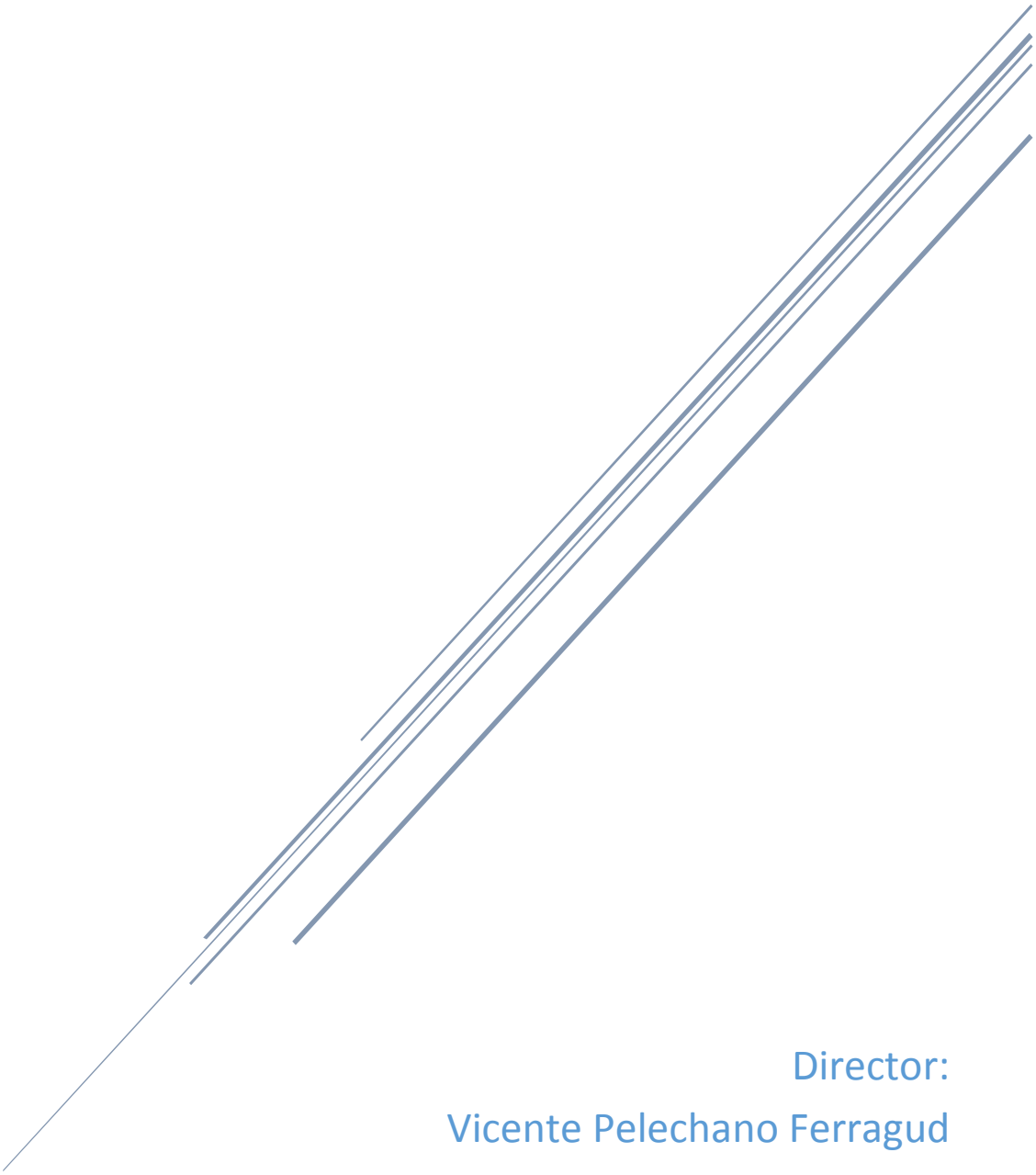
ENTORNO PARA LA ESPECIFICACION, VALIDACION Y GENERACION DE CODIGO PARA ARDUINO

Raul Garcia Alcaraz



Director:
Vicente Pelechano Ferragud

Master en ingeniería del software, métodos formales y sistemas de información

Entorno para la especificación, validación y generación de código para arduino.

Este documento ha sido preparado por

Raul Garcia Alcaraz

Director

Vicente Pelechano Ferragud

Universitat Politècnica de Valencia
Camí de Vera s/n, Edif. 1F
46022 - Valencia, Spain

Tel: (+34) 963 877 007 (Ext. 83530)
Fax: (+34) 963 877 359
Web: http://www.pros.upv.es

Fecha:	Septiembre 2014
Comentarios:	Documento presentado en cumplimiento parcial de los requisitos para el grado de Master en Ingeniería del Software, Métodos Formales y Sistemas de Información por la Universitat Politècnica de València.

A Jazmín
por darme un motivo
para levantarme cada mañana.

Agradecimientos

Este trabajo fin de master ha sido muy costoso, a la vez que enriquecedor. Pero todo este trabajo no hubiese sido posible sin la ayuda de una persona. Me gustaría agradecerle a Ignacio Mansanet todo el tiempo dedicado, toda la paciencia y todos los ánimos que me ha brindado durante el desarrollo de este trabajo de final de master. A mi tutor Vicente Pelechano me gustaría darle las gracias por ser como es y por tener siempre la puerta abierta a todos los alumnos.

También me gustaría agradecerle a Pau Martí todo su apoyo, favores, horas de clase, entretenidas charlas de bar, quejas, risas y llantos y darle las gracias por estar junto a mí desde que nos conocimos ya hace mucho.

Especialmente, me gustaría darle las gracias a la persona que me ha acompañado en esta nueva etapa de mi vida y a la cual espero que nunca me abandone.

Raul Garcia

Resumen

En los últimos tiempos se está detectando una tendencia conocida como Do-It-Yourself, consistente en que son los propios usuarios los que construyen y fabrican sus propios productos.

Enfocado un poco más en el ámbito de la tecnología, en los últimos años la plataforma Arduino ha visto como el número de usuarios ha aumentado en gran medida, gracias a la cantidad de componentes y a su gran comunidad que aporta ayuda a los neófitos.

Arduino es una plataforma de hardware libre, basada en una placa con un microcontrolador y un entorno de desarrollo, diseñada para facilitar el uso de la electrónica en proyectos multidisciplinares. En otras palabras, Arduino es un chip programable, con diferentes sensores y actuadores, que sirve para hacer prototipos de forma rápida.

El gran problema que conlleva iniciarse en el mundo Arduino, es la configuración de los componentes, que es llevada a cabo mediante código C. El mundo de la programación, no es agradable ni fácil para mucha gente, esto conlleva a que mucha gente se lo piense antes de adentrarse en el mundo de Arduino.

En este trabajo presentamos una herramienta web con la cual, se puede generar código para Arduino de forma visual y automática. Esta herramienta genera la mayor parte del código necesario a la hora de crear un circuito con Arduino, ahorrando al usuario mucho tiempo.

El código generado utiliza la librería TatAmi creado por el grupo de investigación PROS de la Universitat Politècnica de València. Dicha librería nos permite utilizar unas clases las cuales hacen el código más entendible al usuario y ahorra líneas de código.

Contenido

Índice de ilustraciones

1. Introducción

1.1. Descripción

Hoy en día, es innegable, que la tecnología avanza a pasos agigantados. Cada año las grandes compañías, en su afán de vender sus productos, lanzan al mercado móviles más potentes, televisores con mayor resolución, coches con mayor tecnología, etc. Pero, no solo las grandes compañías sacan nuevos productos con ideas novedosas.

Gracias a plataformas de "crowdfunding", también denominado financiación masiva, una persona con buenas ideas y pocos recursos, puede sacar a delante proyectos novedosos gracias a la ayuda de otras personas, que demuestran su confianza en forma de ayudas monetarias antes incluso de empezar a producir el proyecto.

Buena prueba de ello es Pebble, uno de los primeros relojes inteligentes capaz de mostrar notificaciones conectado con nuestro móvil. Su pantalla de tinta electrónica y la capacidad de ser conectado a dispositivos iOS y Android, hicieron que un proyecto que necesitaba 100.000 dólares, acabara la ronda de financiación consiguiendo 10 millones, 100 veces más de lo necesitado en un principio.

Con la idea de “a computer anyone can makes”, Kano, otro proyecto de la plataforma kickstarter, consiguió más de un millón de dólares. Kano vende la idea que cualquiera puede con unas simples instrucciones crear cualquier cosa, incluso un ordenador. Dicho proyecto consta de un miniordenador Raspberry Pi, un teclado, un libro de instrucciones y una versión de Linux modificada para ser ejecutada en el Raspberry Pi. Dicha versión de Linux, lleva un ide con código de demostración para hacer juegos como el “Pong” o el “Snake”. En este caso, no sacaron ningún invento nuevo, tan solo se sirven de la idea “créatelo tú mismo”.

En los últimos tiempos se está detectando una tendencia conocida como Do-It-Yourself, consistente en que son los propios usuarios los que construyen y fabrican sus propios productos. Mucha gente se hace sus bufandas de lana para el invierno, se prepara su propio pan, se modernizan su casa. Y es que la tecnología, internet, y el gran boom de compartir casi todo lo que hacemos, aunque a mucha gente que no le parezca buena idea, tiene sus cosas buenas. Existen miles de manuales, recetarios, video blogs que enseñan cómo preparar pan, sushi o cualquier cosa que puedas imaginar. Y es que hoy en día, con toda la información que existe, todos podemos aprender a hacer cosas nuevas con la ayuda de internet.

1.2. Motivación

Como hemos visto en los ejemplos anteriores, cada día más gente se está lanzando a hacer realidad sus ideas. Todos tenemos ideas y proyectos que nos gustaría realizar, pero por falta de tiempo no podemos llevarlos a cabo. Con el proyecto presentado en este trabajo final de master, queremos ayudar a toda la gente que se está iniciando o que ya está metida en el mundo de Arduino.

Pero, ¿Qué es Arduino? Arduino es una plataforma de hardware libre, basada en una placa con un microcontrolador y un entorno de desarrollo, diseñada para facilitar el uso de la electrónica en proyectos multidisciplinares. En otras palabras, Arduino es un chip programable, con diferentes sensores y actuadores, que sirve para hacer prototipos de forma rápida.

Arduino tiene una gran cantidad de sensores y actuadores compatibles con la plataforma. Dichos sensores nos pueden dar información de diverso tipo, como por ejemplo valores de luz, humedad, sonido, proximidad, mientras que con los actuadores podemos hacer que pase la corriente cuando nos interese mediante un relé, mover componentes con un servo motor, etc.

Gracias a Arduino y sus componentes podemos montar nuestro propios gadgets o prototipos electrónicos con mucha facilidad. Por ejemplo, con un Arduino, un sensor de luz, un sensor de proximidad y un relé, podemos hacer que se encienda las luces del pasillo de nuestra casa cuando pasemos por él y esté oscuro.

1.3. Planteamiento del problema

En este proyecto se presenta una herramienta web para la creación rápida de prototipos, utilizando la plataforma Arduino. Al ser una herramienta web, puede ser accedida desde cualquier parte del mundo y con cualquier tipo dispositivo, sin tener un software instalado previamente en nuestro ordenador.

Arduino es actualmente, la plataforma de prototipado rápido con más soporte por parte de la comunidad, tanto de desarrolladores como de fabricantes de componentes. Existe mucha gente que, pese a tener pocos conocimientos sobre programación, se está lanzando a aprender con la excusa de probar Arduino y crear sus propios prototipos. Pero este no es un camino fácil, con nuestra aplicación queremos facilitar ese camino.

Gracias a la aplicación presentada en este TFM, la creación de plantillas de código para Arduino se puede realizar de forma sencilla, por lo que el usuario, tan solo deberá rellenar unas pocas líneas, ahorrándose mucho código de configuración.

1.4. Contexto del trabajo final de master

Este Trabajo Final de Master se ha desarrollado en el contexto del Centro de Investigación en Métodos de Producción de Software (ProS), de la Universidad Politécnica de Valencia (www.pros.upv.es).

1.4.1. Antecedentes

Dentro del mismo grupo de investigación, se imparte en el master una asignatura llamada “Patrones Software y generación de código” (PSW), en la cual se trabajó en una solución del problema muy diferente.

Para realizar la solución en dicha asignatura, se hizo uso de la herramienta de desarrollo “Eclipse Modeling Framework” (EMF) y un componente llamado XPAND, basando el desarrollo en una extensión del mismo entorno de desarrollo.

EMF es un marco de modelado y generación de código basado en Eclipse para la construcción de herramientas y otras aplicaciones basadas en un modelo de datos estructurado. Desde una especificación de modelo descrito en XMI, EMF proporciona herramientas y soporte de ejecución para producir un conjunto de clases Java para el modelo, un conjunto de clases de adaptadores que permiten la visualización y edición basada en comandos de la modelo, y un editor básico. Los modelos pueden ser especificados usando Java, UML, documentos XML, o herramientas de modelado anotado, luego importado a EMF. Lo más importante de todo, EMF proporciona la base para la interoperabilidad con otras herramientas y aplicaciones basadas en EMF.

Xpand es un lenguaje especializado en la generación de código basado en modelos EMF.

Para el desarrollo de nuestra herramienta para eclipse, primero utilizamos las herramientas que nos brinda Eclipse para definir nuestro metamodelo. En el metamodelo definimos un leguaje específico de dominio, en el cual definiremos un leguaje que nos ayude a describir nuestra realidad.

En la imagen siguiente podemos ver una representación gráfica del metamodelo creado.

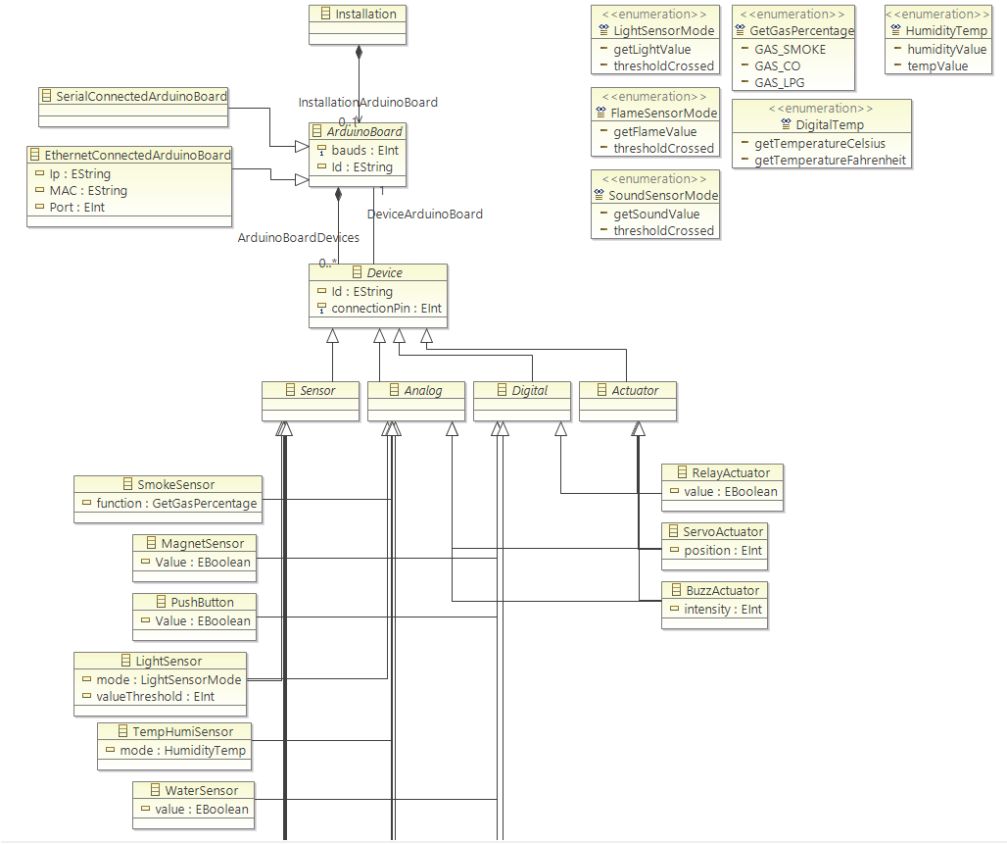


Figura 1 - Metamodelo creado en PSW

Como se puede ver en la imagen anterior, para definir nuestras instalaciones de Arduino, hemos partido de una clase “instalación” la cual tendrá cero o una placa arduino. Dicha placa puede estar conectada a nuestro ordenador mediante un puerto serie o un cable Ethernet. También tendrá conectados diferentes dispositivos, los cuales están pueden ser sensores o actuadores, y estos a su vez pueden ser digitales o analógicos. Para no tener que repetir las clases base de los dispositivos (Sensor, Actuador, Analógico y digital) se optó por usar herencia múltiple y así, que los dispositivos finales, pudieran estar en uno de los cuatro grupos resultantes, sensores analógicos o digitales, y actuadores analógico o digitales. De esta forma quedó un metamodelo muy simple.

Una vez que terminamos el metamodelo, pudimos generar el plugin para eclipse con el cual podremos crear instancias de este metamodelo.

Una vez que creamos y ejecutamos el plugin que creamos mediante el metamodelo, pudimos generar modelos a partir de instancias del metamodelo. Gracias a dicho plugin pudimos modelar nuestra realidad con el lenguaje definido anteriormente.

Por ejemplo, si tenemos una placa Arduino conectado a nuestro pc mediante USB y a ella tenemos conectado un sensor de luz, un sensor de sonido y un relé, podríamos hacerlo de la manera que se en la siguiente imagen.

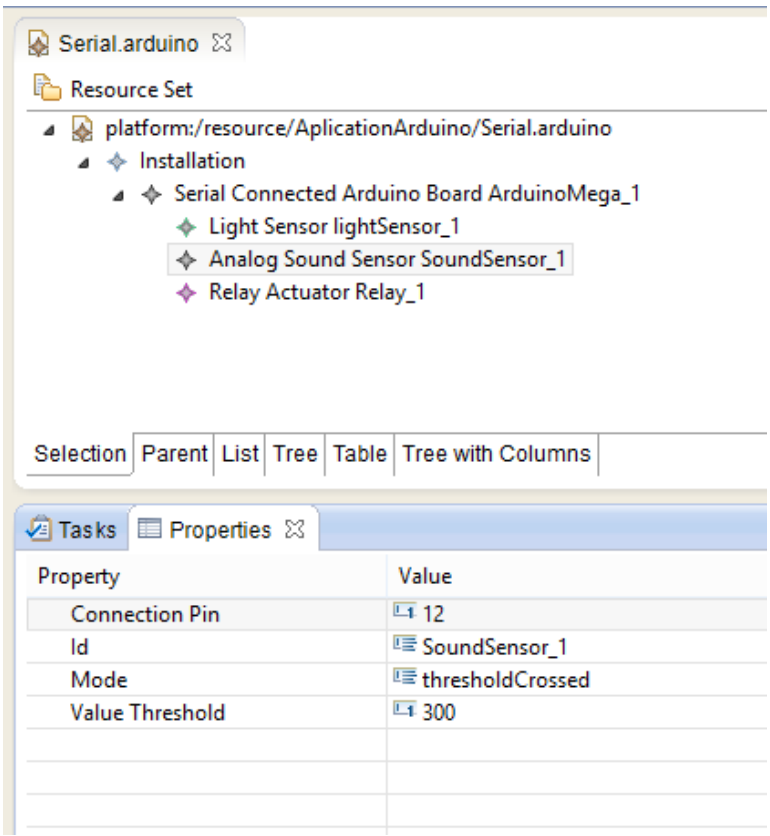


Figura 2 - Modelo creado en PSW

Para que el código generado por el plugin funcione, es muy importante que cuando se crea el proyecto ".arduino" elijamos como primer elemento un nodo instalación, ya que de no ser así, el generador de código no funcionará. También hay que tener en cuenta, que se debe rellenar siempre todas las propiedades de todos los componentes.

Dicha herramienta generará dos ficheros ".ino" con código C para arduino. El primer fichero tendrá el nombre "arduino.ino" el cual tendrá el código principal y el segundo "handlers.ino" contendrá el código de los manejadores de los actuadores.

El fichero "arduino.ino" esta, a su vez, dividido en dos funciones, "Setup" y "Loop".

La función "Setup" inicializa las variables anteriormente creadas, mientras que la función "Loop" se ejecuta de manera infinita ejecutando el código de su interior.

Para crear estos ficheros, y rellenarlos con la información anteriormente creada en el modelo, se ha usado la librería XPAND, la cual nos ayuda en la generación de código a partir del modelo.

arduino2code.xpt | arduino.ino | handlers.ino | Arduino.ecorediag

```
«DEFINE Root FOR arduino::Installation»
    «FILE "arduino.ino"»
#include <Tatami.h>
#include <Ethernet.h>
#include <Wire.h>
#include <SPI.h>

    «EXPAND Constant FOREACH InstallationArduinoBoard.ArduinoBoardDevices»
    «EXPAND Ids FOREACH InstallationArduinoBoard.ArduinoBoardDevices»
    «EXPAND EthernetConst FOR InstallationArduinoBoard»
    «EXPAND Define FOREACH InstallationArduinoBoard.ArduinoBoardDevices»

    void setup(){
        Serial.begin(«InstallationArduinoBoard.bauds»);
        «EXPAND Board FOR InstallationArduinoBoard»
        «EXPAND Init FOREACH InstallationArduinoBoard.ArduinoBoardDevices»
    }

    void loop(){
        «EXPAND Devices FOREACH InstallationArduinoBoard.ArduinoBoardDevices»

        ////////////////////////////////////////////////////////////
        //// Handlers
        ////////////////////////////////////////////////////////////
        Serial.flush();
        char addressBuffer[20]="init";
        char valBuffer[100];
        «EXPAND Handlers FOREACH InstallationArduinoBoard.ArduinoBoardDevices»
    }
    «ENDFILE»

    «FILE "handlers.ino"»
        «EXPAND DefHandlers FOREACH InstallationArduinoBoard.ArduinoBoardDevices»
    «ENDFILE»
«ENDDEFINE»
```

Figura 3 - Codigo XPAND para la generacion de codigo

En la imagen anterior, podemos un trozo de código utilizado para la generación del código. En dicha imagen podemos ver la función principal («DEFINE Root FOR arduino::Installation») la cual será utilizada por XPAND para empezar la generación de código.

La sentencia “«FILE "arduino.ino"»” creará un fichero llamado “arduino.ino”, el cual contendrá todo el código siguiente que no esté entre las llaves “«»”.

La sentencia “«EXPAND Board FOR InstallationArduinoBoard»” será la encargada de llamar a la función que añadirá al fichero “arduino.ino” el código que inicializara la placa.

```
//Propietas de la placa
«DEFINE Board FOR arduino::ArduinoBoard»
    «IF metaType.name == 'arduino::EthernetConnectedArduinoBoard'»
        comm.begin(mac, server, myIP, port);
    «ELSE»
        comm.begin();
    «ENDIF»
«ENDDEFINE»
```

Figura 4 - Funcion XPAND para inicializar una placa Arduino

Aquí podemos ver dicha función, la cual mira si la placa está conectada por cable Ethernet o no. Dependiendo del tipo de conexión, añade un código u otro.

La sentencia
"«EXPAND Init FOREACH InstallationArduinoBoard.ArduinoBoardDevices»" es un bucle que llamara a una función "Init" por cada dispositivo conectado a la placa.

```
// Inicialetcem l'objecte
«DEFINE Init FOR arduino::Device»
        «Id».begin(«Id»_PIN);
«ENDDEFINE»
```

Figura 5 - Funcion XPAND para inicializar un dispositivo

En esta imagen se puede ver como se accede a la propiedad "Id" de la clase base del dispositivo.

