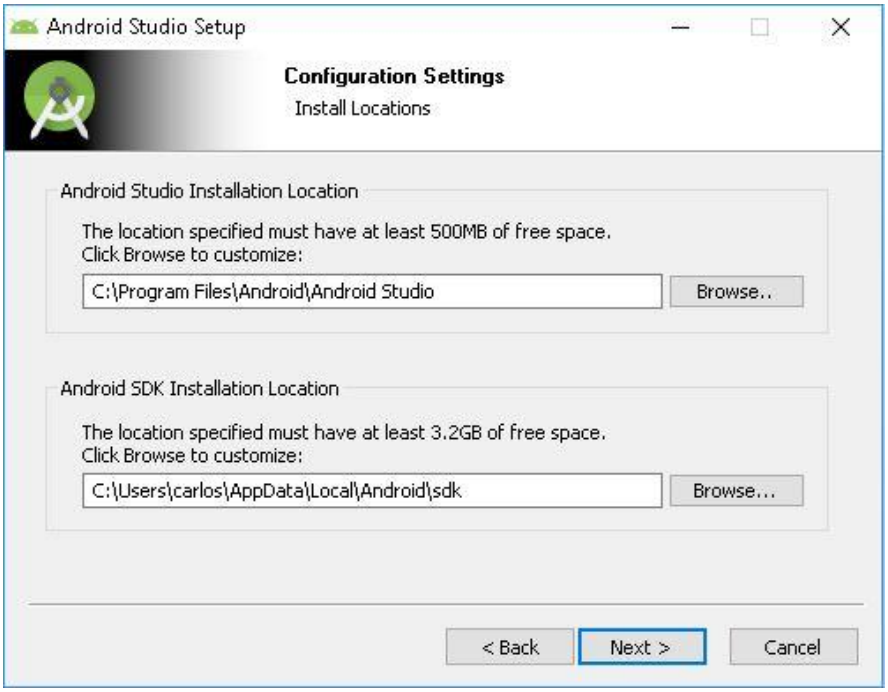


Ilustración 60. Instalación Android Studio 2

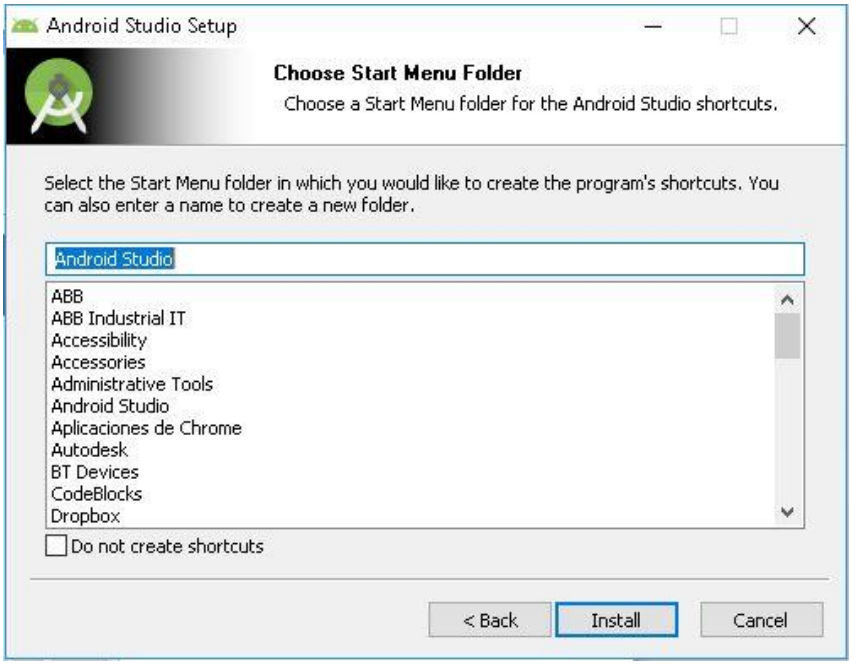


Ilustración 61. Instalación Android Studio 3

Una vez finalizada la instalación, si todo ha ido de manera correcta nos aparece la siguiente pantalla, en la cual podemos abrir por primera vez el programa.

