# A.1. Desarrollo de los Sprints

## A.1.0. Sprint #0: Mitigación de Riesgo Tecnico

Previo al inicio del desarrollo de los Sprint se desarrolló un prototipo inicial, con el fin de disminuir en gran medida los riegos técnicos presentes en el proyecto, o en su defecto eliminarlo como riesgo propiamente tal. Al proceso de desarrollar este prototipo se le denominó «Sprint #0» para todos los efectos del presente documento y Memoria.

Uno de los obstáculos a superar es la ignorancia de parte del equipo de programadores al momento de interconectar estas tecnologías y lenguajes de programación. Si bien el equipo cuenta con conocimientos en IoT, pero sólo tiene experiencia en esta área programando en Python y PHP, mas no en C#. Además, unas de las restricciones que debemos atender es utilizar las librerías proporcionadas por Clayster en el prototipo.

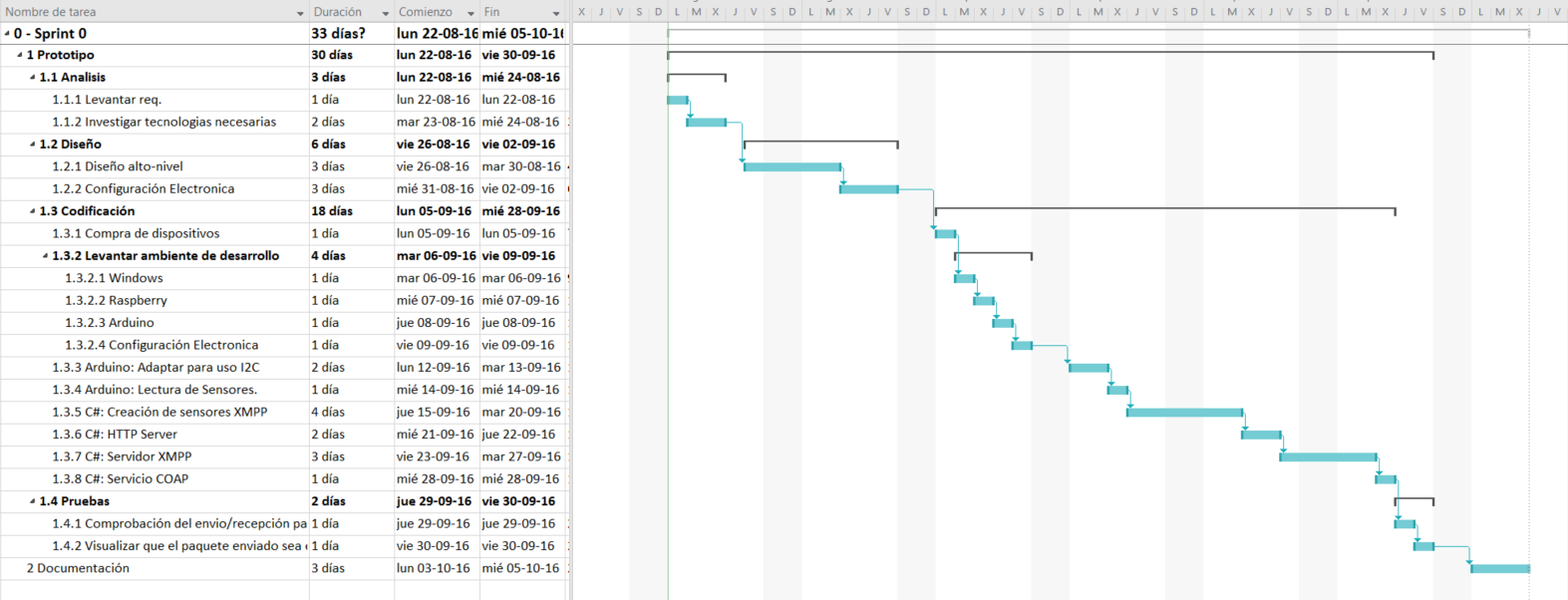
Para esto se acordó realizar los ejemplos del libro «Learning Internet of Things», escrito por Peter Waher. Estos enseñan la utilización de XMPP por medio de C#. Si bien no utilizaremos C# en nuestro proyecto, servirá para entender y aprender cómo trabajar con los flujos XML propios de XMPP. Para que así luego se simplifique el desarrollo en JAVA del plugin y extensión para el servidor XMPP OpenFire y librería SMACK respectivamente.

Si desea ver con mayor detalle de qué trata XMPP y IoT véase el segundo capítulo de la memoria.

### A.1.0.1. Planificación

Para el Sprint #0 se realizó una Gantt con las actividades a realizar, estimando el tiempo a tardar en cada una de estas actividades, las cuales se detallan en la figura 1.