Dentro del código generado "Loop" se llama a una función "«EXPAND Devices FOREACH InstallationArduinoBoard.ArduinoBoardDevices»" la cual será la encargada de acceder a los parámetros internos de cada dispositivo.

```
«DEFINE Devices FOR arduino::Device»
    «IF metaType.name == 'arduino::LightSensor'»
        «EXPAND Light FOR this»
    «ELSEIF metaType.name == 'arduino::SmokeSensor'»
        «EXPAND Smoke FOR this»
    «ELSEIF metaType.name == 'arduino::MagnetSensor'»
        «EXPAND Magnetism FOR this»
    «ELSEIF metaType.name == 'arduino::PushButton'»
        «EXPAND PushButton FOR this»
    «ELSEIF metaType.name == 'arduino::TempHumiSensor'»
        «EXPAND HumidityTemp FOR this»
    «ELSEIF metaType.name == 'arduino::WaterSensor'»
        «EXPAND Water FOR this»
    «ELSEIF metaType.name == 'arduino::TiltSensor'»
        «EXPAND Tilt FOR this»
    «ELSEIF metaType.name == 'arduino::MotionSensor'»
        «EXPAND Motion FOR this»
    «ELSEIF metaType.name == 'arduino::FlameSensor'»
```

Figura 6 - Funcion XPAND para acceder a los parametros de cada dispositivo

En la imagen anterior vemos una función utilizada para ver a qué tipo de sensor estamos accediendo y llamar a una función con su nombre para así poder acceder, ya que en la función actual (Devices) no podemos acceder al contenido de las clases dispositivo.

```
«DEFINE HumidityTemp FOR arduino::TempHumiSensor»
    «IF mode.toString().contains("humidity")»
        comm.writeData("Humidity sensor read value is:\n","");
        comm.writeData(«Id».getHumiValue(),"");
    «ELSE»
        comm.writeData("Temperature sensor read value is:\n","");
        comm.writeData(«Id».getTempValue(),"");
    «ENDIF»
        delay(500);
«ENDDEFINE»

«DEFINE Water FOR arduino::WaterSensor»
        comm.writeData("Water sensor read value is:\n","");
        comm.writeData(«Id».getWaterValue(),"");
        delay(100);
«ENDDEFINE»
```

Figura 7 - Funcion XPAND para acceder a los atributos de los dispositivos

En la imagen anterior vemos como llamando estas funciones podemos generar el código personalizado para cada sensor o actuador.

Para crear el segundo fichero, usamos otra sentencia “«FILE "handlers.ino"»” fuera de la anterior sentencia que creaba el fichero “arduino”, como podemos ver a continuación.

```
«FILE "handlers.ino"»
    «EXPAND DefHandlers FOREACH InstallationArduinoBoard.ArduinoBoardDevices»
«ENDFILE»
```

Figura 8 - Sentencia XPAND para crear un fichero .ino

Una vez ejecutado todo el código XPAND, vemos que el resultado es un código generado para arduino el cual solo queda pegarlo en el “Arduino IDE” para compilarlo y pasarlo la placa. En la imagen siguiente podemos ver un tozo del fichero generado.

```
#include <Tatami.h>
#include <Ethernet.h>
#include <Wire.h>
#include <SPI.h>


    const int lightSensor_1_PIN=10;

    const int SoundSensor_1_PIN=12;

    const int Relay_1_PIN=21;


    char LIGHTSENSOR_1_ID[] = "lightSensor_1";

    char SOUNDSENSOR_1_ID[] = "SoundSensor_1";

    char RELAY_1_ID[] = "Relay_1";


    SerialCommunication comm;


    LightSensor lightSensor_1;

    AnalogSoundSensor SoundSensor_1;

    RelayActuator Relay_1;


    void setup(){
        Serial.begin(9600);


        comm.begin();


        lightSensor_1.begin(lightSensor_1_PIN);

        SoundSensor_1.begin(SoundSensor_1_PIN);

        Relay_1.begin(Relay_1_PIN);

    }

    void loop(){



        comm.writeData("light sensor read value is:\n","");
        comm.writeData(lightSensor_1.getLightValue(),"");

        delay(500);
```

Figura 9 - Ejemplo de fichero generado por la aplicacion de PSW

1.5. Objetivos Globales

El TFM desarrolla una plataforma software compuesta dos elementos, interfaz web y servidor el cual puede ser accedido mediante un API REST y con la cual se puede generar código C, que una vez compilado, funciona en placas arduino.

La aplicación debe cumplir los siguientes objetivos:

- **Multidispositivo**: La aplicación debe poder ser accedida desde cualquier dispositivo sin importar su sistema operativo ni su resolución de pantalla.
- **Persistencia de los datos**: Los proyectos creados podrán ser retomados en cualquier momento aunque no sigamos en el mismo dispositivo donde hemos creado dicho proyecto.
- **Simplicidad**: La aplicación debe ser fácil de usar, ya que uno de los valores principales de la herramienta es ahorrar tiempo y esto se consigue eliminando tiempos de aprendizaje y configuración.
- **Robustez**: La aplicación debe responder en todo momento.
- **Escalable:** No debe importar el número de usuarios activos, el sistema debe funcionar siempre igual de fluido, sin aumentar los tiempos de carga.
- **API de comunicación:** Los datos deben poder ser accedidos desde fuera de la aplicación por medio de los servicios REST.

1.6. Objetivo del trabajo

El objetivo principal es desarrollar una herramienta que ayude a los usuarios en sus diseños de prototipos con Arduino, mediante la generación de código a partir de esquemas diseñados con la propia herramienta y la simulación de dichos esquemas sin necesidad de tener ningún hardware conectado a nuestro ordenador.

El TFM se puede dividir en tres partes:

- Herramienta web con la que el usuario puede diseñar de manera gráfica, compilar y descargar el código fuente para su configuración de arduino.
- Simulador integrado en la herramienta web, en el que se podrá ver una virtualización del funcionamiento del circuito antes de ser montado físicamente.
- Almacenamiento y recuperación de los datos y gestión de la simulación, gracias a un api REST a la que podrán conectar otros dispositivos para consultar dicha información.

1.7. Estructura del documento

El resto de este documento se estructura en ocho Capítulos. Como guía de la estructura se puede consultar la lista siguiente:

Capítulo 2: Aplicaciones relacionadas.

En este capítulo se presentan varias aplicaciones similares a la desarrollada realizando una comparativa entre éstas.

Capítulo 3: Contexto tecnológico.

En este punto se presentan las tecnologías utilizadas en el desarrollo de nuestra parte del proyecto haciendo una especial explicación de aquellas más importantes.

Capítulo 4: Desarrollo de la propuesta.

En este capítulo se explican los requisitos además de todo el proceso de diseño, implantación y pruebas de las tareas realizadas describiendo en profundidad todo lo realizado.

Capítulo 5: Manual de usuario.

En este capítulo se presenta la aplicación desarrollada, explicando cómo utilizarla en cada momento.

Capítulo 6: Casos de uso.

En este capítulo se expone un caso de estudio sobre el que se ha aplicado la propuesta de la tesis para demostrar su viabilidad práctica.

Capítulo 7: Conclusiones y trabajo futuro.

Se plantean las conclusiones extraídas y trabajos de ampliación de la tesina desarrollada.

2. Estudio comparativo

En este capítulo del trabajo final de master, se desarrolla un estudio de las actuales aplicaciones que nos brinda el mercado, asimismo veremos las diferencia y similitudes entre ellas y el valor añadido de nuestra aplicación frente a la competencia.

2.1. Arduino IDE

Arduino IDE, a pesar de no ser un rival para nuestro proyecto, sino más bien un compañero, se ha creído conveniente presentar dicha herramienta ya que es la herramienta oficial de Arduino, y junto a la librería TatAmi, desarrollada por el Pros, serán necesarias para compilar nuestro código y ejecutarlo en nuestra placa.

Arduino IDE es de código libre y multiplataforma. A pesar de ser la herramienta desarrollada por el equipo de Arduino, es la más básica. Ofrece un editor de texto

con opciones de personalización para cada tipo de placa, así como importación de librerías y paso de código compilado directamente a la placa, siempre y cuando esté conectada a nuestro ordenador.

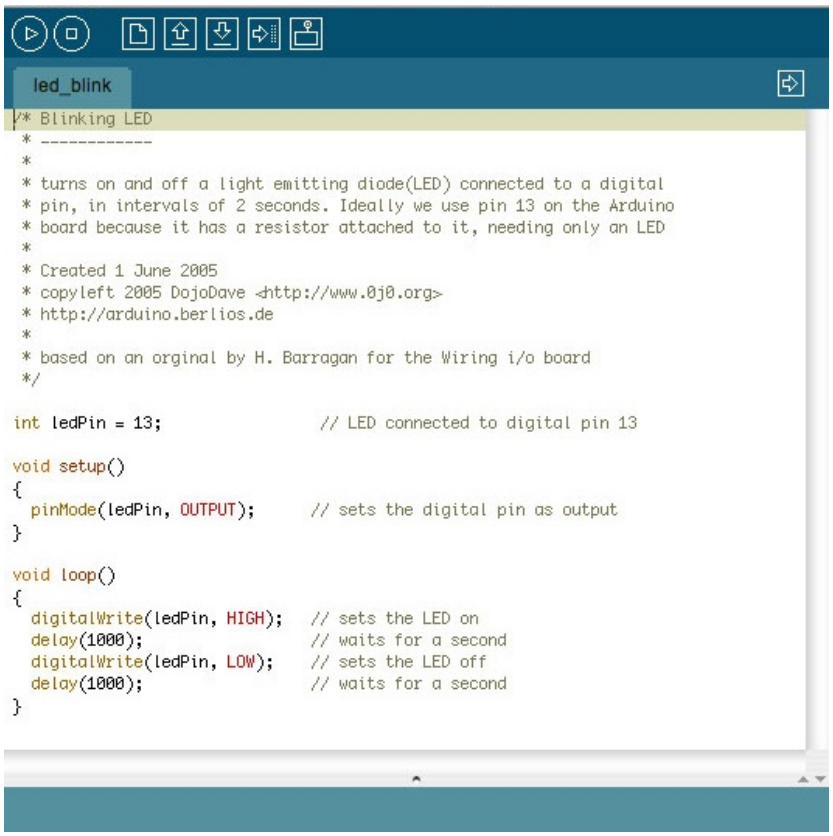


Figura 10 - Arduino IDE

2.2. Atmel Studio + Arduino plugin

Atmel Studio es una herramienta para el desarrollo de código para Arduino, basada en Microsoft Studio 2010. Tiene un compilador ligero y rápido, con el que con un clic se puede compilar y subir el código a la placa arduino que tengamos conectada al ordenador.

Dispone de un debuger en el cual, se puede poner puntos de interrupción para que podamos ver el valor de las variables, ver el flujo del programa y encontrar posibles errores en nuestro código en tiempo real mientras el código se está ejecutando.

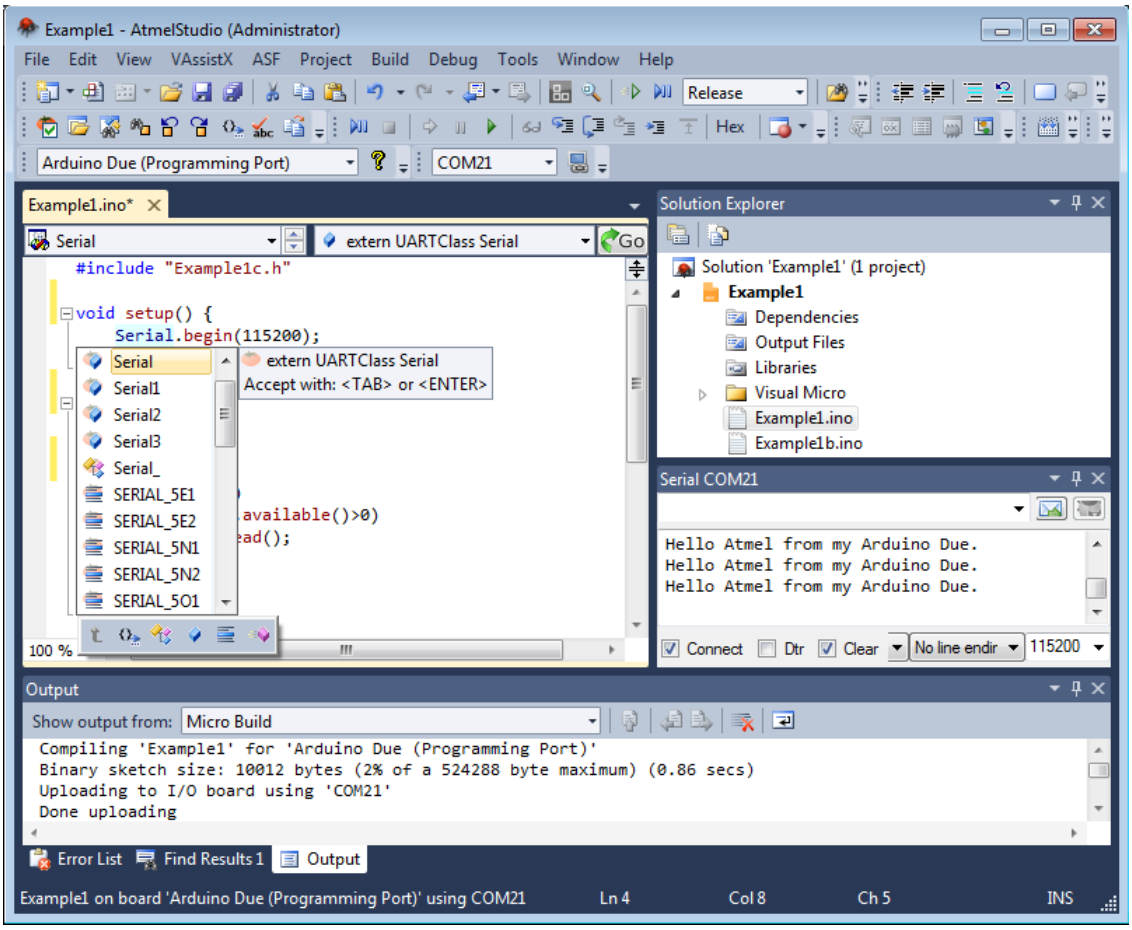


Figura 11 - Interfaz de Atmel Studio

2.2.1. Problemas de Atmel Studio

Atmel Studio es un programa binario que se ejecuta en un ordenador, por lo cual dependemos de esa máquina durante todo el desarrollo. Como es normal, los datos son guardados de forma local y por lo tanto en caso de pérdida de datos, perderíamos el proyecto.

Al estar basado en Microsoft Visual Studio, se necesita tener instalado previamente el entorno de desarrollo de Microsoft, y tan solo se puede ejecutar en máquinas con el sistema operativo de Microsoft, dejando fuera a las diferentes distribuciones de Linux y Mac.

2.2.2. Puntos a destacar

En este apartado vamos a mostrar los puntos positivos y negativos de Atmel:

Positivo	Negativo
La herramienta básica es gratuita.	El debugger no es gratuito, hay que pagarlo a parte.
Ofrece un debugger en tiempo real, con el que puedes ver cómo se va ejecutando el código en nuestra placa.	Hay que tener instalado previamente el IDE Microsoft Visual Studio 2010.
De forma fácil muestra ejemplos.	Solo para la plataforma Windows.
Muestra los errores de código mientras se escribe el código.	

2.3. Pduino

Pduino nace de la fusión de los proyectos Pure Data y Arduino. Ambos proyectos de código abierto. Cargando el firmware de Pure Data (PD) a la placa Arduino se puede acceder a ella mediante el lenguaje de programación gráfico.

En la ilustración 1 se muestra la interfaz del IDE.

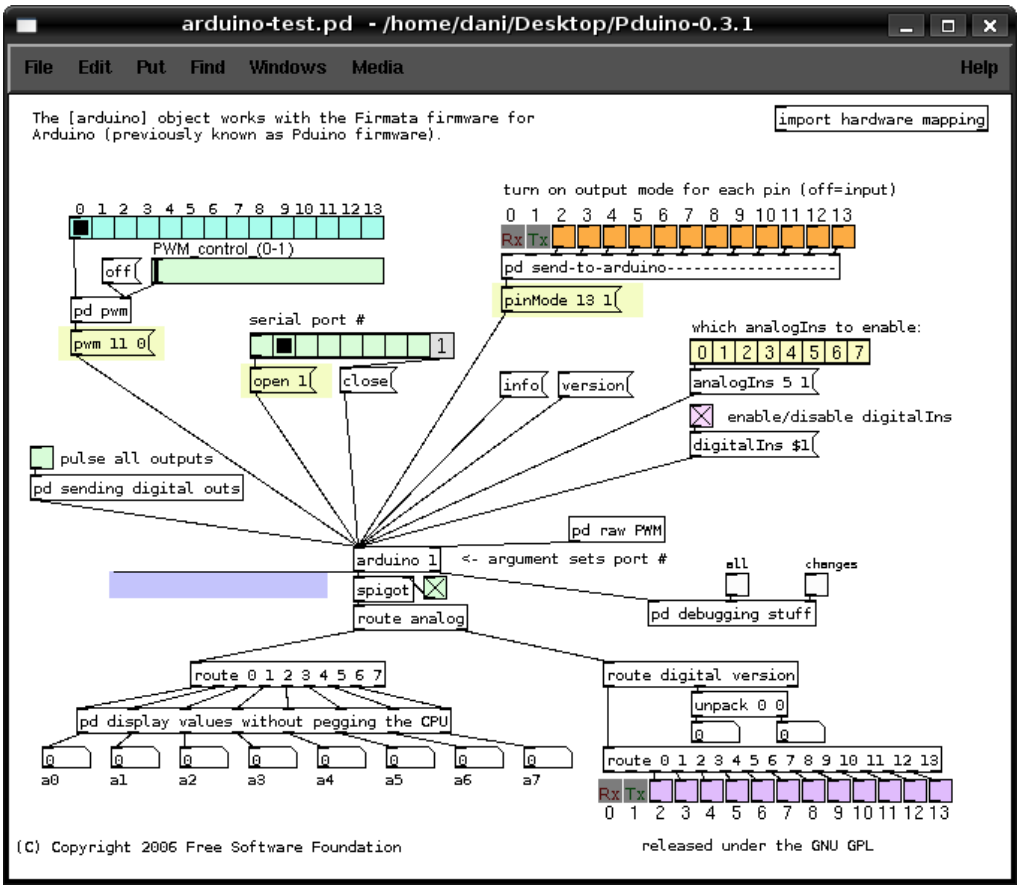


Figura 12 - Pduino

2.3.1. Problemas de Pduino

Como se puede ver en la imagen 3, pduino tiene una interfaz poco intuitiva.

Necesita una configuración inicial de la placa utilizando Arduino IDE.

Como he podido ver en su repositorio de github (https://github.com/reduzent/pduino) el proyecto está parado desde octubre del 2012.

2.3.2. Puntos a destacar

En este apartado vamos a mostrar los puntos positivos y negativos de Pduino:

Positivo	Negativo
Multiplataforma, soporta Linux, Windows y Mac OSx	Requiere de instalación local en una máquina.
En su web, tiene ejemplos de cómo configurarlo.	Configuración inicial complicada.
Es un editor visual.	El proyecto ya no tiene desarrollo.

2.4. Minibloq

Minibloq es un entorno gráfico de programación que puede generar código nativo de Arduino y escribirlo directamente en la memoria flash de la placa. Tiene un modo que permite visualizar el código generado, el cual también puede ser copiado y pegado en el Arduino-IDE, para los usuarios que intentan hacer el pasaje de una herramienta gráfica a la programación en sintaxis C/C++. Minibloq es de uso libre y sus fuentes también están disponibles gratuitamente.

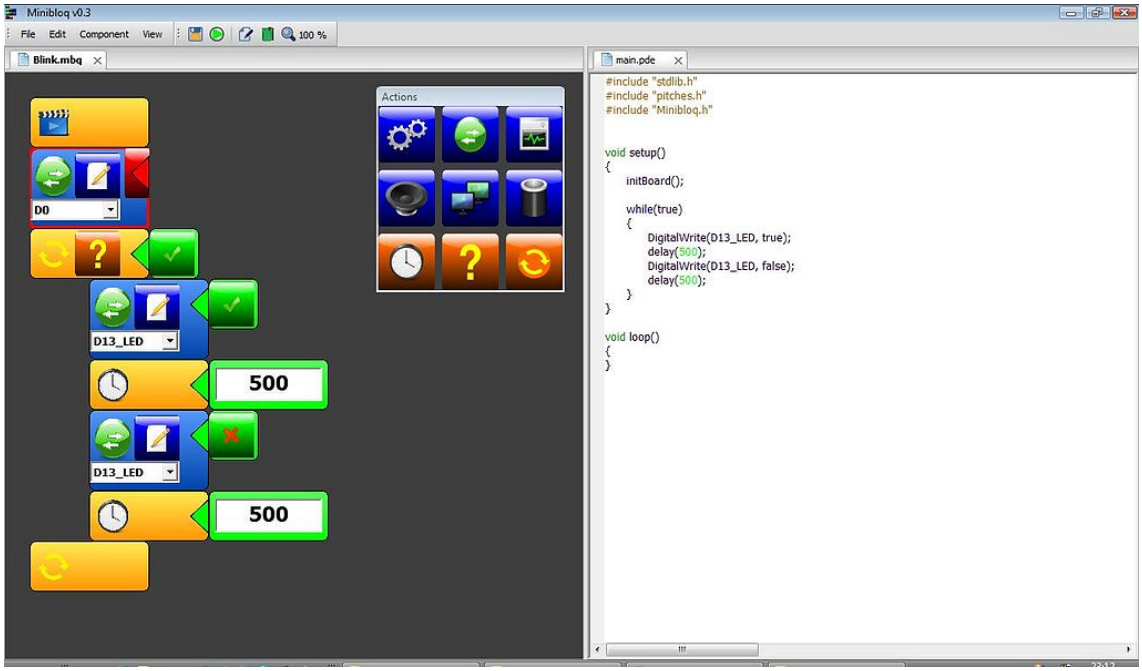


Figura 13 - Minibloq

2.4.1. Problemas de Minibloq

Minibloq al igual que Atmel Studio es una aplicación de escritorio donde tanto el proyecto como el código generado se guardan en la maquina local.

Requiere de un conocimiento anterior del sistema Arduino,

2.4.2. Puntos a destacar

En este apartado vamos a mostrar los puntos positivos y negativos de Minibloq

Positivo	Negativo
Interfaz gráfica fácil de usar si se ha trabajado con Arduino con anterioridad.	La versión instalable del programa, siempre es compilada para Windows, y posteriormente portada a Linux.
En su blog oficial tiene ejemplos y manuales.	No tiene versión para Mac OSx.
Tiene versión “portable”	

2.5. Modkit

Modkit es una herramienta web de programación para dispositivos. No está centrado en arduino, sino en todo tipo de chips programables. En su web nos encontramos con una demo de su programa, pero no está completa. Para poder acceder a todas las herramientas que nos brinda modkit, es necesario pagar una suscripción anual a su programa, la cual nos da derecho a poder usar su herramienta completa de escritorio y su versión web, un programa de auto-detección de dispositivos conectados al ordenador y soporte mediante su canal de google+.

Dicha herramienta permite de forma sencilla y visual, programar cualquier chip programable que tengan en su base de datos. También nos permite, si tenemos nuestra placa Arduino o el dispositivo que estemos programando conectado a nuestro ordenador, pasar el código compilado al dispositivo.

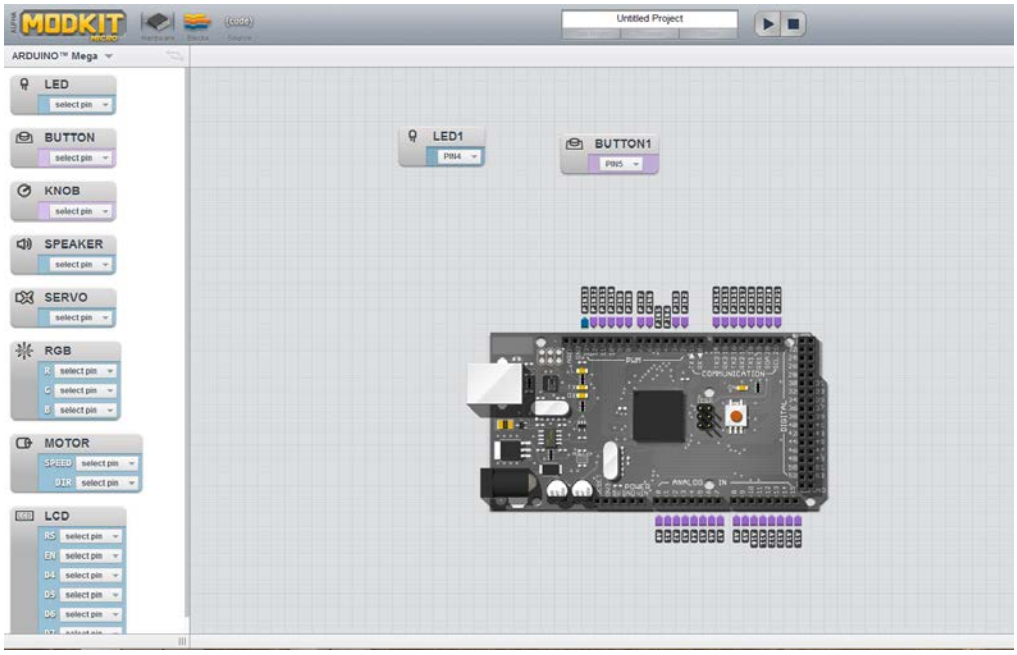


Figura 14 – Interfaz de ModKit

2.5.1. Problemas de Modkit

El primer problema que nos encontramos al intentar probar modkit, es que para acceder a su totalidad hay que pagar la suscripción. También hay que decir que modkit, en su versión gratuita, no permite comprobar si el código es correcto. Por lo que podríamos conectar dispositivos analógicos en pines digitales de la placa y a la inversa sin que la herramienta diese ningún mensaje de error. Su interfaz, a pesar de ser gráfica, no es intuitiva del todo, ya que permite al usuario elegir entre muchas opciones de configuración, las cuales no todas son correctas.

