

▪ Proyecto Fin de Grado ▪
Ingeniería del software

ESTUDIO Y COMPARATIVA DE LAS
DIFERENTES ALTERNATIVAS PARA EL
USO DE SMART CARDS DESDE
APLICACIONES WEB

Autor

Andoni Sánchez Antolín

Director

José Miguel Blanco Arbe

Junio 2016

Me gustaría agradecer en primer lugar a mi grupo de compañeros, Iñaki Latorre, Yongliang Zhan y José Javier Ruiz por estos fantásticos años trabajando mano a mano y ayudándonos unos a otros. No olvidarme también de esas escapadas entre clases para “echar un Ping Pong”.

También me gustaría agradecer a todo el profesorado que me ha guiado en mi aprendizaje, en especial a mi director de proyecto, José Miguel Blanco, en el que me he inspirado y del que he aprendido muchísimo estos dos últimos años del grado.

Agradecer también a la empresa ATELEI por darme esta oportunidad y experiencia tanto en la realización del proyecto como en las prácticas que he realizado con ellos el verano pasado. La exigencia y ayuda recibida por vosotros, Iván, David e Iñigo, ha sido de gran ayuda para explotar mis capacidades y dar el 100% de mí.

Acordarme también de mi grupo de amigos y de mis compañeros de trabajo de Ilcapo, que continuamente me han estado apoyando y se han interesado por mis progresos.

Apreciar inmensamente también, la motivación recibida por parte de mi novia, Carla Aldabaldetrekú, que cuando las cosas se veían muy negras ha sabido darles la vuelta y me ha ayudado a aumentar la confianza en mí mismo creyendo en mis capacidades.

Por último, agradecer y homenajear especialmente todo este esfuerzo y entrega realizado a mis padres, Paco y Candy, como a mi hermana, Marina, que han confiado en mis competencias y me han dado esta oportunidad, apostado por mi formación en aquello que me apasiona.

Gracias a todos por estos estupendos años, no lo hubiese logrado sin vuestro apoyo.

Resumen

Este documento corresponde a la memoria del proyecto de fin de grado, Estudio y comparativa de las diferentes alternativas para el uso de *smart cards* desde aplicaciones web, desarrollado para la titulación de Ingeniería Informática, concretamente en la especialidad de Ingeniería del software, en la facultad de Informática de la universidad UPV-EHU de Donostia/San Sebastián.

En este proyecto se ha llevado a cabo un estudio exhaustivo de las diferentes alternativas a los *Applets* de Java por su aparente desaparición en los navegadores web.

El proyecto ha sido realizado en la empresa ATELEI Engineering SLU, empresa de base tecnológica, especializada en sistemas de control de acceso y monitorización remota. Bajo la supervisión de David Ruiz e Iván Piquer junto con la colaboración de Iñigo Olaso, y dirigido por el profesor de la UPV/EHU José Miguel Blanco.

El principal objetivo de este es comunicar datos entre un lector de tarjetas y una servidor web, mediante un módulo ESP8266 que dispone prestaciones para comunicaciones Wireless utilizando el protocolo TCP/IP mediante SSL.

Índice

RESUMEN	V
ÍNDICE	VII
LISTA DE FIGURAS Y TABLAS	XI
FIGURAS	XI
TABLAS	XIV
1. INTRODUCCIÓN	1
2. ANTECEDENTES DEL PROYECTO	5
2.1. CARACTERÍSTICAS DE LA EMPRESA	5
2.2. EL SISTEMA DE CONTROL DE ACCESOS	6
2.2.1. <i>Familias de dispositivos</i>	6
2.2.2. <i>Tipos de tarjetas y funcionamiento</i>	7
2.3. NECESIDAD DE LA EMPRESA	9
3. OBJETIVOS	11
3.1. ALCANCE DEL PROYECTO	12
3.2. EXCLUSIONES DEL PROYECTO	13
3.3. ESTRUCTURA DE DESCOMPOSICIÓN DEL TRABAJO	13
4. METODOLOGÍA	15
5. ESTUDIO Y ANÁLISIS DE LAS ALTERNATIVAS	17
5.1. JAVA WEB START	18
5.1.1. <i>Funcionamiento</i>	18
5.1.2. <i>Ejecución de la aplicación</i>	19
5.1.3. <i>Requisitos</i>	19
5.1.4. <i>Ejemplo de Java Web Start</i>	20
5.1.5. <i>Aspectos positivos</i>	20
5.1.6. <i>Aspectos negativos</i>	20
5.2. CHROME.USB API	21
5.2.1. <i>Ejemplo de creación</i>	21
5.2.2. <i>Aspectos positivos</i>	21
5.2.3. <i>Aspectos negativos</i>	21
5.3. NATIVE CLIENT	21
5.3.1. <i>Aspectos positivos</i>	22

5.3.2. Aspectos negativos	22
5.4. EDGE (SILVERLIGHT).....	22
5.4.1. Aspectos positivos.....	23
5.4.2. Aspectos negativos	23
5.5. SMARTPHONE - TABLET (NFC)	23
5.5.1. Aspectos positivos.....	23
5.5.2. Aspectos negativos	23
5.6. ANÁLISIS GLOBAL DE LAS ALTERNATIVAS.....	24
6. EL MÓDULO ESP8266	25
6.1. ANÁLISIS DEL MÓDULO ESP8266.....	25
6.1.1. Características principales	26
6.1.2. Ventajas y desventajas de las diferentes versiones del modulo ESP8266.....	26
6.1.3. Tabla de tiendas dónde comprar (ordenadas por prioridad).....	27
6.1.4. Conclusión del análisis	27
6.2. INICIALIZACIÓN EN EL MÓDULO ESP8266	28
6.2.1. Herramientas	28
6.2.2. Conexión del módulo ESP8266.....	29
6.2.3. Instalación del driver del módulo UART PL2303	31
6.2.4. Comunicación con la terminal a través del puerto serie.....	32
6.2.5. Prueba inicial con comandos AT	33
6.2.6. Eclipse Espressif SDK	34
6.2.7. Flasheo del módulo ESP8266 desde Unix.....	35
6.2.8. Arduino	36
6.2.9. Eliminar la memoria flash.....	38
6.3. ANÁLISIS DEL USO DE LOS DIFERENTES SDKS	39
6.4. CREACIÓN DE UN PROYECTO EN ECLIPSE	40
6.5. IDENTIFICACIÓN DE LAS LIBRERÍAS REQUERIDAS	43
7. DESARROLLO DEL SOFTWARE	45
7.1. ESPECIFICACIÓN Y REQUISITOS.....	46
7.2. DISEÑO	47
7.2.1. Casos de uso	47
7.2.2. Diagrama de estados.....	55
7.2.3. Arquitectura.....	57
7.2.3. Estructuras de datos	57
7.3. IMPLEMENTACIÓN	59

7.3.1. Implementación de los casos de uso Establecer la configuración inicial de la instalación y carga de los datos.....	59
7.3.2. Implementación del caso de uso conexión Wi-Fi y cliente de peticiones	60
7.3.3. Implementación de los casos de uso login, obtener acciones y manejo de acciones	61
7.3.4. Implementación del caso de uso del la ejecución de la acción en el encoder	62
7.3.5. Problemas en la implementación	62
7.4. VERIFICACIÓN Y PRUEBAS	63
7.4.1. Verificación del caso de uso Establecer configuración inicial de la instalación .	63
7.4.2. Verificación del caso de uso cargar configuración.....	63
7.4.3. Verificación del caso de uso Conexión Wi-Fi y puesta en marcha del cliente de peticiones.....	64
7.4.4. Verificación del caso de uso Login.	66
7.4.5. Verificación del caso de uso Obtener acciones	67
7.4.6. Verificación del caso de uso Lectura de datos y número de serie y Ejecución en el encoder	68
7.4.7. Verificación y pruebas globales del módulo ESP8266.....	70
8. GESTIÓN DEL PROYECTO.....	73
8.1. GESTIÓN DEL ALCANCE.....	74
8.2. ESTUDIO DE LA VIABILIDAD	75
8.2.1. DAFO.....	75
8.2.2. Identificación de los riesgos.....	76
8.2.3. Cuantificación de los riesgos.....	77
8.2.4. Plan de contingencia.....	77
8.3. GESTIÓN DE CAMBIOS.....	78
8.4. GESTIÓN DE COSTES	79
8.5. PLANIFICACIÓN.....	80
8.5.1. Gráfico Gantt	80
8.5.2. Diagrama de Hitos.....	83
9. CONCLUSIONES.....	85
9.1. CONCLUSIONES SOBRE LOS OBJETIVOS	86
9.2. CONCLUSIONES SOBRE LA GESTIÓN	86
9.3. CONCLUSIONES PERSONALES.....	87
10. PROPUESTAS DE MEJORA	89
11. BIBLIOGRAFÍA	91