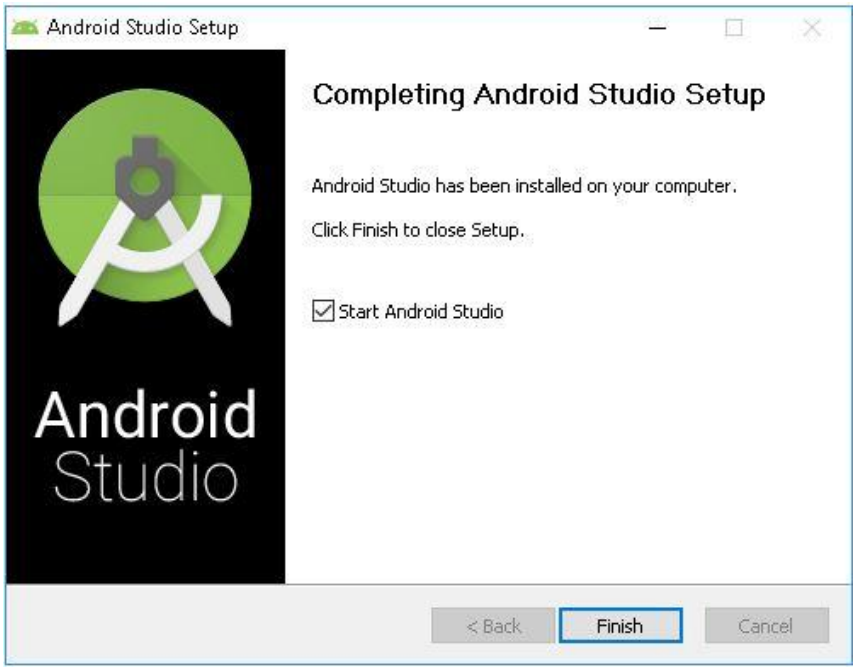


Ilustración 62. Instalación Completada Android Studio

Durante el proceso se instalará el SDK de Android, algunos componentes adicionales para el desarrollo sobre la plataforma, y el entorno de desarrollo Android Studio.

Una vez instalado, la primera vez que abrimos el programa debemos realizar una configuración general del entorno mediante el asistente de inicio de Android Studio.

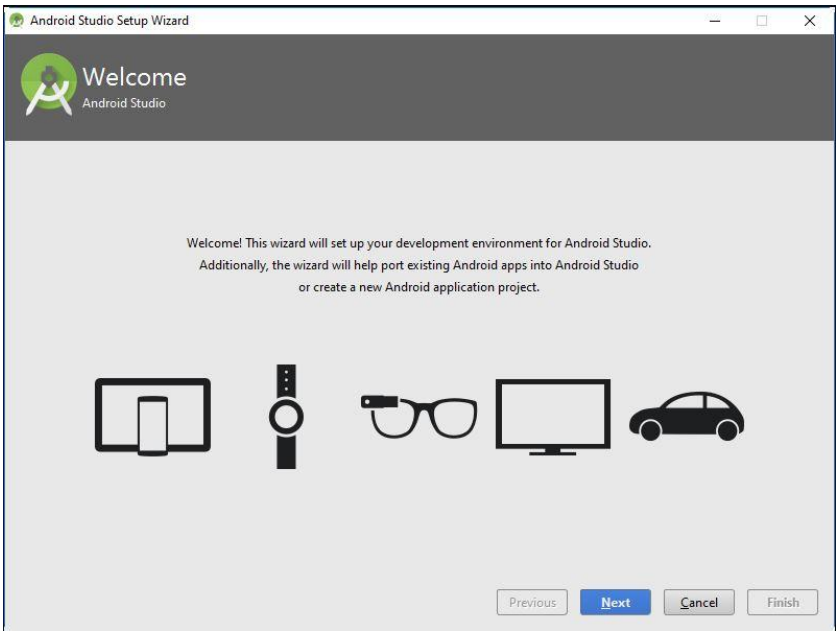


Ilustración 63. Configuración Android Studio 1

A continuación nos aparece el tipo de configuración que queremos elegir.

Hemos optado por realizar la instalación por defecto para evitar problemas.