2.5.2. Puntos a destacar

En este apartado vamos a mostrar los puntos positivos y negativos de Modkit.

Positivos	Negativos
Es una herramienta web	La herramienta es de pago
	No puede accederse desde dispositivos móviles
	No tiene simulador

2.6. Comparación entre aplicaciones

En este último apartado se realiza una serie de comparaciones entre las aplicaciones presentadas.

	Atmel Studio	**Pduino**	**Minibloq**	**Modkit**	**Nuestra Aplicación**
Multidispositivo					x
Datos en la nube				x	x
Simulación					x
Compilación	x	x	x	x	
Gratuito		x	x		x
Corrección	x	x			x
Multiplataforma				x	x
Editor visual				x	x
Debuger	x				

3. Contexto tecnológico

3.1. Tecnologías utilizadas

Durante el desarrollo se han encontrado varios problemas y hemos tenido que tomar decisiones importantes para solucionarlos. Para resolver cada uno de los problemas se han utilizado una serie de tecnologías y patrones más oportunos para la solución propuesta y que no solo deben servir ahora si no en un futuro.

Se han investigado y estudiado la evolución de las tecnologías antes de cada decisión para no usar tecnologías en desuso o con poca proyección futura, que en un corto periodo de tiempo puede ser un problema para el desarrollo temporal del proyecto.

A continuación vamos se explican las tecnologías utilizadas en cada momento para implementar una solución correcta y el por qué se han elegido.

3.1.1. Arduino

En este apartado vamos a explicar cómo es la estructura de un fichero ejecutable por arduino, ya que el objetivo principal de nuestra aplicación es la generación automática de dicho código.

El código de Arduino está basado en el leguaje “C”, por lo que en primera instancia podremos encontrar las llamadas a las librerías externar usando la palabra reservada “#include”. Seguidamente nos encontraremos las constantes y variables que serán necesarias para controlar los dispositivos conectados. En diferencia a los programas escritos en lenguaje “C”, el código arduino no empieza ejecutándose por la función principal “main”, para ello Arduino tiene dos funciones principales, “setup” y “loop”. En la función “setup” inicializaremos todos los objetos definidos anteriormente. La función “loop”, como su nombre indica, es un bucle infinito, por lo que el código que escribamos dentro de dicha función se estará ejecutando constantemente. Esta función se utiliza para crear la funcionalidad que queramos con nuestra paca Arduino y sus componentes, es donde se leen los valores de los sensores y se da órdenes a los actuadores.

3.1.2. TatAmi

TatAmi es una librería desarrollada en el grupo de investigación ProS de la Universitat Politècnica de València, más concretamente por B. Botella e I. Mansanet, la cual facilita la comunicación de los diferentes componentes y la placa arduino. Gracias a ella el código generado es mucho más fácil y entendible.

- *¿Por qué hemos usado TatAmi?*

Tatami facilita una generación de código más limpia y entendible.

3.1.3. HTML

HTML, siglas de HyperText Markup Language (lenguaje de marcas de hipertexto), hace referencia al lenguaje de marcado para la elaboración de páginas web. Es un estándar que sirve de referencia para la elaboración de páginas web en sus diferentes versiones, define una estructura básica y un código (denominado código HTML) para la definición de contenido de una página web, como texto, imágenes, etc. Es un estándar a cargo de la W3C, organización dedicada a la estandarización de casi todas las tecnologías ligadas a la web, sobre todo en lo referente a su escritura e interpretación.

El lenguaje HTML basa su filosofía de desarrollo en la referenciación. Para añadir un elemento externo a la página (imagen, vídeo, script, etc.), este no se incrusta directamente en el código de la página, sino que se hace una referencia a la ubicación de dicho elemento mediante texto. De este modo, la página web contiene sólo texto mientras que recae en el navegador web (interpretador del código) la tarea de unir todos los elementos y visualizar la página final. Al ser un estándar, HTML busca ser un lenguaje que permita que cualquier página web

escrita en una determinada versión, pueda ser interpretada de la misma forma (estándar) por cualquier navegador web actualizado.

- *¿Por qué hemos usado HTML?*

Ya que queríamos hacer una aplicación que no dependiese del ordenador donde estemos en un momento indicado, decidimos hacer la aplicación web.

3.1.4. Raphaël

Raphaël es una biblioteca JavaScript que funciona en cualquier navegador, y sirve para dibujar gráficos escalables vectoriales (Scalable Vector Graphics (SVG)) para sitios web.

El SVG permite tres tipos de objetos gráficos:

- Elementos geométricos vectoriales (p.e. caminos consistentes en rectas y curvas, y áreas limitadas por ellos)
- Imágenes de mapa de bits /digitales
- Texto

Los objetos gráficos pueden ser agrupados, transformados y compuestos en objetos previamente renderizados, y pueden recibir un estilo común. El texto puede estar en cualquier espacio de nombres XML admitido por la aplicación, lo que mejora la posibilidad de búsqueda y la accesibilidad de los gráficos SVG. El juego de características incluye las transformaciones anidadas, los clipping paths, las máscaras alfa, los filtros de efectos, las plantillas de objetos y la extensibilidad.

Utiliza SVG para la mayoría de los navegadores, pero utilizará VML para versiones anteriores de Internet Explorer. Raphaël actualmente soporta Chrome 5.0 + Firefox 3.0 +, Safari 3.0 +, Opera 9.5 + e Internet Explorer 6.0 +.

- *¿Por qué hemos usado Raphaël?*

En nuestra aplicación, debemos mover las representaciones graficas de todos los componentes Arduino, así como también deben poder conectarse unos con otros. La librería Raphaël nos da las herramientas para poder manejar esto de forma sencilla y rápida. Con Raphaël hemos creado los conectores y las conexiones de los componentes.

3.1.5. JavaScript

JavaScript (abreviado comúnmente "JS") es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico. Actualmente está siendo mantenido por Mozilla.

Se utiliza principalmente en su forma del lado del cliente (client-side), implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas aunque existe una forma de JavaScript del lado del servidor (Server-side JavaScript o SSJS). Su uso en aplicaciones externas

a la web, por ejemplo en documentos PDF, aplicaciones de escritorio (mayoritariamente widgets) es también significativo.

Todos los navegadores modernos interpretan el código JavaScript integrado en las páginas web. Para interactuar con una página web se provee al lenguaje JavaScript de una implementación del Document Object Model (DOM).

- *¿Por qué hemos usado JavaScript?*

Como se ha explicado anteriormente, JavaScript utiliza la potencia de la máquina del cliente para crear páginas dinámicas, a raíz de esto y para quitar carga a nuestros servidores, se ha decidido usar JavaScript. También, gracias a su sintaxis, es muy fácil de aprender y también por eso lo hemos usado en la parte del servidor.

3.1.6. JQuery

jQuery es una biblioteca de JavaScript, creada inicialmente por John Resig, que permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con la técnica AJAX a páginas web. JQuery es la biblioteca de JavaScript más utilizada.

JQuery es software libre y de código abierto, posee un doble licenciamiento bajo la Licencia MIT y la Licencia Pública General de GNU v2, permitiendo su uso en proyectos libres y privativos. jQuery, al igual que otras bibliotecas, ofrece una serie de funcionalidades basadas en JavaScript que de otra manera requerirían de mucho más código, es decir, con las funciones propias de esta biblioteca se logran grandes resultados en menos tiempo y espacio.

- *¿Por qué hemos usado jQuery?*

Se ha usado jQuery para ayudar al desarrollo de la parte cliente de la aplicación, ya que aporta una serie de herramientas que facilitan mucho el desarrollo con JavaScript, pero especialmente en la inserción de código HTML en caliente y las peticiones a servidor sin refresco de página gracias a Ajax.

3.1.7. Bootstrap

Twitter Bootstrap es un framework o conjunto de herramientas de software libre para diseño de sitios y aplicaciones web “responsive design” o diseño adaptable. Contiene plantillas de diseño con tipografía, formularios, botones, cuadros, menús de navegación y otros elementos de diseño basado en HTML y CSS, así como, extensiones de JavaScript opcionales adicionales.

Bootstrap ofrece una serie de plantillas CSS y ficheros JavaScript que nos permiten integrar el framework de forma sencilla y potente en nuestros proyectos webs.

- Permite crear interfaces que se adapten a los diferentes navegadores, tanto de escritorio como tablets y móviles a distintas escalas y resoluciones.

- Se integra perfectamente con las principales librerías JavaScript, por ejemplo JQuery.
- Ofrece un diseño sólido usando estándares como CSS3/HTML5.
- Es un framework ligero que se integra de forma limpia en nuestro proyecto actual.
- Funciona con todos los navegadores, incluido Internet Explorer usando HTML Shim para que reconozca los tags HTML5.

Bootstrap ofrece distintos componentes que podemos usar con unos estilos predefinidos y fáciles de configurar.

- Botones. se pueden aplicar tanto a enlaces como a etiquetas button/input simplemente usando la clase btn. Así podemos distinguir el propósito de cada botón con los distintos estilos prefijados o variar su tamaño.
- Dropdown. Con ellos podemos asociar un menú desplegable a un botón de forma sencilla agrupando distintas opciones.
- Formularios. Bootstrap cuenta con distintos layout que podemos adaptar con las principales necesidades. Además de incluir distintos elementos para remarcar distintas acciones sobre los formularios: focused, disabled, control-group,…
- Plugin de jQuery. Bootstap cuenta con distintos plugins que nos permiten crear ventanas modales, o crear tooltip sobre algún elemento de la página de manera sencilla.

Este framework destaca, como ya hemos dicho, por su facilidad para crear sitios adaptables a las diferentes resoluciones de las pantallas que lo visualicen. Para hacer esto, Bootstrap divide la página en filas, y cada fila en 12 columnas, de esta manera, nosotros tan solo tenemos que hacer uso de las clases que nos ofrece para hacer una distribución correcta.

La idea en la que Bootstrap se basa es, primero se diseña para el dispositivo más pequeño y luego vamos adaptando para pantallas mayores. Este nuevo sistema puede resultar confuso y lioso porque la realidad es que estamos acostumbrados a hacerlo justo al revés. Pero una vez empezamos a utilizar este nuevo sistema nos damos cuenta de lo lógico y sencillo que es diseñar de pequeño a grande.

Bootstrap establece varios tamaños de dispositivos diferentes que los identifica con los siguientes nombres y abreviaciones.

- Extra small devices (xs) – Menor que 768px
- Small devices Tablets (sm) – Mayor o igual a 768px
- Medium devices Desktops (md) – Mayor o igual a 992px
- Large devices Desktops (lg) – Mayor o igual a 1200px

Para entender la forma en la que trabaja este framework, vamos a ver un ejemplo:

```
<div class="container">
        <div class="row">
            <div class="col-xs-12 col-sm-6 col-md-4 col-lg-3">
                <p>Lorem ipsum Dolore ea adipisicing est nisi aliquip nostrud occaecat
exercitation amet.</p>
            </div>
```

Figura 15 - Ejemplo de uso Bootstrap

Aquí vemos que especificamos varias clases a las columnas. Esto significa lo siguiente: con col-xs-12 le decimos que en pantallas pequeñas queremos que cada columna ocupe las 12 disponible, esto significa que ocupará todo el ancho de la pantalla. A continuación con col-sm-6 establecemos que cada columna tenga un ancho de 6 logrando que se vean dos por filas. Con col-md-4 tendremos un tamaño de 4 columnas en escritorios medianos, es decir, 3 columnas por fila. Por último establecemos col-lg-3 para escritorios largos o lo que es lo mismo 4 columnas por fila.

Este método puede parecer algo lioso al principio, pero con un par de pruebas comprobamos que es más sencillo de lo que parece y que lo que debemos hacer es establecer el estilo inicial para móviles e ir cambiando lo que queramos en escritorios mayores. Hay que tener en cuenta que lo que aplicamos a un tamaño se lo aplicamos automáticamente a sus tamaños mayores.

También podemos con Bootstrap, tratar imágenes que se adaptan al tamaño del contenedor ocupando el 100% del mismo. Su uso no puede ser más sencillo que:

```
<img src="#" class="img-responsive" alt="Image">
```

Figura 16 - Ejemplo imagen responsive con Bootstrap

Como vemos sólo debemos añadir la clase img-responsive y la imagen se adaptará al contendor. Siempre podemos añadir nosotros un max-width para que no sobrepase un tamaño deseado.

Otra de las ventajas que nos aporta Bootstrap es el soporte para la mayoría los navegadores. La lista de navegadores soportados es la siguiente:

- Chrome (Mac, Windows, iOS, y Android)
- Safari (Mac e iOS solamente)
- Firefox (Mac, Windows)
- Internet Explorer
- Opera (Mac, Windows)

Internet Explorer estará soportado hasta la versión 8, pero para que funcione correctamente en esta necesitaremos incluir respond.js.

- *¿Por qué hemos usado Bootstrap?*

El uso de Bootstrap conlleva unas ventajas muy significativas, y es que con una sola versión en HTML y CSS se cubren todas las resoluciones de pantalla, es decir, el sitio web creado estará optimizado para todo tipo de dispositivos: PCs,

tabletas, teléfonos móviles, etc. Esto mejora la experiencia de usuario a diferencia de lo que ocurre, por ejemplo, con sitios web de ancho fijo cuando se acceden desde dispositivos móviles.

3.1.8. Ajax

AJAX, acrónimo de Asynchronous JavaScript And XML (JavaScript asíncrono y XML), es una técnica de desarrollo web para crear aplicaciones interactivas o RIA (Rich Internet Applications). Estas aplicaciones se ejecutan en el cliente, es decir, en el navegador de los usuarios mientras se mantiene la comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre las páginas sin necesidad de recargarlas, mejorando la interactividad, velocidad y usabilidad en las aplicaciones.

Ajax es una tecnología asíncrona, en el sentido de que los datos adicionales se solicitan al servidor y se cargan en segundo plano sin interferir con la visualización ni el comportamiento de la página. JavaScript es el lenguaje interpretado (scripting language) en el que normalmente se efectúan las funciones de llamada de Ajax mientras que el acceso a los datos se realiza mediante XMLHttpRequest, objeto disponible en los navegadores actuales. En cualquier caso, no es necesario que el contenido asíncrono esté formateado en XML.

Ajax es una técnica válida para múltiples plataformas y utilizable en muchos sistemas operativos y navegadores dado que está basado en estándares abiertos como JavaScript y Document Object Model (DOM).

- *¿Por qué hemos usado Ajax?*

Gracias a Ajax, se ha podido hacer transacciones con el API Rest sin tener que refrescar la página que el usuario estaba usando. Gracias a Ajax, también se han podido hacer modificaciones en el código HTML sin necesidad de hacer una petición al servidor, se puede mostrar formularios para registrarse, identificarse, añadir imágenes, etc. De esta manera quitamos carga al servidor, ya que no tiene que estar sirviendo las páginas web con cada clic de ratón.

3.1.9. JSON

JSON, acrónimo de JavaScript Object Notation, es un formato ligero para el intercambio de datos. JSON es un subconjunto de la notación literal de objetos de JavaScript que no requiere el uso de XML.

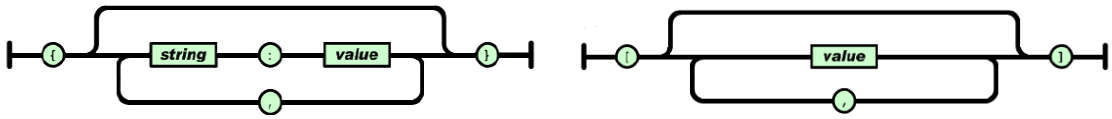
La simplicidad de JSON ha dado lugar a la generalización de su uso, especialmente como alternativa a XML en AJAX. Una de las ventajas de JSON sobre XML como formato de intercambio de datos en este contexto es que es mucho más sencillo escribir un analizador sintáctico (parser) de JSON. En JavaScript, un texto JSON se puede analizar fácilmente usando la función eval(), lo cual ha sido fundamental para que JSON haya sido aceptado por parte de la comunidad de desarrolladores AJAX, debido a la ubicuidad de JavaScript en casi cualquier navegador web.

Estas propiedades, hacen que JSON sea un lenguaje ideal para el intercambio de datos. JSON está constituido por dos estructuras:

- Una colección de pares de ***nombre/valor***. En varios lenguajes es conocido como un **objeto**, registro, estructura, diccionario, tabla hash, lista de claves o un arreglo asociativo.
- Una lista ordenada de ***valores***. En la mayoría de los lenguajes, esto se implementa como arreglos, vectores, listas o secuencias.

Estas son estructuras universales; virtualmente todos los lenguajes de programación las soportan, de una forma u otra. Es razonable que un formato de intercambio de datos, que es independiente del lenguaje de programación, se base en estas estructuras.

Un objeto, es un conjunto desordenado de pares nombre/valor. Un objeto comienza con { (llave de apertura), y termina con } (llave de cierre). Cada nombre es seguido por : (dos puntos), y los pares nombre/valor están separados por ,



(coma).

Un array es una colección de valores. Comienza con [(corchete izquierdo) y termina con] (corchete derecho). Los valores se separan por , (coma).

A continuación, se muestra un ejemplo de uno de los JSON devueltos, en una de las llamadas al servicio web utilizado en este TFM:

```
{
    "date": "23-7-2014",
    "nameUser": "raugaral",
    "fileName": "sim_ple",
    "_id": "53cf8ddc822ebf1418a89c10",
    "circuit": {
        "connections": [
            {
                "_id": "53cf8ddc822ebf1418a89c11",
                "to": {
                    "idDispositive": "arduinoUno_1000",
                    "typeConn": "Analog",
                    "idPin": 5
                },
                "from": {
                    "idDispositive": "LightSensor_1001",
                    "typeConn": "Analog"
                }
            }
        ],
        "actuators": [],
        "sensors": [
            {
                "_id": "53cf8ddc822ebf1418a89c12",
                "attr": {
                    "id": "LightSensor_1001",
                    "typeDisp": "LightSensor",
                    "selectedFunction": "getLightValue",
                    "typeConn": "Analog",
                    "src": "img/light_sensor.png",
                    "width": 40,
                    "height": 110,
                    "x": 749,
                    "y": 189
                }
            }
        ],
        "arduinos": [
            {
                "_id": "53cf8ddc822ebf1418a89c13",
                "attr": {
                    "id": "arduinoUno_1000",
                    "typeDisp": "arduino_uno",
                    "bauds": 9600,
                    "conn": "Serie",
                    "src": "img/arduino_uno.png",
                    "width": 350,
                    "height": 250,
                    "x": 294,
                    "y": 30
                }
            }
        ]
    }
}
```


Figura 18 - JSON devuelto por el API

- *¿Por qué hemos usado JSON?*

Como se ha explicado anteriormente, JSON es un leguaje creado para la gestión de objetos JavaScript, pudiendo así, guardar un objeto directamente en JSON para más tarde ser cargado en memoria. Gracias a las funciones JavaScript “stringify()” y “eval()” puede pasarse un objeto JavaScript a JSON y a la inversa, tan solo llamando a dichas funciones.

3.1.10. REST

REST define un conjunto de principios arquitectónicos por los cuales se diseñan servicios web, haciendo énfasis en los recursos del sistema, incluyendo, cómo se accede al estado de dichos recursos, y cómo se transfieren por HTTP hacia clientes escritos en diversos lenguajes.

Con esta tecnología se permite crear servicios y aplicaciones que pueden ser usadas por cualquier dispositivo o cliente que entienda HTTP, por lo que es increíblemente más simple y usual que tecnologías usadas anteriormente como SOAP o XML-RPC.

REST es el tipo de arquitectura más natural y estándar para crear APIs para servicios orientados a internet.

Para crear la API REST se utiliza el modelo llamado Richardson Maturity Model, donde se dan una serie de patrones y buenas prácticas que hay que utilizar si se quiere crear un software de calidad. Los niveles para ello son el uso correcto de URIs, uso correcto de HTTP e implementar Hypermedia.

1. Utiliza los métodos HTTP de manera explícita.

Una de las características claves de los servicios web REST, es el uso explícito de los métodos HTTP, siguiendo el protocolo definido por RFC 2616. Por ejemplo, HTTP GET se define como un método productor de datos, cuyo uso está pensado para que las aplicaciones cliente obtengan recursos, busquen datos de un servidor web, o ejecuten una consulta, esperando que el servidor web la realice y devuelva un conjunto de recursos.

REST hace que los desarrolladores usen los métodos HTTP explícitamente, de manera que resulte consistente con la definición del protocolo. Este principio de diseño básico, establece una asociación uno-a-uno, entre las operaciones de crear, leer, actualizar y borrar, y los métodos HTTP. De acuerdo a esta asociación:

    1.1. Se usa POST para crear un recurso en el servidor
    1.2. Se usa GET para obtener un recurso
    1.3. Se usa PUT para cambiar el estado de un recurso o actualizarlo
    1.4. Se usa DELETE para eliminar un recurso
2. No mantiene estado.

Los servicios web REST necesitan escalar, para poder satisfacer una demanda en constante crecimiento. Se usan clusters de servidores con balanceadores de carga y alta disponibilidad, proxies, y gateways para conformar una topología útil/práctica, que permita transferir peticiones de un equipo a otro, para disminuir el tiempo total de respuesta de una invocación al servicio web. El uso de servidores intermedios para mejorar la escalabilidad, hace necesario que los clientes de servicios web REST, envíen peticiones completas e independientes, es decir, se deben enviar peticiones que incluyan todos los datos necesarios, para cumplir el pedido, de manera que los componentes en los servidores intermedios, puedan re-direccionar y gestionar la carga, sin mantener el estado localmente entre las peticiones.

3. Despliega URIs con forma de directorios

Desde el punto de vista del cliente de la aplicación que accede a un recurso, la URI determina qué tan intuitivo va a ser el servicio web REST, y si éste va a ser utilizado tal como fue pensado al momento de diseñarlo. La tercera característica de los servicios web REST es justamente sobre las URIs.

Las URIs de los servicios web REST deben ser intuitivas, hasta el punto de que sea fácil adivinarlas. Se ha de pensar en las URI como una interfaz auto-documentada, que necesita de muy poca o ninguna, explicación o referencia, para que un desarrollador pueda comprender a lo que apunta, y a los recursos derivados relacionados.

4. Transfiere XML, JSON o ambos.

Estos son los dos formatos en los que un servicio web REST, puede mostrar los datos devueltos por el servidor.

- *¿Por qué hemos usado REST?*

Han sido varios los motivos por los cuales se ha decidido utilizar REST. Para comenzar vemos el rendimiento, utilizando REST la arquitectura se simplifica consiguiendo así un gran aumento a la hora del rendimiento.

Como segundo punto importante a destacar es el considerable aumento de velocidad debido a que las peticiones se simplifican.

No existe curva de aprendizaje con lo que se consiguen resultados óptimos y visualmente interpretables en poco tiempo.

Otro punto muy importante son las futuras actualizaciones y desarrollos paralelos que se puedan hacer, ya que gracias a esta forma de acceder a los datos se puede ampliar la aplicación o crear nuevas sin tener que desarrollar nada.