12. GLOSARIO Y ACRÓNIMOS	95
A. USO DE JAVA WEB START	101
A.0. HERRAMIENTAS UTILIZADAS	101
A.1. EJEMPLOS APLICACIÓN JWS	101
A.1.1. Pasos de creación	101
A.1.2. Configuración y ejecución del servidor Apache	106
A.1.3. Diagrama de flujo.....	106
A.1.4. Uso de la aplicación java mediante Java Web Start.....	106
A.1.5. Ficheros creados.....	108
A.2. EJEMPLO JWS CON NODE JS	108
A.2.1. Instalación y configuración del servidor Node Js + Express	108
A.2.2. Comunicación cliente-servidor Node Js mediante sockets.....	109
A.2.3. Comunicación desde un servidor apache a un servidor node Js y al cliente (Node Js).....	111
A.2.4. Comunicación desde un cliente Node Js al servidor Node Js y éste a un servidor apache	115
A.2.5. Aplicación Java	117
A.2.6. Ficheros creados para la comunicación.....	117
A.3. BIBLIOGRAFÍA CONSULTADA	118
B. EJEMPLOS DE LOS APIS DEL SDK RTOS DE ESPRESSIF	121
B.1. MANEJO DE TAREAS 1	121
B.2. CONFIGURACIÓN WI-FI.....	122
B.3. MANEJO DE TAREAS 2	124
B.4. USO DE CALLBACKS DENTRO DE UNA TAREA	126
B.5. USO DE LA LIBRERÍA CJSON.....	127
B.6. CREACIÓN DE UN CLIENTE Y SERVIDOR <i>HTTP REQUEST</i>	128
B.6.1. <i>Espconn API</i>	128
B.6.2. <i>Esp-now API</i>	128
B.6.3. <i>Mesh API</i> ^[16]	129
B.6.4. <i>Ejemplo</i>	129
B.7. EJEMPLO TCP - SSL.....	131
B.8. ALMACENAR DATOS EN LA MEMORIA FLASH	133
C. COMANDOS AT.....	135
D. PROTOCOLO DE COMUNICACIÓN	137
E. INFORME DE LA EMPRESA.....	139

Lista de Figuras y Tablas

Figuras

Figura 1. Cerradura electrónica personalizable.....	6
Figura 2. Lector mural.	7
Figura 3. Controlador.....	7
Figura 4. Tarjeta (MIFARE) de Residente de Irún.....	7
Figura 5. Estructura interna de la tarjeta MIFARE.	8
Figura 6. EDT - Estructura de descomposición de trabajo.	13
Figura 7. Desarrollo en espiral. Recuperado de https://es.wikipedia.org/wiki/Desarrollo_en_espiral	16
Figura 8. Aplicación Slack. Recuperado de: https://en.wikipedia.org/wiki/Slack_(software)	16
Figura 9. Antes de la instalación de la aplicación Java.....	19
Figura 10. Una vez instalada la aplicación Java.....	19
Figura 11. Ejemplo del chat entre los servidores Apache y Node Js.	20
Figura 12. Módulo ESP8266 versión ESP-12F.	26
Figura 13. Esquema de conexionado para el envío de comandos.....	29
Figura 14. Esquema de conexionado modo flash.	30
Figura 15. Esquema de conexionado modo boot desde la memoria flash.....	30
Figura 16. Administrador de dispositivos.....	31
Figura 17. Controlador para el módulo UART PL2303.	31
Figura 18. Configuración del puerto en RealTerm.	32
Figura 19. Envío de comandos con RealTerm.	32
Figura 20. Configuración de la aplicación CoolTerm.....	33

Figura 21. Paquetes MinGW.	34
Figura 22. Campo Additional Boards Manager URLs.	36
Figura 23. Instalación de la placa base.	36
Figura 24. Configuración de la placa base.	36
Figura 25. Verificación y compilación de la aplicación.	38
Figura 26. Carga (flasheo) de la aplicación.	38
Figura 27. Creación del proyecto C.	40
Figura 28. Carpetas necesarias para el proyecto.	41
Figura 29. Configuración del builder.	42
Figura 30. Crear un nuevo Make Target.	42
Figura 31. Configuración del Make target a crear.	42
Figura 32. Casos de uso.	47
Figura 33. Flujo de eventos establecer la configuración inicial de la instalación.	48
Figura 34. Flujo de eventos cargar configuración inicial de la instalación.	49
Figura 35. Flujo de eventos Conexión Wi-Fi y puesta en marcha del cliente de peticiones.	50
Figura 36. Flujo de eventos login.	51
Figura 37. Flujo de eventos obtener acciones.	52
Figura 38. Flujo de eventos leer datos y/o leer número de serie de la tarjeta.	53
Figura 39. Flujo de eventos ejecutar acción en el encoder.	54
Figura 40. Diagrama de estados de la aplicación.	55
Figura 41. Arquitectura de la instalación.	57
Figura 42. Curl desde la terminal al módulo ESP8266.	63
Figura 43. Carga de datos almacenados en la flash en la estructura esp_install. ...	64
Figura 44. Intento de conexión a la red Wi-Fi.	64
Figura 45. Conexión establecida a la red Wi-Fi.	65
Figura 46. Captura mediante Wireshark de mensajes enviados y recibidos entre cliente-servidor.	65
Figura 47. Logueo del módulo ESP8266 en el servidor (visto desde el cliente).	66
Figura 48. Logueo del módulo ESP8266 en el servidor (visto desde el servidor).	66
Figura 49. Obtención de acciones – No hay acciones (visto desde el cliente).	67
Figura 50. Obtención de acciones – No hay acciones (visto desde el servidor).	67

Figura 51. Lectura correcta de datos y número de serie de la tarjeta (visto desde el cliente).....	68
Figura 52. Lectura correcta de datos y número de serie de la tarjeta (visto desde el servidor).	68
Figura 53. Acción de lectura fallida (visto desde el cliente).	69
Figura 54. Acción de lectura fallida (visto desde el servidor).	69
Figura 55. Acción de lectura tiempo de espera superado (visto desde el cliente). ...	70
Figura 56. Acción de lectura tiempo de espera superado (visto desde el servidor)..	70
Figura 57. Memoria RAM (heap) del módulo ESP8266 estable en cada petición. ..	71
Figura 58. Gantt general del proyecto.....	80
Figura 59. Gráfico Gantt del estudio de las alternativas.....	81
Figura 60. Gráfico Gantt etapa de aprendizaje del módulo ESP8266.	81
Figura 61. Gráfico Gantt de la especificación del funcionamiento esperado de la aplicación.....	81
Figura 62. Gráfico Gantt del ciclo de vida de desarrollo software de la aplicación. 82	
Figura 63. Gráfico Gantt de la redacción de la memoria y preparación de la presentación.....	82
Figura 64. Diagrama de flujo de la aplicación JWS.	106
Figura 65. Diagrama de flujo de la aplicación JWS.	106
Figura 66. Imagen de la página web que ofrece la aplicación JWS.	106
Figura 67. Ejemplo para permitir la ejecución de una aplicación JWS.	107
Figura 68. Imagen de ejecución de un etiqueta dentro de un JFrame en una aplicación JWS.....	107
Figura 69. Imagen de ejecución de un una página HTML dentro de un Web View en una aplicación JWS.	107
Figura 70. Diagrama de flujo de comunicación entre el cliente y el servidor de Node JS.	111
Figura 71. Chat de mensajes entre los servidores Apache y Node JS.....	111
Figura 72. Consola de Node JS, conexión de un cliente y mensajes escritos en el socket.....	111
Figura 73. Diagrama de flujo de comunicación entre el cliente y el servidor de Apache con Node JS y su cliente.	114
Figura 74. Servidor Apache, Servidor Node Js y consola con mensajes del servidor Node Js. Mensaje sin enviar.	114

Figura 75. Servidor Apache, Servidor Node Js y consola con mensajes del servidor Node Js. Mensaje enviado.	115
Figura 76. Diagrama de flujo, Cliente Node Js escribe en socket y el servidor Node Js petición a servidor Apache.	117
Figura 77. Servidor Node Js en funcionamiento.	117
Figura 78. Envío de comandos AT con RealTerm.....	135
Figura 79. Instantánea del índice del protocolo de comunicación.....	137
Figura 80. Instantánea de los casos de uso del protocolo de comunicación.	138
Figura 81. Informe empresa. Parte 1/2.	139
Figura 82. Informe empresa. Parte 2/2.	140

Tablas

Tabla 1. Tabla comparativa de las diferentes tiendas, precios y tiempo de envío del módulo ESP8266.	27
Tabla 2. Caso de uso establecer la configuración inicial de la instalación.	48
Tabla 3. Caso de uso cargar configuración inicial de la instalación.....	49
Tabla 4. Caso de uso Conexión Wi-Fi y puesta en marcha del cliente de peticiones.	50
Tabla 5. Caso de uso login.	51
Tabla 6. Caso de uso obtener acciones.	52
Tabla 7. Caso de uso leer datos y/o leer número de serie de la tarjeta.....	53
Tabla 8. Caso de uso ejecutar acción en el encoder.....	54
Tabla 9. Parámetros de configuración de la instalación.	58
Tabla 10. Análisis DAFO del proyecto.....	75
Tabla 11. Tabla de cuantificación de riesgos.	77
Tabla 12. Gestión de costes del proyecto.....	79
Tabla 13. Diagrama de hitos.	83

Capítulo 1

1. Introducción

Las vulnerabilidades derivadas del uso las aplicaciones Java en los navegadores web han ido creciendo progresivamente, lo que ha llevado a plantear las políticas de seguridad de los navegadores. En su intento por mitigar estas brechas de seguridad Google ha introducido en Chrome la restricción de uso para estas aplicaciones. También el navegador Firefox ha impedido que se ejecute el *plugin* de Java automáticamente debido al problema de seguridad anteriormente comentado, aunque éste aún permite su activación en sitios de confianza.

