MAY 6, 2016

TRABAJO PRÁCTICO INTEGRADOR

TEMA: “Verificador Remoto de Temperatura”

PRIMERA ENTREGA – VERSIÓN 1.1

Curso: 5K4

Integrantes:

Andrada, Emiliano	Legajo: 59809
Armella, Darío	Legajo: 45570
González Martínez, Lucas	Legajo: 59919

Docente:

Frías, Pablo Sebastián

Romani, Germán Ariel

UTN FRC

ARQUITECTURA DE SOFTWARE

HISTORIAL DE REVISIÓN

Fecha	Versión	Descripción	Autor
16/04/2016	1.0	Versión Inicial	Andrada - González - Armella
06/05/2016	1.1	Se agrega descripción de arquitectura, mención del repositorio, objetivos generales y específicos, implementación de funciones principales, diagramas de arquitectura y proyecciones del trabajo.	Andrada - González - Armella

ÍNDICE

Contents

INTRODUCCIÓN

El presente informe describe la realización del **trabajo práctico integrador** de la cátedra de Arquitectura de Software. Enfocados principalmente en el desarrollo de la arquitectura a construir, este documento se estructura en las siguientes partes:

- **Descripción del dominio**: explicación breve del escenario sobre el cual será desarrollada la arquitectura. En este caso, se deberá diseñar un software que permita la verificación de la temperatura en forma remota.
- **Especificación de Requerimientos**: se describirán en forma breve cuáles son las historias de usuario planteadas por el equipo, qué se espera de cada una de ellas y cómo deberá diseñarse la arquitectura en función de estos requisitos. Además, se armará un backlog especificando la prioridad de cada US y los Requerimientos No Funcionales.
- **Objetivos generales y específicos**: se resumen cuáles son los objetivos generales que se persiguen con el desarrollo de esta actividad práctica, en función del software que estemos desarrollo. También se agregan las actividades u objetivos específicos a cumplir para complementar los objetivos globales. Intentaremos dar respuesta a estas preguntas:
    - ¿Qué hacemos?
    - ¿Por qué lo hacemos?
    - ¿Qué esperamos aprender?
    - ¿Cómo lo hacemos?
- **Descripción de la arquitectura**: una primera versión de la arquitectura será expuesta en este documento en forma de diagramas y explicaciones que serán los cimientos sobre los cuáles será implementado el código.
- **Implementación**: la implementación de lo que se ha desarrollado hasta el momento se encuentra en un repositorio externo. Se dará información de acceso al repositorio, y además, especificaciones sobre qué lenguaje de programación y tecnologías se han utilizado.
- **Proyección**: ¿qué esperamos de este proyecto y qué otros usos le podemos dar fuera de lo que ya conocemos?
- **Conclusiones:** entendemos que es importante dejar en claro cuál es la importancia de diseñar la arquitectura dentro de cualquier proyecto de software.

DESCRIPCIÓN DEL DOMINIO

Historia

Un nuevo emprendimiento tecnológico está investigando acerca de nuevas herramientas para tomar mediciones de ambiente y poder extrapolar los resultados a nuevos proyectos de mercado.

La primera etapa consiste en lograr visualizar de manera remota la temperatura de un lugar específico. El Cliente está interesado en los siguientes requerimientos:

- Poder medir la temperatura con una placa compatible con Arduino.
- Tener un servidor Web en la placa, que permita recibir peticiones externas para conocer la temperatura y visualizar un gráfico a lo largo del tiempo. Para esto se deberán tomar mediciones cada 5 segundos.

Actividades

1. Código fuente versionado en GitHub
2. Documentación de Arquitectura
    a. Definición de Requerimientos y Supuestos
    b. Definición de Requerimientos no funcionales
    c. Diagrama de Arquitectura
    d. Dependencias

ESPECIFICACIÓN DE REQUERIMIENTOS

Historias de Usuario

A continuación, vamos a describir todas las historias de usuario con las que vamos a trabajar a lo largo del proyecto. El objetivo es, además de cumplir con la funcionalidad indicada en la historia principal, agregar otras historias adicionales que permitan expandir la descripción de la arquitectura y, por consiguiente, del software.