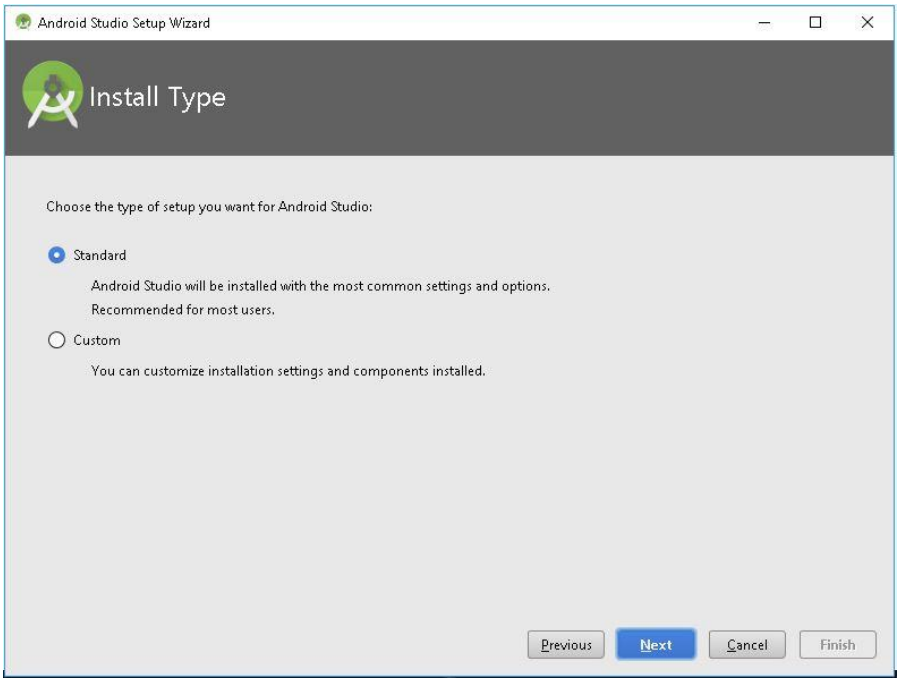


Ilustración 64. Configuración Android Studio 2

Una vez pulsamos el botón Next, comienza la descarga e instalación de los componentes necesarios.

Una vez finalizado el asistente de inicio nos aparecerá la pantalla de bienvenida de Android Studio.

3.3 Conocimientos básicos Android

Para comenzar a trabajar con la aplicación debemos tener unos conocimientos básicos sobre el sistema operativo Android, terminología y estructura de un proyecto.

Como comentamos en la memoria principal, Android es un sistema operativo basado en el núcleo Linux.

Respecto al lenguaje de programación necesario para empezar a programar, Java es el principal, ya que es la base sobre la que se construyen todas las Apps para Android.

A pesar de que el Java que se usa para Android no es exactamente el original, se diferencia muy poco.

A continuación definiremos algunos de los términos básicos que nos permitirán entender las explicaciones posteriores.

Proyecto: La entidad *proyecto* es única, y engloba a todos los demás elementos. Dentro de un proyecto podemos incluir varios *módulos*, que pueden representar aplicaciones distintas, versiones diferentes de una misma aplicación, o distintos componentes de un sistema (aplicación móvil, aplicación servidor, librerías, etc). En la mayoría de los casos, trabajaremos con un proyecto que contendrá un sólo módulo correspondiente a nuestra aplicación principal.

Módulo: cómo se explica en la definición anterior, un módulo puede representar versiones diferentes de una misma aplicación o distintos componentes de un sistema.

Activity: Corresponde a una ventana o pantalla de una aplicación de escritorio.

Un Activity es una clase donde mostraremos Views (Vistas) para generar la interfaz de usuario y responder a acciones que se realicen sobre ella.
Una aplicación, generalmente cuenta con múltiples actividades independientes, capaces de llamarse entre sí, pasándose parámetros y recibiendo respuestas, de modo que su funcionamiento sea un conjunto.
A cada Activity se le asigna una ventana sobre la que dibujar una interfaz de usuario.
Este contenido se muestra de un modo ordenado mediante vistas, que son objetos que implementan la clase View (Vista). Esta clase es el nexo de unión

entre las Activity y el usuario, ya que se encargan de recibir los eventos realizados y reaccionar ante ellos.

Views: Las vistas son los elementos que componen la interfaz de usuario de una aplicación, son empleados específicamente para dibujar contenido en la pantalla del dispositivo Android.
View es la clase base de widgets, que se utilizan para crear componentes de interfaz de usuario interactivos (botones, campos de texto, etc.)