Se desconoce si el resto de navegadores van a preservar su continuidad a corto y a largo plazo, por lo que queda en cuestión el futuro de una gran cantidad de aplicaciones Java que siguen en el mercado ejecutándose en navegadores web.

Una de estas aplicaciones, la cual usaba la empresa ATELEI dentro de su sistema de control de accesos, se utilizaba para programar o leer tarjetas *smart card* mediante estos *applets* y un dispositivo *card reader* instalado en el equipo.

El propósito de este proyecto es buscar y realizar un estudio de las diferentes alternativas existentes a los *applets* e implementar la más óptima. Además, existen unos objetivos mínimos que hay que cumplir; ha de ser una aplicación multiplataforma que abarque un gran número de dispositivos electrónicos a los que se pueda integrar, además de ser altamente configurable para cualquier tipo de instalación, entre otros más. Para conocer todo el abanico de objetivos que se quieren alcanzar, estos se detallan más adelante en el capítulo: 3.1. Alcance del proyecto.

En este documento se describe el trabajo realizado por el autor en la empresa ATELEI Engineering SLU durante el segundo cuatrimestre del curso académico 2015/2016. La estructura de capítulos se describe a continuación:

- El documento de antecedentes del proyecto, que describe las características de la empresa y su especialización, así como el producto estrella en el que trabajan y las necesidades que surgen en torno a él.
- El documento de objetivos del proyecto, cuya subdivisión está formada por 3 subapartados: El alcance del proyecto y necesidades a satisfacer del proyecto, por otro lado, las exclusiones que no forman parte del alcance, y por último, la estructura de descomposición de trabajo la cual divide las diferentes tareas que se identifican en el proyecto.
- Un documento que recoge el estudio completo realizado sobre las diferentes alternativas que pueden escogerse para dar solución a la problemática surgida, con las ventajas y desventajas de cada una de ellas.
- Un documento que describe la metodología de desarrollo utilizada y las diferentes herramientas en las que se ha apoyado para facilitar el uso de la misma.
- Ciclo de vida del desarrollo de software, que está estructurado desde el análisis de requerimientos pasando por el diseño e implementación de los casos de uso recogidos hasta las pruebas y la verificación de su correcto funcionamiento, con diferentes instantáneas que muestran la verificación del funcionamiento.
- Documentación de la gestión del proyecto que se ha realizado desde el estudio de las alternativas hasta el proceso de desarrollo, que describe cada una de las decisiones adoptadas durante el transcurso del mismo, evaluando si se han seguido las medidas establecidas, así como si las estimaciones iniciales de las tareas fijadas al inicio del proyecto fueron las correctas.

- Un documento que capta las diferentes reflexiones realizadas tanto de las tecnologías de desarrollo utilizadas, las competencias adquiridas, como la experiencia vivida.

Es usual que a lo largo del proyecto aparezcan términos o acrónimos que podrían resultar desconocidos para el lector. Por ello, se adjunta un glosario que describe el significado de cada término, a fin de que sea consultado en su caso.

Capítulo 2

2. Antecedentes del proyecto

2.1. Características de la empresa

ATELEI Engineering SLU, con sede en el polígono Arretxe-Ugalde de Irún, es una empresa fundada por el CEO Iván Piquer en el año 2012. Está especializada en sistemas de control de acceso y monitorización remota, definiéndose como “diseñadores de productos, especializados en el diseño y desarrollo de productos electrónicos desde su conceptualización hasta su fabricación”. Pero no toda su

ingeniería reside en la electrónica, también son especialistas en desarrollo de sistemas software de alto nivel para cumplir con las funcionalidades de sus productos. Así mismo consta de un departamento de I+D en el que desarrollan esos productos y servicios.

Este proyecto en particular ha sido supervisado por David Ruiz (Ingeniero Informático de la especialidad del Software) e Iván Piquer (Ingeniero Informático Superior) y llevado a cabo junto con la colaboración de Iñigo Olaso (Ingeniero Mecánico y Electrónico).

2.2. El sistema de control de accesos

Los dispositivos fabricados por ATELEI, forman una red inalámbrica invisible, que permite que éstos se comuniquen entre sí mediante mensajes. De esta manera es posible configurar la red para que, ante ciertos eventos en un dispositivo, cualquiera que se encuentre en la misma red, esté o no alejado de él, pueda llevar a cabo acciones que vayan desde la apertura de una puerta o barrera de parking hasta el encendido de la iluminación de una estancia.

2.2.1. Familias de dispositivos

Cerraduras Electrónicas: En ATELEI prestan especial atención a los pequeños detalles. Esos pequeños detalles les diferencia del resto de competidores. Por este motivo, han creado una cerradura electrónica en la que el cliente elige el diseño. Tanto si es una empresa que desea usar su imagen corporativa, como un particular con grandes dotes como artista.



Figura 1. Cerradura electrónica personalizable.

Lectores Murales: Dan al cliente la posibilidad de elegir el diseño de sus mecanismos. Gracias a los lectores fabricados por ATELEI, sus clientes pueden elegir entre una amplia gama de colores que permiten al sistema de control y acceso integrarse a la perfección dentro de la estética de los edificios. Todo ello además mejorado con los últimos avances tecnológicos de un sistema *wireless* innovador.



Figura 2. Lector mural.

Controladores: Un sistema de control, tiene que permitir integrar productos de terceros, de forma que estos se integren en la instalación como un elemento más. Con este objetivo en mente, en ATELEI han diseñado un grupo de dispositivos que les permiten llevar a cabo esta integración. De esta manera, gestionar el encendido de la calefacción o activar el aire acondicionado se convierte en una tarea trivial que puede ser llevada a cabo desde cualquier lugar del planeta.



Figura 3. Controlador.

2.2.2. Tipos de tarjetas y funcionamiento

En ATELEI hacen uso de la tarjetas MIFARE, tarjetas inteligentes sin contacto o *contactless*. Estas tarjetas de memoria están protegidas, divididas en sectores, bloques y mecanismos simples de seguridad para el control de acceso. Debido a su capacidad de computo no permite realizar operaciones criptográficas o de autenticación mutua a alto nivel. Su uso está orientado a monederos electrónicos simples, control de acceso, identidad corporativa o como tarjeta de transporte urbano.



Figura 4. Tarjeta (MIFARE) de Residente de Irún.

En cuanto a su estructura, los sectores se dividen en cuatro bloques, de los cuales tres pueden o no contener información de usuario y el restante guarda las claves y los permisos de acceso a los tres bloques anteriores, que pueden ser: lectura, escritura, descuento o incremento.

■	Manufacturer data, including UID.
■	Mifare application directory.
■	Key A.
■	Access Bits.
■	General purpose byte.
■	Key B.
■	Usable space.

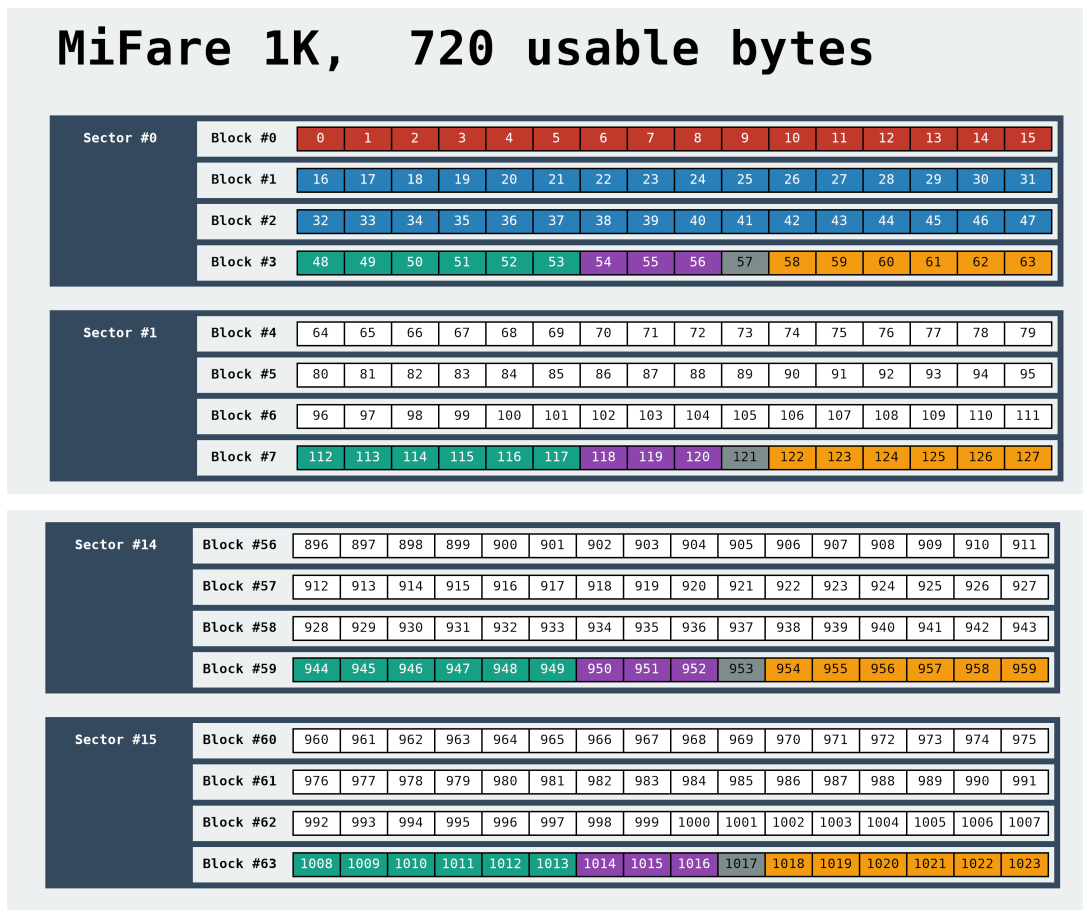


Figura 5. Estructura interna de la tarjeta MIFARE.

