UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Título del Proyecto:

Hola Mundo CMU

Autor:

Fernández Ippólito, Juan Carlos

Director:

Pérez Llorens, Rubén

TESINA PARA LA OBTENCIÓN DEL TÍTULO DE:

Máster en Desarrollo de Aplicaciones sobre Dispositivos Móviles

Septiembre del 2017

Contenido

Introducción

Descripción del problema

Cuando se comienza a trabajar o aprender una tecnología que es nueva, para quien emprende esa faena, existe una barrera inicial que suele ser ardua y molesta de atravesar.

Allá por 1978, el legendario libro "El Lenguaje de Programación C", de Kernighan & Ritchie, en el primer ejercicio del primer capítulo, invitaba a escribir y ejecutar este programa:

```
#include <stdio.h>

main()

{
        printf("hola, mundo\n");
}
```

Si se obtenía en el monitor el saludo, implicaba que un sinnúmero de cosas que se habían llevado a cabo previamente para llegar hasta ese punto funcionaban bien, y esa tediosa barrera inicial estaba superada.

Y así, a lo largo de los años, hemos visto "hola mundos" en cada lenguaje de programación, framework, modelo de trabajo o cualquier herramienta nueva que aparecía en el universo del software.

El problema es que la barrera mencionada es, en algunos casos, capaz de desalentar a tal punto el impulso inicial, que se corre el riesgo de dejar de lado una valiosa tecnología o paradigma y usar otra cosa, simplemente por conocida.

Objetivos

En cada asignatura, aun siendo el dictado de ellas intenso, o tal vez gracias a eso, quedaron líneas abiertas de estudio y algunas extensiones propuestas a las prácticas, que la falta de tiempo me impidió recorrer o concretar.

En este contexto, el objetivo del proyecto es llevar a cabo un "Hola Mundo", tomando como base los conocimientos adquiridos en muchas de las materias cursadas en el Master en Desarrollo de Aplicaciones sobre Dispositivos Móviles y transitar algunas de las líneas abiertas mencionadas.

Tiene el valor, para mí y para quien quiera usarlo (pues será de código abierto), de tener superadas las barreras de ingreso a la implementación de cada parte que lo compone, o al

menos servir como guía para ello, como si fuese un "Hola mundo" del último año de la Maestría, del CMU.

Una lista breve de los temas involucrados podría ser esta:

- Arquitecturas ARM
- Sensorización, MEMS, I2C
- IoT
- Windows UWP
- Unity 3D
- Arquitecturas SOA, Servicios REST
- Cloud Computing, Node.js, Express.js, Heroku, HTML5, javascript, CSS3, jquery...
- Phonegap

El objetivo entonces, no consiste en realizar un producto o aplicación con un fin muy específico, sino más bien, en un marco o esqueleto que enhebra gran cantidad de las tecnologías que aprendí durante la cursada del Máster en Desarrollo de Aplicaciones sobre Dispositivos Móviles, superando así algunas de las barreras de entrada en las líneas de estudio abiertas.

Motivación

La motivación es fundamentalmente personal: siempre me sentí atraído por los sistemas SCADA (Supervisión, Control y Adquisición de Datos).

IoT, las tecnologías móviles y la Computación Móvil y Ubicua abren un universo nuevo y apasionante para transitar y explorar. En este sentido, el proyecto presentó algunos retos que no quise privarme de disfrutar.

Tecnologías utilizadas

Una breve descripción de las tecnologías, paradigmas de diseño o plataformas utilizadas y no descriptas durante el Máster, es la que sigue:

Ecosistema de ST Microelectronics.

http://www.st.com/en/ecosystems.html

ST y empresas asociadas ofrecen un ecosistema completo de hardware y software para facilitar la rápida evaluación, creación de prototipos y la producción de sistemas completos utilizando microcontroladores STM32 con actuadores, conectividad, sensores, unidades de potencia y periféricos de E/S estándar.

El entorno de desarrollo abierto STM32 es una colección de componentes compatibles e interoperables de hardware y software que permite el desarrollo y creación de prototipos de aplicaciones innovadoras basadas en el microcontrolador STM32 de 32 bits combinado con otros componentes ST.

La suite de desarrollo de ST "Open Software Expansion", ofrece drivers, middleware y software de aplicación para iniciar diseños con diveras tecnologías (MEMS, Bluetooth debaja energía , sensores, etc.) utilizando tanto entornos de desarrollo abiertos de ST como propietarios de diversas marcas (IAR, Keil...).

El entorno de desarrollo abierto ST

http://www.st.com/en/ecosystems/stm32-open-development-environment.html

El entorno de desarrollo abierto STM32 (STM32 Open Development Environment) es una forma abierta, flexible y relativamente económica de desarrollar dispositivos y aplicaciones basados en la familia de microcontroladores STM32 de 32 bits combinados con otros componentes de ST conectados a través de tarjetas de expansión. Permite el prototipado con componentes que pueden transformarse rápidamente en diseños finales.

El entorno de desarrollo abierto STM32 incluye estos cinco componentes:

- STM32 Nucleo development boards: placas de desarrollo con microcontroladores STM32 con programador y debugger incluidos. Tiene grandes posibilidades de expansión mediante los conectores que dispone.
- STM32 Nucleo expansion boards: tarjetas con funcionalidad adicional para sensado, control, conectividad, audio, etc.
- STM32Cube software: un conjunto de herramientas gratuitas y componentes de software para facilitar y acelerar el desarrollo. Incluye una capa de abstracción del hardware (HAL: Hardware Abstraction Layer), middlewares y la aplicación para

ordenador STM32CubeMX que es un configurador de los microcontroladores y generador de código.
- STM32Cube expansion software: software gratuito compatible con el framework de STM32Cube.
- STM32 ODE Function Packs: un conjunto de ejemplos de funciones para algunos de los casos de aplicación más comunes, construidos aprovechando la modularidad e interoperabilidad de las placas de desarrollo y ampliaciones STM32 Nucleo, con el software STM32Cube.