3.1.11. Node.js

Node.js es un entorno de programación en la capa del servidor basado en el lenguaje de programación JavaScript, con Entrada/Salida de datos en una arquitectura orientada a eventos y basado en el motor JavaScript V8. Fue creado con el enfoque de ser útil en la creación de programas de red altamente escalables, como por ejemplo, servidores web. Fue creado por Ryan Dahl en 2009 y su evolución está apadrinada por la empresa Joyent, que además tiene contratado a Dahl en plantilla.

Node.js es similar en su propósito a Twisted o Tornado de Python, Perl Object Environment de Perl, React de PHP, libevent o libev de C, EventMachine de Ruby y de Java existe Apache MINA, Netty, Akka, Vert.x, Grizzly o Xsocket. Al contrario que la mayoría del código JavaScript, no se ejecuta en un navegador, sino en el servidor. Node.js implementa algunas especificaciones de CommonJS. Node.js incluye un entorno REPL para depuración interactiva, el cual es difícil y poco intuitivo, ya que se maneja mediante terminal de comandos.

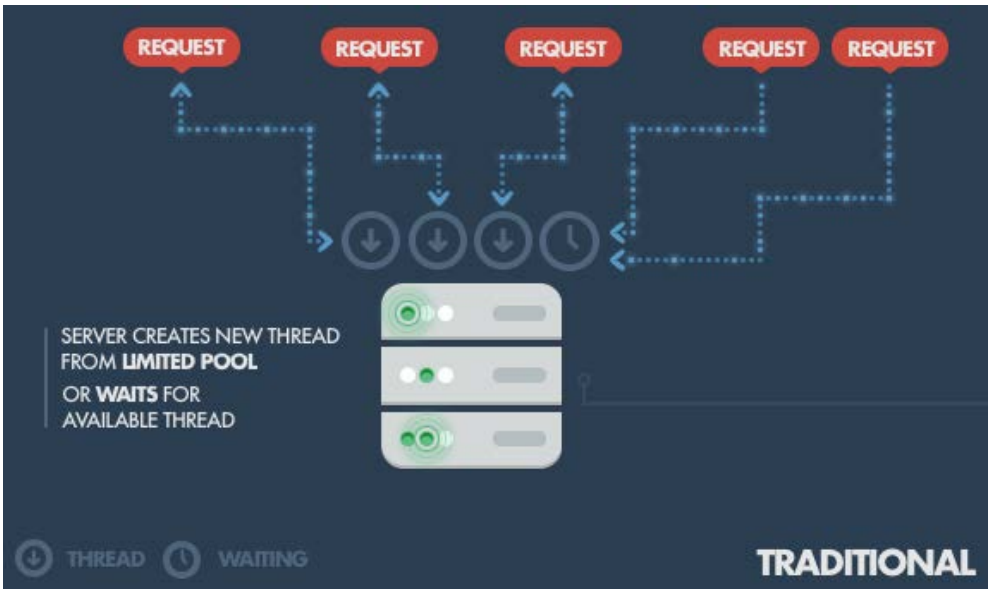


Figura 19 - Servidor web tradicional

En comparación con los servidores web tradicionales donde cada conexión (Request) genera un nuevo hilo, ocupando memoria RAM del sistema y, finalmente, el gasto excesivo de RAM disponible (como podemos ver en la ilustración anterior), Node.js opera en un solo hilo, usando sin bloqueo de Entrada/Salida, que le permite soportar decenas de miles de conexiones simultáneas (organizadas en el bucle de eventos) como podemos ver en la ilustración siguiente.

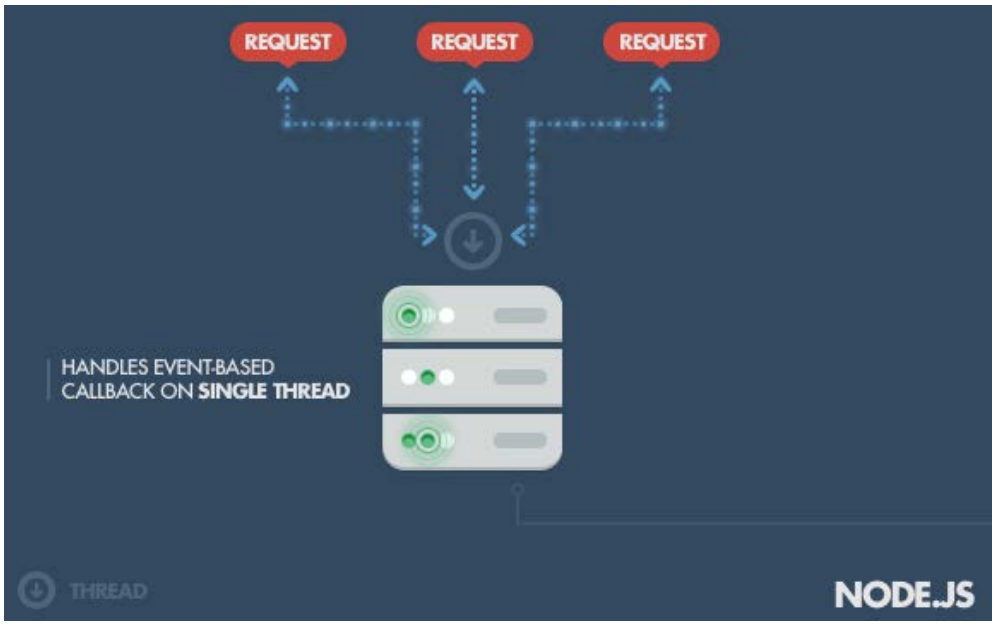


Figura 20 - Servidor node.js

Un cálculo rápido : suponiendo que cada hilo tiene potencialmente un uso 2 MB de memoria, que se ejecutan en un sistema con 8 GB de memoria RAM nos sitúa en un máximo teórico de 4.000 conexiones simultáneas, más el costo de contexto (cambio entre hilos). Ese es el escenario que normalmente ocupa los servidores

web tradicionales. Al evitar todo eso, Node.js logra niveles de escalabilidad de más de 1 millón de conexiones simultáneas (como prueba de concepto).

A parte de todo lo referido a su gestión de la memoria, Node.js tiene un punto a favor muy importante contra los servidores web tradicionales, este punto es su gestor de paquetes NPM (Node Package Manager). Gracias a NPM podemos instalar extensiones o plugins del servidor con un simple comando.

- *¿Por qué hemos usado Node.js?*

Como en el cliente se usa JavaScript, se crean los mensajes con JSON, necesitábamos una forma fácil de manejar la información que fluye entre el cliente y el servidor, ya que el lenguaje de programación para Node.js es JavaScript, tan solo hace falta utilizar la función "eval()", para convertir JSON en objetos JavaScript.

3.1.12. Express.js

Express.js es un liviano framework para Node.js que ayuda a organizar las aplicaciones web utilizando la arquitectura MVC (modelo - vista - controlador) en el lado del servidor. Express.js ayuda a manejar todo con rutas, manejadores de peticiones y vistas. Gracias a su fácil configuración y la gran ayuda que ofrece a Node.js se está convirtiendo en un estándar de facto.

- *¿Por qué hemos usado Express.js?*

Se ha usado Express.js ya que es el complemento perfecto de Node.js para crear APIs RESTfull. Ayuda a todo lo que es la propia estructuración de los ficheros creando un árbol de directorios, facilita la configuración del servidor, engloba las peticiones y respuestas del servidor para hacerlas más fáciles.

3.1.13. MongoDB

MongoDB es un sistema de base de datos NoSQL orientado a documentos, desarrollado bajo el concepto de código abierto.

MongoDB forma parte de la nueva familia de sistemas de base de datos NoSQL. En vez de guardar los datos en tablas como se hace en las base de datos relacionales, MongoDB guarda estructuras de datos en documentos tipo JSON con un esquema dinámico (MongoDB llama ese formato BSON), haciendo que la integración de los datos en ciertas aplicaciones sea más fácil y rápida.

MongoDB tiene la capacidad de realizar consultas utilizando JavaScript, haciendo que estas sean enviadas directamente a la base de datos para ser ejecutadas.

MongoDB tiene las siguientes características:

- Orientado a guardar documentos JSON.
- Cualquier atributo puede ser índice.
- Fácil replicación y alta disponibilidad.
- Escalar horizontalmente sin comprometer la funcionalidad.
- Consultas ricas basadas en documentos.

- Actualizaciones rápidas in situ.
- Agregación flexible y procesamiento de datos.
- Almacene archivos de cualquier tamaño.

- *¿Por qué hemos usado MongoDB?*

Mongo de almacena los datos usando colecciones de JSON y se puede acceder a ellos mediante consultas resueltas en JavaScript, ya que nosotros usamos JSON y JavaScript tanto en el cliente como en el servidor, optamos por usar esta tecnología que es la que mejor nos encaja con todo el sistema que hemos montado.

3.2. Conclusiones

En este capítulo se ha presentado la tecnología empleada para llevar a cabo el desarrollo de este proyecto final de master. Como hemos podido ver, se ha escogido el lenguaje JavaScript como base para todas las herramientas utilizadas, tanto en la parte del cliente como en el servidor, y para hacer el intercambio información se ha optado por JSON mediante un api REST. En cuanto al almacenamiento de la información, se ha decidido utilizar una base de datos orientada a documentos, ya que nos permite poder guardar los JSON directamente sin transformar.

4. Desarrollo de la propuesta

Este cuarto capítulo se divide en cinco partes. Primero veremos una visión global de la aplicación presentada. Después, se van a explicar los requisitos del trabajo, bien sean funcionales o no funcionales. En la tercera parte se desarrolla el proceso de diseño del TFM. Para continuar, en el cuarto apartado del capítulo se va a explicar el proceso de implementación y por último la parte encargada de comprobar el correcto funcionamiento del trabajo propuesto.

Hay que comentar que se han utilizado diferentes metodologías de trabajo para cada una de las propuestas ya que estas se adaptan mejor para solucionar los problemas. Para el desarrollo de la API se ha optado por utilizar una metodología guiada por pruebas y para el desarrollo de la aplicación web se ha decidido utilizar un desarrollo incremental.

4.1. Visión global de la solución

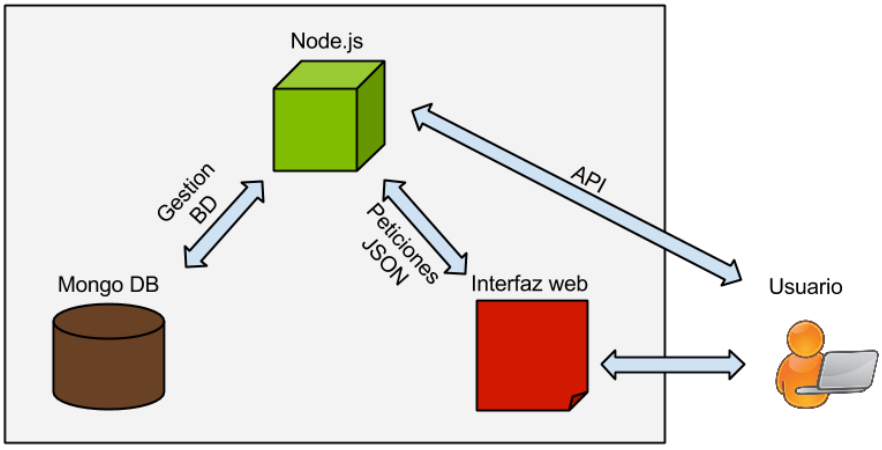


Figura 21 - Diagrama general de la aplicación

En la anterior imagen, podemos ver un esquema de nuestra solución. Cabe destacar, como pieza central, el servidor Node.js, ya que además de servir la página web y todos sus ficheros necesarios, es la que interactúa con los ficheros guardados en la base de datos MongoDB. Para acceder a dicha información almacenada, Node.js proporciona un API de comunicación REST. En ella nos encontramos, la lógica necesaria para crear los diagramas de los circuitos almacenados. Gracias a dicha API los usuarios pueden acceder y modificar los datos sin tener que acceder a nuestra interfaz web, incluso crear sus propias aplicaciones.

Para facilitar la gestión y la creación de los modelos, se proporciona una aplicación web, creada en su totalidad con la tecnología HTML, css y JavaScript. Dicha aplicación web facilita la creación de modelos mediante dibujos de los componentes, los cuales se pueden conectar entre ellos. Además, proporciona una validación del circuito para comprobar que todos los componentes están correctamente configurados y conectados. También, proporciona los datos de la simulación de forma estructurada.

4.2. Requisitos

En el TFM se ha desarrollado una herramienta web para la creación grafica de programas para arduino, con un simulador para, sin tener que cargar nuestro programa realizado en una placa arduino, para ver cuál sería el resultado.

A continuación vamos a detallar las funcionalidades y requisitos que debe cumplir.

4.2.1. Requisitos funcionales

Un requisito funcional define una función del sistema software. Una función es descrita como un conjunto de entradas, comportamientos y salidas. Los detalles

funcionales pueden ser: cálculos, detalles técnicos, manipulación de datos etc. Es decir, funcionalidades que un sistema debe cumplir.

Los requisitos funcionales más importantes son los siguientes:

- **Aplicación web:** Nuestro objetivo es proporcionar una aplicación web donde el usuario pueda usar el servicio el cual ponemos a su disposición.
    - Crear un circuito gráficamente.
    - Registrarse como usuario.
    - Entrar en el sistema una vez se haya registrado.
    - Poder guardar los circuitos creados.
    - Recuperar los circuitos guardados anteriormente.
    - Verificar que el circuito este correctamente conectado y que no tenga errores de configuración.
    - Simular el funcionamiento del circuito.
    - Crear código Arduino listo para compilar y cargar en el dispositivo.
- **API de comunicación:** Se debe proporcionar un API que sirva para comunicarse con todos los datos de la aplicación. El API debe dar soporte para:
    - Registro de usuario.
    - Validación de usuario.
    - Almacenamiento de circuito.
    - Recuperación de circuito.
    - Empezar una simulación.
    - Cambiar el estado de los actuadores.
    - Parar la simulación.
    - Crear código Arduino.

La aplicación web, a los usuarios no registrados, tan solo dejara crear el circuito, para activar las demás funciones el usuario debe estar registrado y haber entrado en el sistema.

El código deberá ser guardado antes de comprobar si está bien configurado. En el caso que este correcto, dará la opción de descargar el código.

El código resultante, será empaquetado y comprimido en formato zip, para poder descargar todos los ficheros de una sola vez.

Para realizar la simulación el código deberá ser guardado con anterioridad y, el modelo creado, deberá ser correcto.

4.2.2. Requisitos no funcionales

Un requisito no funcional es un requisito que especifica criterios que pueden usarse para juzgar la operación de un sistema en lugar de sus comportamientos específicos. Los requisitos no funcionales más importantes del proyecto son:

- **Rendimiento:** La aplicación debe tener un rendimiento aceptable en todo momento.

- **Usabilidad:** La curva de aprendizaje para el uso de nuestra aplicación deberá ser lo menor posible.
- **Portabilidad:** La aplicación debe ser accesible desde cualquier parte del mundo y desde cualquier dispositivo.
- **Escalabilidad:** El usuario no debe percibir la carga del sistema, deberá comportarse con la misma rapidez sin importar el número de usuarios.

4.3. Diseño

El proyecto se ha desarrollado siguiendo el patrón MVC (modelo, vista, controlador), para conseguir un nivel de acoplamiento bajo y hacer la aplicación más escalable. Vamos a ver en más profundidad como se ha diseñado la aplicación.

4.3.1. Base de datos

Como hemos dicho anteriormente, se ha usado MongoDB como almacenamiento interno de la aplicación. MongoDB es una base de datos no relacional y "no sql", como se ha explicado anteriormente, en esta base de datos se guardara código JSON, lo que implica que nuestras tablas no estarán conectadas entre sí. En MongoDB el concepto "tabla" se llama "mongo schema" ya que no es una tabla como tal lo que guarda. Vamos a ver más en detalle los schemas creados para dar soporte a nuestra aplicación.

- *Schemas utilizados*

Para dar soporte a nuestra aplicación hemos necesitado dos schemas.

- **UserSchema:** En este schema se guardan los usuarios registrados en el sistema. Este esquema es el más sencillo, ya que tan solo almacena la información básica de los usuarios registrados. Sus campos son:
  - **Name:** Campo de tipo String donde se guarda el nombre con el que el usuario podrá entrar en el sistema.
  - **Pass**: Campo de tipo String donde se guarda la contraseña que el usuario usará para entrar en el sistema.
  - **Email**: Campo String donde se guarda el email del usuario.
- **CircuitSchema:** Este schema guardara toda la información relativa a los circuitos creados por los usuarios. En él se guarda:
  - **NameUser:** Campo de tipo String. Es el nombre del usuario que ha creado el circuito.
  - **Date:** Campo de tipo String. Guarda la fecha de la última modificación del circuito.
  - **Filename:** Campo de tipo String. Guarda el nombre identificativo que le ha dado el usuario al crear el circuito.
  - **Circuit:** Esta tupla contendrá los componentes del circuito. Está formado, a su vez, por tres tuplas más.

- **Arduinos:** Campo de tipo Array. Está formado por todos atributos básicos necesarios que debe tener un Arduino.
    - **Id:** Campo de tipo String. Identificador único del Arduino.
    - **Src:** Campo de tipo String. Ruta donde se encuentra la imagen que simboliza a ese arduino.
    - **Height:** Campo de tipo Number. Representa la altura de la imagen del Arduino.
    - **Width:** Campo de tipo Number. Representa la anchura de la imagen del Arduino.
    - **X:** Campo de tipo Number. Representa el punto horizontal en la pantalla donde se muestra el dibujo del Arduino.
    - **Y:** Campo de tipo Number. Representa el punto vertical en la pantalla donde se muestra el dibujo del Arduino.
    - **TypeDisp:** Campo de tipo String. Tipo dentro de la familia Arduino.
    - **Baunds:** Campo de tipo Number. Baundios a los que el Arduino se comunica con el ordenador.
    - **Conn:** Campo de tipo String. Tipo de conexion con el ordenador.
    - **Ip:** Campo de tipo String. ip que tendrá asignada el Arduino, en caso de que la conexión sea Ethernet.
    - **Port:** Campo de tipo String. Puerto al que accederá el ordenador en caso de que la conexión sea Ethernet.
    - **Mac:** Campo de tipo String. Mac del arduino en caso de que la conexión sea por Ethernet.
- **Sensors:** Campo de tipo Array. Almacena todos los atributos de los sensores.
    - **Id:** Campo de tipo String. Identificador único del sensor.
    - **Src:** Campo de tipo String. Ruta donde se encuentra la imagen que simboliza a ese sensor.
    - **Height:** Campo de tipo Number. Representa la altura de la imagen del sensor.
    - **Width:** Campo de tipo Number. Representa la anchura de la imagen del sensor.
    - **X:** Campo de tipo Number. Representa el punto horizontal en la pantalla donde se muestra el dibujo del sensor.
    - **Y:** Campo de tipo Number. Representa el punto vertical en la pantalla donde se muestra el dibujo del sensor.
    - **TypeDisp:** Campo de tipo String. Tipo de sensor.

- **TypeConn:** Campo de tipo String. Tipo de conexión. Esta puede ser analógica o digital.
- **SelectedFunction:** Campo de tipo String. Este campo guarda la función que se usará del sensor, en el caso que el sensor pueda hacer más de una función.
- **Umbral:** Campo de tipo String. Valor, en caso de que la función seleccionada necesite guardar un umbral.

- **Actuators:** Campo de tipo Array. Almacena todos los atributos de los actuadores.

  - **Id:** Campo de tipo String. Identificador único del sensor.
  - **Src:** Campo de tipo String. Ruta donde se encuentra la imagen que simboliza a ese sensor.
  - **Height:** Campo de tipo Number. Representa la altura de la imagen del sensor.
  - **Width:** Campo de tipo Number. Representa la anchura de la imagen del sensor.
  - **X:** Campo de tipo Number. Representa el punto horizontal en la pantalla donde se muestra el dibujo del sensor.
  - **Y:** Campo de tipo Number. Representa el punto vertical en la pantalla donde se muestra el dibujo del sensor.

  - **TypeDisp:** Campo de tipo String. Tipo de sensor.
  - **State:** Campo de tipo String. Representa el estado del actuador.
- **Connections:** Campo de tipo Array. Almacena las conexiones entre Arduinos y dispositivos.
  - **From:** Campo de tipo Array. Guarda la referencia al dispositivo origen de la conexión.
    - **IdDispositive:** Campo te tipo String. Id del dispositivo origen.
    - **TypeConn:** Campo de tipo String. Tipo de conexión del dispositivo origen.
    - **IdPin:** Campo de tipo Number. Numero de pin dentro del dispositivo.
  - **To:** Campo de tipo Array. Guarda la referencia al dispositivo destino de la conexión.

    - **IdDispositive:** Campo te tipo String. Id del dispositivo origen.
    - **TypeConn:** Campo de tipo String. Tipo de conexión del dispositivo origen.

- **IdPin:** Campo de tipo Number. Numero de pin dentro del dispositivo.

A continuación podemos ver un diagrama del schema anteriormente descrito:

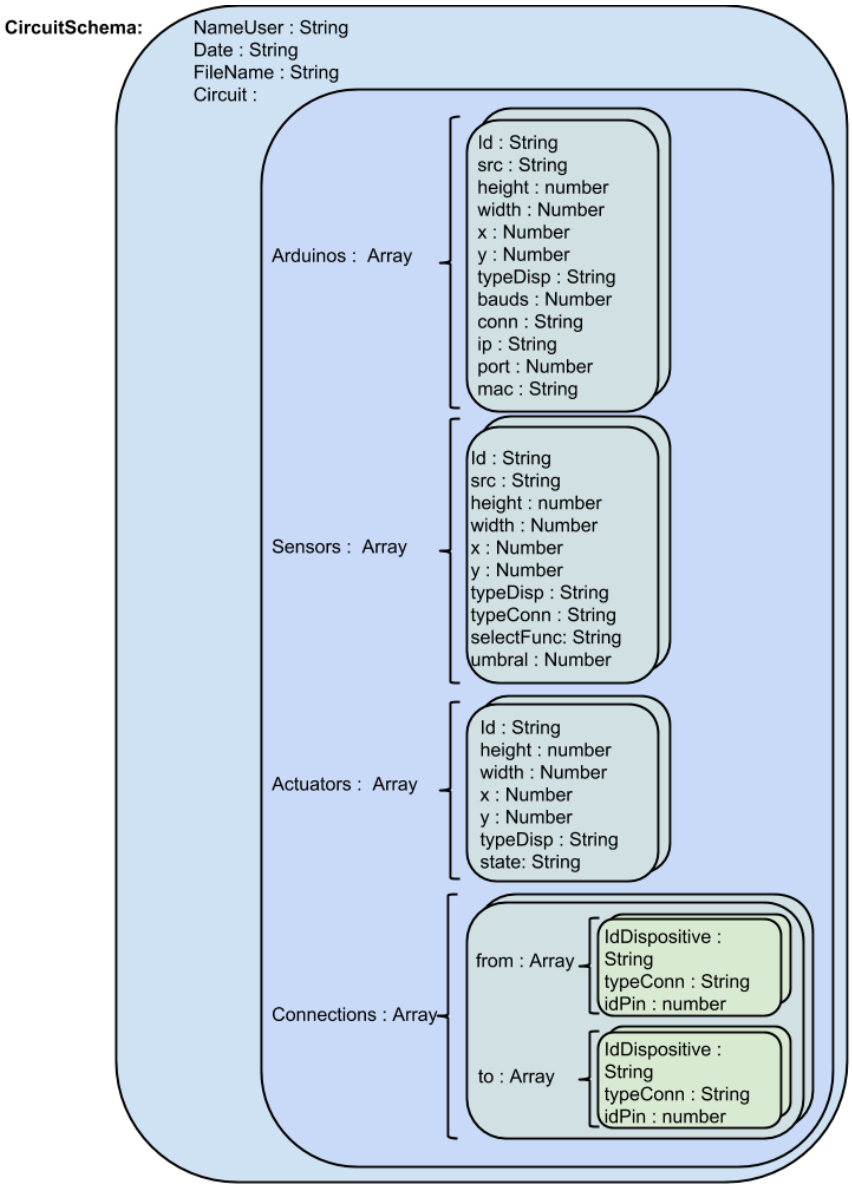


Figura 22 - Schema "Circuit"

4.3.2. API de comunicación

Vamos a comenzar este punto explicando que es, una API (Application Programming Interface) o IPA (Interfaz de programación de aplicaciones) es un conjunto de funciones y métodos que ofrece una biblioteca para ser utilizada por otro software como una capa de abstracción.