Cuando se acerca la tarjeta a un lector, ésta se activa e inicia un proceso de intercambio de datos con el lector para establecer una comunicación cifrada. Una vez establecido el canal cifrado, la tarjeta envía un código de identificación de conexión (siendo normalmente el número de serie de la tarjeta). Con el número de serie identificado por el lector y una vez presentadas las claves de acceso a los respectivos sectores éste está capacitado para hacer operaciones sobre la tarjeta.

En concreto la empresa ATELEI trabaja con la versión MIFARE Plus de 1K^[20]. Ésta ofrece un grado de seguridad mayor que la MIFARE Classic sin necesidad de actualizar la instalación. La seguridad para la autenticación, integridad de los datos y cifrado se basa en la criptografía estandarizada AES.

Soporta la norma ISO/IEC 1443-3 con identificadores de 7 bytes, soporte para IDs aleatorios, las claves pueden almacenarse como claves de 2 x 48 bits MIFARE Crypto1 o como claves AES de 2x 128 bits y alcanza una velocidad de comunicación de hasta 848kbit/s.

Las aplicaciones que se les pueden dar son las siguientes:

- Control de accesos, colegios, residencias, empleados, hostelería.
- Aparcamientos de coches.
- Fidelidad.
- Como método de pago.

2.3. Necesidad de la empresa

Durante años los distintos navegadores para plataformas de escritorio han convivido con los *plugin* de Java que Oracle suministraba. Tanto ha crecido su popularidad que se ha convertido en un complemento casi indispensable para los navegadores. Como aplicación famosa, es común que numerosos atacantes traten de explotar al máximo estas aplicaciones, para dar con vulnerabilidades y llegar a afectar a un número considerado de usuarios. Así es que se han ido descubriendo tal cantidad de vulnerabilidades en Java, cuyas repercusiones en el ámbito de la ciberseguridad fueron tales que Oracle tuvo que eliminar el soporte^[21] de estos *plugins*.

Al final el *plugin* de Java, en la versión Java 9, no ha desaparecido completamente de todos los navegadores sino que se ha adaptado^[22]. Oracle está tratando de convencer a usuarios y desarrolladores que dejen de utilizar los *applets* de Java y adapten sus desarrollos a la tecnología Java Web Start. Esta medida tomada por Oracle es consecuencia del abandono del *plugin* por parte de Firefox y Chrome, este último lo llevaba anunciando desde hace más de dos años, por lo que en la actualidad Chrome ha desechado cualquier opción de usar Java en su navegador y algunos navegadores como Firefox lo permiten, siempre que se accione su uso de forma manual para sitios de confianza.

La empresa ATELEI Engineering SLU, hasta la fecha del fin de soporte, hacía uso de estos *applets* de Java, para que su aplicación de control de accesos hiciera uso del dispositivo (lector de tarjetas) instalado en el ordenador del usuario. Estos *applets* se ejecutaban a través del navegador y hacían uso de librerías de Java que requerían tener instalada la JVM (Java Virtual Machine) y asignarle una serie de permisos. Así conseguía realizar acciones de lectura y escritura sobre tarjetas *smart card* desde un entorno web, resultando ser una aplicación multiplataforma accesible desde cualquier ordenador.

Capítulo 3

3. Objetivos

La informática avanza a un ritmo vertiginoso y son constantes los cambios que hay que realizar en los productos ya existentes para adaptarlos a las nuevas necesidades. En este caso por la desaparición de un complemento software de los navegadores.

La aplicación web de control de acceso de la empresa, se beneficiaba de ese complemento para hacer uso del lector de tarjetas que se requería estar conectado al ordenador del cliente.

3.1. Alcance del proyecto

El alcance del proyecto se divide en dos fases bien diferenciadas, la primera fase está dedicada al estudio de las alternativas a los *applets* de Java disponibles y la segunda fase al desarrollo de la aplicación de la alternativa escogida. De este modo se separan los objetivos para cada una de las distintas fases, a pesar de la concordancia entre ellas en el aspecto de desarrollo. En rasgos generales, se trata de crear una aplicación similar que permita alcanzar la calidad que disponía la anterior, creada en Java.

- Un análisis de las posibles herramientas o alternativas encontradas, con sus informes correspondientes que contendrán los resultados obtenidos y ejemplos de uso con carácter didáctico.
- El módulo o aplicación ha de ser configurado de manera externa para las necesidades de cada cliente.
- Lograr un funcionamiento multiplataforma: PC, MAC, Android, IOS y cualquier otro dispositivo que disponga de un navegador web.
- El módulo o aplicación ha de enviar peticiones al servidor indicado en la configuración, peticiones bajo el protocolo TCP-IP mediante SSL, para iniciar sesión en el servidor y preguntar por las acciones de lectura de tarjetas.
- Una vez el módulo termine la lectura realizará otra petición al servidor con la información de la transacción realizada.
- Mostrar notificaciones al usuario que utilice el lector, notificación de conexión con el servidor, lectura correcta o incorrecta y espera al posado de tarjeta, con el fin de conseguir una aplicación lo más accesible posible.
- Establecer un interruptor de reseteo para restablecer los ajustes de fábrica.

El desarrollo realizado se basa en los siguientes puntos:

1. Un cliente, el módulo ESP8266 con la ayuda de un *encoder* que es capaz de leer tarjetas.
2. El módulo es independiente del sistema operativo que se utilice, únicamente requiere de alimentación por puerto USB y de conexión a una red Wireless que disponga de acceso a internet.
3. Un servidor que se rige por el protocolo descrito para la realización de las comunicaciones entre cliente-servidor.
4. El cliente, el módulo ESP8266, crea una red Wireless o Access Point en la que en la dirección de su puerta de enlace tiene un servidor escuchando, a la espera de recibir peticiones POST con los parámetros de configuración descritos en el protocolo en formato JSON. Esos parámetros se almacenan en la memoria flash que carga en cada reinicio del mismo.

3.2. Exclusiones del proyecto

El siguiente listado hace referencia a los puntos que se excluyen del alcance del proyecto:

1. Se excluye toda documentación relativa a la configuración y gestión del servidor que brinda las peticiones, más allá de la que se describe en el protocolo de comunicación, recogido en el anexo: D. Protocolo de comunicación.
2. De la misma forma, se excluye la documentación y desarrollo realizado del *encoder* o *transceiver* encargado de la lectura de tarjetas.
3. Se excluye también la validación de los datos de la configuración inicial de la instalación que recibe el servidor del módulo ESP8266 en modo punto de acceso (AP).
4. Así mismo, se excluye también el análisis de accesibilidad relacionado con la interacción del usuario con el módulo y el manual de usuario del mismo.
5. Además se excluyen del alcance las pruebas unitarias de cada módulo o método de código como las pruebas de integración de la unión de cada uno de los métodos.

3.3. Estructura de descomposición del trabajo

La siguiente figura muestra las diferentes tareas y paquetes de trabajo en el proyecto.

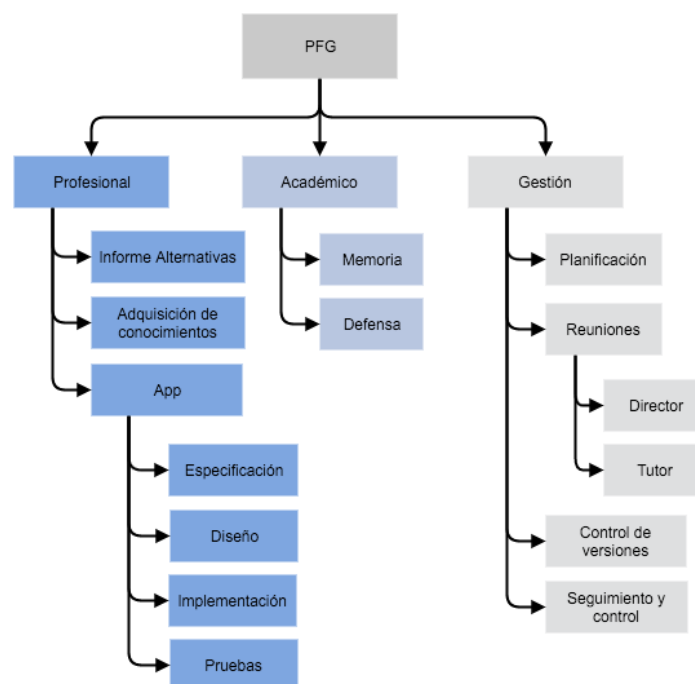


Figura 6. EDT - Estructura de descomposición de trabajo.