Layout: los layout son elementos no visuales, dedicados a controlar la distribución, posición y dimensión de los controles que se insertan en su interior.
Existen una gran variedad de layouts, en función de su posicionamiento en la pantalla.
Intents: Los intent sirven para invocar componentes, en android entendemos por componentes las activities, por tanto, la comunicación entre los distintos componentes y aplicaciones en Android, como por ejemplo lanzar una actividad, lanzar un servicio, enviar un anuncio broadcast, etc, se realiza mediante Intents.

3.4 Creación primer proyecto.

Lo primero que nos encontramos al iniciar el entorno de desarrollo es la siguiente pantalla.

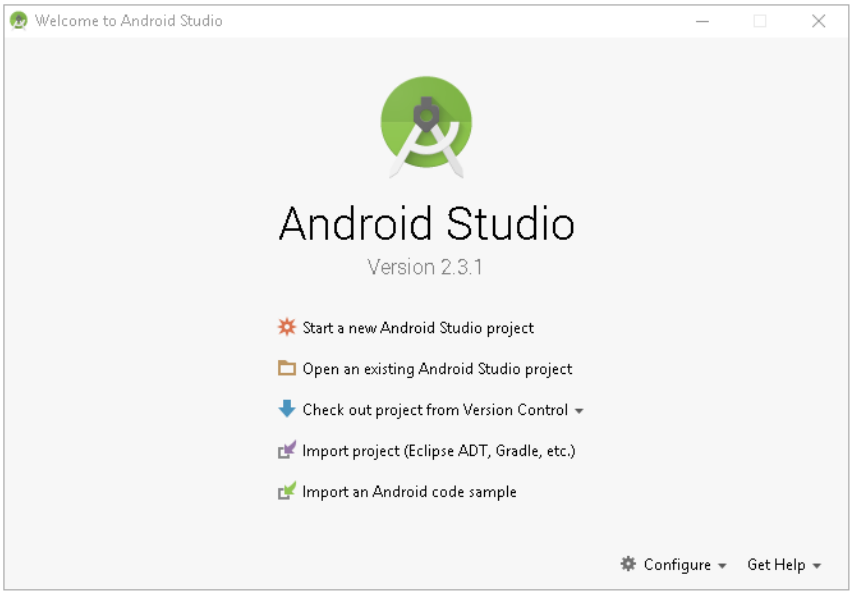


Ilustración 65. Inicio proyecto Android Studio.

Vamos a configurar un nuevo proyecto para entender el funcionamiento del entorno de desarrollo y poder comprobar el funcionamiento.

Iniciamos un nuevo proyecto y nos muestra la siguiente pantalla, en la cual ponemos nombre al programa y pulsamos el botón Next.

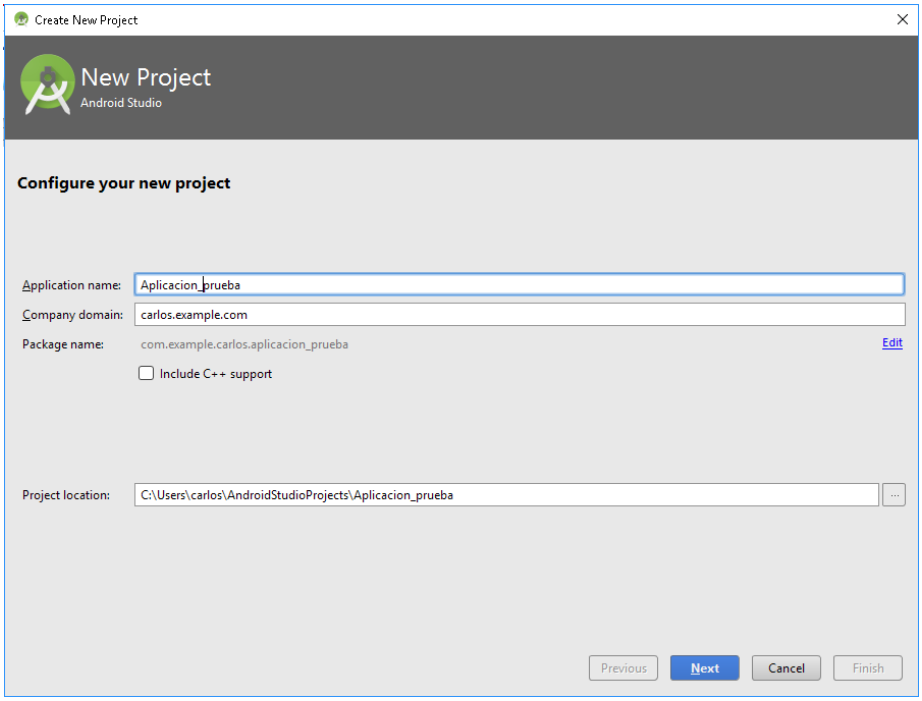


Ilustración 66. Nuevo proyecto Android Studio

En esta pantalla debemos configurar las plataformas y APIs que va a utilizar nuestra aplicación.