Decidimos crear una API para nuestro proyecto por las siguientes razones:

- **Rapidez**: Podemos recuperar de una forma muy rápida toda la información que necesitamos para nuestra aplicación, sin necesidad de almacenarla en nuestros dispositivos la información.

- **Localización**: Da lo mismo el lugar en el mundo en el que te encuentres, simplemente teniendo acceso a internet, tenemos acceso a nuestra información almacenada
- **Mantenimiento**: Es mucho más fácil crear un mantenimiento y una evolución aceptable si accedemos a nuestros recursos mediante una API, ya que las mejoras son totalmente transparentes a los programadores que las usan.
- **Robustez**: Nuestra API se ha construido buscando la robustez y seguridad de los datos, dado que no sabemos el número de personas que van a consultarla a la vez. Esto se ha podido gestionar fácilmente gracias a node.js.

Para lograr todo esto se ha utilizado un servidor Node.JS, el cual se ha ganado una tremenda fama por utilizar JavaScript en la parte del servidor, así como por su ejecución asíncrona. Gracias a esto Node.JS es un servidor perfecto para crear APIs.

4.3.3. Diseño aplicación web

El diseño web está dividido en dos partes, la parte de lógica programada en JavaScript y guardada en ficheros ".js" y la vista de la web se ha desarrollado en HTML y se ha guardado en ficheros ".HTML". Se ha hecho de esta forma para asegurar un grado de mantenibilidad alto, ya que toda la lógica del cliente la tenemos agrupada en un directorio con diferentes ficheros ".js" y todo lo referente a la vista, está en otro directorio formado por un fichero ".HTML"

Al ser una aplicación web el apartado de portabilidad está totalmente cubierto, ya que se puede acceder en cualquier instante, desde cualquier parte del mundo y desde cualquier tipo de dispositivo. Para conseguir esto, se ha usado tecnología "Responsive Design" con ayuda de la librería "Bootstrap", gracias a ella da igual la resolución de pantalla del dispositivo con el que se acceda a la aplicación. Por ejemplo, si entramos a la aplicación desde un móvil o desde un ordenador con resolución Full HD, esta se verá igual.

Diseño de la interfaz

Este ha sido uno de los puntos que más dificultad ha conllevado al desarrollo total del proyecto, ya que se quería crear una interfaz que se adecuara a cualquier dispositivo, además de ser fácil e intuitiva para el usuario novel.

La interfaz de usuario se ha construido, intentando que la experiencia para el usuario sea lo más fácil posible. Por ello se ha construido la aplicación utilizando una barra de menú superior y otra en el lateral izquierdo, dejando así, la mayor parte de la pantalla para el diseñador.

En la barra superior, utilizando botones grandes, se pueden ver las opciones de "Guardar", "Guardar Como", "Cargar", "Borrar Todo", "Compilar" y "Simular". En la barra de menú izquierda, se pueden ver claramente diferenciados 3 grupos, "Arduinos", "Sensores", "Actuadores". Al lado de cada título de grupo se ha puesto el número de componentes que constituye el grupo. Si se pincha en el título, esta

sección se contrae dejando así, más sitio para el diseñador. Definimos como diseñador, a la parte blanca donde el usuario podrá ver los componentes y conectarlos entre sí.

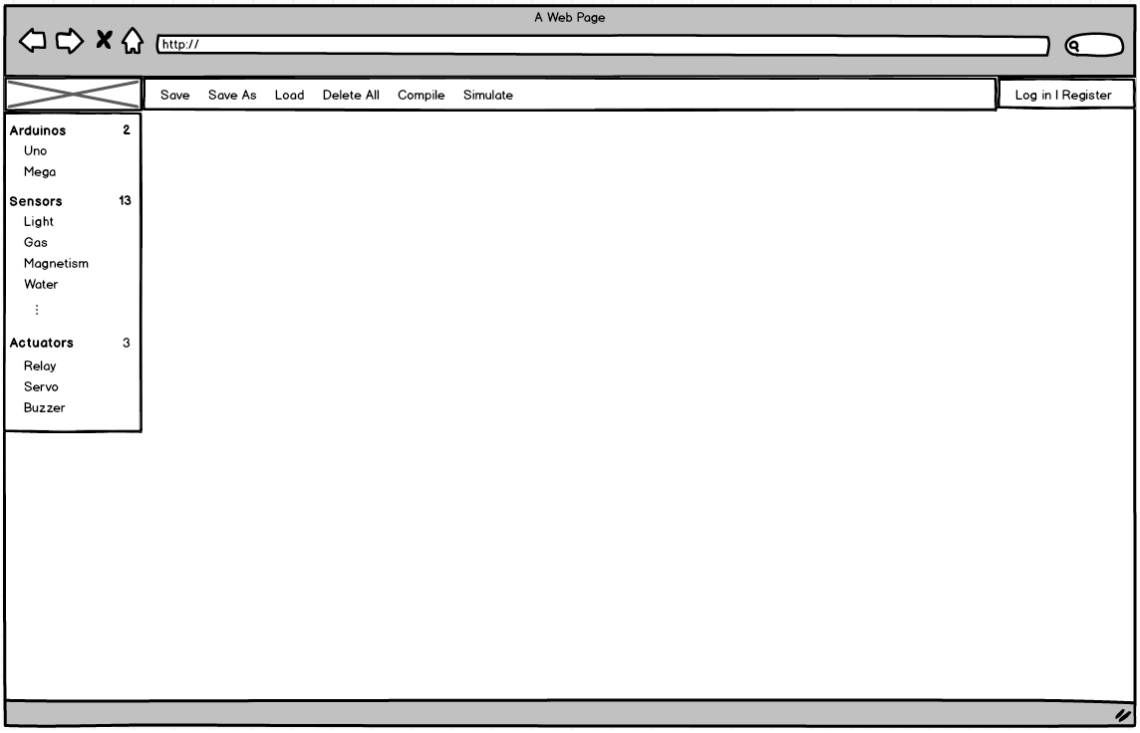


Figura 23 - Boceto de la interfaz web

4.4. Implementación

En este tercer apartado vamos a hablar de la implementación de cada uno de los bloques, viendo para cada uno de éstos la parte de código más importante.

4.4.1. Implementación del API

Como bien se ha descrito anteriormente, una vez el sistema ha generado los modelos es necesario almacenarlos en algún lugar para ser consultados por los usuarios si estos lo requieren. Estos se almacenan en una base de datos y debemos acceder a ella para poder recuperar los modelos creados anteriormente. Este acceso se realiza mediante una API que haga de capa de abstracción de la base de datos. Esta parte es la que se va a explicar a continuación.

Se ha podido crear el API rápidamente, de forma robusta y separando la parte que usaran las aplicación para acceder a los datos, de la lógica del servidor, gracias a la utilización del framework “Express.js”.

Express.js nos crea un árbol de directorios y ficheros de configuración predefinido que nos ayuda a separar, las peticiones que recibirá el servidor, de la lógica y de la vista de la web.

```
// Calls
app.get('/', routes.index);

app.post('/tesina/save', files.save);
app.post('/tesina/load', files.load);
app.delete('/tesina/files/:nameUser', files.deleteAllForUser);
app.get('/tesina/files/:nameUser', files.showAllForUser);
app.delete('/tesina/files', files.deleteAll);
app.get('/tesina/files', files.showAll);
app.post('/tesina/makeFile', files.createPhisicalFiles)

app.post('/tesina/createuser', users.create);
app.get('/tesina/viewusers', users.view);
app.get('/tesina/cleanusers', users.clean);

app.post('/tesina/login', users.login);

app.put('/tesina/simulator/start', simulator.startSimulation);
app.get('/tesina/simulator/:nameUser/:fileName', simulator.getSimulationValues);
app.put('/tesina/simulator/actuator', simulator.changeStateActuator);
app.get('/tesina/simulator/:nameUser/:fileName/:idDisp', simulator.getSimulationDispositive);


app.options('/tesina/simulator/sensor/:typesensor', simulator.optionsSensor);
app.head('/tesina/simulator/sensor/:typesensor', simulator.infoSensor);
```

Figura 24 - Creación Handlers API

En la ilustración anterior se puede ver cómo están creados los “handlers” que recibirán la llamada de las aplicaciones externas.

Teniendo en cuenta que “app” es el objeto servidor, podemos ver que los “handlers” se crean llamando al tipo función que queremos que responda a la petición y pasándole la ruta que escuchara y la función que lanzara cuando reciba una petición.

De esta manera, podemos encontrar que para crear un usuario, hemos definido un manejador de la siguiente manera, “`app.post('/tesina/createuser', users.create);`” por lo que cuando un cliente haga una llamada a nuestra ip y puerto, y seguidamente escriba “/tesina/createuser” se lanzara la función créate, dentro del controlador (“route”) “users”.

Los “routes” son los ficheros que contienen la lógica del servidor, contienen las funciones que se lanzaran al recibir una petición. Siguiendo con el ejemplo anterior, vamos a ver la función que lanza para crear un registro de usuario.

Como hemos visto anteriormente, la llamada se realizaran utilizando el método “post”, por lo que lo primero que se hace es recuperar los datos del cuerpo del mensaje. Una vez obtenidos, y usando el “Schema” de “MongoDB” creamos un objeto con los datos del usuario. Seguidamente, buscamos si dicho usuario ya existe, en cuyo caso, devolvemos en la respuesta el código de error 409 (Conflict) y el texto “User exist!”, después mostramos en el terminal del servidor un aviso con el nombre del usuario y lo ocurrido. Si en la comprobación resulta que el usuario no existe, intentamos guárdalo en la base de datos, en el caso de que hubiese algún problema, se le devolvería al usuario un código 420 y el texto del error; si se guardase correctamente, devolveríamos, únicamente un código 200.

```
exports.create = function(request, response){
    response.header("Access-Control-Allow-Origin", "*");
    params = request.body;

    user = new Users({
        name: params.name,
        pass: params.pass,
        email: params.email
    });

    //busca usuaris en eixe nom, pero no torna ni el __v ni el pass
    Users.find({name: params.name},{__v: 0, pass: 0})
    .exec(function(err, data){
        if(data.length > 0 ){
            response.send(409, "User exist!"); // 409 conflict
            console.log(">> Usuario "+params.name+" intento resgistrarse, pero ya esxiste.");
        }
        else{
            user.save(function(err, data){
                if(err){ response.json(420, err); console.log("Error al crear el usuario "+name
                else{ response.send(200); console.log(">> User "+params.name+" created!"); }
            })
        }
    });
}
```

Figura 25 - Funcion para crear un usuario

Durante el desarrollo de las respuestas a las peticiones, se tuvo problemas al intentar acceder a los recursos del API desde fuera del servidor local, ya que el navegador del cliente ve que la respuesta no se estaba emitiendo de la misma red, por lo tanto la bloqueaba. Para poder solucionar este problema, desde el servidor, se tuvo que añadir la cabecera "`Access-Control-Allow-Origin = *`" a los paquetes devueltos.

4.4.2. Implementación de la interfaz web

En este apartado se va a explicar toda la creación de la aplicación web.

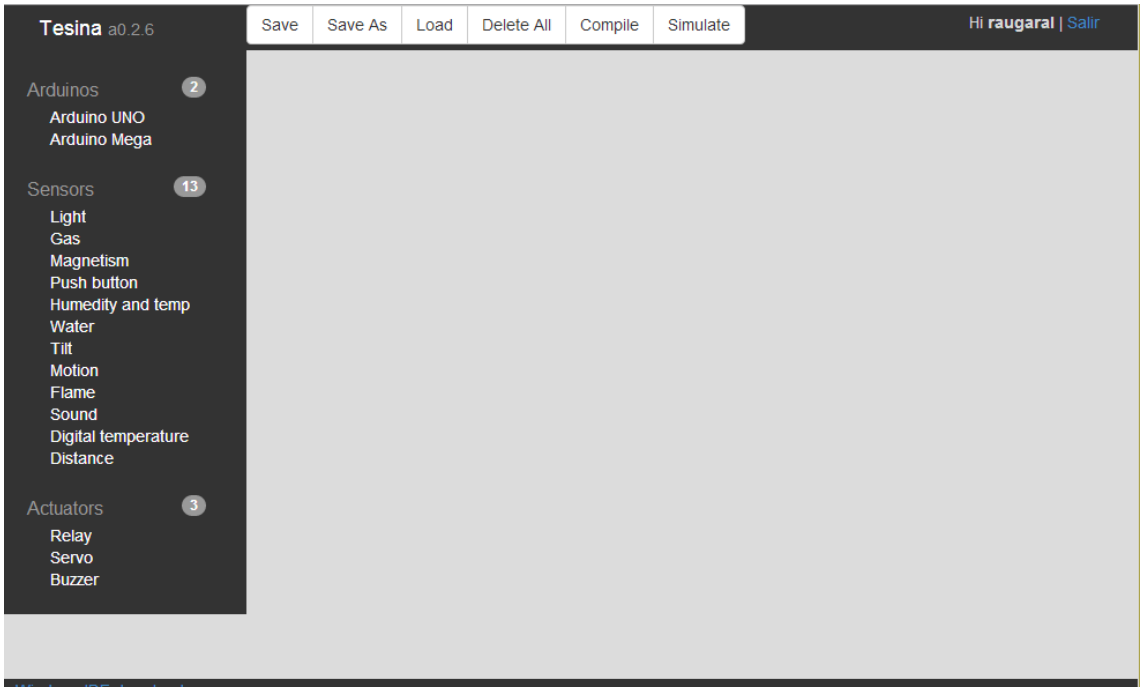


Figura 26 - Interfaz web final de nuestra aplicación

Como se puede ver en la ilustración anterior, este es el aspecto final de nuestra aplicación. Como mostramos en el boceto anteriormente, la página web está formada por una barra superior y otra en el lateral izquierdo.

A simple vista, se puede ver la gran incursión de “Bootstrap” en toda la web, ya que el tipo de botones utilizados, los menús de la barra izquierda y los colores, son característicos de “Bootstrap”.

- *Disposición interna de la web*

Si viésemos la disposición interna de la web, veríamos que está formada por un gran “div” que utiliza una clase llamada “container-fluid” que engloba todos los elementos visibles de la web. La clase “container-fluid” indica a “Bootstrap" que es un contenedor de ancho variable, que usa el 100% de la página y que contendrá filas.

Dentro de él, encontramos tres elementos “div” más, llamados “top”, “center” y “bottom”. Cada uno usa la clase “row” para indicar a “Bootstrap” que es una fila dentro del “container”. Como sus propios nombres indican, cada “div” engloba los elementos de arriba, centro y abajo.

Si nos centramos en el div “top”, podremos ver que este está dividido a su vez en tres div más, “topRight”, “topCenter”, “topLeft”.

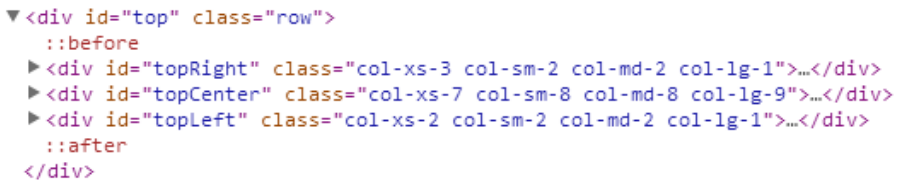


Figura 27 – Codigo de la región "top" de la interfaz web

Como podemos ver en la ilustración anterior, cada región tiene unas clases asociadas que vamos a analizar a continuación, ya que este es el propósito del uso de “Bootstrap”. (Para comprender la disposición de las regiones, cabe recordar que “Bootstrap” divide el área en 12 columnas)

La región “topRight”, está pensada para albergar el logotipo o nombre del proyecto, al estar ubicado entre el menú superior y la barra lateral izquierda tiene que tener la misma altura que el menú superior y el mismo ancho que la barra lateral izquierda, para ello se han usado las siguientes clases:

- Col-xs-3: esto le dice al navegador que en navegadores muy pequeños, esta región debe ocupar un 3/12 del ancho de la pantalla.
- Col-sm-2, col-md-2: estas clases le indican a los navegadores pequeños y medianos, que esta región debe ocupar 2/12 del ancho de la pantalla.
- Col-lg-1: por último, con esta clase se le indica al navegador que en navegadores con gran resolución, tan solo ocupe 1/12 del ancho de la pantalla.

La región “topCenter” contiene el menú superior de la página, por ello, dentro tiene otro “div” con la clase “btn-group” que hace que todos los botones que metamos

dentro de esta región, se agrupen formando una botonera. Cada botón tiene asociada una función, la cual se ejecutara al pulsar sobre el mismo. Más a delante veremos dichas funciones. Para hacer que la botonera tenga un diseño adaptable, se ha usado las siguientes clases:

- Col-xs-7: esto le dice al navegador que en navegadores muy pequeños, esta región debe ocupar un 7/12 del ancho de la pantalla.

- Col-sm-8, col-md-8: estas clases le indican a los navegadores pequeños y medianos, que esta región debe ocupar 8/12 del ancho de la pantalla.

- Col-lg-9: por último, con esta clase se le indica al navegador que en navegadores con gran resolución, tan solo ocupe 9/12 del ancho de la pantalla.

Por último, la región “topLeft” contiene el enlace de registro e inicio de sesión. Esta región contiene las clases con el ancho restante para completar los 12/12, por ello podemos decir que:

- Col-xs-2, col-sm-2, col-md-2: esto le dice al navegador que en navegadores muy pequeños, esta región debe ocupar un 3/12 del ancho de la pantalla.

- Col-lg-1: por último, con esta clase se le indica al navegador que en navegadores con gran resolución, tan solo ocupe 1/12 del ancho de la pantalla.

Si sumamos, por ejemplo, los tres “col-xs” podremos ver que dan 12, pero al sumar los “col-lg” vemos que nos da 11, esto es porque vimos que con las medidas estándar de “Bootstrap”, las columnas en las pantallas de mayor resolución, se quedaban pequeñas. Para mejorar la vista en dispositivos de gran resolución, modificamos la clase “col-lg” haciendo que ocupase el 11% de la página en lugar del 8’33% que ocupaba inicialmente.

Pasamos a ver ahora la parte central de la página, en la que nos encontraremos el menú lateral izquierdo, la zona de diseño y la consola de simulación, esta última estando oculta por defecto. En el menú izquierdo, nos encontramos separados por los títulos, todos los componentes que podemos añadir a la zona de diseño, tan solo hace falta pinchar con el ratón en el nombre del componente que queramos añadir, y este aparecerá en la zona de diseño. Si pinchamos en el título de la sección de los componentes, dicha sección se plegara no dejando ver ningún componente de dicha sección y dejando más espacio a la zona de diseño. La zona de diseño, está creada para que ocupe el cien por cien de la página, quedando los menús por encima de esta, de esta manera podremos aprovechar todo el espacio que nos permita nuestro dispositivo.

El menú lateral izquierdo, debe tener el mismo tamaño que la zona donde está alojado el logotipo o nombre de nuestra aplicación, por eso tiene las mismas clases que la región “topRight”. Dentro de este “div”, nos encontramos una lista que contiene el título de sección, la lista de los componentes de esa sección y un separador de sección.

```
▼<div id="botonera" class="col-xs-3 col-sm-2 col-md-2 col-lg-1">
  ▼<ul role="menu" class="list-group">
    ▶<li class="dropdown-header" data-toggle="collapse" href="#listaArduinos">…</li>
    ▶<ul id="listaArduinos" class="panel-collapse collapse in" style="height: auto;">…</ul>
     <li class="divider list-group-item"></li>
    ▶<li class="dropdown-header" data-toggle="collapse" href="#listaSensores">…</li>
    ▶<ul id="listaSensores" class="panel-collapse collapse in" style="height: auto;">…</ul>
     <li class="divider list-group-item"></li>
    ▶<li class="dropdown-header" data-toggle="collapse" href="#listaActuadores">…</li>
    ▶<ul id="listaActuadores" class="panel-collapse collapse in">…</ul>
   </ul>
  </div>
```

Figura 28 – Codigo del menú lateral izquierdo de la interfaz

Para crear el efecto de plegado y desplegado del menú al pinchar sobre el título de la lista, se ha usado el atributo “data-toggle=collapse” y el enlace al identificador de la lista a plegar en el título de cada sección. Como se puede ver en la imagen, cada lista de dispositivos tiene un identificador claro el cual es enlazado desde el título anteriormente descrito.

En el segundo lugar de la parte central nos encontramos la consola de simulación, donde veremos los datos que nos arrojan nuestros dispositivos.

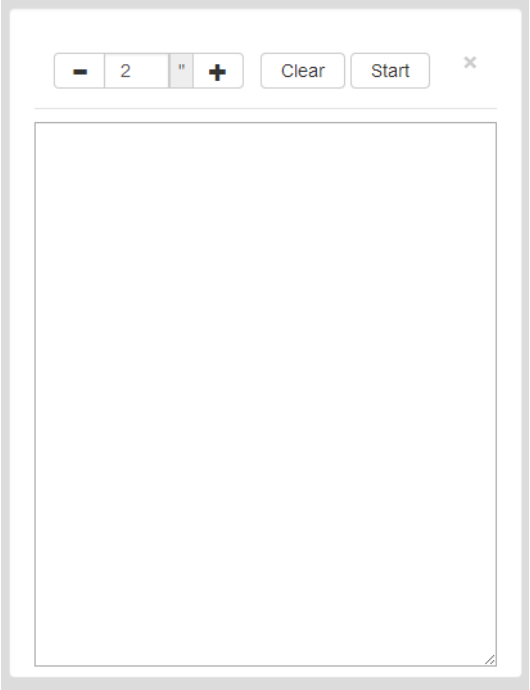


Figura 29 - Consola de simulación

Como se puede ver la ilustración anterior, la consola de simulación está formada por un panel superior y un área de texto. En el panel superior podremos encontrar una celda donde veremos el tiempo con él se realizan las peticiones al simulador; a su lado, vemos dos botones con los símbolos “-” y “+” respectivamente, estos botones sirven para añadir o disminuir tiempo a la simulación. Al lado nos encontramos los botones de “Clear” para limpiar la consola y “Start”, el cual enciende la simulación y se convierte en “Stop” una vez ha empezado. Al cuadro de texto se le ha añadido la propiedad “readonly” para que no se puedan modificar los resultados obtenidos.

Por último, tenemos la región de diseño, siendo un “div” de posición absoluta que ocupa toda la pantalla y está alojado por debajo de todos los menús.

4.4.3. Generación de dispositivos

En este apartado vamos a ver como se crea un dispositivo en la zona de diseño cuando pinchamos en el menú.

- *Generar el esquema un Arduino*

Para crear un arduino en el diseñador, empezamos por pulsar en el panel lateral izquierdo el tipo de arduino que queramos añadir, este tiene un evento el cual lanzara una función al pichar sobre el con el ratón o con el dedo, en caso de estar en un Tablet o móvil.

Cuando creamos un arduino, primero comprobamos si todos sus atributos están inicializados, en caso de que no lo estén los inicializamos nosotros. Esto se hace porque en JavaScript se puede llamar a una función sin pasar ningún parámetro aunque esta los necesite.

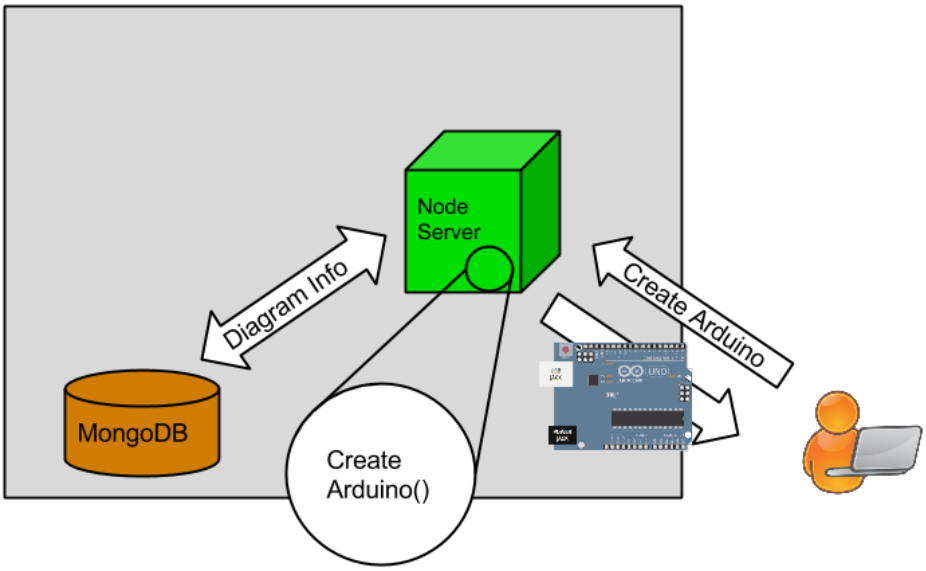


Figura 30 – Representacion de la creacion de una arduino

Una vez todos los parámetros han sido inicializados, creamos los conectores de la placa.