Raspberry Pi 3

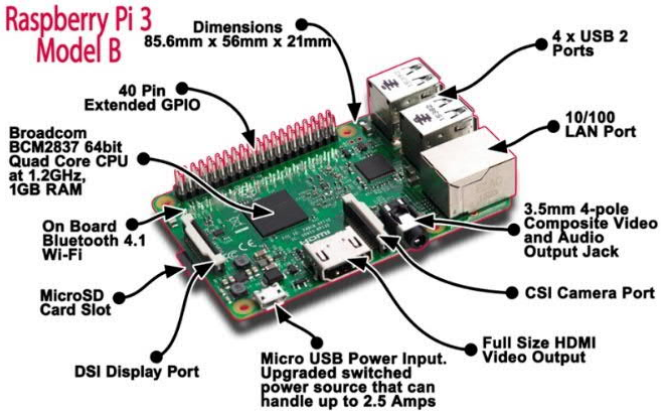
Raspberry Pi 3 es un ordenador pequeño (aprox. 5.5 x 8.5 cm) y económico (alrededor de 37 €), montado en un solo circuito impreso y que dispone de:

- CPU 1.2GHz 64-bit quad-core ARMv8
- 802.11n Wireless LAN
- Bluetooth 4.1
- Bluetooth Low Energy (BLE)
- 4 USB ports
- 40 GPIO pins
- Full HDMI port
- Ethernet port
- Combined 3.5mm audio jack and composite video
- Camera interface (CSI)
- Display interface (DSI)
- Micro SD card slot (now push-pull rather than push-push)
- VideoCore IV 3D graphics core

En este ordenador pueden instalarse distintos sistemas operativos, siendo el nativo Raspbian, un derivado de Debian, una de las distribuciones de Linux.

En este proyecto el sistema operativo será Windows 10 para la Internet de las cosas: Windows 10 IoT Core.

En la imagen que sigue podemos observar la Raspberry Pi 3 utilizada en este proyecto:



Windows 10 IoT Core

Windows 10 IoT Core es el sistema operativo que ofrece hoy (2017) Microsoft para construir proyectos de internet de las cosas.

La elección de utilizar este sistema operativo en el proyecto se fundamenta en la posibilidad de utilizar las herramientas que provee Microsoft para la Universal Windows Platform (UWP), como los son el Visual Studio, Blend, .NET y mucho del software disponible en NuGet (https://www.nuget.org/).

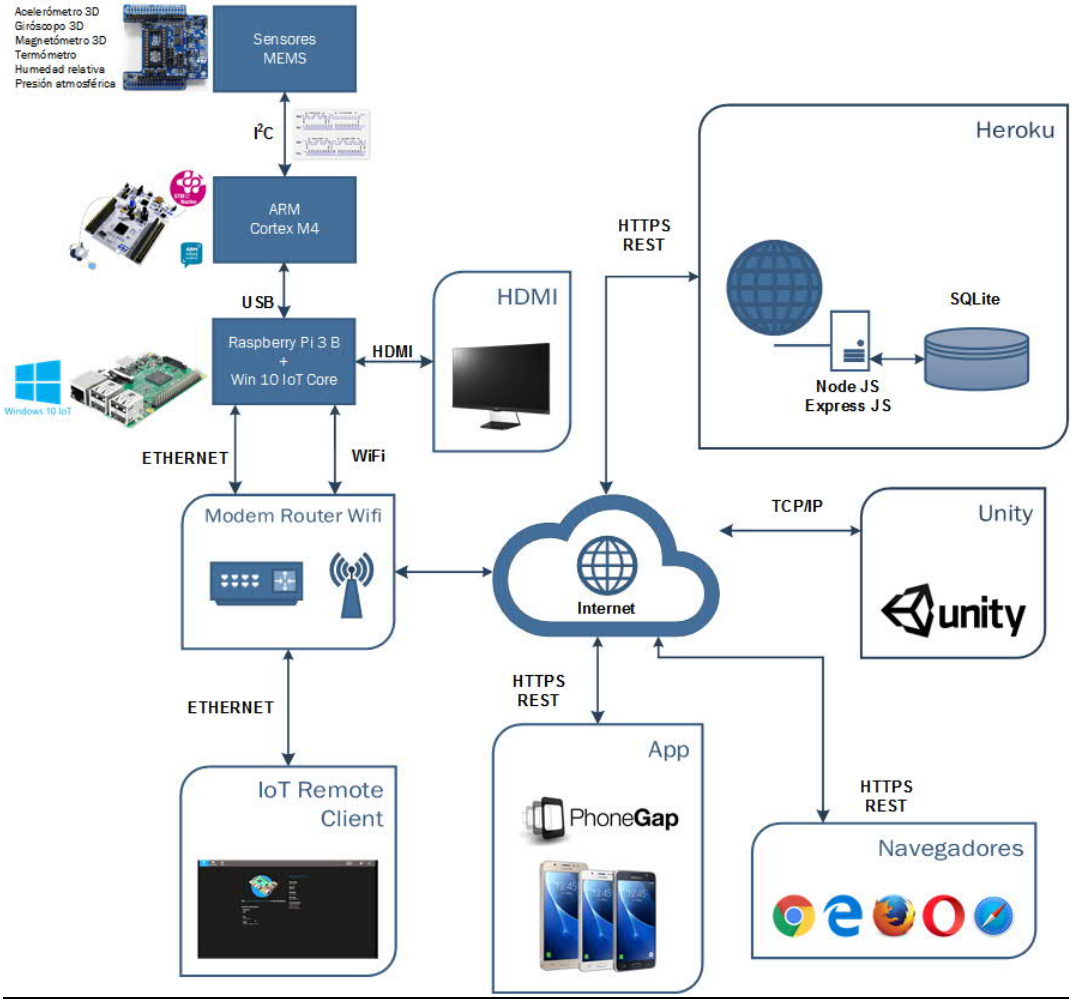
Sin embargo, tal vez habrá que enfrentar algunos problemas, pues la instalación de drivers no es trivial.

Para más información sobre Windows IoT Core, puede comenzarse por el siguiente link:

https://developer.microsoft.com/es-es/windows/iot

Arquitectura del sistema

Diagrama en bloques



Modelo de datos

La base de datos que almacena los datos de los sensores es Postgres, reside en la nube en la plataforma como servicio (PaaS) Heroku. Consta de una sola tabla denominada Sensores.

La tabla sensores tiene una estructura de acuerdo a la siguiente sentencia SQL de creación:

```
CREATE TABLE `Sensores` (`id` INTEGER PRIMARY KEY AUTOINCREMENT, `Hours` INTEGER, `Minutes`
INTEGER, `Seconds` INTEGER, `subSec` INTEGER, `ACC_X` INTEGER, `ACC_Y` INTEGER, `ACC_Z` INTEGER,
`GYR_X` INTEGER, `GYR_Y` INTEGER, `GYR_Z` INTEGER, `MAG_X` INTEGER, `MAG_Y` INTEGER, `MAG_Z`
```

```
INTEGER, `PRESS` FLOAT, `HUM` FLOAT, `TEMP` FLOAT, `createdAt` DATETIME NOT NULL, `updatedAt`
DATETIME NOT NULL)
```

En la siguiente imagen se muestra la estructura de la tabla Sensores:

Column ID	Name	Type	Not Null	Default Value	Primary Key
0	id	INTEGER	0	null	1
1	Hours	INTEGER	0	null	0
2	Minutes	INTEGER	0	null	0
3	Seconds	INTEGER	0	null	0
4	subSec	INTEGER	0	null	0
5	ACC_X	INTEGER	0	null	0
6	ACC_Y	INTEGER	0	null	0
7	ACC_Z	INTEGER	0	null	0
8	GYR_X	INTEGER	0	null	0
9	GYR_Y	INTEGER	0	null	0
10	GYR_Z	INTEGER	0	null	0
11	MAG_X	INTEGER	0	null	0
12	MAG_Y	INTEGER	0	null	0
13	MAG_Z	INTEGER	0	null	0
14	PRESS	FLOAT	0	null	0
15	HUM	FLOAT	0	null	0
16	TEMP	FLOAT	0	null	0
17	createdAt	DATETIME	1	null	0
18	updatedAt	DATETIME	1	null	0

Listado de Servicios Web

El servidor instalado en Heroku y utilizando Node JS ofrece los siguientes servicios web:

Operaciones en https://holamundocmu.herokuapp.com		
uri	método	Servicio en
sensores/recall	GET	https://holamundocmu.herokuapp.com/sensores/recall
sensores/store	POST	https://holamundocmu.herokuapp.com/sensores/store

Ruta Sensores/recall

Request:

El body del request de esta ruta está vacío

Response:

El siguiente es un ejemplo del body del response Json:

```
{
        fecha: {object date},
        temperatura: 1.26743233E+15,
        humedad: 1.26743233E+15,
        presion: 1.26743233E+15,
        acc_x: 2147483647,
        acc_y: 2147483647,
        acc_z: 2147483647,
        gyr_x: 2147483647,
        gyr_y: 2147483647,
        gyr_z: 2147483647,
        mag_x: 2147483647,
        mag_y: 2147483647,
        mag_z: 2147483647
}
```

Nota: si se accede desde un navegador a esta ruta, se observarán los datos correspondientes a la última adquisición realizada. Dependiendo si Raspberry está en línea o no, la fecha puede ser antigua.

Esto es un ejemplo de lo que se observa como respuesta en un navegador:


```
{"fecha":"2017-09-
21T19:30:29.132Z","temperatura":24.1,"humedad":61.4,"presion":1019.82,"acc_x"
:-113,"acc_y":-39,"acc_z":1038,"gyr_x":399,"gyr_y":-483,"gyr_z":-
287,"mag_x":-138,"mag_y":-223,"mag_z":634}
```

Ruta Sensores/store

Request:

El siguiente es un ejemplo del body del request Json:

```
{
        Hours: 2147483647,
        Minutes: 2147483647,
        Seconds: 2147483647,
        subSec: 2147483647,
        ACC_X: 2147483647,
        ACC_Y: 2147483647,
        ACC_Z: 2147483647,
        GYR_X: 2147483647,
        GYR_Y: 2147483647,
        GYR_Z: 2147483647,
        MAG_X: 2147483647,
        MAG_Y: 2147483647,
        MAG_Z: 2147483647,
        PRESS: 1.26743233E+15,
        HUM: 1.26743233E+15,
        TEMP: 1.26743233E+15
}
```

El siguiente es un ejemplo del body del response:

```
"OK"
```

Componentes del sistema

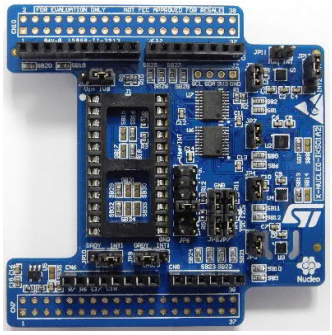
Sensores MEMS

Los sensores MEMS (Micro electro-mechanical systems) están montados sobre un circuito impreso fabricado por ST Microelectronics y cuenta con los siguientes sensores:

- Acelerómetro 3D
- Giróscopo 3D
- Magnetómetro 3D
- termómetro
- sensor de humedad relativa
- sensor de presión atmosférica

El circuito impreso está diseñado para conectarse directamente sobre la placa NUCLEO-F401RE.

En la imagen que sigue podemos observar la placa de expansión X-NUCLEO-IKS01A2 utilizada en este proyecto:



El intercambio de datos con los sensores se realiza mediante el protocolo I^2C.

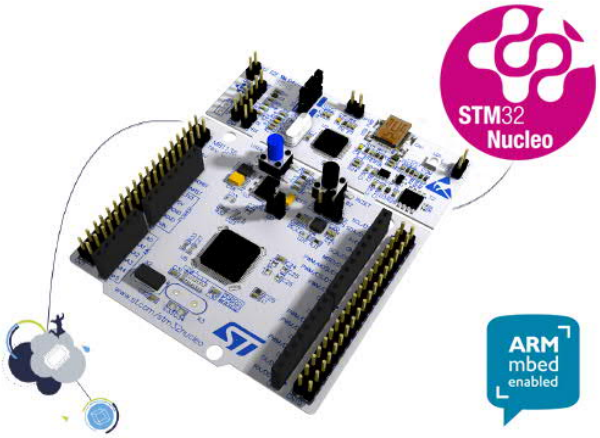
Una descripción completa de la placa de expansión X-NUCLEO-IKS01A2 puede encontrarse en esto link:

http://www.st.com/en/ecosystems/x-nucleo-iks01a2.html

Microcontrolador ARM Cortex M4

Este microcontrolador está montado sobre placa NUCLEO-F401RE de ST Microelectronics. Es una buena opción para hacer pruebas de concepto, pues ST la rodea de todo un ecosistema que permite realizar desarrollos muy velozmente y a bajo costo.

En la imagen que sigue podemos observar la MCU ARM (el integrado central más grande entre las dos hileras de expansión y pines), y los componentes asociados:



Una descripción completa de la placa y de la MCU pueden encontrarse en estos links:

http://www.st.com/content/st_com/en/products/evaluation-tools/product-evaluation-tools/mcu-eval-tools/stm32-mcu-eval-tools/stm32-mcu-nucleo/nucleo-f401re.html

http://www.st.com/content/st_com/en/products/microcontrollers/stm32-32-bit-arm-cortex-mcus/stm32-high-performance-mcus/stm32f4-series/stm32f401/stm32f401re.html

Además de una gran variedad de circuitos impresos de extensión, como el de sensores que utilizamos en este proyecto, están disponibles una importante cantidad de herramientas gratuitas, como lo son aplicaciones para configurar los diferentes subsistemas del microcontrolador ARM en forma gráfica y generar código, hasta ambientes integrados de desarrollo, estos últimos tanto gratuitos como de pago.

Ejemplo de esto es la aplicación STM32CubeMX, utilizada durante este desarrollo y el IDE CoIDE, ambos gratuitos y obtenibles mediante registro en el sitio de ST.