Capítulo 4

4. Metodología

La metodología de desarrollo que se ha empleado, tanto en el estudio de las alternativas, como en el ciclo de vida de la aplicación, ha sido una metodología de desarrollo en espiral. Esta metodología de desarrollo consta de diversas actividades, que se conforman en una espiral, en la que cada bucle o iteración representa un conjunto de actividades. Las actividades no siguen ninguna prioridad sino que se escogen en función del análisis de riesgo, comenzando por el bucle interior.

Este modelo de desarrollo consta de cuatro fases:

- En primer lugar, al inicio de cada iteración se determinan los objetivos que se quieren alcanzar para esa iteración.
- A continuación, se realiza un análisis para identificar los posibles riesgos que se puedan originar y se realiza un plan para prevenirlos o mitigarlos.
- Acto seguido, se comienza a desarrollar los casos de uso siguiendo el ciclo de vida del desarrollo software, desde el análisis hasta las pruebas, pasando por el diseño, la implementación y la verificación del caso de uso.
- Por último, se planifica los casos de uso de la siguiente iteración y se vuelve al punto de determinar los objetivos.

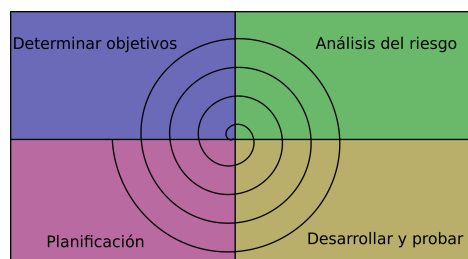


Figura 7. Desarrollo en espiral. Recuperado de https://es.wikipedia.org/wiki/Desarrollo_en_espiral

Una vez terminadas las cuatro fases de cada iteración se comienza con la siguiente, repitiendo continuamente este proceso hasta terminar cumpliendo con el alcance.

El seguimiento y desarrollo de ésta metodología se ha realizado en las reuniones pertinentes en la empresa, a través de la aplicación Slack, dónde se fijaba una planificación con los siguientes pasos a realizar.



Figura 8. Aplicación Slack. Recuperado de: [https://en.wikipedia.org/wiki/Slack_\(software\)](https://en.wikipedia.org/wiki/Slack_(software))

La aplicación Slack, es una herramienta o método de comunicación instantáneo, especializada para el trabajo en equipo. Una de sus mayores ventajas, es la creación de canales (*channels*), organizados por temas para poder comunicarse con el resto de componentes que forman el equipo. Esta aplicación permite la integración de muchas aplicaciones de terceros, como son Google Drive, utilizada para compartir informes y actas de reunión, permiten ser añadidas en los canales a disposición del resto del equipo, y Bitbucket, que es un repositorio remoto de Git, permite un control de versiones del código que se va implementando, repositorio del cual se reciben notificaciones en la aplicación Slack, para que el resto de integrantes se percaten de las nuevas actualizaciones que ha recibido el código.

Capítulo 5

5. Estudio y análisis de las alternativas

Este capítulo hace referencia al estudio de alternativas a los *applets* de Java que recientemente finalizó el soporte de la tecnología que utilizaban hasta ahora llamada NPAPI. Este *plugin* de Java para los navegadores web se basaba en una arquitectura de *cross-plataform*, arquitectura que permitía el acceso a los recursos del ordenador en el que se ejecutaba e interaccionaba con el navegador. Ha sido apoyado por navegadores web durante más de una década hasta verse como víctima de muchos ataques por su falta de seguridad. A causa de esto tanto el navegador Chrome, como

Firefox, entre otros, eliminan su soporte de NPAPI impactando en *plugin* para Silverlight, Java, Facebook Video y otros *plugins* similares que se basaban en esta misma arquitectura.

La finalidad de este estudio es conseguir una solución que permita interactuar a través un navegador con un dispositivo instalado en un equipo. Para ello a continuación se listan posibles alternativas que podrían dar con la solución.

5.1. Java Web Start

Java Web Start^[1] (JWS) es la alternativa más próxima a los *applets* ya que está creada por Oracle que es quien ha cambiado la perspectiva del uso de aplicaciones Java en la web. Esta tecnología permite a los usuarios arrancar aplicaciones desde la plataforma de Java directamente desde internet utilizando el navegador. Pero al contrario que los *applets* de Java esta tecnología no interacciona con el Navegador una vez que ha sido lanzada sino que directamente es la aplicación la que envía los datos al servidor.

Además puesto que utiliza el lenguaje de programación Java no sería muy costoso adaptar la aplicación (*applet*) que ya estaba creada, la cual ya interactúa con los dispositivos conectados en un equipo.

5.1.1. Funcionamiento

Las aplicaciones hechas para esta aplicación se encuentran alojadas en servidores web y se ejecutan por medio de enlaces puestos en páginas HTML a las aplicaciones Java. Cada uno de estos enlaces apunta a ficheros JNLP (*Java Networking Launching Protocol*) que indican la ruta de la aplicación en el mismo u otro servidor.

Java Web Start se ejecuta sobre una máquina virtual java como una aplicación de ventanas hecha con la biblioteca gráfica *swing*.

Una vez abierto el enlace, se descarga el fichero JNLP y se arranca automáticamente Java Web Start, comprueba la seguridad y si el usuario tiene la última versión instalada en su equipo; si no es así, la descarga y ejecuta.

Una vez obtenida la aplicación JWS se podrá ejecutar sin la necesidad de abrir el navegador.

5.1.2. Ejecución de la aplicación

- Desde un navegador: Clicar en un vínculo de una página Web.
- Desde un icono del escritorio: Si se utiliza una aplicación con frecuencia, se puede crear un acceso directo en el escritorio o en el almacenamiento del equipo. Java Web Start le preguntará si desea crear accesos directos o una entrada en el menú Inicio. Si se responde "sí", todas las ejecuciones posteriores de la aplicación se podrán iniciar sin necesidad de un navegador.
- Desde el Visualizador de la memoria caché de aplicaciones de Java: Java Web Start también proporciona un Visualizador de la memoria caché de aplicaciones, que puede ejecutar desde el Panel de control de Java. Este visualizador permite ejecutar directamente las aplicaciones descargadas.

Un pequeño ejemplo de su ejecución ha sido verificado en los siguientes navegadores:

- Chrome: Cuando se quiere ejecutar/descargar una aplicación creada para JAWAWS^[1] se redirecciona a la página de Oracle y te pide actualizar o verificar la versión existente, una vez instalada dice que Chrome no es compatible^[23].

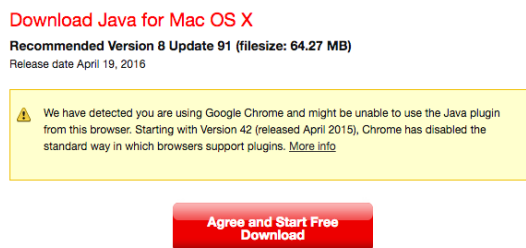


Figura 9. Antes de la instalación de la aplicación Java.

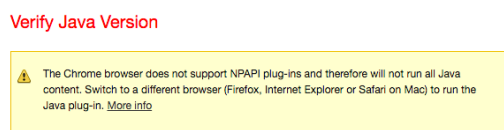


Figura 10. Una vez instalada la aplicación Java.

- Firefox: Al contrario en este otro navegador permite la descarga de la aplicación (por el momento) y su ejecución en el cliente.

5.1.3. Requisitos

Los requisitos necesarios para la ejecución en un cliente de las aplicaciones Java Web Start son los siguientes:

- Sistema Operativo: Windows 7, Linux, OS X.
- Tener instalada alguna máquina virtual Java superior a la versión 1.2.
- Tener instalada la última versión de java disponible.

- Configurar el navegador para que ejecute los ficheros con *mime-type* "application/x-java-jnlp-file" con Java Web Start (configuración por defecto).
- Conexión a Internet.

5.1.4. Ejemplo de Java Web Start

Véase el ejemplo creado en el anexo A. Uso de Java Web Start de la creación de una aplicación JWS y la comunicación mediante un "chat" entre un servidor Apache y un servidor Node Js brindado por una aplicación JWS.

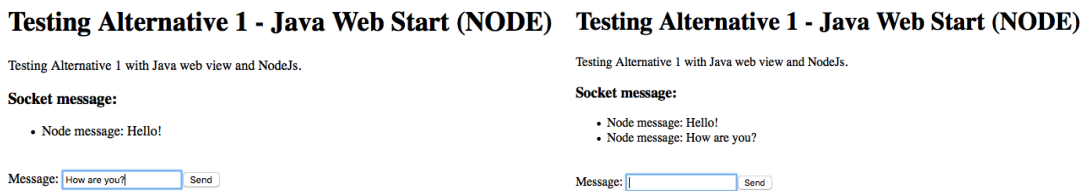


Figura 11. Ejemplo del chat entre los servidores Apache y Node Js.

5.1.5. Aspectos positivos

- Únicamente sería necesaria la adaptación del código Java del *applet* anterior a la tecnología JWS.