	Medición de temperatura	**Criterios de aceptación**	**Prioridad**
1	Como usuario quiero poder tomar la temperatura para poder visualizarla en un tablero de control.	Medición en forma remota	Alta
		Placa compatible con Arduino	
		Medición cada 5 segundos	
		Servidor web instalado en la placa	
		Acceso web al tablero de control	

	Medición de humedad	**Criterios de aceptación**	**Prioridad**
2	Como usuario quiero poder tomar la humedad para poder visualizarla en un tablero de control.	Medición en forma remota	Baja
		Placa compatible con Arduino	
		Medición cada 5 segundos	
		Servidor web instalado en la placa	
		Acceso web al tablero de control	

Esta funcionalidad se complementa con la indicada en la US Nº 1. Sin embargo, se asigna prioridad baja, ya que se incluirá en la lista de deseos. Se intentará describir una arquitectura que soporte la medición de temperatura, y que sea lo suficientemente flexible y robusta para agregar funciones como la medición de otros parámetros, en este caso, la humedad.

	Histórico de temperaturas	**Criterios de aceptación**	**Prioridad**
3	Como usuario quiero poder guardar una medición de temperatura para poder armar un historial de lecturas.	Al menos 2 o más temperaturas almacenadas	Media

	US N° 4: Histórico de humedad	Criterios de aceptación	Prioridad
4	Como usuario quiero poder guardar una medición de humedad para poder armar un historial de lecturas.	Al menos 2 o más porcentajes de humedad almacenados	Baja

Esta historia también formará parte de futuros reportes y/o estadísticas del software a desarrollar, ya que no podremos preparar un historial de un parámetro que aún nuestro producto no está preparado para recibir.

	Evolución de temperatura	Criterios de aceptación	Prioridad
5	Como usuario quiero visualizar la evolución de la temperatura a lo largo de un período de tiempo.	Al menos dos valores de temperaturas almacenados en el período	Alta

Las gráficas deberán reflejar cómo se comportó la temperatura a lo largo de un período de tiempo ingresado por parámetro. La gráfica a utilizar puede ser de barras.

	Notificaciones push	Criterios de aceptación	Comentarios
6	Como usuario quiero recibir notificaciones de lecturas en los sensores para estar informado todo el tiempo.	Valor de temperatura válido.	Media

Las notificaciones deberán ser enviadas vía mail, a la casilla que el usuario deje configurada previamente. Para ello requeriremos de **tecnología push,** la cual es una forma de comunicación a través de internet en la que la petición de envío tiene origen en el servidor, que en este caso será nuestro **web service.** Se podría también diseñar algún esquema ***"publicador/suscriptor"*** para que el usuario decida si desea o no recibir estas notificaciones.

REQUERIMIENTOS NO FUNCIONALES

Definimos un conjunto de requerimientos **no asociados** a la funcionalidad del software, pero que deberán tenerse en cuenta para diseñar la arquitectura. El sistema deberá estar preparado estructuralmente para poder soportar los siguientes requisitos:

Nº	Nombre	Descripción	Categoría	Significativo	Prioridad	Decisión Arquitectónica
1	Conexión con el Arduino	Se deberá instalar un web service en la placa compatible con Arduino.	Restricciones Técnicas	Sí	Alta	Para poder hacer las primeras mediciones es necesario realizar la instalación. El lenguaje de programación debe ser compatible con este escenario.
2	Notificaciones vía mail de temperatura	Se permitirá al usuario suscribirse a un sistema de notificaciones que le enviará cada 30 minutos la última toma de temperatura.	Producto	Sí	Alta	Se deberá desarrollar algún módulo que realice el envío de notificaciones y que se conecte al servidor de correo electrónico
3	Entorno Web	Todas las funcionalidades del sistema se integrarán en una página web.	Producto	Sí	Alta	Se deberá preparar una arquitectura que permita la conexión al web service, y utilizar un lenguaje de programación acorde.
4	Colores en gráficos	Evitar colores fluorescentes, llamativos, o que perturben al usuario.	Producto	No	Baja	
5	Tecnología websocket	Implementar sobre un web service que tenga disponibilidad para esta tecnología de comunicación.	Restricciones Técnicas	Sí	Media	Se restringen los navegadores y las versiones disponibles que se comunicarán con el web service. El expolorador a utilizar tiene que tener incorporada esta tecnología.