Esta placa de evaluación consta de un circuito impreso cuyo componente principal es la MCU ARM Cortex M4 STM32F401RE, rodeado de la "lógica de pegamento", fuente de alimentación, circuitos drivers para los puertos de comunicaciones, la electrónica que permite descargar un programa a la MCU y hacer debug desde un ordenador de escritorio, conectores de expansión, algunos leds y botones para comprobar el funcionamiento de los programas descargados, memori, etc.

El código fuente que adquiere los datos de los sensores y los transmite mediante el puerto USB de la placa NUCLEO-F401RE puede encontrarse en la carpeta "ARM y MEMS" en el siguiente link: https://drive.google.com/drive/folders/0B-2HrrU-WoSgRUJmcjdJWlptazg?usp=sharing

Este código es una modificación del ejemplo DataLog de ST para la placa de expansión X-NUCLEO-IKS01A2, disponible en la instalación de la herramienta STM32CubeMX.

En este caso, los datos de los sensores son adquiridos cada 500 ms (ver el delay en el archivo STM32F401XX-NUCLEO\Src\main.c, linea 234) mediante un bucle while infinito, en el que primero se adquieren los valores medidos por cada sensor, utilizando las siguientes funciones:

- Pressure_Sensor_Handler(&Msg);
- Humidity_Sensor_Handler(&Msg);
- Temperature_Sensor_Handler(&Msg);
- Accelero_Sensor_Handler(&Msg);
- Gyro_Sensor_Handler(&Msg);
- Magneto_Sensor_Handler(&Msg);

Y luego se los envía al Raspberry mediante la función SendDataToRaspberry(); generando previamente un string igual a lo que sería el resultado de un JSON.stringify, construyéndolo en base a este string de formato de un snprintf:

```
char dataOutrb[MAX_BUF_SIZE];
char jsonFormat[] =
        "{\
                \"Hours\": %d, \
                \"Minutes\": %d, \
                \"Seconds\": %d, \
                \"subSec\": %d, \
                \"ACC_X\": %d, \
                \"ACC_Y\": %d, \
                \"ACC_Z\": %d, \
                \"GYR_X\": %d, \
                \"GYR_Y\": %d, \
                \"GYR_Z\": %d, \
                \"MAG_X\": %d, \
                \"MAG_Y\": %d, \
                \"MAG_Z\": %d, \
                \"PRESS\": %d.%02d, \
                \"HUM\": %d.%02d, \
                \"TEMP\": %c%d.%02d\
        }";
```

Los datos son entonces así enviados mediante el puerto USB hacia el Raspberry.

Raspberry Pi 3B con Windows 10 IoT Core

Este dispositivo tendrá la misión de colectar los datos de los sensores enviados por el procesador ARM Cortex M4 a su puerto USB y luego derivarlos a la aplicación Unity mediante un socket TCP/IP y a un servidor instalado en Heroku, utilizando en este caso servicios web REST.

Instalación de Windows 10 IoT Core en Raspberry Pi 3B

Hay varios caminos para llevar a cabo la instalación de Windows 10 IoT Core en Raspberry. La elección de uno u otro puede depender de las limitaciones del hardware del cual disponemos. En mi caso, solo tenía disponible un drive de SD card en una notebook con Windows 7, mientras que el ordenador con Windows 10 no tenía la capacidad de formatear y escribir en una memoria de ese tipo.

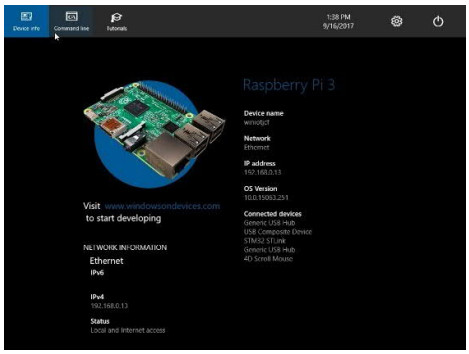
Los pasos, entonces, podrían describirse de esta manera:

1. Descargar la NOOBS desde https://www.raspberrypi.org/downloads/noobs/ : con esto se obtiene un archivo compactado, el cual al descomprimirlo genera una carpeta cuyo contenido son carpetas interiores y archivos.
2. Formatear la SD Card. Utilicé una de 32 GBytes de capacidad y el formateo lo realicé con la aplicación SD Formatter 4.0 tool, que puede descargarse desde https://www.sdcard.org/downloads/formatter_4/
3. Copiar los archivos extraídos de NOOBS en la SD formateada
4. Insertar la SD en el Raspberry, el cual debe tener conectado un monitor (HDMI), un teclado y un mouse (ambos USB)

5. Alimentar el Raspberry y una vez que inicia, seguir los pasos indicados para conectarse a internet.
6. En la lista de sistemas operativos disponibles, escoger el release oficial de Windows 10 IoT Core (también puede instalarse Windows 10 IoT Core Insider, si uno está dispuesto a lidiar con las inestabilidades de versiones avanzadas).

Recomiendo que la conexión a internet se realice mediante el puerto ethernet y no wifi durante la descarga, pues el tamaño de ella es grande.

Cuando este procedimiento finaliza, Raspberry solicita reiniciar y si todo anduvo bien, nos encontraremos con una pantalla similar a esta:



En este punto, es posible que uno se sienta algo decepcionado, pues estamos acostumbrados a que un sistema operativo nos ofrezca una enorme cantidad de herramientas y aplicaciones que no encontraremos en esta instalación de Windows 10 IoT Core.

De hecho, la instalación de Raspbian, sobre Rasperry Pi 3B, ofrece muchas de las aplicaciones que encontramos habitualmente en cualquier distribución de Linux: navegador, block de notas, editores y un largo etcétera. Nada de esos sucede en Win IoT Core. El resultado es bastante espartano.

Sin embargo, disponemos de la enorme ventaja de poder utilizar todo el poder de la Plataforma Universal de Windows (UWP) para desarrollar aplicaciones y, considerando que el sistema operativo está orientado a IoT, el balance es en mi opinión, positivo.

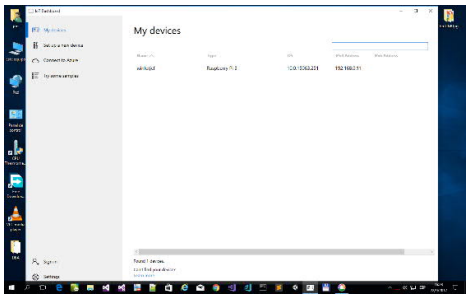
De todas maneras, existen ciertas pegas como por ejemplo la instalación de drivers, la cual no es trivial.