5.1.6. Aspectos negativos

- Tener que descargar y/o arrancar para cada interacción (inicialización de la tarjeta) el fichero .jnlp.
- Java en el navegador tiende a desaparecer por su falta de seguridad.
- Para pasar argumentos desde el navegador al fichero .jnlp tienen que ser argumentos estáticos, dos posibilidades:
 - Crear mediante PHP el fichero .jnlp.
 - Con Jsp, requiere de servidor Tomcat.
- Necesidad de un ordenador personal y permisos de administrador para ejecutar la aplicación.
- Se requiere la apertura de puertos para la comunicación de socket.io entre cliente y servidor Node Js.

5.2. Chrome.usb API

Google dispone de un API llamada *Chrome.usb* que permite interactuar con los dispositivos USB conectados al equipo. Mediante esta API las aplicaciones pueden funcionar como *drivers* para los dispositivos hardware.

Esta tecnología está actualmente en una fase de desarrollo muy temprana. Aunque la empresa Google está muy convencida de que ésta será la manera mediante la cual, en un futuro, los dispositivos conectados al equipo interactuarán con la web.

5.2.1. Ejemplo de creación

Aún no habiéndose verificado el funcionamiento del mismo, disponen de un tutorial muy completo en el apartado “*app_usb*” de desarrolladores de Chrome. Ese apartado corresponde al siguiente enlace: https://developer.chrome.com/apps/app_usb.

También hay un ejemplo de uso del mismo en el repositorio de Git Hub de Google Chrome accesible mediante en enlace: <https://github.com/GoogleChrome/chrome-app-samples/tree/master/samples/usb/device-info>.

5.2.2. Aspectos positivos

- Google es una fuente que ha mostrado confianza y seguridad.
- Uso de dispositivos conectados al equipo.
- No depende del sistema operativo.

5.2.3. Aspectos negativos

- No hay multiplataforma, al centrarse únicamente en el navegador Google Chrome.

5.3. Native Client

Native Client es un *sandbox* o caja de arena para ejecutar código nativo compilado de C y C++ en esa “caja de arena” de pruebas. Permite la ejecución segura de código nativo en un navegador web, además de extender las tradicionales tecnologías web aprovechando la potencia de cómputo de una máquina cliente. Es independiente del sistema operativo del usuario, permitiendo a las aplicaciones web ejecutarse casi a la misma velocidad que si la aplicación fuera nativa con la potencia total de la CPU,

incluyendo los vectores SIMD y el procesamiento multi-núcleo con memoria compartida.

Existen dos versiones de esta tecnología:

- PNaCl (*Portable Native Client*) extiende la tecnología con arquitectura independiente, deja a los desarrolladores compilar su código una vez para ejecutarlo en cualquier página web y en cualquier arquitectura con (AOT) *Ahead-of-time*.
 - Compilar el código en un ejecutable *bitcode*.
 - Traducir el código *bitcode* en un ejecutable del host específico tan pronto como se cargue el modulo antes de cualquier ejecución.
- NaCl: *Native Client* da rendimiento y el control de código nativo a bajo nivel en navegadores web modernos, sin sacrificar la seguridad y la portabilidad de la web.

5.3.1. Aspectos positivos

- Una de los mayores beneficios del lenguaje de programación C y la causa por la que es muy popular para aplicaciones embebidas, es que es un lenguaje estructurado de alto nivel pero también con capacidades de bajo nivel. La posibilidad de escribir código y conseguir acercarse al hardware es esencial y C dota esa habilidad.
- Uso de librerías del sistema que podrían acceder a dispositivos instalados en el equipo e interactuar con ellos.

5.3.2. Aspectos negativos

- El uso de llamadas del sistema únicamente funcionan en sistemas UNIX, como lo son Linux y OS X, por lo que no sería multiplataforma.

5.4. Edge (Silverlight)

Microsoft dispone una tecnología llamada Silverlight 5 que es capaz de llamar a código nativo incluido en las APIs de Win32. Esta tecnología es bien conocida y ha estado en .NET desde su primera versión, la 1.0. Llamar a APIs nativas requiere que previamente se haya indicado que la aplicación es de plena confianza. Es una restricción de seguridad, puesto que la API de Windows tiene muchas utilidades y puede dotar de acceso pleno al equipo. La aplicación ejecuta la llamada a *pInvoke* para invocar ese código nativo del sistema operativo.

Silverlight 5 permite que su aplicación sea de plena confianza y su uso tanto dentro como fuera del navegador, pero el uso de ésta API solo está disponible en Windows, no hay soporte para MAC.

5.4.1. Aspectos positivos

- Uso de llamadas al sistema para el uso de dispositivos conectados al equipo.
- Posibilidad de trabajar a través del navegador o en local.

5.4.2. Aspectos negativos

- No es multiplataforma, únicamente disponible para sistemas operativos Windows.

5.5. Smartphone - tablet (NFC)

Actualmente los teléfonos y tabletas inteligentes están siendo una herramienta muy extendida en el sector laboral por su cómoda portabilidad y uso de los mismos.

En concreto para este estudio su tecnología NFC es la necesaria. Gracias a esa tecnología y el desarrollo de una aplicación para Android podría ser una solución a la problemática surgida. La tecnología NFC (*Near Field Communication*) podría comunicarse con las tarjetas MIFARE para programarlas u obtener datos de ellas. También podría ser el mismo dispositivo móvil el que programe o lea información de otros dispositivos a través de NFC.

5.5.1. Aspectos positivos

- Desarrollo de la aplicación a partir del *applet* de Java, reescribir código y adaptarlo a Android.
- Eliminación del lector de tarjetas y posibilidad de uso de un teléfono móvil con tecnología NFC.

5.5.2. Aspectos negativos

- Disponer de un teléfono móvil con tecnología NFC.
- Portabilidad.
- No es multiplataforma, solo disponible para Android.

5.6. Análisis global de las alternativas

Cada una de ellas tiene sus ventajas y desventajas, pero ninguna de ellas ha conseguido satisfacer las necesidades del cliente, de manera que se ha buscado una alternativa más, ésta trata de añadir un componente hardware, el módulo creado por la empresa Espressif llamado ESP8266. El correspondiente estudio del mismo se recoge en el siguiente capítulo: 6. El módulo ESP8266.

Capítulo 6

6. El módulo ESP8266

6.1. Análisis del módulo ESP8266

El módulo ESP8266 es un microcontrolador que contiene la prestación de realizar comunicaciones inalámbricas mediante su conectividad Wi-Fi. Esta pequeña placa permite conectar microcontroladores a una red Wi-Fi y realizar comunicaciones mediante el protocolo TCP/IP. Existen diferentes versiones de la placa de este módulo con mayor o menor número de conexiones para diferentes aplicaciones.



Figura 12. Módulo ESP8266 versión ESP-12F.

Gracias a la posibilidad de realizar estas comunicaciones, se puede conectar un *transceiver* que interactúe con las tarjetas para enviar los datos al servidor. Además de cumplir con las necesidades del cliente las cuales se recogen en el apartado 7.1. Especificación y requisitos.

El bajo coste y las dimensiones de este módulo lo convierten en un módulo muy atractivo para su explotación.

6.1.1. Características principales

- Una CPU de 32-bit: Tensilica Xtensa LX106 con una frecuencia de reloj de 80 MHz.
- Memoria RAM de 64Kib para instrucciones y 96KiB para datos.
- Soporta QSPI flash externa desde 512KiB a 4 MiB y algunas hasta 16MiB.
- Protocolo IEEE 802.11 b/g/n.
- Wi-Fi Direct (P2P).
- 16 pins GPIO.

6.1.2. Ventajas y desventajas de las diferentes versiones del módulo ESP8266

- Ventajas
 - Posee una memoria flash en la que se puede almacenar contenido, pudiendo ser éste algún fichero HTML.
 - Posibilidad de almacenar el algoritmo de cifrado en el módulo.
 - Mejor antena para comunicación en zonas en la que la señal es débil o una ubicación con mucha sombra, versión E12-F.
 - *Pines* extra para:
 - Activación de *leds* de comprobación.
 - Alimentar mediante un decodificador un *display* mostrando las directrices a seguir.

- Desventajas
 - Más costosa las versión con NodeMCU.
 - Tiempo de envío inversamente proporcional al precio.
 - Necesidad de soldar los pines.

6.1.3. Tabla de tiendas dónde comprar (ordenadas por prioridad)

Módulo	Precio/unidad	Tiempo de envío	Web	Producto
ESP8266 ESP-12F con microUSB	4,58€*	13 días	dx.com	Ver
ESP8266 ESP-12E con placa NodeMCU	13,90€*	3-8 días	ebay.es	Ver
ESP8266 ESP-12E con placa NodeMCU Flash 4MiB	3,16€*	15-45 días laborables	aliexpress.com	Ver
ESP8266 ESP-12F	6,20€*	3-8 días	ebay.es	Ver
ESP8266 ESP-12E	7,70€*	Amazon Premium desde china	amazon.es	Ver
ESP8266 ESP-12F	2,81€*	11-27 días laborables	banggood.com	Ver
ESP8266-12E con placa NodeMCU	8,07€*	5-7 días laborables	banggood.com	Ver

Tabla 1. Tabla comparativa de las diferentes tiendas, precios y tiempo de envío del módulo ESP8266.