Nosotros vamos a realizar una aplicación para teléfonos y tablets, por lo que tan sólo tendremos que seleccionar la API mínima (es decir, la versión mínima de Android) que soportará la aplicación.

En mi caso selecciono la API 14, correspondiente a Android 4.0 ya que con ello se abarca el 95% de los teléfonos disponibles en el mercado.

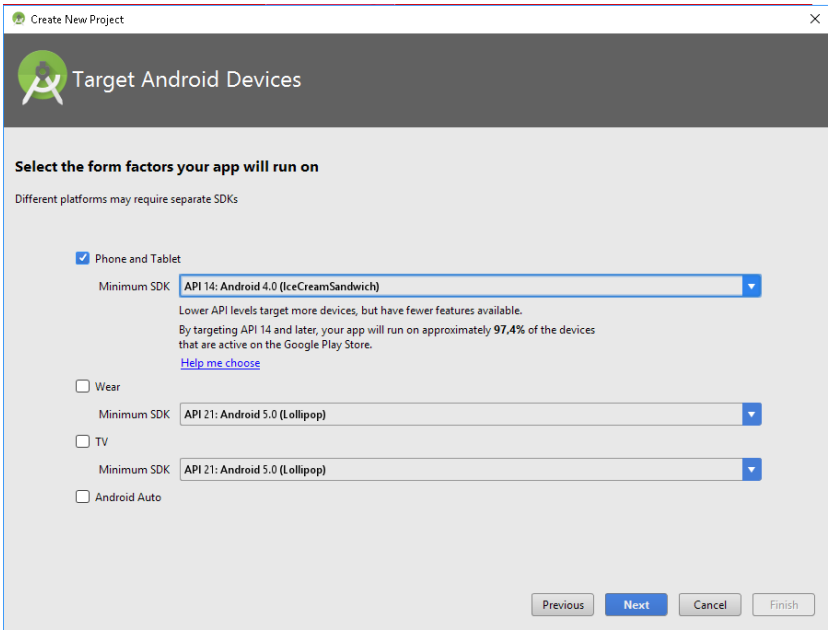


Ilustración 67. Selección API Android Studio

En la siguiente pantalla del asistente vamos a elegir el tipo de *actividad* principal de la aplicación. En este ejemplo sencillo vamos a seleccionar *Empty* Activity, que es el tipo más sencillo.

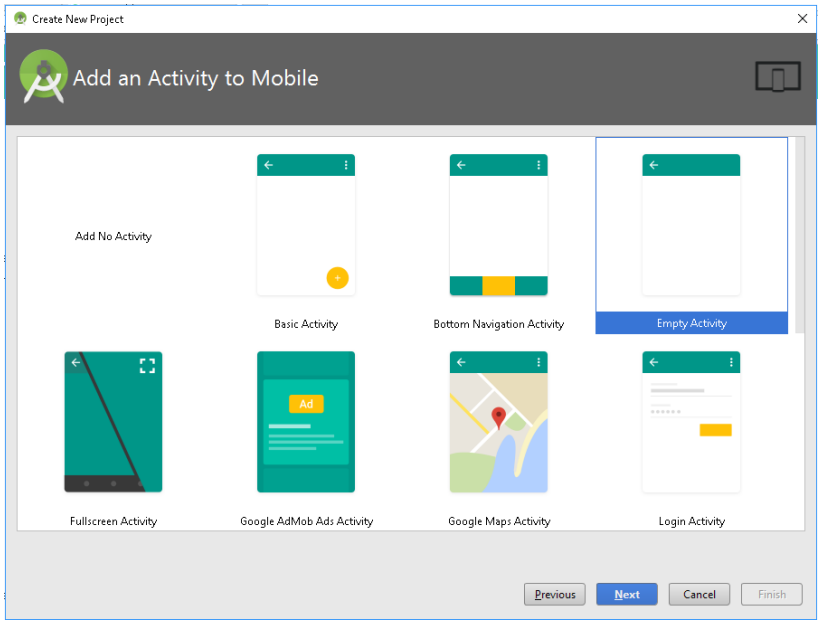


Ilustración 68. Selección activity Android Studio

Por último debemos dar un nombre a la actividad que acabamos de crear.