6	Bases de datos	Almacenamiento persistente en base de datos con motor SQL Server 2008.	Restricciones Técnicas	Sí	Alta	Se deberá describir la arquitectura de manera tal que soporte la conexión con este DBMS. Además, los lenguajes de programación a utilizar deberán ser compatibles
7	Diseño responsivo	El diseño de la página web accesible desde cualquier dispositivo deberá ser responsivo, es decir, que deberá adaptarse a la pantalla.	Producto	Sí	Baja	Se deberá utilizar algún framework que permita la realización del diseño responsivo de una manera más sencilla, tal como Bootstrap.
8	Extracción de datos	El mecanismo de búsqueda y carga de los datos deberá flexibilizarse en caso de cambiar el Arduino en el futuro.	Producto	Sí	Media	Se deberá implementar algún framework que permita estandarizar el proceso de materialización y desmaterialización de forma tal que podamos extenderlo a otras marcas de Arduino.

OBJETIVOS GENERALES Y ESPECÍFICOS

A semejanza de los planos de un edificio o construcción, la arquitectura de un software indica la estructura, funcionamiento e interacción entre las partes del software. Es, en definitiva, el diseño de más **alto nivel** de la estructura de un sistema.

Por tal definición, podemos definir los siguientes objetivos generales en torno a nuestro trabajo integrador:

1. Conocer los diferentes estilos arquitectónicos existentes que podemos aplicar al diseño del software “Verificador de temperatura remota”

2. Conocer y aplicar los patrones de diseño que sean necesarios para el desarrollo de alguna funcionalidad del software que vayamos a desarrollar.

3. Entender la importancia y el significado de la Arquitectura en el contexto del desarrollo de software.

4. Expandir esta importancia al desarrollo e implementación de otros sistemas

Algunas tareas u objetivos específicos planteados por el equipo para alcanzar las metas globales:

- ✓ Aplicación de los estilos arquitectónicos aprendidos en clase.
- ✓ Investigación y desarrollo de tecnologías necesarias para conectar un Arduino a nuestro software e instalar en el los componentes necesarios.
- ✓ Diseñar componentes que tengan una granularidad baja, sean cohesivos y de esta forma, re utilizables en el futuro.
- ✓ Crear relaciones y dependencias que nos permitan identificar cómo se van a comunicar los módulos de nuestro sistema, y qué rol cumple cada uno de ellos en la comunicación.
- ✓ Aplicar algún patrón que resuelva alguna situación que podamos contextualizar en el dominio de nuestro problema. **Por ejemplo**: la suscripción de uno o más usuarios a recibir tomas de temperatura cada 30 minutos.

DESCRIPCIÓN DE LA ARQUITECTURA

En este segmento vamos a mostrar cómo se organizará nuestro software en término de capas y también en función de los componentes más importantes. La comunicación de las capas y los componentes define todas las funcionalidades del sistema, y es importante que esta información comunique de manera eficiente a los interesados **cómo será el desempeño del sistema, cuáles serán sus funciones y módulos más importantes, y qué recursos externos necesitará para poder funcionar.**

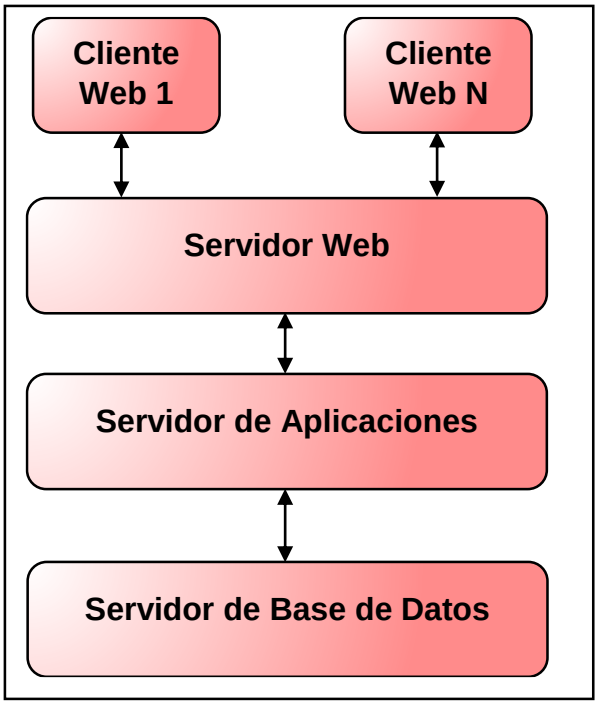
Patrón N-Layer (Client-Server)