Imagen : Sprint #0 – Carta Gantt



#### A.1.0.1.1 Riegos

A continuación en la tabla, vemos los riesgos identificados para el Sprint #0 junto con su estrategia de mitigación y finalmente, visualizamos el valor de riesgo aplicando dicha estrategia.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | Tipo | Riesgos | Estrategia de mitigación | Probabilidad | Impacto | Valor de riesgo |
| 002 | Técnico | Dificultad del equipo de desarrollo para aprender el uso de XMPP. | Se realizará un «Sprint #0» en el cual se desarrollará un prototipo con las funcionalidades de XMPP. | Alta | Muy Alto | 10 |
| 003 | Técnico | Dificultad del equipo de desarrollo para aprender a usar librerías Clayster o WaherData. | Se realizará un «Sprint #0» en el cual se desarrollará un prototipo con las funcionalidades de las librerías de Clayster y/o WaherData. | Alta | Alto | 10 |
| 004 | Externo | El cliente no satisfecho con el resultado del Sprint. | Se diseñarán criterios y pruebas de aceptación en el inicio de cada sprint, los cuales él cliente deberá aceptar. | Media | Alto | 6 |
| 007 | Externo | Desperfectos en los dispositivos electrónicos. | Se tendrá doble cantidad de dispositivos. Además, se realizarán conexiones y manipulaciones de la electrónica con los dispositivos desconectados de la toma de electricidad, además eliminando la estática del sujeto que intervenga los dispositivos. | Media | Alto | 3 |

Tabla : Sprint #0 - Riesgos

### A.1.0.2. Diseño de Prototipo

Se procederá a crear un prototipo funcional, con la configuración electrónica de la Imagen 1. Es decir, se interconectará una Raspberry Pi 3 con Arduino por medio del bus que utiliza el protocolo I2C, por el cual se pasarán, desde Arduino a Raspberry, por medio de una solicitud de esta última, del sensor de temperatura «DFRobot LM35» y humedad «DFRobotSteam V2» conectados al microcontrolador. La unidad de computo Raspberry tendrá conectados 4 LEDs de colores blanco, rojo, verde y azul, cuyas funciones son alertar del estado del software que se encuentra en marcha.  
Cada LEDs significa:

* Blanco: Programa en ejecución.
* Rojo: Error de ejecución del programa.
* Verde: Estado de la red.
* Azul: Lectura o interacción con sensores.

Esto cuanto al diseño de la configuración electrónica, la cual se puede apreciar a continuación:

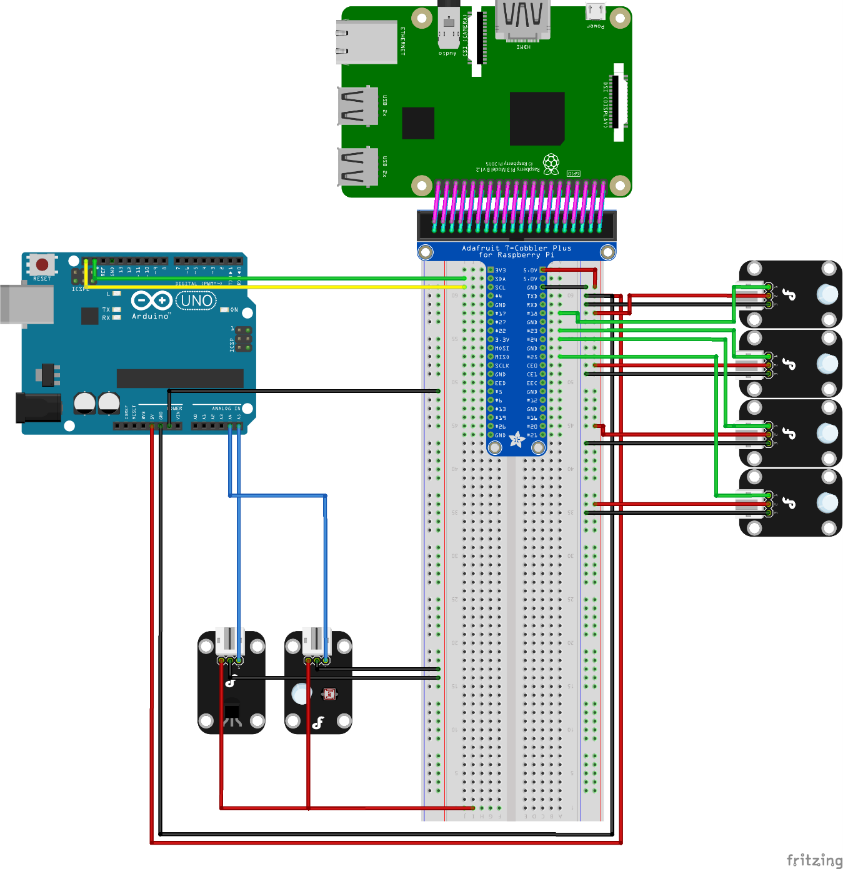


Imagen : Prototipo - configuración electrónica

La Raspberry estará conectada como un cliente XMPP al servidor ubicado en el dominio binarylamp.cl, el cual tiene lo básico necesario para funcionar como un servidor XMPP con funciones implementadas para Internet de las Cosas. A su vez, existirá un cliente el cual será un computador el que tendrá la función de comprobar los valores de los sensores.