En este caso la denominaremos MainActivity.

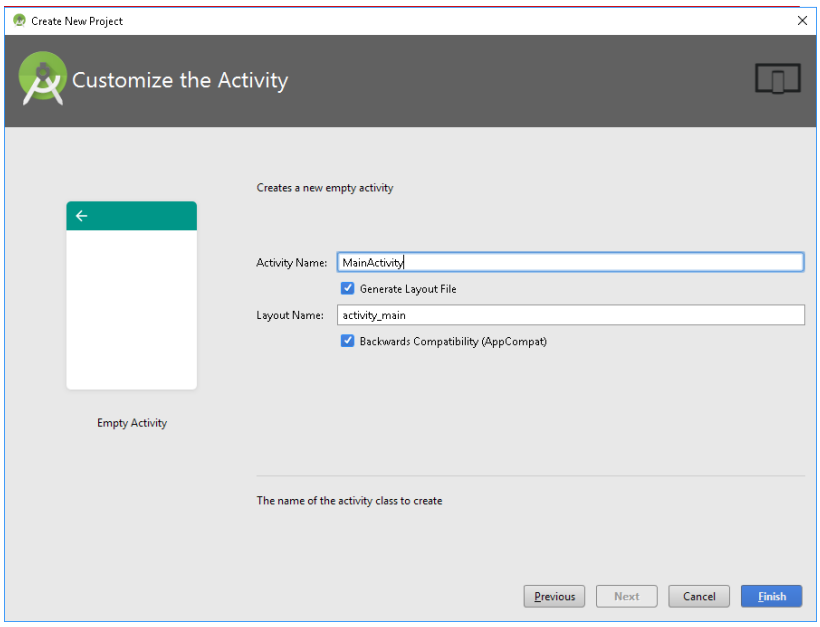


Ilustración 69. Nombrar activity Android Studio

Una vez seguidos todos los pasos pulsamos el botón Finish y Android Studio creará toda la estructura del proyecto con los elementos necesario.

3.5 Entorno de desarrollo

Si todo ha ido correctamente aparecerá ante nosotros la pantalla principal del entorno de desarrollo Android Studio.

En esta pantalla podemos observar varias partes claramente diferenciadas que comentaremos a continuación.

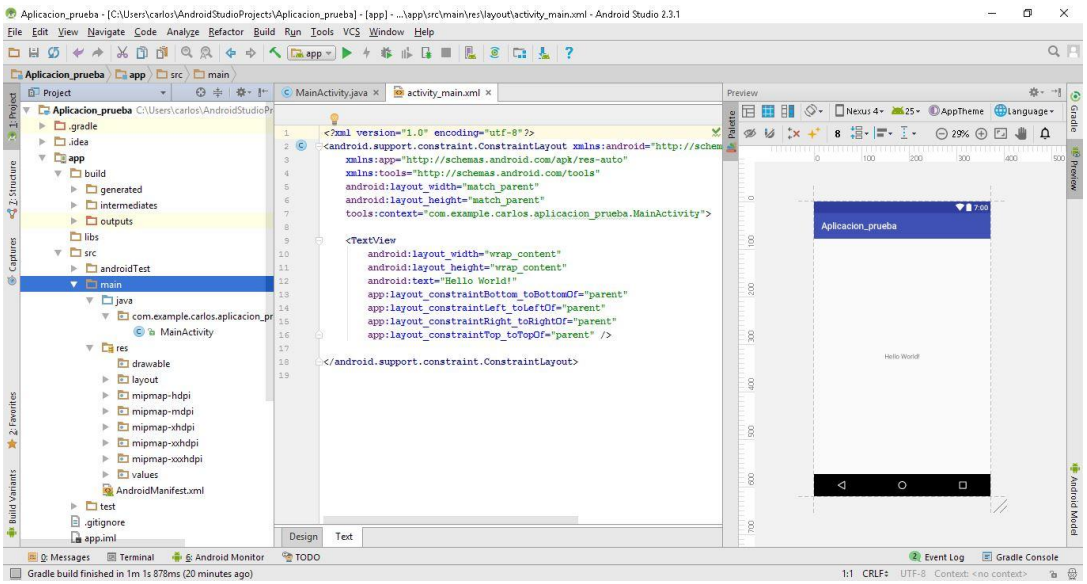


Ilustración 70. Entorno de desarrollo Android Studio

Barra de herramientas: en la barra de herramientas de Android Studio nos encontramos accesos rápidos a algunas de las acciones más comunes como pueden ser compilar el proyecto o ejecutarlo.