Capa de Presentación

Capa de Servicios Web

Capa de Lógica de Negocio

Capa de Acceso a Datos

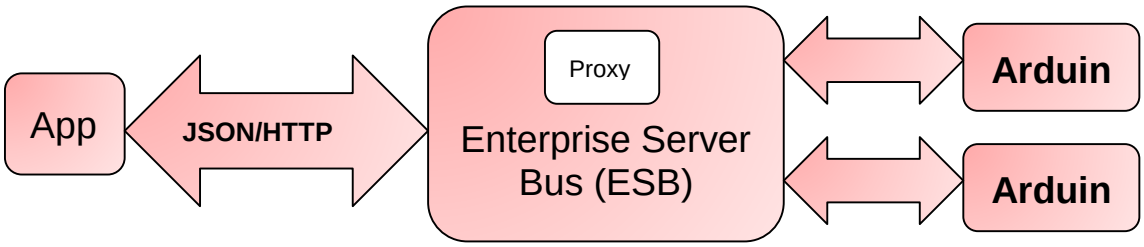


Este patrón arquitectónico aplicado en el contexto del **"Verificador Remoto de Temperatura"** es importante para definir la distribución lógica de las capas en términos de la estructuración del código. Sin embargo, cabe aclarar que esto **no responde** a una estructura **N-Tier**, ya que sólo tenemos separado físicamente el web-service, pero el motor de base de datos y el servidor de aplicaciones van a correr físicamente en el mismo ordenador. Lo importante de este patrón, es que a nivel de código se realiza una separación de intereses en términos de lo que cada uno provee y necesita.

Patrón ESB: Bus de Servicio Empresarial

Es un modelo de arquitectura que gestiona la comunicación entre **servicios web**. Es un componente fundamental de la **Arquitectura Orientada a Servicios.**

La palabra **bus** viene del bus que transporta los bits entre los distintos dispositivos. Si bien nuestro software no es empresarial, el Arduino y la conexión al mismo son dos puntos muy importantes a tener en cuenta en el planteamiento de nuestra arquitectura y que hacen muy útil la aplicación del patrón.



Motivaciones:

- Podemos conectar a muchas placas Arduino pero seguir teniendo un solo punto de entrada que pueda encaminar las solicitudes a la placa que corresponde.
- Es posible que necesitemos transformar el formato de los datos captados por el Arduino a varios formatos, no solo web. **En un futuro se pueden desarrollar aplicaciones móviles por ejemplo.**
- Queremos implementar lógica de autenticación para que solo clientes autorizados tengan acceso a la información.
- No queremos exponer directamente a internet a la placa Arduino.

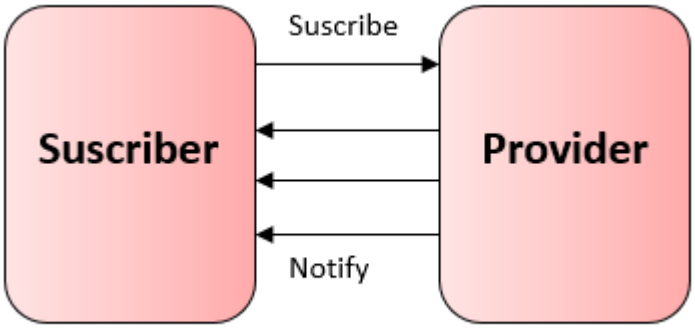
Nota: El proxy dentro del ESB permite recibir las solicitudes de los clientes y enviarlos a las placas correspondientes y luego leer la respuesta y mostrarla a los clientes.

Patrón Suscribe-Notify

A este patrón lo utilizamos para que los clientes se puedan comunicar con el servidor y así obtener los datos de las mediciones. Lo aplicamos con la librería Signal-R.

Nos permite una comunicación asíncrona entre el servidor y los clientes, en ambos sentidos. El cliente indica al servidor los tópicos (temperatura, humedad, etc.) a los que desea suscribirse mediante una llamada de suscripción al servidor. Este

almacena las suscripciones junto con las direcciones de los clientes que quieren ser notificados y les envía las notificaciones cuando estén listas. Mientras los clientes esperan ser notificados pueden continuar con otras tareas y solo procesan la notificación una vez que la reciben.



Otra gran ventaja que simplifica el trabajo al servidor es que las notificaciones se envían por tópico. Todas las notificaciones del mismo tópico se envían a todos los clientes al mismo tiempo (en realidad con muy poca latencia entre el primer y último cliente en ser notificados).