Cada placa Arduino tiene un tamaño y un número de conectores diferente, por lo que cada Arduino tiene un algoritmo de representación de los conectores diferente. Por ejemplo, en la ilustración anterior, podemos ver el código de representación de conectores del Arduino uno.

Esta placa tiene los conectores dispuestos de derecha a izquierda, por ello ponemos el primer conector digital a 10 pixels de la parte superior (y+10) y utilizamos la constante 328 para ubicar el primer conector en la parte izquierda, a partir del cual se irán restando 13 pixeles para poner el siguiente. Dichos

conectores estarán representados por círculos, por lo que usamos la función de Raphäel “ellipse” dándole como valor de alto y ancho 6 pixeles.

Una vez se crea un conector, se le hace opaco y se pinta de gris con el código #808080, y se le añade el nombre “digital” o “analogic” dependiendo del caso, se le pone una referencia a la placa a la que pertenecen, se les pone el tipo de conector que es (“digital” o “analogic”) y se le asigna un identificador interno, que en este caso será el número que ocupa dentro del array.

Seguidamente, hacemos que la placa se pueda mover por toda el área y que todos los conectores la sigan. Para que esto sea posible, capturamos el evento de teclado cuando se pincha sobre la imagen del arduino y se arrastra, en ese momento modificamos las coordenadas de la imagen y de los conectores para que se muevan con el ratón.

Después de dar movimiento a la placa y los conectores, hacemos que se puedan conectar unos conectores con otros. Para ello llamamos a la función “setConnectable” pasándole el conector. Más a delante veremos este proceso con más detalle.

```
//Añadimos la funcion doble click y boton derecho
arduino.mousedown( function(){ contextMenu(); } );
arduino.dblclick( function(){ doubleClick(arduino); } );
```

Figura 31 – Función para representar un arduino.

Una vez podemos conectar diferentes dispositivos a la placa, añadimos las funciones necesarias para que al pinchar con el botón derecho del ratón sobre una placa, esta muestre un menú contextual, el cual nos permitirá eliminar la placa. En el caso de hacer doble clic, se nos mostrara una ventana contextual en la cual podremos configurar la placa, cambiando su identificador, el tipo de conexión, los baudios de frecuencia, etc.

Una vez finalizado todo lo anteriormente descrito, se añada el Arduino al panel y se almacena dicho objeto en un array global.

- *Creación del esquema de un sensor y actuador*

Para crear un sensor o un actuador en el diseñador, pulsamos en el panel lateral izquierdo el tipo de dispositivo que queramos añadir, este tiene un evento el cual lanzara una función al pichar sobre él. Esta función se llama “makeTipoDispositivo” siendo “TipoDispositivo” el tipo del dispositivo que queramos crear. Podemos ver un ejemplo en la siguiente imagen.

```
function makeLightSensor(){
    makeSensor(40, 110, null, null, 'LightSensor', 'img/light_sensor.png', 'Analog');
}
```

Figura 32 – Función llamada al crear un sensor de luz

Dicha función, tan solo llama a una función genérica llamada “makeSensor”, a la cual se le pasa el ancho y alto, la posición “X” e “Y”, el tipo de sensor, la imagen y el tipo de conexión.

Parecido a la función para crear un arduino, primero comprobamos que tenga tamaño y posición donde mostrar el sensor, de no tener, se le asigna uno por defecto. Seguidamente, se crea la imagen, y se le añaden los atributos.

Después, le añadimos las funciones para mostrar el menú contextual y mostrar las opciones del sensor dando doble clic sobre él. Esta función solo será posible en caso de ser un sensor, ya que los actuadores no hace falta configurarlos.

También hacemos que se pueda mover por la pantalla igual que con el arduino. Creamos el conector, ya que los sensores tan solo tienen un conector, en la parta baja del conector. Hacemos que sea conectable y que al mover el sensor, el conector lo siga. Una vez formado el sensor lo añadimos al panel y se almacena dicho objeto en un array global.

4.4.4. Crear una conexión

Los sensores y actuadores deben estar conectados a los puertos de una placa, para ello usamos un código de colores en los conectores. Al pinchar sobre un conector, este cambia de color gris a verde, y se guarda dicho conector en una variable interna. Al pinchar sobre un segundo conector, estos se conectan usando una función propia “connect”. Al conectarse dos conectores cambian de color, de verde a amarillo, y se unen mediante una línea curva que va cambiando de posición conforme movemos los dispositivos.

Una vez terminada la conexión, se guarda el objeto en un array global.

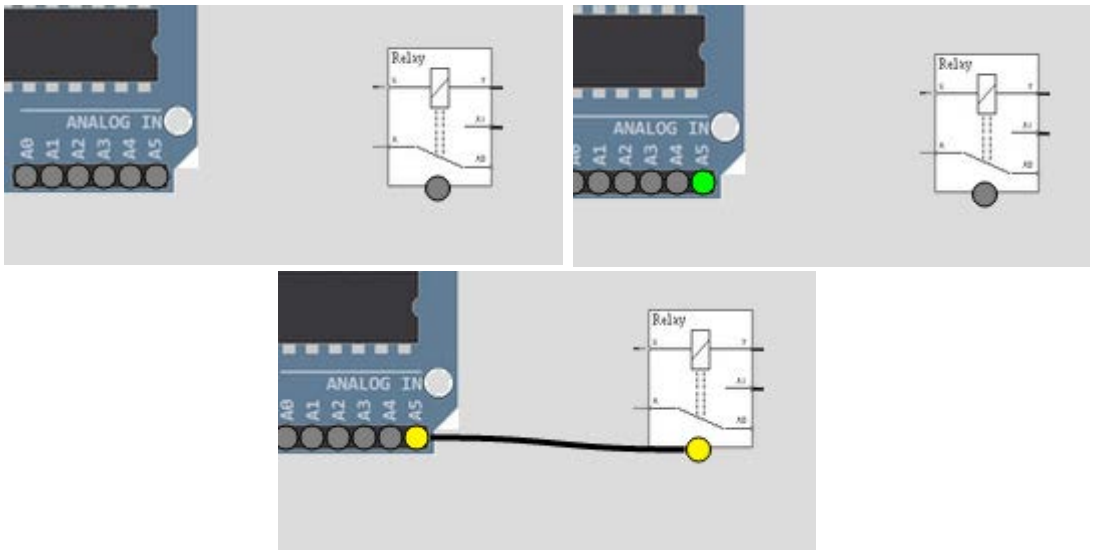


Figura 33 – Pasos para conectar un dispositivo

En el caso que dos dispositivos estén conectados y los conectores estén en amarillo, no podremos conectar nada a dichos conectores. En el caso de querer cambiar la conexión, deberíamos borrarla, haciendo botón derecho sobre la línea de conexión, al hacer esto se nos mostrará un menú contextual dándonos la opción de eliminar la conexión. Al eliminar la conexión, se eliminará la línea que une a los dos conectores y dichos conectores cambiaran nuevamente a color gris. Podemos ver el resultado final en la imagen anterior

4.4.5. Configuración de un dispositivo

Tanto las placas Arduino como los sensores, llevan una configuración por defecto, pero esta se puede cambiar. Para acceder al panel de configuración del dispositivo, tan solo hace falta hacer doble clic sobre dicho dispositivo como se ha explicado en los apartados anteriores.

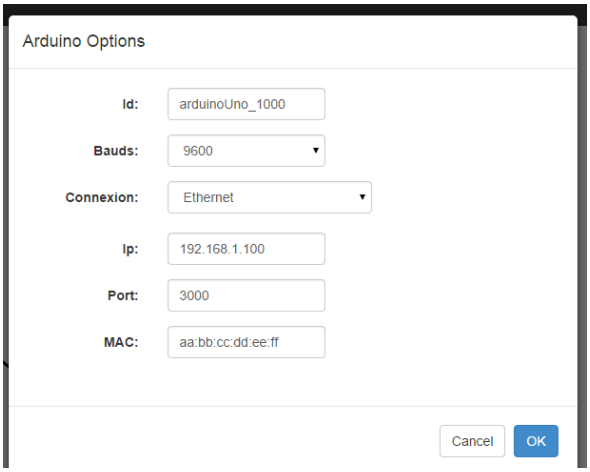


Figura 34 - Ventana de configuracion del arduino

La ventana de configuración de un Arduino, nos mostrara un campo de texto donde poder modificar el identificador por defecto asignado, un desplegable, donde seleccionar los baudios, y por último, un desplegable donde seleccionar el tipo de conexión de la placa con el ordenador. En caso de seleccionar la conexión por red ("Ethernet"), se nos mostraran tres campos de texto más donde deberemos escribir la ip, puerto y dirección MAC del dispositivo al que nos conectaremos.

En caso de hacer doble clic en un sensor, se nos abrirá una ventana modal parecida a la del Arduino, en ella encontraremos un campo de texto donde cambiar el identificador por defecto y un desplegable para seleccionar la función a realizar por el sensor, ya que un sensor puede tener varias funciones, por ejemplo el sensor de luz puede captar los valores de luz o enviar una alerta a la placa cuando se pase cierto umbral de luminosidad. En el caso de seleccionar la función "thresholdCrossed" (umbral cruzado) nos aparecerá un campo de texto para que introduzcamos el número de umbral.

4.4.6. Registro y entrada de un usuario

Tanto la creación de dispositivos, como la conexión y configuración de ellos, se podría hacer sin estar dado de alta en el sistema, pero la finalidad de esta herramienta es convertir nuestros diseño del circuito en condigo arduino, y para hacer eso, hay que registrarse e iniciar sesión en el sistema. En este apartado explicaremos como se ha creado el registro e iniciar sesión de un usuario en el sistema.

- *Aspecto web*

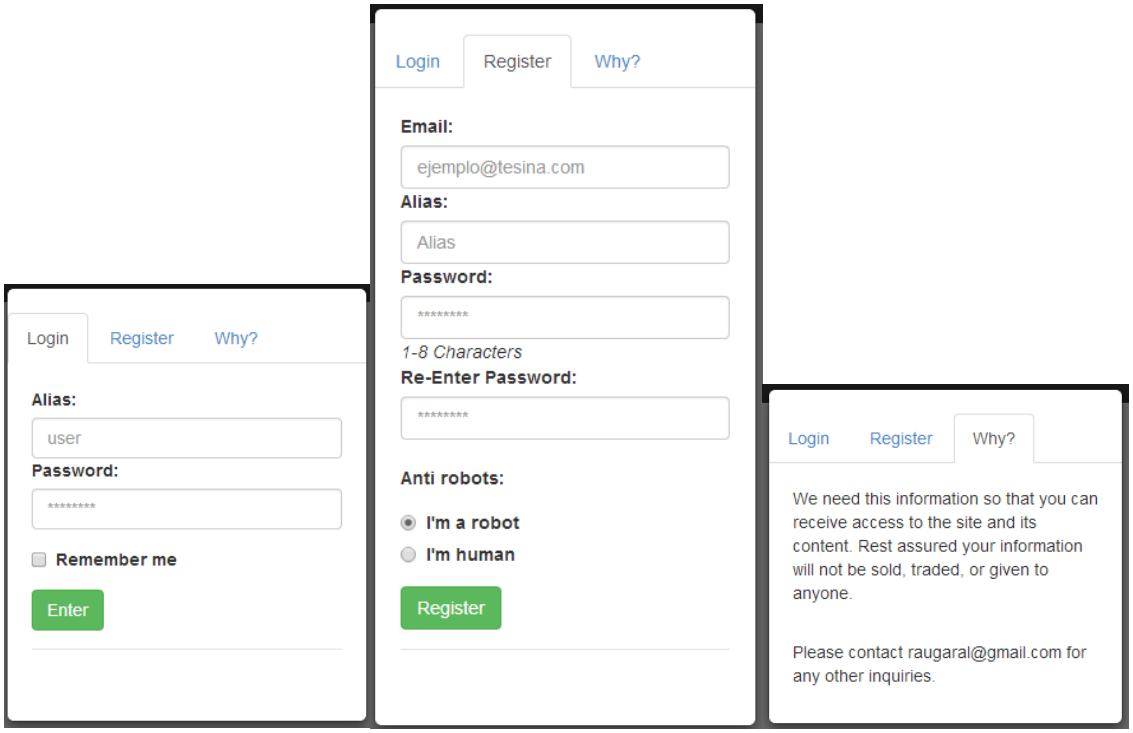


Figura 35 - Pagina de inicio de sesion, registro y ¿por qué?

Como se puede ver en la imagen anterior, se ha creado una ventana modal, partida con pestañas donde el usuario puede iniciar sesión, registrarse o ver porque necesitamos sus datos de forma fácil e intuitiva.

Esta ventana esta oculta por defecto, tan solo se mostrara en caso de pinchar sobre el enlace ubicado a la derecha en la barra superior “Login | Register”, o al intentar acceder a cualquier función de la barra superior, guardar, cargar, simular, etc.

Para crear esta ventana, se ha hecho recurrido a la clase “modal” que nos ofrece Bootstrap. Siguiendo el procedimiento que marca dicha librería, añadimos un div con la clase “modal-dialog” y dentro de él, otro marco con la clase “modal-content” el cual se encargara de contener todos los elementos. Dentro del “modal-content” encontramos las pestañas y el cuerpo.

```
<!-- Entrar -->
<div class="modal fade bs-modal-sm in" id="loginbox" tabindex="-1" role="dialog" aria-hidden="false" style="display: block;">
  <div class="modal-dialog modal-sm">
    <div class="modal-content">
      <br>
      <div>
        <ul id="myTab" class="nav nav-tabs">
          ::before
          <li class="active">
            <a href="#signin" data-toggle="tab">Login</a>
          </li>
          <li class>
            <a href="#signup" data-toggle="tab">Register</a>
          </li>
          <li class>
            <a href="#why" data-toggle="tab">Why?</a>
          </li>
          ::after
        </ul>
      </div>
      <div class="modal-body">…</div>
    </div>
```

Figura 36 - Codigo de la ventana de registro

Dentro del área de las pestañas, nos encontramos con una lista con las clases “nav” y “nav-tabs”, las cuales le dan el aspecto de pestañas. Dentro de la lista, nos encontramos con los tres enlaces los cuales, con ayuda de JavaScript, mostraran la región de inicio de sesión, registro o ¿Por qué? según pinchemos.

La región central esta formateada por la clase “modal-body” la cual contendrá tres regiones, una para cada pestaña.

```
<div class="modal-body">
  <div id="myTabContent" class="tab-content">
    <div class="tab-pane fade" id="why">…</div>
    <!-- Sign In Form -->
    <div class="tab-pane fade active in" id="signin">…</div>
    <!-- Sign Up Form -->
    <div class="tab-pane fade" id="signup">…</div>
  </div>
</div>
```

Figura 37 - codigo de la ventana de registro

Cada región, contiene las clases “tab-pane” (la cual hace que el centro se acople al contenido) y “fade” (al cambiar de pestaña hace un efecto de fundido). Tan solo el div con la clase “active” e “in” será el div mostrado. Dentro de la región con el identificador “signin” y “signup” contendrán formularios, de inicio de sesión y registro.

- *Petición de inicio de sesión*

En el caso que el usuario tuviera creado un circuito, si hiciésemos una petición al servidor normal, el servidor nos respondería con una página web nueva y tendríamos que refrescar la página actual, perdiendo el usuario su circuito. Para prevenir esto, se ha usado la tecnología Ajax, la cual como ya se ha explicado, hace peticiones al servidor de forma asíncrona y modifica la estética de la web sin necesidad de refrescar la página entera.

Para que esto suceda, el botón del formulario de inicio de sesión lanza una función “login” al ser pulsado.

La función “login” inicia rescatando los valores del formulario, nombre de usuario, contraseña y si desea que el navegador recuerde la sesión. Una vez se tienen los valores, se comprueba que no estén vacíos y se crea la información que va a ser enviada al servidor en forma de JSON.

Haciendo uso de la librería JQuery, hacemos una petición por POST a la dirección del servidor seguido de la ruta del servicio “/tesina/login”, y le pasamos los datos recibidos.

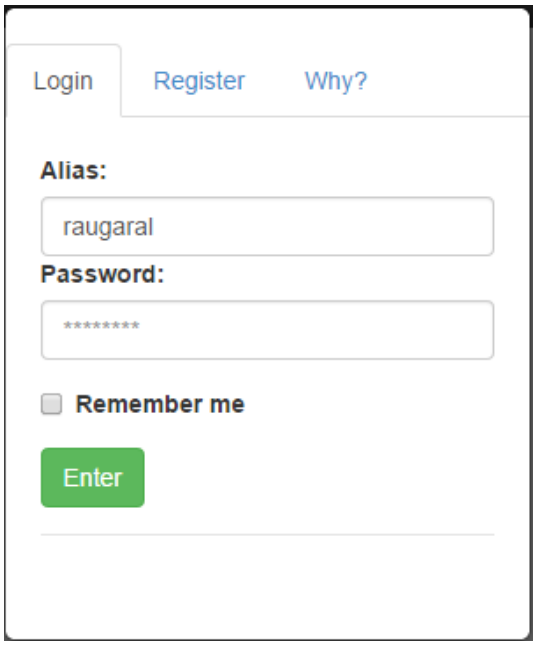


Figura 38 - Pantalla de inicio de sesión

En caso que el servidor no encuentre el usuario, entraremos en la función “.fail” la cual mostrara el mensaje “User/Password no valid”. Si el usuario ya está registrado, guardaremos su nombre y su correo en el navegador. Dependiendo si marco la casilla para que el navegador recuerde su sesión, se guardara en el “localStorage”, en cambio, si no se marcó la casilla, se guardará en la “sessionStorage”.

Una vez hecho esto, se cambia el enlace de “Login | Register” por el mensaje “Hi nombre del usuario | Logout”

En caso de pinchar en “logout”, se cerrara la sesión y se eliminaría toda la información del navegador.

- *Petición de registro*

Estando en la pestaña “Register”, se encuentra el formulario de registro. Al pulsar el botón “Register” se lanza la función que podemos ver en la siguiente imagen.

Primero eliminamos los mensajes que hubiesen de otras interacciones con la ventana de registros, seguidamente, almacenamos en variables los valores de los campos para poder comprobar si son correctos. Se observa si todos los campos están rellenos y si se ha marcado la opción anti robots. En el caso que todos los valores sean correctos, se hace una petición asíncrona POST al servidor mediante Ajax, utilizando la librería JQuery, pasándole en JSON los datos subministrados por el usuario. En caso de guardar los datos de forma exitosa, se mostrara el mensaje en verde diciendo “Saved Correctly” en caso contrario, se mostrara el mensaje de error en rojo.

4.4.7. Guardado de circuitos

En este apartado vamos a explicar cómo se guardan los circuitos diseñados en base de datos.

Si vemos la interfaz, observamos que hay dos botones para guardar el diseño de nuestro circuito, “Save” y “Save As”. La diferencia entre ambos, es que “Save As” abre una ventana para pedir el nombre del fichero, mientras que “Save” busca en la sesión del usuario si existe un nombre para el circuito actual, en cuyo caso guardara el circuito automáticamente. En caso de no encontrar el nombre del fichero, se comportará igual que “Save As” mostrando la ventana de inserción de un nombre para el circuito.

Una vez se tiene un nombre valido para el circuito, se llama a la función “save” pasándole el nombre del fichero y el nombre del usuario.

Dicha función, se encarga de acceder al API de comunicación, y hacer la petición de guardado. Para ello necesita crear en una estructura JSON, igual a la del servidor, donde tengamos almacenado el nombre de usuario, el nombre del circuito y todos los dispositivos que componen el circuito como vimos en el apartado 4.2.1.

Para crear la estructura que contendrá todos los elementos representados en el esquema del circuito, se utiliza la función “makeJSONcircuit”. Dicha función se encarga de recorrer la lista donde guardamos cada componente que se creó con anterioridad, inspecciona de que tipo es el elemento seleccionado (Arduino, sensor o actuador) y copia sus atributos convirtiendo en pares {clave : valor}.

Una vez generado el JSON con el formato visto en la ilustración 10, se manda al servidor mediante una petición Post, usando la ruta dirección_del_servidor/tesina/save. Para hacer dicha petición, se usa la librería “JQuery” y usamos su función “post”, pasándole como parámetro la ruta del servidor, los datos y el formato en el que están codificados los datos. Adicionalmente, se sobrescriben sus métodos “done” y “fail”, para que en caso de éxito muestre una alerta señalando que se ha guardado correctamente, o en caso contrario, un mensaje avisando del error de guardado.

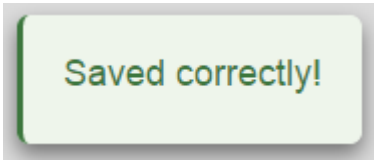


Figura 39 - Confirmacion de guardado

4.4.8. Recuperación de circuitos

En este apartado explicaremos como un usuario registrado y que haya entrado en el sistema, puede recuperar sus circuitos guardados previamente.

Si pulsamos el botón “Load” de la interfaz web, este lanzara una función llamada “load”, la cual después de comprobar si el usuario ha entrado en el sistema, hará una petición GET mediante Ajax, a la ruta “/tesina/files/nombre_del_usuario”. Esta consulta al API, nos devolverá un JSON con todos los circuitos del usuario con el formato visto anteriormente. Una vez recuperados los circuitos, se llama a una función “boxLoad” la cual mostrara al usuario una tabla con el nombre y la fecha de todos los circuitos guardados.

Al pinchar sobre un registro de la tabla, esta llama a otra función “import_shapes” pasándole el nombre del circuito seleccionado por el usuario. Esta función busca

entre todos los circuitos recuperados anteriormente, el circuito seleccionado. Una vez encontrado, se llama a la función “makeShapesFronJSONcircuit” pasándole como parámetro el JSON del circuito seleccionado.

La función “makeShapesFromJSONcircuit” recorre los elementos del JSON llamando a las funciones pertinentes para crear los diferentes componentes (arduino, sensor o actuador), por ejemplo, cuando recorra los sensores, llamara a la función “makeSensor” pasándole sus atributos. Una vez creados todos los componentes, se crean las conexiones. Ya que el JSON solo guarda cada extremo de la conexión (el identificador del dispositivo y el identificador del pin dentro del dispositivo), se debe buscar previamente el dispositivo y el pin de conexión de ambos extremos para poder completar la conexión. Una vez tenemos los dos pines de la conexión, se llama a la función “connect”, como se hace al pinchar con el ratón en los dos pines.

Una vez terminada la conexión de los componentes, podremos ver el esquema del circuito igual a la última vez que lo guardamos.

4.4.9. Validación del circuito

Antes de generar código o simular el funcionamiento del circuito, necesitamos comprobar que el circuito es correcto. Para ello tenemos la función “valide” que es llamada cada vez que se necesita saber si el modelo es válido.

Al llamar a la función “valide”, primero, cambiamos el símbolo del cursor a “esperar” en toda la página. Seguidamente hacemos una serie de comprobaciones:

- Que exista algún componente.
- Que haya componentes sin conectar a la placa.
- Que todos los componentes estén bien conectados (dispositivos analógico en pines analógico y digitales en pines digitales)
- Que estén configurados

En caso de error, cada comprobación añade un mensaje a una lista, la cual es devuelta para que otras funciones puedan mostrar los errores al usuario.

```
function valide(){
    $("body")[0].setAttribute("style","cursor: wait;");

    message = Array();
    if(shapes.length == 0) //Miramos que haya componentes
        message.push("Components not found!");
    if(countDispo()-1 != connections.length) //Que no haya dispositivos sin conectar
        message.push("Device not connected!");
    wc = wellConnected();
    if (wc.length > 0) // Que esten conectados en sus respectivos conectores
        for (var i = 0; i < wc.length; i++)
            message.push("Device "+wc[i].id
                +" bad connected, should be connected to a "+wc[i].typeConn+" pin.");
    ac = areConfigured();
    if (ac.length > 0)  // Que esten configurados
        for (var i = 0; i < ac.length; i++)
            message.push("Device "+ac[i].id+" not configured!");
    if(allDiferentId().length > 0)
        message.push("Two devices have same Id!");

    $("body")[0].removeAttribute("style");

    return message;
}
```

Figura 40 - Funcion para validar el circuito

4.4.10. Generación de código

En este apartado vamos a adentrarnos en la generación de código para arduino.