*Los precios pueden llegar a variar, datos recogidos a fecha de 08/02/2016.

6.1.4. Conclusión del análisis

Se recomiendan las versiones ESP-12E o ESP-12F del módulo ESP8266. El resto de versiones tiene un precio similar y limitan mucho la arquitectura a utilizar. Un módulo con más pines (aparte de los cuatro de Vcc, GND RX y TX) permite mostrar señales de verificación así como mensajes en un *display*.

6.2. Inicialización en el módulo ESP8266

Este apartado del módulo EP8266 recoge la información necesaria para interconectar el módulo ESP8266 a un equipo con un sistema operativo Windows 10 u OS X “El Capitán”. Mediante una aplicación, que se detalla más adelante, se realizan comunicaciones mediante el puerto serie con el módulo para obtener y/o modificar información del mismo.

También, en este apartado, se recoge el estudio que describe los pasos necesarios para establecer el entorno de desarrollo en los IDEs de Eclipse y Arduino, junto con el SDK que proporcionan los fabricantes del módulo ESP8266.

Las pruebas y ejemplos recogidos en este apartado del módulo ESP8266 están realizadas en los sistemas operativos Microsoft Windows 10 y Mac OS X El Capitán.

6.2.1. Herramientas

A continuación se listan las herramientas y módulos utilizados en el informe para realizar la comunicación serie y establecer el entorno de desarrollo:

- Módulo ESP8266 ESP-12F.
- Conversor de serie a USB: UART PL2303.
- Driver PL2303:
 - Windows 10:
 - <https://drive.google.com/file/d/0B3nCzPJqtgWvX3LzRS1RbXpjWnM>
 - MAC OS X El Capitán:
 - <https://drive.google.com/file/d/0B3nCzPJqtgWveVlzUjBmS1U5Mjg>
- Terminal para comunicación por puerto serie:
 - Windows 10: Realterm.
 - <https://sourceforge.net/projects/realterm/files/Realterm/2.0.0.70/>
 - MAC OS X El Capitán: Coolterm.
 - <https://drive.google.com/file/d/0B3nCzPJqtgWvdmvtvOXQxRnNUaU0/>
- SDK Espressif v2.0.9 (incluye RTOS V1.3.0):
 - <https://drive.google.com/open?id=0B3nCzPJqtgWvMUJrVmlYLU9JeG8>
- SDK Espressif v2.1.0 (incluye RTOS V1.4.0):
 - <https://drive.google.com/open?id=0B3nCzPJqtgWvOeZ2RmN4M0RUZIE>

- Compilador GCC para Windows: MinGW.
 - <https://drive.google.com/file/d/0B3nCzPJqtgWvVkeE0aGM2bUhOXzQ>
- Eclipse + CDT:
 - <http://www.eclipse.org/downloads/packages/eclipse-ide-cc-developers/mars1>
 - <https://www.eclipse.org/cdt/>
- Arduino:
 - <https://www.arduino.cc/en/Main/Software>
- Esptool:
 - <https://github.com/themadinventor/esptool>

6.2.2. Conexión del módulo ESP8266

La instalación de los drivers del convertor UART de serie a USB se detalla en el apartado: 6.2.3. Instalación del driver del módulo UART PL2303.

6.2.2.1. Conexión para el envío de comandos AT mediante puerto serie

Conectar el módulo ESP8266 siguiendo el esquema que aparece a continuación para realizar comunicaciones mediante el puerto serie a través del módulo UART PL2303.

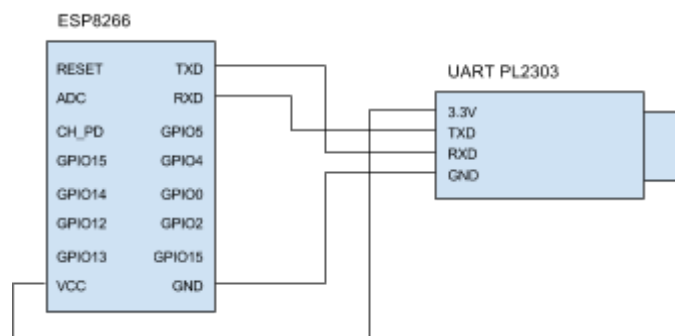


Figura 13. Esquema de conexión para el envío de comandos.

A continuación, para enviar comandos AT al módulo ESP8266, conectar el módulo UART al equipo y configurar la terminal siguiendo los pasos detallados en el apartado: 6.2.4. Comunicación con la terminal a través del puerto serie.

Los comandos AT soportados por el módulo ESP8266 se detallan en el documento oficial de Espressif^[3] y una prueba de los mismos en el apartado: 6.2.5. Prueba inicial con comandos AT.

6.2.2.2. Modo *flash*

Conectar el módulo ESP8266, siguiendo el esquema que aparece a continuación, para *flashear*¹ la memoria EEPROM del módulo ESP8266.

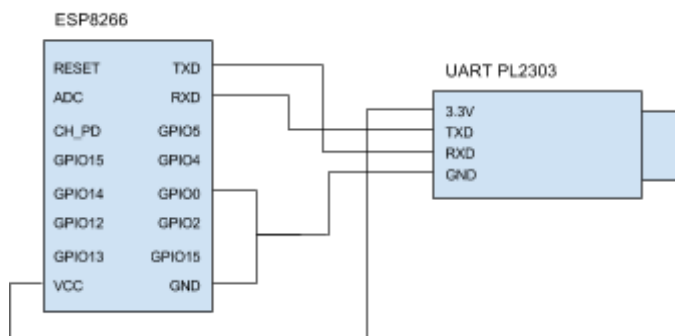


Figura 14. Esquema de conexionado modo flash.

6.2.2.3. Modo *boot* desde la memoria *flash*

Conectar el módulo ESP8266, siguiendo el esquema que aparece a continuación, para iniciar el programa cargado en la memoria EEPROM del módulo ESP8266.

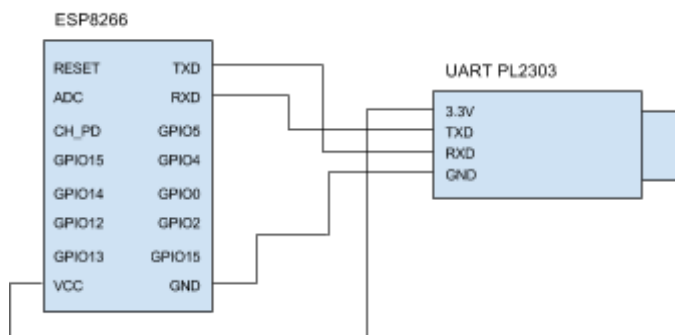


Figura 15. Esquema de conexionado modo boot desde la memoria flash.

¹Flashear: Cargar un programa en una memoria *flash*.

6.2.3. Instalación del driver del módulo UART PL2303

6.2.3.1. Windows 10

Conectar el módulo UART PL2303 al equipo e instalar el driver correspondiente que se recoge en el apartado: 6.2.1. Herramientas. Una vez instalado dirigirse al administrador de dispositivos a través de la barra de búsqueda de Windows como aparece en la siguiente figura.

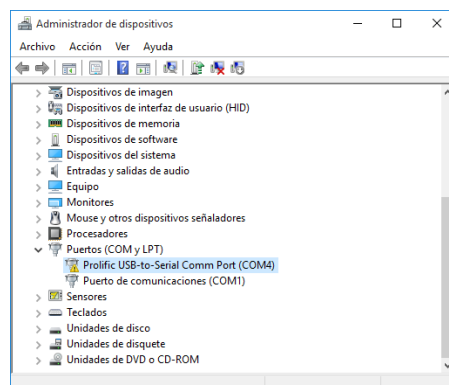


Figura 16. Administrador de dispositivos.

En el apartado Puertos (COM y LPT) clic derecho en *Prolific USB-toSerial Comm Port (COMx)* → *Actualizar software del controlador...* → *Buscar software de controlador en el equipo* → *Elegir en una lista de controladores de dispositivo en el equipo* y escoger el controlador de la siguiente figura y aceptar.



Figura 17. Controlador para el módulo UART PL2303.

Los pasos realizados también se recogen en un vídeo tutorial^[4].

6.2.3.2. MAC OS X El capitán

Conectar el módulo UART PL2303 al equipo e instalar el driver correspondiente que se recoge en el apartado: 6.2.1. Herramientas.

Una vez instalado para comunicarse con el módulo dirigirse al apartado: 6.2.4. Comunicación a través del puerto serie, concretamente al apartado 6.2.4.2. MAC OS X El capitán.

6.2.4. Comunicación con la terminal a través del puerto serie

6.2.4.1. Windows 10

Para establecer una comunicación con el módulo ESP8266 a través del puerto serie con el conversor UART PL2303, se requiere tener instalados los drivers para módulo UART que se recogen en el apartado: 6.2.3. Instalación del driver del módulo UART PL2303, concretamente en el apartado 6.2.3.1. Windows 10.