Ilustración 71. Barra de herramientas Android Studio

Entorno de programación: una de las ventajas de Android Studio es que combina la consola de desarrollo con un editor de diseño que permite arrastrar y soltar componentes de la interfaz de usuario.

Por otra parte observamos que la pantalla está dividida en dos secciones, a la izquierda, se observa el código, mientras que a la derecha se observa un emulador, que nos permite revisar todos los cambios que hagamos al código casi al instante, pudiendo editar código y ver las consecuencias inmediatamente.

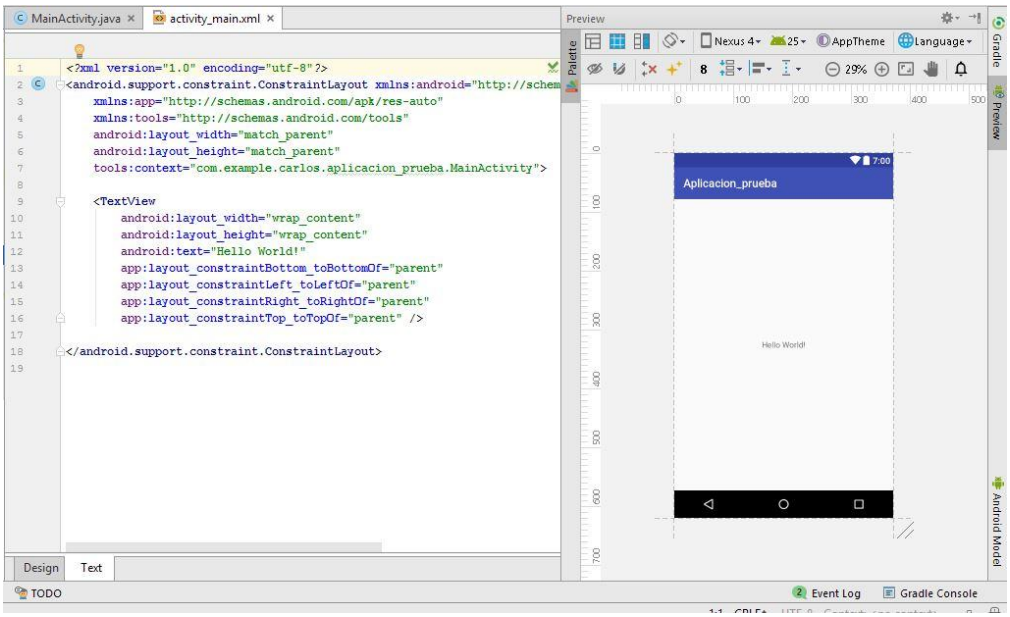


Ilustración 72. Entorno de programación Android Studio

Ficheros y carpetas: en la parte izquierda, podemos observar la estructura completa del proyecto que acabamos de crear.

Simplificando al máximo la explicación de un proyecto, se podría decir que básicamente cada pantalla se compone de dos partes.

-En primer lugar la carpeta Java, esta carpeta contiene todo el código fuente de la aplicación, clases auxiliares, etc. Inicialmente Android Studio ha creado por nosotros el código básico de la pantalla principal, el fichero MainActivity. Pero todas las modificaciones que realicemos sobre este fichero, o nuevas pantallas que añadamos aparecerán aquí.

Estos ficheros definen la LÓGICA DE LA PANTALLA.

-En segundo lugar la carpeta RES, esta carpeta contiene todos los ficheros de recursos necesarios para el proyecto, imágenes, layouts, cadenas de texto, etc.

Los recursos se distribuyen en subcarpetas.

Cabe destacar la carpeta Layout, la cual contiene los ficheros de definición XML de las diferentes pantallas de la interfaz gráfica, a cada pantalla (activity) que se genere, le corresponderá un fichero de esta carpeta. En este caso el fichero activity_main.xml.

Estos ficheros definen el DISEÑO VISUAL de la pantalla.