La desventaja es que el servidor debe mantener **registro de los suscriptores que tiene y del tipo de suscripción si existen diferentes opciones.**

Patrón ETL (Extraer, transformar y cargar)

Con ETL dividimos el uso de la base de datos en tres fases:

- ✓ Extracción de datos de fuentes de datos, ya sean homogéneas o heterogéneas.
- ✓ Transformación de los datos para almacenarlos en el formato o estructura correcta para los propósitos de consulta y análisis.
- ✓ Carga de los datos a la base de datos.

La primera fase suele tardar mucho tiempo, por eso deberíamos ejecutar cada una de estas fases en paralelo. Mientras los datos se van extrayendo, un proceso de transformación se va ejecutando a la par. Procesa los datos ya capturados y los prepara para almacenarlos a la base de datos. En cuanto ya se encuentren datos disponibles para cargar, el proceso de carga inicia sin esperar a que finalicen las fases anteriores.

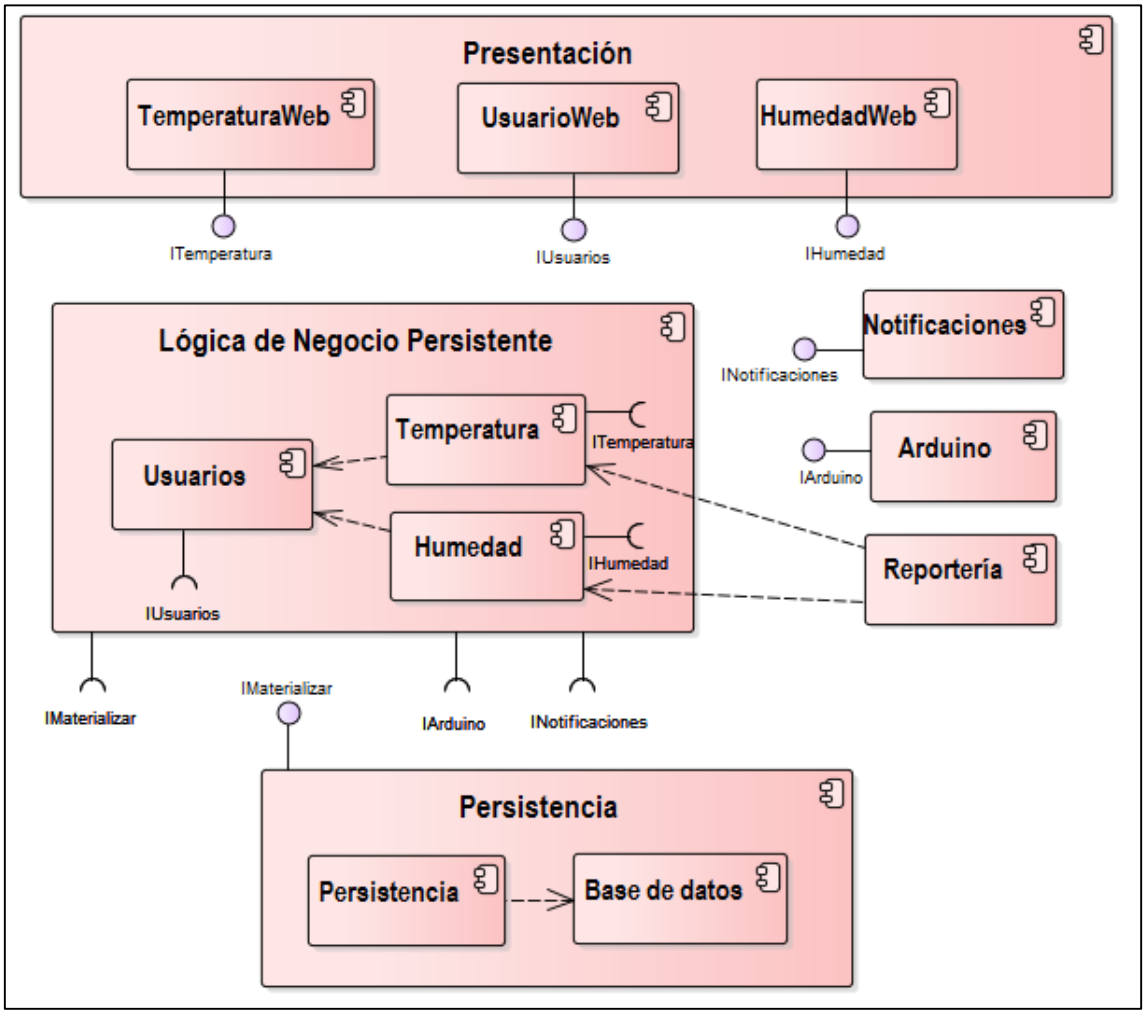
La parte más importante es la primera porque es la que obtiene los datos de forma correcta para que las siguientes fases puedan completarse exitosamente. La extracción de los datos nos ofrece la **ventaja de que se pueden extraer datos de diferentes fuentes**, cada una con su propio formato. Por ahora tenemos solo un **arduino**, por ende una sola marca, pero podríamos recolectar los datos de diferentes placas, diferentes marcas, versiones, desarrolladores donde cada uno de estos aplicó el formato que le pareció conveniente. En esta etapa debemos poder llevar todos los datos al formato elegido por nosotros. El objetivo de esta fase es obtener los datos en un único formato para que el proceso de transformación pueda proceder. Para conseguir que los datos cumplan con cierto formato primero se valida que los datos tengan valores esperados dentro de cierto rango. Si esta validación falla, los datos no se incluirán.