Herramientas para trabajar con Windows 10 IoT Core

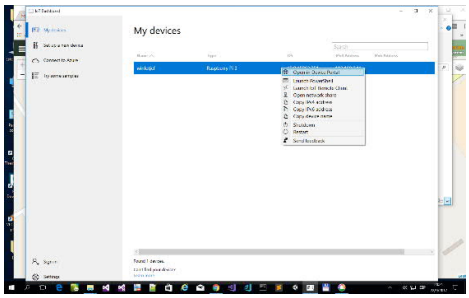
Es necesario descargar la aplicación IoT Core Dashboard:

https://developer.microsoft.com/en-us/windows/iot/downloads

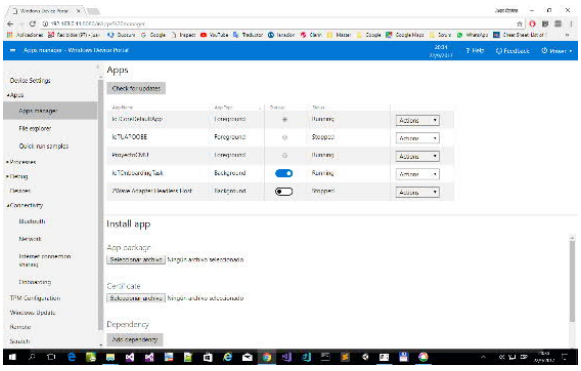
Esta aplicación (que requiere Windows 10 para ejecutarse) permite configurar más (muchos más) parámetros del sistema operativo y del hardware que lo que nos permite la pantalla que genera el Raspberry sobre el monitor HDMI conectado.



Si seleccionamos el dispositivo encontrado y hacemos click con el botón derecho, se despliega un interesante menú que nos da acceso a un escritorio remoto (liberándonos del monitor HDMI) y a un portal que se accede con un navegador, pues el Raspberry con Win 10 IoT dispone de un servidor web que expone su configuración, permite administrar las aplicaciones, tiene un File Explorer, se pueden ver los dispositivos instalados, los procesos, cambiar el nombre y el password, etc.



En la siguiente imagen vemos el administrador de aplicaciones mostrando el programa ProyectoCMU ejecutándose:



El Apps Manager del portal permite iniciar y detener la aplicación y también es posible configurarla para que se ejecute automáticamente en el inicio del sistema.

Para que Win 10 IoT autorice el acceso mediante la app cliente de escritorio remoto, es necesario habilitar esta característica desde el mencionado portal (Windows Device Portal), en la opción Remote de su barra de navegación lateral.

Importante: las claves de acceso iniciales por defecto son:

- Username: Administrator
- Password: p@ssw0rd

Instalación de drivers en Windows IoT Core sobre Raspberry

Históricamente la instalación de drivers ha sido problemática sea cual fuere el sistema operativo: drivers que funcionaban bien en XP no lo hacían en Win7 y así otros problemas como versiones de 32 o de 64 bits, para no mencionar el problema que se enfrenta instalando una impresora nueva en alguna distribución de Linux.

Win IoT Core no le escapa a esto y si bien es de esperar que el teclado y el mouse no tengan problema al conectarse a los puertos USB del Raspberry para ser reconocidos, si puede haber algún inconveniente si intentamos conectar un dispositivo no tan esperable.

El procedimiento que en la fecha de realización de esta memoria suministra Microsoft para instalar drivers de periféricos USB, está descripto en este enlace:

https://docs.microsoft.com/en-us/windows/iot-core/learn-about-hardware/PeripheralDrivers

En los primeros renglones encontramos una frase que puede resultarnos devastadora:

> "For ARM, please contact the supplier of the peripheral to get the sys/inf files."

y conseguir drivers para ARM puede resultar muy cuesta arriba.

Si tenemos la suerte de que el dispositivo a instalar esté basado en un chip de Future Technology Devices International, comúnmente conocida por su abreviatura FTDI, lo cual no es demasiado infrecuente, un camino alternativo puede ser el que propone David Jones, es su artículo en el blog de embedded101.com:

http://embedded101.com/Blogs/David-Jones/entryid/666/win-10-iot-core-ftdi-serial-driver

En particular y en este proyecto, el dispositivo a conectar fue fabricado por ST Microelectronics, y es la placa de evaluación y desarrollo denominada NUCLEO-F401RE, siendo el nombre del driver STM32 STLink.

Meses atrás, en las etapas iniciales del proyecto no lograba que la Raspberry con Win IoT reconociera que el dispositivo estaba conectado y utilicé el procedimiento descripto por David Jones, indicado más arriba en esta memoria, con éxito.

Sin embargo, tiempo después y durante un inicio del sistema, Win IoT Core se actualizó en forma automática (y sin pedir permiso). Como el dispositivo no respondía y no tenía conectado el monitor en la interfaz HDMI, lo desconecté de la alimentación intentando que se inicie, sin éxito esta vez.

Con un monitor conectado pude ver lo que estaba sucediendo, pero las reiteradas desconexiones de la alimentación provocaron (al menos supongo que esa fue la causa) que el sistema no respondiera.

Al reinstalar todo desde cero, para mi sorpresa el dispositivo USB fue reconocido sin intervención de mi parte. Debido a esto me quedó la duda acerca de si el afortunado hecho se debió a que la nueva versión de sistema operativo había incorporado este dispositivo (STLink V2) a la lista que reconoce, o que se siempre lo reconoció y las dificultades del principio para poner en funcionamiento el sistema me confundieron y creí que era necesaria la instalación de un driver, cuando no era así.

Como conclusión de esta experiencia, es muy recomendable asegurarse de que el dispositivo es reconocido o no, haciendo uso del portal web y de la entrada Devices de su menú lateral, recorriendo minuciosamente el árbol del Device Manager, tratando de encontrar la entrada del dispositivo que intentamos conectar. Si estamos seguros de que no está, entonces tendremos que lidiar con otros engorrosos procedimientos.

Aplicación UWP

El desarrollo de la aplicación que se ejecuta en el Raspberry se realizó con Visual Studio 2017 y Blend.

Su código fuente puede encontrarse en la carpeta “Windows 10 IoT Core y Raspberry” en el siguiente link:

https://drive.google.com/drive/folders/0B-2HrrU-WoSgRUJmcjdJWlptazg?usp=sharing

Si bien es posible instalar Visual Studio 2017 en sistemas operativos anteriores a Windows 10, si como en este caso queremos desarrollar en la UWP, las versiones anteriores a Windows 10 no nos servirán.