Otro de los ficheros importantes que se observan es el siguiente:

AndroidManifest.xml: Este fichero describe la aplicación de Android. En él se indican las actividades, las intenciones, los servicios y los proveedores de contenido de la aplicación. También se declaran los permisos que requerirá la aplicación. Se indica la versión mínima y máxima de Android para poder ejecutarla, el paquete Java, la versión de la aplicación, etc.

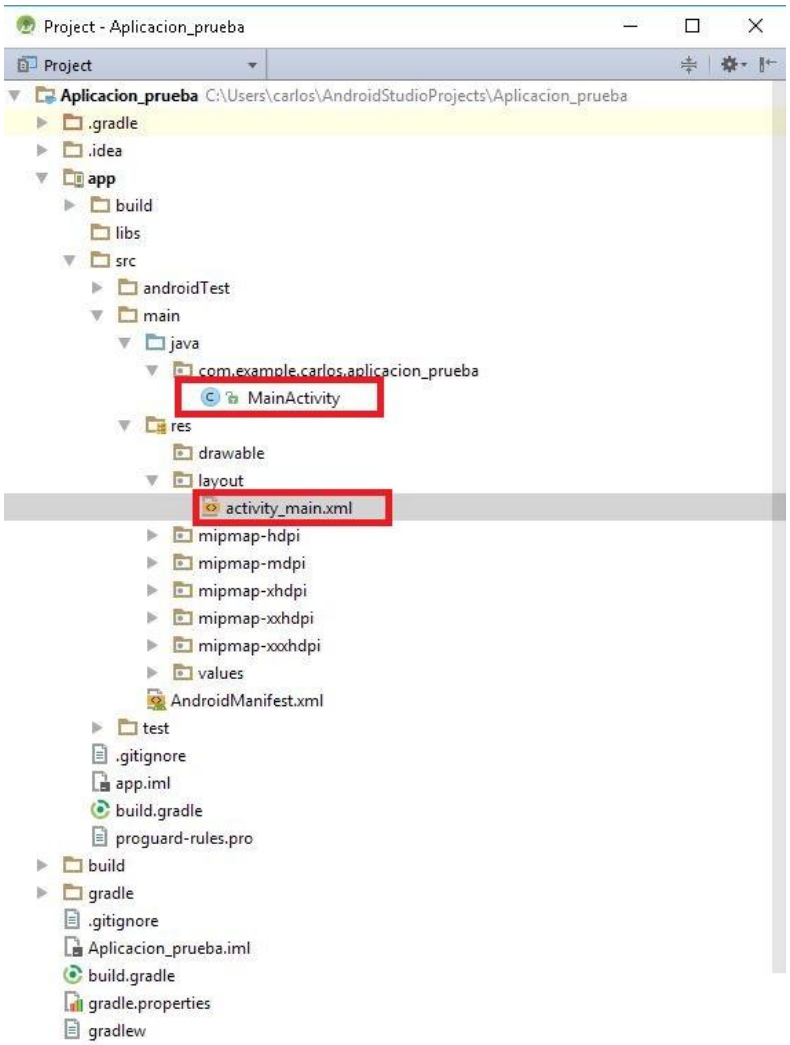


Ilustración 73.Ficheros y carpetas Android Studio

BIBLIOGRAFÍA

Publicaciones escritas

Jiménez Marín, Alfonso; Pérez Montes, Francisco Manuel Aprende a programar con Java. 2º Edición. Paraninfo, 2016

Joyanes Aguilar, Luis; Zahonero Martínez, Ignacio. Programación en C. 2º Edición. Mc Graw Hill, 2004

Ribas Lequerica, Joan. Desarrollo de aplicaciones para Android. Edición 2017. Anaya, 2017.

Ribas Lequerica, Joan. Arduino Práctico. Edición 2014. Anaya, 2014

Publicaciones Web

Android - Wikipedia, la enciclopedia libre

https://es.wikipedia.org/wiki/Android

Arduino Yun. Arduino

https://www.arduino.cc/en/Guide/ArduinoYun

Curso de Programación Android | sgoliver.net

http://www.sgoliver.net/blog/curso-de-programacion-android/

Hell-desk.com: Termostato Inteligente para Arduino YUN

https://www.hell-desk.com/

Métodos de petición HTTP - HTTP | MDN

https://developer.mozilla.org/es/docs/Web/HTTP/Methods

PanamaHitek. Arduino

http://panamahitek.com/category/arduino/