En el proceso de transformación se realiza la limpieza de los datos seleccionando solo aquellos que nos interesa para nuestro sistema. También se puede realizar una traducción de códigos u obtener valores a partir de un cálculo, entre otras.

En la carga se almacena los datos ya transformados a la base de datos.

ETL:

DISEÑO DE LA ARQUITECTURA – VISTA ARQUITECTÓNICA DE SUBSISTEMAS E INTERFACES



La arquitectura de software describe cómo un sistema es descompuesto en componentes, cómo éstos son interconectados y la manera en que éstos se comunican e interactúan. Existen diferentes vistas para representar un **comportamiento particular del sistema.** En este caso, la vista de subsistemas e interfaces apoya principalmente a los requisitos funcionales, pero tampoco olvida a los no funcionales.

El sistema se descompondrá en una serie de **abstracciones primarias**, tomadas principalmente del dominio del problema:

- ✓ **Temperatura**: todas las clases y objetos instancias de clases que recolecten datos relacionados a la entidad temperatura. El verificador remoto tiene como función principal tomar la temperatura y luego enviarlo a través de un server a la base de datos.

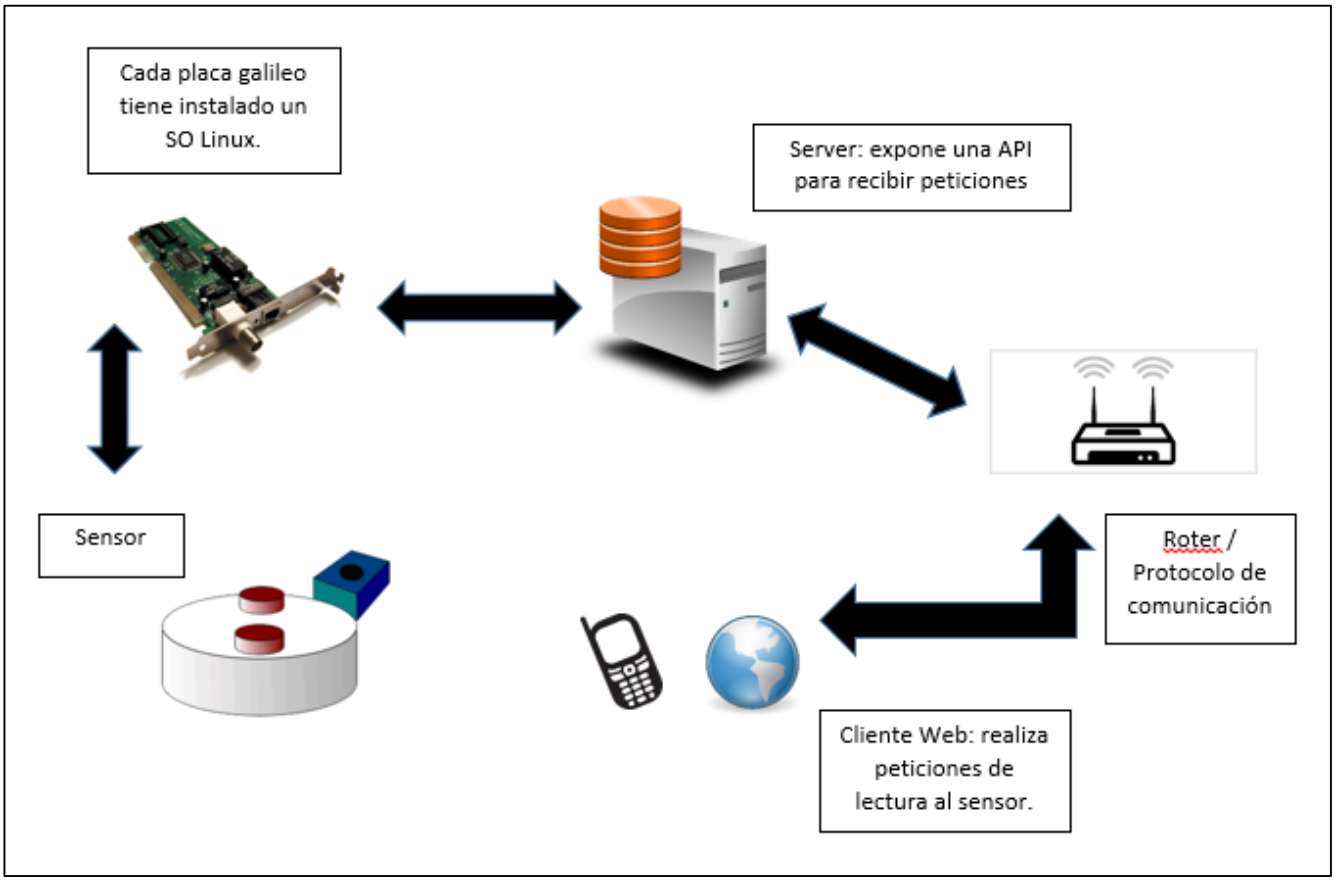
- ✓ **Humedad**: representa lo mismo que temperatura, pero en este caso, el parámetro a utilizar es la humedad. Al ser un dominio acotado, se hizo la diferenciación de las entidades en dos componentes distintos.
- ✓ **Usuarios**: representan las entidades asociadas a los datos de quienes interactúan con el sistema.