En la interfaz web, nos encontramos un botón llamado “Compile” que al pulsarlo lanza la función “validate”, esta comprueba si el usuario ha entrado en el sistema y si el circuito ha sido guardado previamente, si ese no ha sido el caso, se muestra la ventana para guardar. Una vez hechas dichas comprobaciones, hacemos un guardado silencioso (sin que muestre ninguna información por pantalla), para evitar que el modelo actual del usuario sea diferente al guardado en base de datos, y validamos el circuito con la función explicada en el punto anterior. Si no ha habido ningún error, se procede a hacer la llamada POST mediante Ajax, llamando al recurso ubicado en “/tesina/makeFile” y pasándole un JSON con el nombre de usuario y nombre del circuito el cual queremos generar el código. Dicho JSON debe estar formado de la siguiente manera:

“{ nameUser : user, fileName : file }” siendo “user” y “file” el nombre de usuario y nombre de circuito a generar.

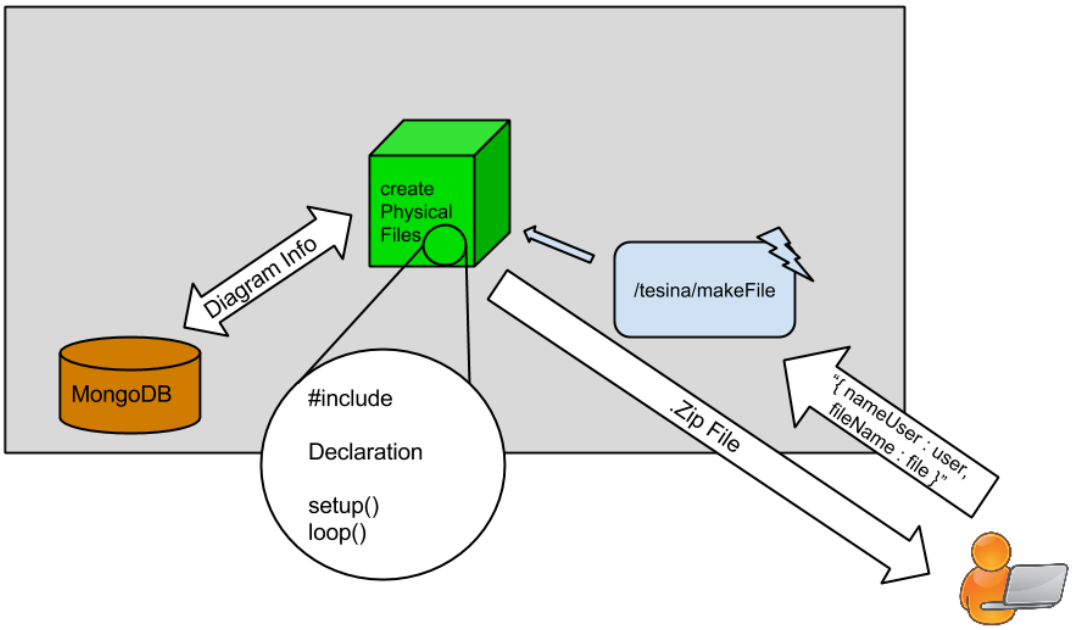


Figura 41 - Diagrama de la generacion del codigo

Esta petición llega al servidor y es respondida por la función “createPhysicalFiles” que es la encargada que generar tanto el fichero “.json”, como el archivo comprimido en “.zip” que almacena los dos ficheros “.ino” de código Arduino. Para crear el archivo comprimido “.zip” utilizamos el plugin para node.js llamado “jszip”.

Los ficheros “.ino” son creados utilizando las funciones “makeMainInoFile” y “makeHandlersFile” las cuales explicaremos a continuación:

- *Creación del fichero principal*

La función “makeMainInoFile” generará el código del fichero principal. Como se ha explicado anteriormente, un programa para arduino está formado por: introducción de librerías (includes), declaración de constantes y variables, métodos “setup” y “loop”.

Para generar dicho código, hemos optado por dividir el código en los siguientes registros:

- Comments: Comentarios posicionados arriba del código.
- Includes: Introducción de librerías.
- Contants: Definición de constantes.
- HandlersVar: Declaración de variables para los manejadores.
- Declare: Declaración de los dispositivos.
- Setup: Método “Setup”
- Loop: Método “Loop”
- Handlers: Todo lo relacionado con los manejadores.

- Close: Clave de cierre.

```
function makeMainInoFile (data) {
    var circuit = data.circuit;

    var comments = "/*\nYou need the TatAmI library\nCan download here:\n";
        comments +="http://www.pros.upv.es/m4spl/tatami\n*/";
    var includes = "\n\n#include <Tatami.h>\n";
        includes +="#include <Wire.h>\n";
        includes +="#include <SPI.h>\n";
        includes +="#include <Ethernet.h>\n";
    var constant = "\n";
    var handlersVar = "\n";
    var declare = "\n";
    var setup = "\nvoid setup(){\n"
    var loop = "\nvoid loop(){\n"
    var handlers = "\n";
    var close = "}\n";
```

Figura 42 - funcion generadora del fichero principal

Como se puede ver en la imagen anterior, cada registro es una variable, a la cual se le añadirá cierto código según el número y tipo de los componentes. Para ello nos encontraremos con tres grandes bucles que recorrerán el circuito en busca de la configuración necesaria para generar el fichero.

Empezamos viendo el bucle de los arduinos:

```
for (var i = 0; i < circuit.arduinos.length; i++) {
   arduinoActual = circuit.arduinos[i].attr;
   setup += "\tSerial.begin("+arduinoActual.bauds+");\n";
   if (arduinoActual.conn.indexOf("Ethernet") >= 0){
      declare += "byte mac"+i+"[] = {0x"+arduinoActual.mac.toUpperCase().replace(/:/g, ", 0x")+"};\n";
      declare += "IPAddress server"+i+"("+arduinoActual.ip.replace(/\./g, ",")+");\n";
      declare += "IPAddress myIP"+i+"("+arduinoActual.ip.replace(/\./g, ",")+");\n";
      declare += "int port"+i+" = "+arduinoActual.port+";\n\n";
      declare += "EthernetClient "+arduinoActual.id+";\n";
      declare += "EthernetCommunication comm;\n";
      setup += "\tcomm.begin(mac"+i+", server"+i+", myIP"+i+", port"+i+");\n";
   }
   else{
      declare += "SerialCommunication comm;\n";
      setup += "\tcomm.begin();\n\n";
   }
};
```

Figura 43 - funcion generadora del fichero principal. Bloque arduino

Como podemos ver, se recorre todos los arduinos que se encuentren en el circuito, y se añade al método “setup” la configuración de los baudios. Seguidamente se comprueba si la placa está conectada por puerto serio o Ethernet; en caso de ser Ethernet, se añade a la variable de declaración la mac del dispositivo, reemplazando los dos puntos (:) que delimitan un bloque de la dirección mac por “0x”, señalando así que es un número hexadecimal. Continuamos añadiendo las ip, cambiado los puntos (.) por comas (,), seguidamente, se añade el puerto y el identificador de la placa, declaramos el objeto para la comunicación por Ethernet y llamamos al constructor del objeto pasándole la configuración anteriormente

creada. En caso de la comunicación ser por puerto serie, se declara el objeto y se llama al constructor.

En segundo lugar, vemos el iterador de los sensores. Dicho bucle recorre todos los sensores, definiendo las constantes, declarando los objetos, añadiendo la configuración necesaria en el método “setup” y mostrando los valores obtenidos por los sensores en el método “loop”.

Antes de añadir el código de los actuadores, añadimos un código estático que prepara a la placa para atender a los manejadores de los controladores.

Por último, al igual que con los sensores, los recorremos y vamos añadiendo las constantes, declarando los objetos, y añadiendo la configuración al método “setup” y en el método “loop” el código necesario para mostrar su funcionamiento. Además, se añade cierto código necesario para los manejadores.

Una vez se ha llenado las variables con todo el código necesario, se devuelven formando una cadena de caracteres. Para crear dicha cadena, las variables siguen el siguiente orden, “comments”, “includes”, “constant”, “handlersVar”, “declare”, “setup”, “close”, “loop”, “handlers”, “close”. De esta manera se crea una cadena que da lugar a un fichero totalmente estructurado.

Una vez finalizado el fichero principal, se llama a la función “makeHandlersFile” para crear un fichero secundario que contendrá toda la configuración de los manejadores (handlers).

- *Creación del fichero de manejadores*

El fichero de manejadores, se encarga de gestionar la entrada/salida de los actuadores. Este fichero es más sencillo que el principal, ya que tan solo crea un método que escribe en el buffer los datos del actuador.

Para ello, se recorren todos los actuadores almacenados en la lista y para cada uno se extrae el identificador, que será el único parámetro que nos hará falta para crear el manejador.

Una vez creados todos los métodos, se devuelve a la función principal, el texto creado en una variable para ser volcado a un fichero de texto.

- *Resultado final*

Al finalizar todo el proceso obtendremos los ficheros comprimidos en un paquete “.zip”, como se ha explicado anteriormente, y el cual podremos ver en la imagen siguiente.

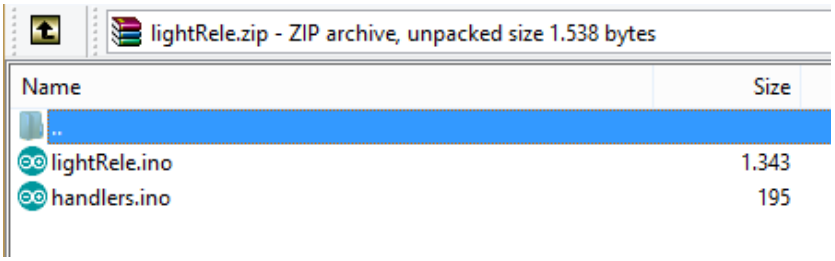


Figura 44 - Paquete .zip con el codigo generado

4.4.11. Implementación del simulador

En este apartado vamos a ver como se ha desarrollado el simulador.

En la interfaz web nos encontramos un botón “Simulate”, que al ser pulsado, lanza una función llamada “showSimulator” que comprueba que el usuario haya iniciado sesión en el sistema, que el circuito sea válido y que este guardado. En caso de no estar iniciado en el sistema, mostrara la ventana de inicio de sesión. Para comprobar la corrección del circuito, se usa la función “validate”, la cual es la misma que lanza el botón “Compile”. Por último, si el circuito no se ha guardado nunca, se muestra la ventana para que de un nombre al circuito y guardarlo. Una vez hechas todas las comprobaciones y superados los requisitos, se muestra a la derecha, la consola de simulación.

- *Consola de simulación*

La consola de simulación nos mostrará valores aleatorios, dentro de un rango, de los componentes que hayamos añadido a nuestro circuito. La consola consta de cuatro componentes, temporizador, botón “Clear”, botón “Start” y área de texto.

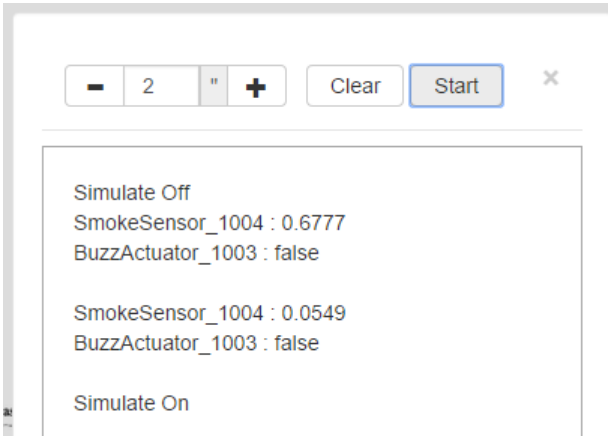


Figura 45 - Interfaz de la consola de simulacion

El temporizador nos servirá para modificar el tiempo con el que se muestra los resultados en la consola. Podremos aumentar o reducir el tiempo, hasta un mínimo de 1 segundo, pulsando los botones con el símbolo menos “-” o más “+”.

Pulsando el botón “Clear” limpiaremos toda el área de texto de simulaciones anteriores.

El botón “Start” iniciará o parará la simulación, la cual, en el caso de estar en marcha, irá añadiendo los valores recibidos en la parte superior del área de texto.

- *Proceso de simulación del cliente*

Vamos a profundizar en cómo se generan de los valores mostrados por el simulador.

Al pulsar el botón “Start” se lanza una función “simulate” comprobará si el simulador está apagado, de ser así, comprueba que sea válido y hace un guardado silencioso. A continuación, se llama a la función “startStopSimulate”, la cual hace una petición PUT al servidor pasándole el nombre del usuario, el nombre del sistema a simular y el estado al que queremos poner el simulador, en caso de querer ponerlo en marcha pasaremos “true”, en caso contrario “false”. También cambiamos el texto del botón “Start” a “Stop”.

Una vez se ha llamado a la función “startStopSimulate” para encender la simulación, se coge el tiempo mostrado en el temporizador para indicarle al iterador, que llame a la función “getSimulateData” cada vez que pase ese lapso de tiempo. Para crear el iterador, se ha utilizado la función nativa de JavaScript “setTinterval” a la cual se pasa la función que se va a ejecutar de forma cíclica y el intervalo de tiempo el cual pasado se ejecutara dicha función.

La función “getSimulateData” tan solo consulta el recurso del api haciendo uso de la ruta “/tesina/simulator/nameUser/fileName”, donde “nameUser” es el nombre del usuario y “fileName” es el nombre del circuito guardado. Cuando el api devuelve los valores, se pasan a la función “showResults” que formatea el JSON recibido y lo muestra en el área de texto de la consola.

- *Proceso de simulación del servidor*

En el servidor tenemos tres recursos que pueden ser accedidos:

Para arrancar la simulación de un sistema, en node dejamos a disposición de los usuarios la ruta “/tesina/simulator/start” el cual será respondida por la función “startSimulation”. Dicha función recoge el nombre de usuario, el nombre del circuito y el estado al cual se va a cambiar la simulación, “true” para arrancar la simulación y cualquier otra para pararlo. Una vez obtenidos los datos de la petición, se busca en la base de datos el circuito indicado y una vez encontrado se comprueba el estado al cual se quiere pasar, en caso de ser “true”, se añade el circuito a un array, en caso de no ser “true” se elimina el circuito del array. En los dos casos se manda un mensaje al usuario señalando que la petición ha sido atendida.

Para obtener los valores de la simulación, podemos hacer una petición get a la ruta “/tesina/simulator/nameUser/fileName” cambiando “nameUser” por el nombre de usuario y “fileName” por el nombre del circuito a buscar. Esta petición será respondida por la función “getSimulationValues”, la cual tras obtener el nombre de usuario y del circuito los busca en el array de simulaciones en marcha. Una vez obtenida la simulación del circuito, se recorren los sensores obteniendo el tipo de sensor, el tipo de función seleccionada y el umbral en caso de tener. Esto se pasa a la función “getSensorValue” la cual, dependiendo del tipo de sensor y la función seleccionada, generará un valor aleatorio dentro de un rango coherente para el dispositivo. Una vez obtenido el valor, se obtiene el tipo del valor, llamando a la función “getTipeSensorValue”, por ejemplo en caso de ser un sensor de luz, devolverá la cadena de texto “luxes”. Obtenido el valor se crea un JSON del tipo “{idSensor : idSens, value : value, typeValue : typeValue}” y se añade a una lista. Cuando se termina de obtener los valores de los sensores, se examinan los actuadores, obteniendo el valor actual de los sensores (ya que para cambiar el valor tenemos otra función que después explicaremos), y se añaden a la misma lista en forma de “{idSensor : idActuat, value : value}”. Una vez obtenidos todos los valores, se devuelve la lista de JSON al usuario.

Para cambiar el estado de un actuador se puede hacer una petición PUT a la dirección “/tesina/simulator/actuator” pasando el usuario, nombre del circuito, identificador del actuador y estado nuevo. Esta petición la responderá la función

“changeStateActuator”, el cual buscara en la lista de simulaciones activas la simulación de dicho circuito, buscará el simulador en cuestión y cambiara su estado al nuevo estado pasado.

4.5. Pruebas

Por último, se va a explicar la forma en que ha sido testeada cada una de las partes de la aplicación. En este apartado se va a seguir una estructura similar a los anteriores, explicando en primer el testeo de la aplicación web y a continuación las pruebas para el Api de comunicación, y para finalizar probaremos el simulador.

4.5.1. Pruebas de la aplicación web

En primera instancia, empezamos probando la aplicación web, ya que es la encargada de crear el JSON que representa el modelo del circuito. Ya que este va a ser transferido por un servicio REST y guardado en una base de datos orientada a documentos como es MongoDB, necesitamos validar que el código generado por la aplicación es correcto.

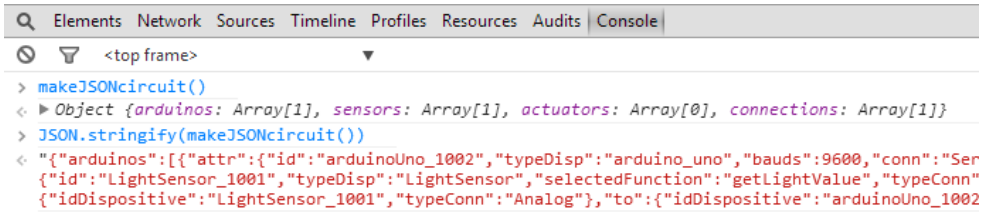


Figura 46 - Obtencion de codigo JSON de un circuito

Para obtener el JSON de un circuito previamente creado, primero abrimos la consola JavaScript de nuestro navegador. Usaremos la función nativa de JavaScript “JSON.stringify()” y le pasemos la función propia “makeJSONcircuit()”, la cual nos devuelve los objetos representados, para obtener el JSON en una cadena de caracteres, como podemos ver en la imagen anterior.

Para validar que el código creado por la aplicación es correcto, usaremos la herramienta web “JSONLint” (http://jsonlint.com/) en la cual, podemos pegar el código obtenido anteriormente en nuestra aplicación y pulsando en el botón “Validate”, JSONLint nos dará el resultado de la validación.

Después de hacer diferente comprobaciones, siempre con resultado positivo, podemos decir que el código JSON generado por nuestra aplicación es correcto.

4.5.2. Pruebas del API

El API REST es el encargado de proveer a la aplicación web los datos de la base de datos. Por ello es tan importante que funciones correctamente.

Para llevar a cabo las pruebas del API, se utilizaron dos herramientas:

- Postman es una extensión para google Chrome, que nos ayuda a hacer nuestras peticiones a servicios web. También nos permite guardar las peticiones en carpetas para tenerlas ordenadas.

Hemos usado la herramienta “Postman” para verificar que las peticiones eran respondidas y ver la respuesta del servicio.

Nos ha sido de gran ayuda ya que su agradable interfaz y su facilidad de uso, han hecho el desarrollo del API mucho más sencillo.

- Apache AB es una herramienta de rendimiento, con la que se puede calcular el tiempo que tarda en responder el servidor a las peticiones.

Hemos usado “Apache AB” para comprobar cuantas peticiones sería capaz de aguantar nuestro servidor actual sin tener demasiada demora o llegar a la denegación de servicio. Para ello, hemos comprobado cómo se comportaría nuestro servidor haciendo diferente número de peticiones simultáneas.

  - Para 100 peticiones con una concurrencia de 10 peticiones simultaneas, en total nuestro servidor dio un tiempo medio de respuesta de 89 ms. Podemos ver la tabla devuelta por AB en el anexo 1.
  - Para 100 peticiones con una concurrencia de 100 peticiones simultaneas, en total nuestro servidor dio un tiempo medio de respuesta de 475 ms. Podemos ver la tabla devuelta por AB en el anexo 2.
  - Para 500 peticiones con una concurrencia de 100 peticiones simultaneas, en total nuestro servidor dio un tiempo medio de respuesta de 1014 ms. Podemos ver la tabla devuelta por AB en el anexo 2.
  - Para 500 peticiones con una concurrencia de 500 peticiones simultaneas, en total nuestro servidor dio un tiempo medio de respuesta de 2662 ms. Podemos ver la tabla devuelta por AB en el anexo 2.

Con este test pudimos comprobar que nuestro servidor podría aguantar unas 100 peticiones simultáneas sin tener un retraso demasiado importante en las respuestas.

4.5.3. Pruebas del simulador

Para comprobar el correcto funcionamiento de la aplicación, se hizo uso de dos aplicaciones, Postman y el debugger de Node.js.

Como se ha explicado anteriormente, Postman es una herramienta para crear peticiones al servidor. Mientras que el debugger de Node.js nos ayudó a ver el flujo del programa.

El debugger en node.js es muy sencillo, para poner un punto de parada en nuestro código, tan solo hay que escribir la palabra “debugger” dentro de la función donde queramos que el debugger se pare. Una vez puestos todos los puntos de interrupción, debemos arrancar el servidor en modo debug, para ello tan solo debemos escribir en la consola “node debug app”, y automáticamente se lanzara el servidor en modo debug. Una vez iniciado, deberemos utilizar los comandos “cont” o “c” para continuar la ejecución, “next” o “n” para ir al siguiente paso, “step” o “s” para entrar en la función, “out” u “o” para salir de la función actual y “pause” para pausar la ejecución del servidor.

Con este simple método pudimos ver el flujo de programa y corregir diferentes errores.

4.6. Conclusiones

En este capítulo se ha explicado cómo se ha llevado a cabo el diseño, la implementación y las pruebas de nuestro trabajo de fin de master. Hemos podido ver como se ha diseñado la herramienta siguiendo unos requisitos básicos, se ha utilizado la tecnología expuesta en el capítulo anterior para llevar el desarrollo de nuestra aplicación y se han probado todos los componentes una vez implementados.

5. Manual de usuario

En el capítulo que vamos a describir cómo se puede utilizar el sistema desarrollado, viendo cómo funciona y como debemos utilizarlo.

Registro y entrada en el sistema

Para poder disfrutar de todas las funciones que brinda la plataforma, es esencial registrarse y entrar en el sistema. Para ello debemos pinchar el en enlace “Login | Register” como vemos en la siguiente ilustración. Seguidamente nos aparecerá una ventana modal con tres pestañas:

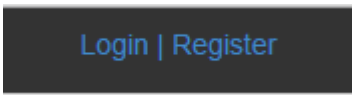


Figura 47 - Enlace de registro y entrada al sistema

- “Login” la cual nos permitirá entrar en el sistema con nuestro usuario y contraseña.
- “Register” en la que tendremos que introducir nuestro correo electrónico, un nombre de usuario y la contraseña para poder registrarnos.
- Por ultimo tenemos la pestaña “Why?”, en la cual se muestra información por la cual pedimos el registro del usuario.

Creación de un circuito

Para llegar a generar el código, primero tenemos que crear el circuito usando los dispositivos que nos encontramos en la barra lateral izquierda. Dichos dispositivos están agrupados en tres listas: Arduinos, Sensores y actuadores.

Al pinchar sobre el título, este recogerá los dispositivos englobados en dicha categoría.

Cada enlace de los dispositivos, mostrará en la pantalla una representación del dispositivo seleccionado

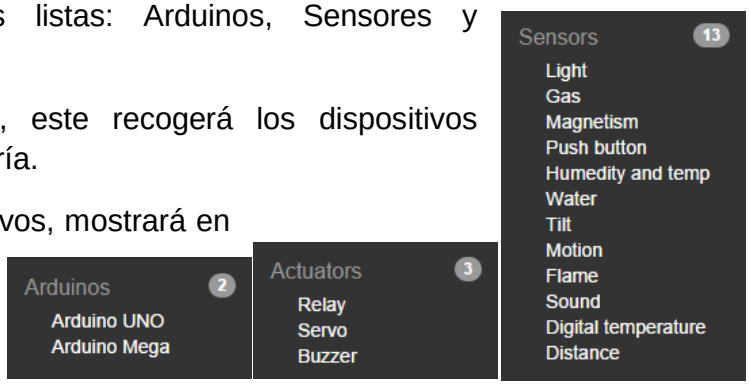


Figura 48 - Lista de dispositivos

Una vez elegidos los componentes a utilizar podemos conectarlos a la placa pinchando en su conector, el cual se pondrá verde para indicar que será conectado al próximo pin en el que pinchemos. Una vez conectado a la placa, este se pondrá amarillo y aparecerá una línea que irá de un conector a otro.