El código de lectura del puerto serie es una modificación del que reside en el repositorio de ejemplos para Windows 10 IoT Core en GitHub cuya url es:

https://github.com/ms-iot/samples/tree/develop/SerialUART/CS

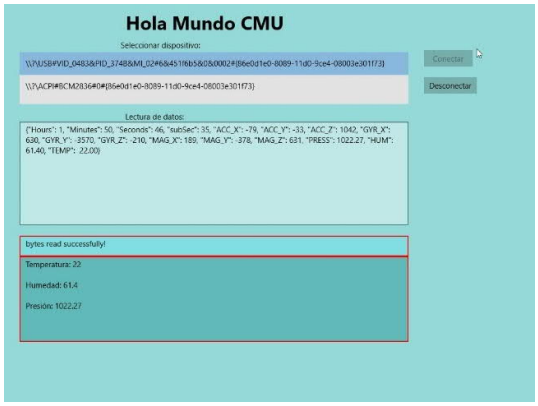
En el archivo MainPage.xaml.cs, el método asíncrono ReadAsync espera los datos enviados por la MCU ARM y que recibe el dispositivo serie. Una vez obtenido el string, lo deserializa y

convierte en un objeto de la clase Mediciones, para llevar a cabo un proceso de calibración de los mismos. Luego vuelve a serializarlos y los envía hacia dos destinos:

- El listener TCP/IP que reside en la aplicación Unity
- El web server NodeJS en Heroku, mediante el uno de sus servicios web: /sensores/store

Esta aplicación puede iniciarse desde el Apps Manager del portal mencionado en el punto Herramientas para trabajar con Windows 10 IoT Core y ser controlada tanto desde el monitor HDMI conectado a la Raspberry como desde la aplicación de escritorio remoto a la que se accede desde la IoT Core Dashboard, también mencionada en el mismo punto.

Atención a este dato: Visual Studio 2017 tiene un error conocido en el Manifest Designer (el editor visual de los archivos appxmanifest) que afecta a la capacidad de comunicación en serie. Si su appxmanifest agrega la capacidad serialcommunication, la modificación de su appxmanifest con el diseñador corromperá su appxmanifest (el dispositivo xml se perderá). Puede solucionar este problema manualmente editando appxmanifest haciendo clic con el botón derecho del ratón en appxmanifest y seleccionando Ver código en el menú contextual.



Aplicación Unity

La aplicación Unity consta de un listener TCP/IP que queda a la espera de la conexión de la aplicación UWP en el Raspberry.

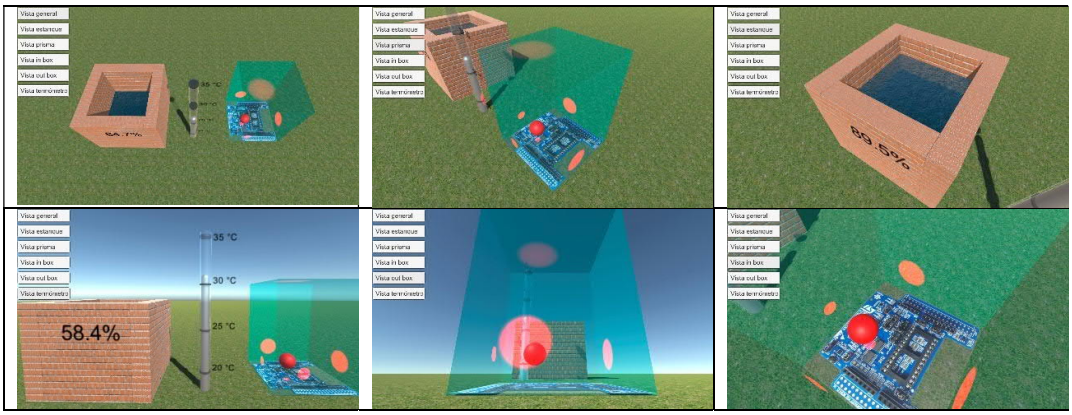
Cuando se establece la conexión y comienza a recibir datos de los sensores, con los datos del acelerómetro 3D, del termómetro y del sensor de humedad relativa ambiente, anima Gameobjects representado los parámetros recibidos.

Un vídeo donde se ve la aplicación funcionando junto a una breve explicación del código y algunos aspectos interesantes de la configuración de los gameobjects, puede encontrarse en youtube en esta url:

https://www.youtube.com/watch?v=zI4NWD_VVf4

También puede descargarse el proyecto Unity completo y el mismo vídeo en formato mp4 desde la carpeta Unity accesible en esta url:

https://drive.google.com/drive/folders/0B-2HrrU-WoSgRUJmcjdJWlptazg?usp=sharing



Servicios y aplicación Web en Heroku

El código fuente de este componente del proyecto, pueden descargarse desde la carpeta NodeJS en Heroku en la siguiente url:

https://drive.google.com/drive/folders/0B-2HrrU-WoSgRUJmcjdJWlptazg?usp=sharing

Esta aplicación web corre sobre NodeJS. El desarrollo de la misma lo realicé primero en forma local, con NodeJS instalado en mi ordenador.

La instalación de Node puede realizarse desde https://nodejs.org/es/.

Para instalar paquetes adicionales de Node, es necesario el Node Package Manage npm. Al instalar node, automáticamente se instala nmp en el ordenador.

El despliegue en Heroku requiere los siguientes pasos:

1. Crear una cuenta en Heroku: https://www.heroku.com
2. Instalar Heroku CLI
3. Tener la aplicación node funcionando y tan depurada como sea posible en local
4. Adaptar la aplicación para su despliegue en Heroku
5. Realizar la versión git y desplegarla en Heroku

Para el punto 1, recordar que Heroku tiene cuentas gratuitas y de pago. Puede comenzarse tranquilamente con la gratuita y escalarla en caso de acuerdo a la necesidad.

https://devcenter.heroku.com/articles/heroku-cli

Adaptar la aplicación para su despliegue en Heroku consistió en la modificación de algunos archivos de configuración, para "hacerla consciente" del entorno en el que se ejecuta.

Trabajando en local, en el ordenador propio, la base de datos utilizada para almacenar las mediciones de los sensores es sqlite, mientras que en Heroku, utilicé Postgres.

Esto requiere que el archivo package.json utilice la dependencia a sqlite en desarrollo y la dependencia a Postgres en producción. Vemos a continuación como se indica esto en dicho archivo:

```
{
	"name": "WebHolaMundoCMU",
	"version": "0.0.0",
	"private": true,
	"scripts": {
		"start": "node ./bin/www"
	},
	"devDependencies": {
		"sqlite3": "^3.1.9"
	},
	"dependencies": {
		"body-parser": "~1.17.1",
		"cookie-parser": "~1.4.3",
		"debug": "~2.6.3",
		"ejs": "~2.5.6",
		"express": "~4.15.2",
		"express-partials": "^0.3.0",
		"jquery": "^3.2.1",
		"morgan": "~1.8.1",
		"pg": "^7.2.0",
		"request": "^2.81.0",
		"sequelize": "^4.7.5",
		"serve-favicon": "~2.4.2"
	}
}
```

En las dependencias de desarrollo se especifica sqlite en su versión 3.1.9 o mayor, mientras que para producción, la base de datos es Postgres 7.2.0 o mayor, que es la elegí instalar como recurso en Heroku.