También podemos encontrar otro tipo de abstracciones que resuelvan los requerimientos **no funcionales o no persistentes:**

- ✓ **Notificaciones:** porción de software que permitirá la comunicación de nuestro sistema con un servidor de correo para enviar datos de temperatura a los interesados. Este componente contendrá la funcionalidad del patrón **Suscribe-Notify.**
- ✓ **Arduino:** porción de software que permitirá la comunicación de nuestro sistema con el web service y la placa de Arduino durante el intercambio de datos. Este componente contendrá la funcionalidad del patrón **ESB.**
- ✓ **Reportería:** procedimientos almacenados o porción de código que resuelva la generación de informes y reportes de interés. Esta información no es persistente.
- ✓ **Persistencia:** resuelve el requerimiento no funcional de bases de datos, el cual solicita que la comunicación sea lo suficientemente flexible para admitir conexiones a otras marcas de Arduino en el futuro. Este componente contendrá la funcionalidad del patrón **ETL.**

DIAGRAMA DE LA ARQUITECTURA – DESPLIEGUE

A continuación, mostramos cómo se distribuyen los diferentes elementos de hardware que componen la arquitectura del sistema. En cada uno de los nodos, **se instalará uno o más componentes de software a desarrollar:**



IMPLEMENTACIÓN

El código asociado a este proyecto se encuentra en un repositorio externo accesible mediante este **link**.

Los lenguajes de programación elegidos para el desarrollo son:

- Lenguaje JavaScript, entorno node.js (lado del server)
- Una app con framework MVC (lado del cliente)

El objetivo es mantener todo el código lo mayor comentado posible para que sea más fácil modificar luego, y que sea entendible para todos los miembros del equipo.

En principio, los esfuerzos para codificar se centraron en la configuración del web service y la conexión al Arduino para realizar algunas de las funciones principales. Se espera en un futuro ampliar la codificación para tener disponible más código que cubra un mayor porcentaje de funcionalidad. De igual manera, ha sido interesante desde el punto de vista arquitectónico investigar cómo llevar a cabo la conexión, la configuración y la aplicación de los patrones descriptos anteriormente.

PROYECCIÓN

Este proyecto se podrá utilizar para fabricar ollas a presión inteligentes. Con el uso de estas ollas, el usuario ingresa lo que desea cocinar y cantidad, tanto de la comida en sí como del agua, y el sistema tendrá cargado la cantidad de calorías que deben ser entregadas a la olla para que esa comida ya esté cocinada, el tamaño de los cortes de comida y calores específicos. Como a cada persona le gusta la comida un poco más o menos cocida, el usuario tendrá la opción de calificar el resultado indicando cómo hubiese sido mejor para que la olla aprenda sus preferencias y así ofrecer mejores platos.