Como petición especial, se nos solicitó por ProductOwner que las «Things»(es decir los sensores), tuviesen implementado el concepto de «Friendship». En términos prácticos, es que las dos identidades se suscriban al estado de «presence» de la otra.

Diagrama de despliegue del prototipo es ilustrado en la imagen 3, a continuación.

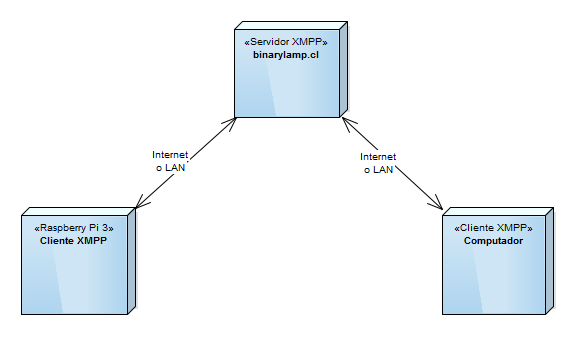


Imagen : Prototipo - Diagrama de Despliegue

### A.1.0.3. Resultados

Dentro de los primeros descubrimientos importantes acerca XMPP, se encuentran una gran variedad de patrones de diseño soportados. Comenzaremos listando y explicando brevemente el funcionamiento de cada uno de estos.

* Publish-Subscribe:

Este patrón de diseño esta descrito en el XEP-0060: Publish-Subscribe.

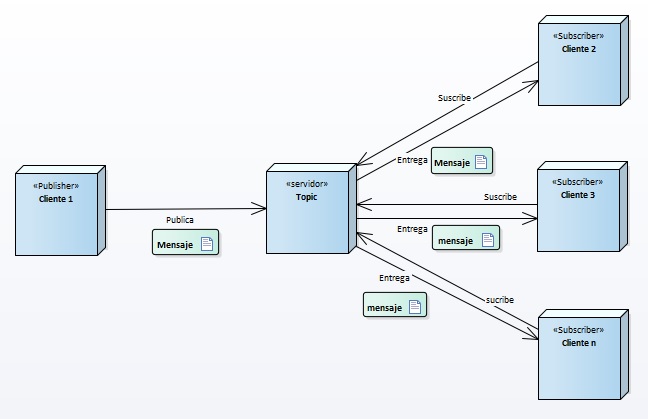


Imagen : Patrón de diseño Publicador/Suscriptor

* El funcionamiento de este patrón de diseño en XMPP es el siguiente. Tal como grafica la imagen, tenemos los clientes dos y tres que son los «Suscriber» o suscriptores, los cuales se suscriben al «Topic» o Tema, con el fin de recibir los mensajes que se envíen. Este mensaje es enviado por algún cliente el cual es el «Publish» o publicador, este se encarga de enviar el mensaje al «Topic» quien a su vez los entrega a él/los mensajes a los «Suscriber».
* Request/Response:  
  Este patrón de diseño esta descrito ampliamente en el RFC 6120: XMPP core, su funcionamiento lo podemos visualizar a continuación en la imagen 5.

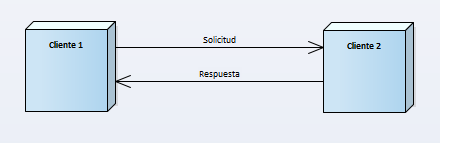


Imagen : Patrón de diseño Request/Response

Como podemos apreciar, este es un patrón intrínseco en el funcionamiento del protocolo XMPP, ya que en gran cantidad de oportunidades las solicitudes que se están realizando son solitudes con el fin de esperar una respuesta. Este tiempo de conexiones, es sincrónica.

* Asynchronous Messaging:

De igual modo que el patrón de diseño Request/Response, el patrón Asynchronous es parte fundamental en el núcleo de XMPP, por ende, está en RFC 6120: XMPP core. A continuación, en la Imagen X, podemos ver su funcionamiento.

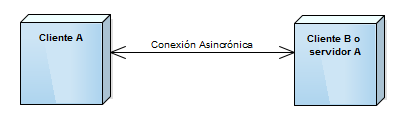


Imagen : Patrón de diseño Asynchronous Message

Debido a problemas con la compra de los dispositivos en la región, particularmente el conversor análogo a digital «AD799X» el cual no se encontraba disponible, una opción viable era comprarlo en el extranjero, sin embargo, no era una alternativa viable debido a la tardanza en el envío. Debido a esto, se procedió a utilizar Arduino para emular el funcionamiento del conversor análogo-digital «AD799X», el cual hace uso del del bus I2C para atender las solicitudes realizadas de la Raspberry. Debido a lo anteriormente expuesto, no fue posible cumplir el tiempo pactado en la planificación.

El código se generado para atender las solicitudes se puede ver a continuación.



Por otra parte además, se nos presentó un problema IPv6 en Raspiaban, lo cual nos tomó una semana extra el codificar el software, el cual no permitía levantar tanto IPv4 como IPv6. Para solucionarlo, establecimos comunicación con Peter Waher, quien nos asistió para encontrar la solución al problema, el cual consintió en bajar la interfaz IPv6 para que así solo quedase IPv4 funcionando. Lo cual resultó exitosamente.