El paquete sequelize (instalar utilizando npm install --save sequelize), nos provee de una abstracción para que el código de acceso y manipulación de la base de datos sea independiente de la que se utilice. Es interesante pegarle un vistazo al archivo \models\models.js, para ver como se asignan los diversos parámetros que requiere la creación del objeto sequelize, en función de la variable de entorno de Heroku process.env.DATABASE_URL, cuando se ejecuta en producción.

Sequelize: http://docs.sequelizejs.com/manual/installation/getting-started

Si el archivo Procfile no existe, es necesario crearlo y debe tener el siguiente contenido:

web: node ./bin/www

Recordar que, a partir de estos cambios, para ejecutar la aplicación en modo local debe invocarse desde la línea de comandos la siguiente sentencia: heroku local

Finalmente, el despliegue en Heroku se realiza utilizando git. Es posible hacerlo desde un repositorio local, como así también desde el repositorio de nuestra aplicación e github.

Los comandos de creación de un repositorio git local, resumidamente y ya parados en la carpeta raiz del proyecto, son estos:

```
git init
git add .
git commit -m “escribir aquí un texto que describe el commit”
```

luego nos conectamos a Heroku, con el siguiente comando:

```
heroku login
```

luego de ingresar nuestras credenciales estaremos conectados a nuestra cuenta en Heroku. Ahora configuramos el repositorio remoto git como heroku en nuestro ordenador:

```
heroku create
```

y transferimos nuestra aplicación al repositorio remoto en heroku con:

```
git push heroku master
```

Si no hay errores, nuestra aplicación estará accesible desde la web. Podemos obtener su url en el dashboard de Heroku. Para “Hola Mundo CMU” la url es:

https://holamundocmu.herokuapp.com/

PhoneGap: aplicación Android en Google Play

El código fuente y archivos de comando .bat de ayuda, pueden descargarse desde la carpeta PhoneGap en la siguiente url:

https://drive.google.com/drive/folders/0B-2HrrU-WoSgRUJmcjdJWlptazg?usp=sharing

El desarrollo de la app Android mediante PhoneGap lo lleve a cabo con la herramienta cliente de PhoneGap: PhoneGap CLI.

Su instalación se realiza por medio de npm, con el siguiente comando:

```
npm install -g phonegap
```

Una vez desarrollada la aplicación web que se transformará en una webapp Android, para obtener el apk se siguen estos pasos:

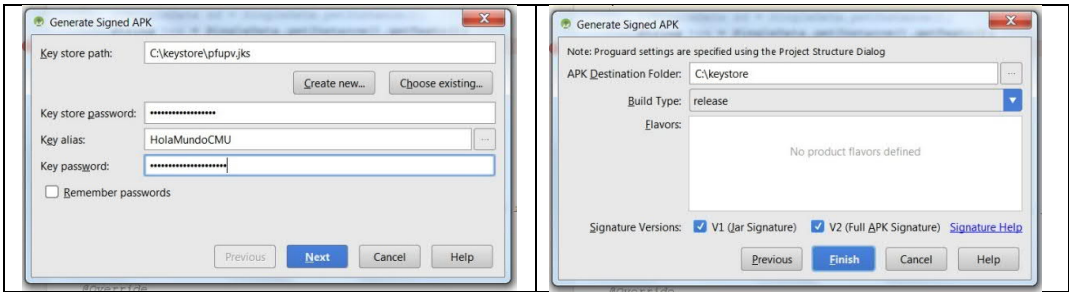
1. Creación del esqueleto de la aplicación PhoneGap:
    a. phonegap create hmCMU --id "ar.com.hmcmu" --name "HolaMundoCMU"
2. Agregar las plataformas en las que se desea generar la webapp, en este caso, Android
    a. phonegap platform add android
3. Copiar la aplicación web bajo la carpeta www de la estructura generada por phonegap en el paso 1
4. Construir la aplicación nativa
    a. phonegap build android --release

nota: en el paso 4, es MUY importante el parámetro release. Si no se incluye, será imposible publicar la aplicación en Google Play.

El siguiente paso consiste en firmar la aplicación. Este paso es obligatorio si se desea publicar la webapp en Google Play.

Para llevar a cabo la firma, primero hay que generar las claves que se utilizarán para firmarla y guardarlas en un almacén de claves. Esto puede hacerse cómodamente desde Android Studio, desde la opción de menú Build-Generate Signed APK, en dos pasos simples:

- Elegir las claves
- Elegir la carpeta donde se almacenarán



Luego, con esas claves, podemos firmar la webapp usando una aplicación de línea de comandos provista por Google llamada apksigner. Puede descargarse desde este link:

https://developer.android.com/studio/command-line/apksigner.html

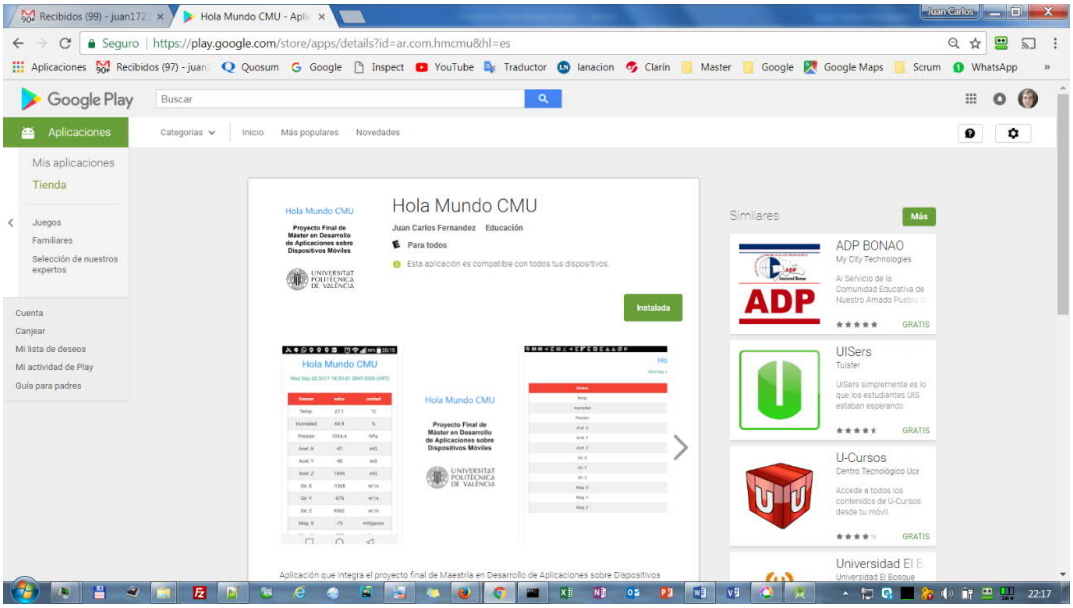
esta herramienta es parte del Android SDK Build Tools desde la versión 24.0.3.