En el caso que nos equivoquemos al hacer una conexión, podremos eliminar dicha conexión pinchando con el botón derecho sobre la línea y haciendo clic en la opción “Delete” del menú contextual que aparecerá, como podemos ver en la imagen siguiente.

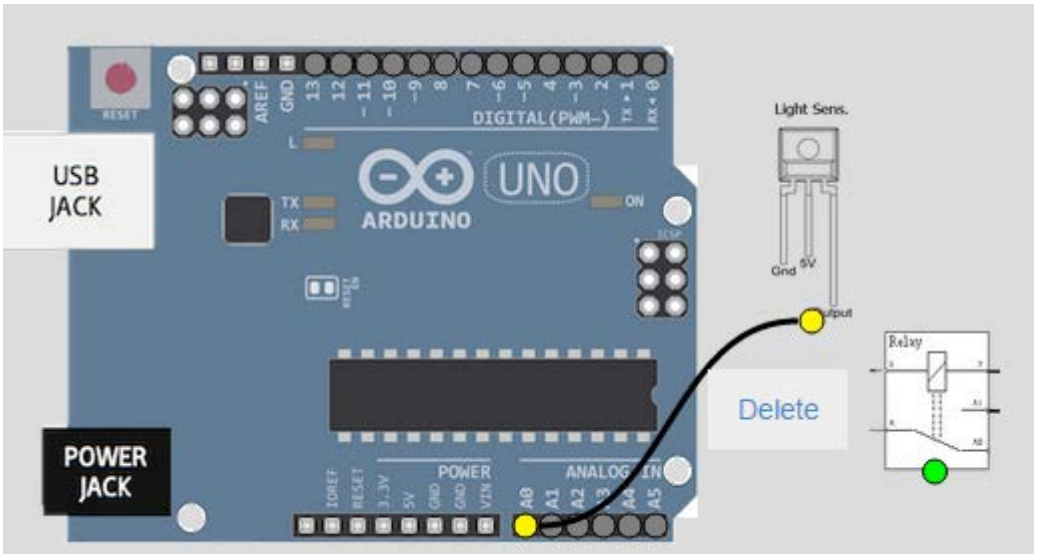


Figura 49 - Eliminar una conexión

Simulación

Podemos comprobar los valores que nos generaran nuestros sensores mediante el simulador. Para ello, solo tenemos que pinchar en el botón de la barra superior “Simulate”, el cual nos abrirá la ventana de simulación en la parte derecha de la pantalla.

La consola de simulación consta de cuatro botones.

- Los botones con los símbolos “+” y “-” permiten subir o bajar el tiempo en el que se muestran los valores de la simulación.
- El botón “Clear” el cual nos permite limpiar la consola, eliminando todos los valores anteriores.
- El botón “Start/Stop” el cual dará comienzo a la simulación o parará la simulación en caso de estar en marcha.

Comprobación y generación de código

Una vez generado nuestro modelo, podemos pinchar en el botón “Compile”, el cual nos dará la opción de guardar el modelo en caso de no haberlo hecho antes. Si el modelo del circuito está correctamente conectado, nos aparecerá una ventana indicándolo y nos dará la opción de descargar el código, como podemos ver en la imagen siguiente. En caso que hubiese algún componente mal conectado, nos aparecería un error y no nos indicaría la forma de corregirlo.

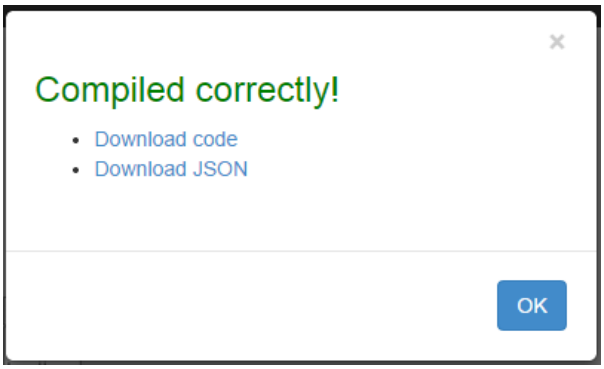


Figura 50 - Ventana para descargar el codigo

6. Caso de uso

En este capítulo se presenta un caso de estudio para presentar el funcionamiento de nuestra herramienta en un caso real.

6.1. Presentación del caso de uso

En el caso de estudio vamos a presentar como se podrían crear unas herramientas de seguridad para nuestra casa. Dichas herramientas consisten en una alarma de humos y un detector del movimiento.

6.1.1. Alarma de humos

Empezaremos explicando cómo crear una alarma de humos, la cual podríamos tener en la cocina para que nos avise en caso que se queme la comida o también podríamos tenerla en el centro de la vivienda para que nos avise en caso de incendio.

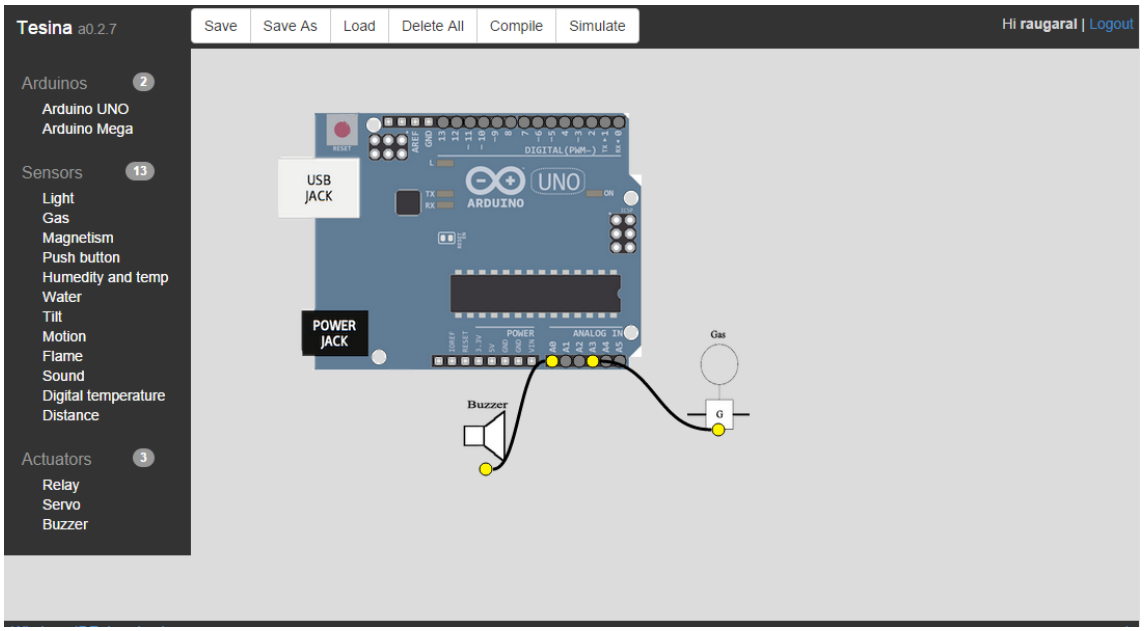


Figura 51 - Diseño de una alarma de humo

Para crear la alarma de humos, necesitaremos una placa arduino, un sensor de gas y un altavoz. El objetivo es tener en una caja, una placa arduino con un sensor de humo y un altavoz para que nos avise en caso que detecte humo. En este caso supondremos que tenemos la placa conectada por Ethernet.

Para obtener el código necesario, en primer lugar navegamos hasta la web y nos damos de alta en el sistema, accediendo a la pantalla que se nos muestra al hacer clic sobre el enlace de la parte superior derecha de la aplicación.

Una vez que hemos entrado en el sistema, utilizamos la barra de componentes izquierda para modelar nuestra alarma. Para ello, escogemos una placa Arduino (en nuestro caso hemos escogido la placa Arduino UNO), un sensor de gas (Gas sensor) y un altavoz (Buzzer). Conectamos los componentes a la placa, como vemos en la figura anterior y guardamos el modelo pulsando en el botón “Save As”, le damos un nombre identificativo, en nuestro caso “alarmaHumo”.

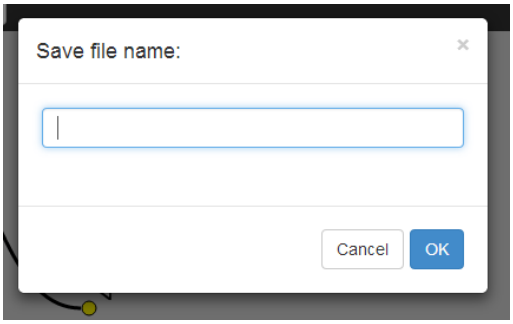


Figura 52 - Ventana de guardado

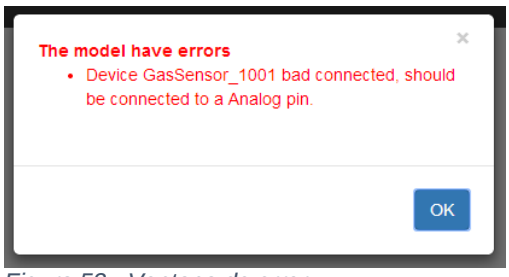


Figura 53 - Ventana de error

Una vez guardado, pulsamos el botón “Compile” para verificar que los componentes están correctamente conectados en sus correspondientes pines, de no ser así, la aplicación nos avisará mediante una ventana emergente como podemos ver en la ilustración 51, indicándonos el error, en este caso, hemos conectado por error el sensor de gas en un

pin digital, para corregir la conexión, solo tenemos que hacer clic con el botón derecho del ratón en el hilo conector y pinchar en “Delete” para eliminar la conexión actual y conectarla a un pin analógico.

Una vez que tenemos todos los componentes correctamente conectados, pasamos a configurar los componentes. La aplicación da una configuración por defecto, pero es recomendable repasarla ya que puede no ser la deseada. Para ello hacemos doble clic encima del componente a configurar. Empezando por la placa, vemos que estaba configurada con la conexión serie pero nosotros tendremos el Arduino conectado por Ethernet, para ello en conexión marcamos “Ethernet” y veremos cómo nos aparecen los campos de entrada “IP”, “Port” y “MAC”, los cuales tendremos que rellenar con la ip, el puerto y la dirección MAC del ordenador

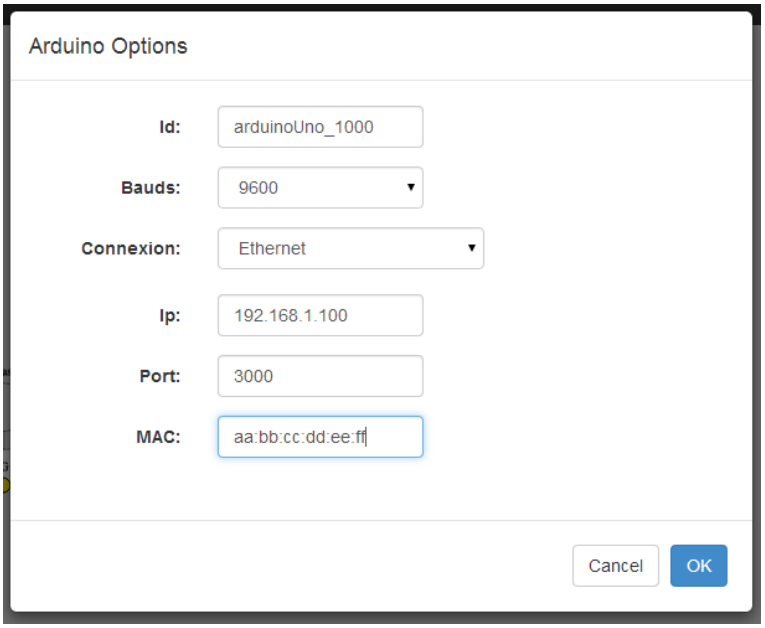


Figura 54 - Configuracion de la placa

servidor al cual estará conectado. Podemos ver un ejemplo en la imagen siguiente.

Una vez configurada la placa, pasamos al sensor de gas, el cual al hacer doble clic sobre él, se nos abre una ventana parecida a la de configuración de la placa en la cual deberemos elegir el tipo de función que deseemos, en nuestro caso elegiremos “GAS_SMOKE” el cual nos devolverá el porcentaje de humo en el habiente. Si quisiésemos crear una alarma que detectase el dióxido de carbono en el aire, tendríamos que usar los mismos componentes pero cambiaríamos la función del sensor de gas, el cual debería ser “GAS_CO”. En el caso de un detector de gas butano seleccionaríamos “GAS_LPG”.

Al hacer doble clic sobre el “Buzzer” nos encontramos que no tiene parámetros de configuración, ya que no los necesita.

Una vez que tengamos todos los componentes conectados y configurados, podemos hacer una simulación para ver que rango de valores puede devolver nuestro sensor de gas. Para ello vamos a la barra superior y pinchamos en “Simulate”. Al hacerlo, vemos como nos aparece la consola de simulación por la derecha, en la cual podemos modificar el tiempo en el cual nos aparecen los datos modificando el valor de tiempo pinchando sobre los botones con el símbolo “+” y “-”. Al hacer pinchar en “Start” empezaremos a recibir valores de la simulación como vemos en la imagen siguiente. Como vemos el sensor de gar devuelve valores comprendidos entre 0 y 1, ya que es el porcentaje de humo en el aire.

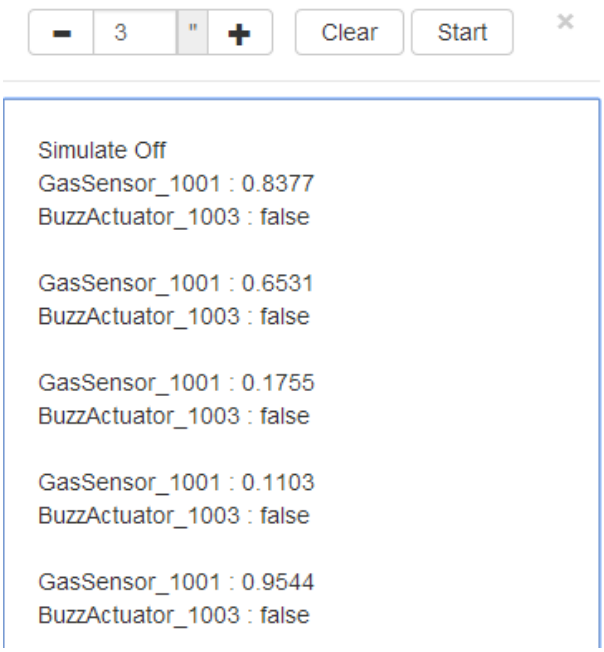


Figura 55 - Simulacion de un sensor de gas

Una vez que tenemos el modelo terminado, generamos el código haciendo clic en “Compile”, el cual nos mostrara una ventana diciendo que el código ha sido compilado correctamente y nos ofrecerá la opción de descargar el código. Al hacerlo, podremos ver como se nos descarga en nuestro ordenador un archivo comprimido en formato “zip” con el mismo nombre que nuestro proyecto, en nuestro caso “alarmaHumo.zip”. Descomprimimos los ficheros y los metemos en un directorio con el mismo nombre que el proyecto.

```
alarmaHumo | handlers
12 const int SmokeSensor_1004_PIN = 3;
13 const int BuzzActuator_1003_PIN = 0;
14
15 char BUZZACTUATOR_1003_ID[] = "BuzzActuator_1003";
16
17 byte mac0[] = {0xAA, 0xBB, 0xCC, 0xDD, 0xEE, 0xFF};
18 IPAddress server0(192,168,1,100);
19 IPAddress myIP0(192,168,1,100);
20 int port0 = 3000;
21
22 EthernetClient arduinoUno_1000;
23 EthernetCommunication comm;
24 SmokeSensor SmokeSensor_1004;
25 BuzzActuator BuzzActuator_1003;
26
27 void setup(){
28         Serial.begin(9600);
29         comm.begin(mac0, server0, myIP0, port0);
30         SmokeSensor_1004.begin(SmokeSensor_1004_PIN);
31         BuzzActuator_1003.begin(BuzzActuator_1003_PIN);
32 }
33 void loop(){
34         comm.writeData("SmokeSensor read value is: \n","
35         comm.writeData(SmokeSensor_1004.MQGetGasPercenta
36
37         BuzzActuator_1003.playWithBuzzIntensity(500, 100
38         delay(1000);
```

Figura 56 - Codigo generado automaticamente

En caso de tener instalado en nuestro sistema el entorno Arduino IDE, podremos hacer doble clic sobre el fichero con el mismo nombre que el proyecto y se nos abrirá dicho entorno con los ficheros cargados. En el caso de no tener instalado Arduino IDE deberemos instalarlo, o en el caso de usar otro entorno de programación, deberemos importar los ficheros a un proyecto de propio entorno. En nuestro caso utilizaremos Arduino IDE ya que es el

entorno de programación oficial.

En la ilustración anterior podemos ver el código obtenido. En el caso de tener una placa Arduino conectada a nuestro ordenador, podríamos compilar y cargar el código para ver su funcionamiento. En este caso, nos mostraría por la consola de depuración el texto “SmokeSensor read value is:” seguido del valor obtenido por el sensor, a continuación haría que el “buzzer” pitara dos veces con un intervalo de un segundo entre pitido y pitido.

Para finalizar nuestra alarma de humos, tan solo tendríamos que comparar el valor recibido por el sensor con un umbral que nosotros creamos, entonces llamaremos la sentencia para hacer que suene el altavoz.

Como podemos ver en la imagen siguiente, el usuario tan solo debería de introducir el código bordeado por el cuadro verde.

```
void loop(){

    comm.writeData("SmokeSensor read value is: \n","");
    comm.writeData(SmokeSensor_1004.MQGetGasPercentage(SmokeSensor_1004.MQRead(),GAS_SMOKE)+"\n","");

    float value = SmokeSensor_1004.MQGetGasPercentage(SmokeSensor_1004.MQRead(),GAS_SMOKE);
    if(value > 0.5){
      BuzzActuator_1003.playWithBuzzIntensity(500, 1000);
      delay(1000);
      BuzzActuator_1003.playWithBuzzIntensity(1000, 1000);
      delay(1000);
    }
    Serial.flush();
    String cadena=comm.readData();
    char addressBuffer[20]="init";
    char valBuffer[100];
```

Figura 57 - Codigo que debería rellenar el usuario

7. Conclusiones y trabajo futuro

En este último capítulo vamos a presentar las conclusiones extraídas una vez desarrollada la propuesta que motivó esta tesis de máster, así como que el trabajo que aún se podría mejorar una vez completada esta.

7.1. Conclusiones

Gracias a las tecnologías actuales, se ha podido diseñar e implementar una herramienta web mediante JavaScript, la cual genera código arduino de forma automática, verifica la correcta conexión de los componentes y emula el código generado antes de ser descargado.

Para este cometido se ha utilizado el lenguaje de programación JavaScript tanto en la parte del cliente como en la del servidor, usando el framework Node.js y JSON para el intercambio de información. Para que la interfaz de usuario se adapte a la resolución de la mayor parte de los dispositivos en los que puede ser mostrada, se ha utilizado la librería Botstrap.

Por otro lado, como medio para guardar la información generada por el usuario, se ha utilizado MongoDB en el cometido de base de datos. Gracias a que JSON es la

forma natural de pasar objetos JavaScript y que MongoDB guarda dichos objetos, el paso de datos ha sido muy natural.

Para no convertir esta herramienta en una aplicación cerrada, se creó un servicio web con el cual poder interactuar con los datos, dejando de esta manera vía libre para futuros desarrollos.

7.2. Trabajo futuro

El trabajo realizado en esta tesis de máster no es trabajo cerrado, sino todo lo contrario, es una línea de investigación abierta donde se pueden abordar muchísimas mejoras. A continuación vamos a proponer una serie de principales trabajos futuros que se podrían realizar:

- **Ampliar el abanico de sensores y actuadores:** Actualmente el desarrollo de la librería TatAmi está parado, si la ampliásemos tendríamos una mayor variedad de sensores y actuadores los cuales generar código automáticamente.
- **Añadir la lógica al programa:** Un punto muy interesante para un futuro desarrollo, es la ampliación de la aplicación para que el código final sea completo y correcto. Actualmente, si queremos que cuando no haya suficiente luz se encienda una lámpara, la aplicación genera el código necesario para avisarnos cuando los niveles de luz bajen de un umbral, así como también para encender y apagar un relé, pero debemos completar el código nosotros mismos haciendo que cuando nos llegue la señal de que la luz ha bajado del umbral, encienda el relé.
- **Plugin para pasar el código a la placa:** Otro punto te desarrollo, podría ser la creación de una herramienta opcional, la cual compile y envíe el código a una placa que tuviésemos conectada a nuestro ordenador, evitando de esta manera el tener que depender de arduino IDE.

Bibliografía

Raphaël: http://raphaeljs.com/reference.html

Bootstrap: http://getbootstrap.com/

Node.JS: http://nodejs.org/api/fs.html

Express.js: http://expressjs.com/3x/api.html

JQuery: http://api.jquery.com/

MongoDB: http://docs.mongodb.org/manual/

Mongoose.js: http://mongoosejs.com/docs/

Errores HTTP: http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

TatAmi: http://www.pros.upv.es/m4spl/tatami/thelibrary.php

JavaScript: https://developer.mozilla.org/en-US/docs/Web/JavaScript

Servicios web: Leonard Richarson and Sam Ruby (2007). Restful web services ed: O'Reilly

Anexos

Pruebas de carga del servidor

Se decidió hacer unas pruebas de carga en el actual servidor, para comprobar cuantas peticiones podrían ser soportadas. Para realizar dicho test se utilizó la herramienta ApacheBench (ab).

En este anexo se pueden ver los resultados de los diferentes test realizados.

A.1. Test de carga con 100 peticiones y una concurrencia de 10

Server Software:	
Server Hostname:	raulhome.ddns.net
Server Port:	3001
Document Path:	/tesina/files/raugaral
Document Length:	4199 bytes
Concurrency Level:	10
Time taken for tests:	0.942 seconds
Complete requests:	100
Failed requests:	0
Total transferred:	441700 bytes
HTML transferred:	419900 bytes
Requests per second:	106.11
Transfer rate:	468673.07 kb/s received

Connnection Times (ms)

	min	avg	max
Connect:	4	9	16
Processing:	21	80	89
Total:	25	89	105

A.2. Test de carga con 100 peticiones y una concurrencia de 100

Server Software:	
Server Hostname:	raulhome.ddns.net
Server Port:	3001
Document Path:	/tesina/files/raugaral
Document Length:	4199 bytes
Concurrency Level:	100
Time taken for tests:	0.998 seconds
Complete requests:	100
Failed requests:	0
Total transferred:	441700 bytes
HTML transferred:	419900 bytes
Requests per second:	100.21
Transfer rate:	442606.01 kb/s received

Connnection Times (ms)

	min	avg	max
Connect:	5	9	16
Processing:	16	466	925
Total:	21	475	941

A.3. Test de carga con 500 peticiones y una concurrencia de 100

Server Software:	
Server Hostname:	raulhome.ddns.net
Server Port:	3001
Document Path:	/tesina/files/raugaral
Document Length:	4199 bytes
Concurrency Level:	100
Time taken for tests:	5.590 seconds
Complete requests:	500
Failed requests:	0
Total transferred:	2208500 bytes
HTML transferred:	2099500 bytes
Requests per second:	89.45
Transfer rate:	395103.40 kb/s received

Connnection Times (ms)

	min	avg	max
Connect:	5	10	30
Processing:	17	1004	1331
Total:	22	1014	1361

A.4. Test de carga con 500 peticiones y una concurrencia de 500

Server Software:	
Server Hostname:	raulhome.ddns.net
Server Port:	3001
Document Path:	/tesina/files/raugaral
Document Length:	4199 bytes
Concurrency Level:	500
Time taken for tests:	5.309 seconds
Complete requests:	500
Failed requests:	0
Total transferred:	2208500 bytes
HTML transferred:	2099500 bytes
Requests per second:	94.17
Transfer rate:	415960.84 kb/s received

Connnection Times (ms)

	min	avg	max
Connect:	6	10	35
Processing:	25	2652	5265
Total:	31	2662	5300

Anexo 1 - Test de carga con 500 peticiones y una concurrencia de 500