Después de que el usuario prenda la hornalla, la placa calculará el tiempo que requerirá para terminar de cocinar para que el usuario pueda desentenderse completamente hasta que le llegue la notificación de que ya debe apagar el fuego.

Optamos por una olla a presión para que el agua no se evapore y quitarle al usuario la tarea de revisar la olla o de tener que reponer agua en el caso de que el sistema notificara esta situación.

En la base de datos guardaremos la cantidad de calorías a entregar a partir de que el agua alcance cierta temperatura para que no dependa de la temperatura del agua antes de prender la hornalla. Claramente esta temperatura debería ser mayor a la temperatura ambiente pero lo más lejos del punto de ebullición del agua para poder estimar más rápido el tiempo de cocción necesario.

La placa tendrá la tarea de calcular las calorías absorbidas por la olla. Para ello requiere almacenar la temperatura inicial y de ir tomando temperaturas para calcular la variación de esta. A esta diferencia se la multiplica con la suma de los productos calor específico del agua y su cantidad y calor específico de la comida y su masa.

$$Q = \Delta t \,.\, (m_{H_2O} \,.\, Ce_{H_2O} + \sum m_{ingrediente} \,.\, Ce_{\,ingrediente})$$

La temperatura inicial de cada parte que se ponga en la olla debe estar cerca de la temperatura del agua para obtener un mejor resultado. Al ser poca la diferencia entre las temperaturas iniciales podemos obviarlo sin cambios significativos con el fin de conseguir una experiencia de uso más simplificada al no tener que tomar la temperatura de cada ingrediente.

Además del cálculo de calorías, la placa tendrá almacenado los valores de las calificaciones del usuario para cada comida para obtener las calorías necesarias para cocinar teniendo en cuenta las preferencias del usuario. También deberá guardar por cada uso el promedio de las calorías otorgadas en 5 segundos, pero solo del último trimestre porque el calor del fuego puede variar en el tiempo. Esta variación puede ocurrir si se utiliza el servicio de distribución de gas natural de la ciudad o garrafa, que el calor que otorga es proporcional a la temperatura atmosférica. Por ende, un dato extra que brinda más precisión todavía es el de la temperatura actual en la ciudad.

El fabricante de estas ollas se puede beneficiar de los datos generados por las mismas para aprender sobre la vida útil de sus productos, cuáles son las que más duraron y las que menos y analizar qué cambió en el lote de producción de esas ollas. Asimismo, conoce las costumbres de los usuarios: la comida que come, cuándo la come y cantidad. Con esta información se puede realizar mejores publicidades, se sabría las zonas donde se comen ciertas comidas, el momento del día y el año que más se cocina, con más producción, y con la cantidad se puede deducir si tiene visitas.

CONCLUSIÓN

Finalizada la primera versión del trabajo práctico integrador, podemos abordar a las siguientes conclusiones:

- ✓ Para un mismo problema, y utilizando diferentes tecnologías, podemos encontrar muchos patrones/estilos arquitectónicos o problemas resueltos de diseño que nos permitan llegar a la misma solución de diversas maneras. Es importante hacer uso de un buen criterio de selección para utilizar el que mejor se adecúe a nuestro desarrollo y habilidades.
- ✓ Tenemos un sinfín de librerías que podemos utilizar dentro de los lenguajes de programación adaptados para solucionar problemas de implementación. Lo cual, nos lleva a decir que la arquitectura diseñada a veces se modifica o no se respeta en su totalidad, para simplificar ciertos aspectos que no quedaron claramente definidos.
- ✓ La importancia de la arquitectura ha sido bien entendida como un instrumento que permite crear las bases sobre las cuales se construirá el software. Además, es una herramienta muy poderosa de comunicación con las diferentes partes del equipo. Si trasladamos esta situación a un contexto más numeroso, o a una organización, será de utilidad también para transmitir aspectos funcionales y no funcionales a los distintos stakeholders.
- ✓ Es importante mantener la consistencia entre lo que está diseñado en la arquitectura y la codificación. La documentación siempre es un reflejo de nuestro producto, y la arquitectura como instrumento de comunicación, no puede dar falsas impresiones de lo que hace y/o no hace el software que hemos desarrollado.
- ✓ Queda pendiente para una próxima versión la aplicación e implementación de algún patrón que resuelva algún aspecto del diseño de nuestro software.