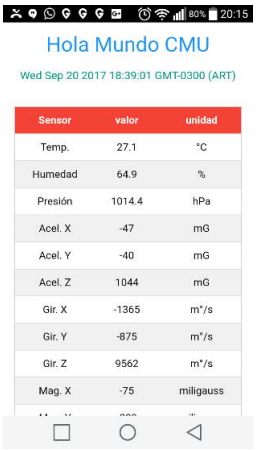
Un ejemplo de la línea de comando para firmar el apk, es el que sigue:

```
apksigner.bat sign --ks path_clave\archivo_con_claves.jks --out
path_destino_apk_firmado\HolaMundoCMU.apk --v2-signing-enabled true --verbose
path_del_apk_sin_firma\android-release-unsigned.apk
```

El proceso de publicación de la aplicación es el explicado en la asignatura Android. Básicamente es necesario tener una cuenta en la Google Play Console. Este proceso consiste en acceder con la cuenta Google propia, aceptar el acuerdo para desarrolladores, pagar la tarifa de registro y completar los datos faltantes.

Luego, publicar la aplicación es un proceso algo tedioso, pero guiado: se requieren varias imágenes de pantalla e íconos de las vistas de la app, en distintos tamaños, calificar la aplicación de acuerdo a su contenido y luego las publicaciones pueden ser de test, tanto alfa como beta y en grupos de prueba restringidos para luego pasar a versiones de producción.





Conclusiones

Grado de cumplimiento de los objetivos planteados

Considero que el objetivo humilde de allanar el primer paso en muchas líneas abiertas del Master fue alcanzado en gran medida. En concreto, algunos de esos pasos, en esas líneas fueron estos:

- Programar una MCU ARM haciendo interfaz con sensores y obteniendo datos físicos.
- Realizar una aplicación que se ejecute en “open hardware” aunque estrictamente sea un tema de debate si Raspberry Pi lo es o no.
- Realizar la instalación y los primeros pasos en un sistema operativo diseñado para IoT, como Windows 10 IoT Core, utilizando a su vez lo aprendido en la asignatura Programación en Windows.
- Utilizar sockets en Unity, para así enviarle datos desde el mundo real y mediante su enorme capacidad para modelar y animar ambientes en 3D, hacer experiencia en representar datos complejos de un modo amigable pero contundente.
- Lograr la publicación en Google Play de una aplicación realizada en PhoneGap, tarea que, si bien no es ciencia espacial, requiere una serie de pasos que por alguna razón es difícil encontrarlos juntos.
- Utilizar servicios web en la PaaS Heroku.
- Dejar una documentación del camino recorrido tal que sea útil para mí en futuros desarrollos y compartirla para intercambiar experiencia y mejorar procedimientos.

Líneas abiertas

- Utilizar PhoneGap para desplegar la misma aplicación en los almacenes de aplicaciones de Windows y de Apple.
- Procesar los datos de los sensores en su origen, utilizando la MCU ARM, y enviar datos "digeridos" al resto de la cadena del proyecto. Esto no es especialmente útil en el sensor de temperatura, pero por ejemplo en el caso del giróscopo, para tener una mejor aproximación de la posición angular de un cuerpo físico, procesar muchos más datos por unidad de tiempo lograría paro no perder datos intermedios necesarios, dada la naturaleza de los cambios de la aceleración angular.
- Los servicios web que expone la aplicación en Heroku, responden a quien hizo la petición. Por ejemplo, en el caso de la ruta /sensores/store, con el verbo POST, utilizada por la aplicación UWP residente en Raspberry PI, una vez procesado el request, se envía un response al Raspberry PI. Sin embargo, hay o puede haber muchos otros clientes interesados en conocer la novedad del arribo de una nueva medición. Actualmente eso está resuelto por polling: periódicamente, cada cliente realiza un petición GET a la ruta /sensores/recall y obtiene los datos nuevos. Me gustaría que, en lugar de esa estrategia, los clientes interesados se registraran en la aplicación en Heroku y fueran notificados cuando (y solo cuando) alguna novedad sucede. Creo que quedó esa línea abierta para el uso de notificaciones push.
- Implementar el envío de órdenes, tanto desde la app móvil como desde la página web alojada en Heroku, para accionar actuadores comandados desde el circuito impreso con la MCU ARM y así poder activar o desactivar luces, motores, etc.

Consideraciones personales

El trabajo en este proyecto me fue útil y enriquecedor. Me hizo tomar conciencia de muchas cosas que ignoraba o conocía más superficialmente y me permitió explorar áreas que de otra manera por el trajín diario no hubiera transitado.

Agradezco sinceramente a todas las personas que me ayudaron en el trayecto: profesores, alumnos y al personal administrativo que resolvió amablemente problemas que por la distancia se vuelven más complejos.

Especialmente agradezco la pasión y la seriedad para enseñar que encontré muy seguido a lo largo de toda la cursada.

Saludo cordiales!
Juan Carlos
juan1721@gmail.com

Anexos

Listado de fuentes entregadas

El código fuente de las aplicaciones, video explicativo de la aplicación Unity, transparencias de la lectura del proyecto y esta misma memoria, pueden ser descargados desde las carpetas accesibles en este link:

https://drive.google.com/drive/folders/0B-2HrrU-WoSgRUJmcjdJWlptazg

El listado es el siguiente:

1. Carpeta ARM y MEMS
    a. STM32F401RE-Nucleo.rar
2. Carpeta Memoria
    a. Memoria Hola Mundo CMU.pdf
3. NodeJS en Heroku
    a. WebHolaMundoCMU.rar
4. Carpeta PhoneGap
    a. hmCMU.rar
    b. hmCMUDeploy.bat
    c. hmCMUFull.bat
    d. hmCMUpg.rar
    e. hmCMUPhoneGap.bat
    f. HolaMundoCMU.apk
5. Carpeta Unity
    a. ProyectoFinal_UPV_2017.mp4
    b. ProyectoFinalCMU.rar
6. Carpeta Windows IoT Core y Raspberry
    a. ProyectoCMU.rar