A continuación, instalar y abrir la aplicación RealTerm, con el conversor UART conectado, que se recoge en el apartado: 6.2.1. Herramientas. Configurar la aplicación RealTerm de la siguiente manera:

- En la pestaña Port (como aparece en la figura siguiente):
 - Baud: 115200.
 - Port: (el puerto que se le asigna al conversor UART, éste se puede obtener en el apartado de configuración de administrar dispositivos).
 - *Open* activado.
 - *Change* activado.

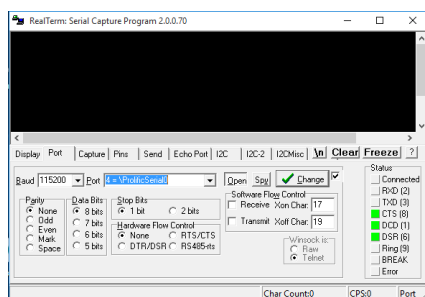


Figura 18. Configuración del puerto en RealTerm.

- En la pestaña *send* (como aparece en la figura siguiente):
 - EOL: Seleccionar +CR (*carriage return*) y +LF (*end of line*).
 - En el izquierdo escribir el comando a enviar.
 - Send ASCII.

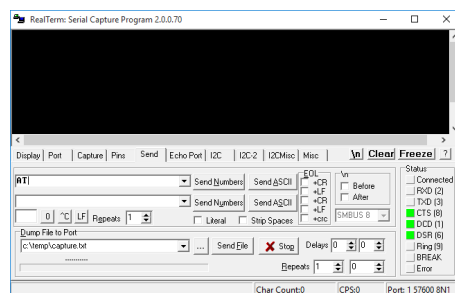


Figura 19. Envío de comandos con RealTerm.

6.2.4.2. MAC OS X El capitán

Para establecer una comunicación con el módulo ESP8266 a través del puerto serie con el conversor UART PL2303, se requiere tener instalados los drivers para módulo UART que se recogen en el apartado 6.3.3. Instalación del driver del módulo UARTPL2303, concretamente en el apartado de 6.2.3.2. MAC OS X El capitán.

A continuación, abrir la aplicación CoolTerm, con el conversor UART conectado, que se recoge en el apartado: 6.2.1. Herramientas. Configurar la aplicación CoolTerm de la siguiente manera:

- En el apartado *option* del menú superior (como aparece en la figura):
 - Port: usbserial.
 - Baudrate: 115200.

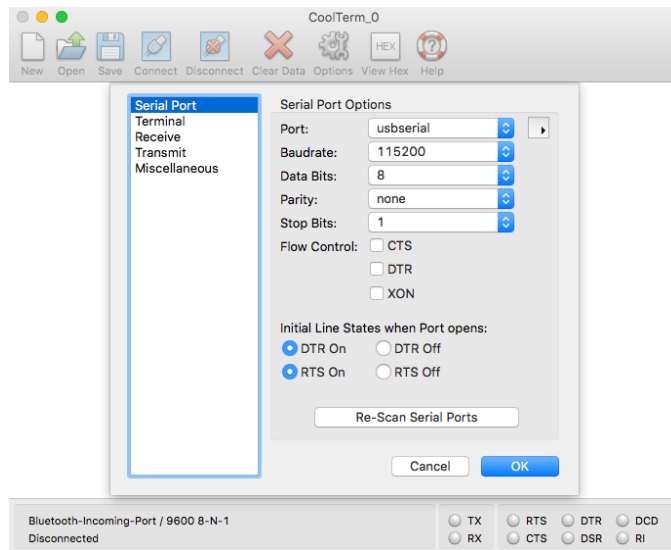


Figura 20. Configuración de la aplicación CoolTerm.

Aceptar la nueva configuración con *OK* y en el menú superior clicar en *Connect*. A partir de este momento ya se pueden realizar envío y recepción de datos.

6.2.5. Prueba inicial con comandos AT

Una vez establecida la comunicación entre el equipo y el conversor UART, recogido el proceso en el apartado: 6.2.4. Comunicación con la terminal a través del puerto serie, se han ejecutado comandos AT para cambiar la configuración del módulo ESP8266 y observar las aplicaciones que tiene. Un ejemplo de los comandos AT utilizados se recoge en el anexo: C. Comandos AT.

Se pueden ver los comandos en el libro ESP8266^[2] en las páginas 46 – 50 y en la documentación de Espressif de los Comandos AT para el módulo ESP8266^[3].

6.2.6. Eclipse Espressif SDK

6.2.6.1. Instalación del SDK, MinGW y Eclipse + CDT

Instalar las aplicaciones siguientes desde el apartado 6.2.1. Herramientas:

- Eclipse para desarrolladores C/C++ junto con el *plugin* CDT.
- SDK de Espressif Versión 2.1.0 (incluye RTOS v1.4.0).
- El compilador GCC para Windows, MinGW.

Seguir las indicaciones de instalación de cada una de la aplicaciones. Para instalar el *plugin* de CDT se puede realizar la instalación mediante el fichero ZIP o mediante el *Eclipse Marketplace de Eclipse* del menú *Help*.

Una vez instaladas ejecutar la aplicación *MinGW Intallation Manager* y seleccionar los paquetes que aparecen en la siguiente figura y aplicar los cambios en *Installation* → *Apply changes*.

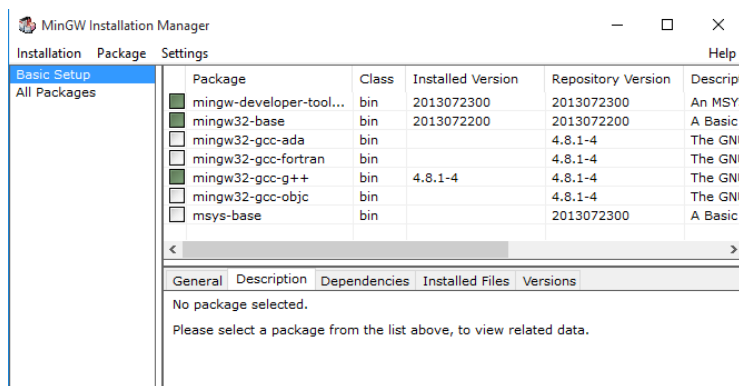


Figura 21. Paquetes MinGW.

Terminado el paso anterior se dispondrá de todo lo necesario para compilar y *flashear* el módulo ESP8266.

6.2.6.2. “Hola mundo” en eclipse

Abrir eclipse e importar el proyecto de ejemplo “*Hello Word*” que está en la carpeta *examples* del junto con el SDK de Espressif instalado: *File* → *Import...* → *General* → *Existing Project into Workspace* → *Select Root Directory* → *Browse* → *C:\Espressif\examples\hello_world*.

Para compilar y *flashear* la aplicación ir al apartado 6.2.6.4. *Flashear* en eclipse.

6.2.6.3. Ejemplo de un Servidor Web en eclipse

Abrir eclipse e importar el proyecto de ejemplo “*esphttpd*” que está en la carpeta *examples* del junto con el SDK de Espressif instalado: *File* → *Import...* → *General* → *Existing Project into Workspace* → *Select Root Directory* → *Browse* → *C:\Espressif\examples\esphttpd*.

Para compilar y *flashear* la aplicación ir al apartado 6.2.6.4. *Flashear* en eclipse.

6.2.6.4. *Flashear* en eclipse

Dentro del proyectos de ejemplo anteriores, abrir el fichero *Makefile* y cambiar la configuración del puerto de comunicación con la que el sistema operativo le establece al módulo UART y establecer el módulo en modo *flash* que se recoge en el apartado: 6.2.2.2. Modo *flash*.

A continuación en la pestaña *Make Target* (de la ventana que está por defecto en el lado derecho) realizar las operaciones *clean* y *flash*.

Consejo: Asegurarse de que el puerto coincide con el del fichero de comunicación, está desconectado de cualquier otra comunicación y el módulo está preparado en modo *flash*, si no permite el *flashear* desconectar el módulo UART, volver a conectarlo y volver a intentarlo, no siempre *flashea* a la primera.

Makefile RTOS v1.4.0:

- <https://drive.google.com/file/d/0B3nCzPJqtgWvbFY2U3dTVVAzaWc>

6.2.7. *Flasheo* del módulo ESP8266 desde Unix

Uno de los métodos para *flashear* el módulo ESP8266 se realiza con la ayuda de la librería *esptool*, disponible en el apartado 6.2.1. Herramientas, y el fichero copiado *.bin* de la aplicación que se desea cargar.

Para ello desde una máquina Unix:

- Descargar la librería *esptool* del repositorio Github.
- Instalar la librería como lo indica el enlace anterior o simplemente con el siguiente comando en la terminal, situados en la raíz de la librería descarga:
 - `python ./setup.py install`
- Ejecutar el siguiente comando en la terminal para flashear el módulo ESP8266:
 - `python ./esptool.py -p /dev/tty.usbserial write_flash 0x00 ./fichero_compilado.bin`

6.2.8. Arduino

6.2.8.1. Instalación y configuración Arduino para ESP8266

Instalar Arduino desde su página web que se recoge en el apartado 6.2.1. Herramientas.

Una vez instalado ha de configurarse Arduino para compilar y *flashear*. Para ello acceder a las preferencias. *File* → *Preferences* en sistemas Windows o *Arduino* → *Preferences...* en sistemas OS X.

Rellenar el campo *Additional Boards Manager URLs* con la siguiente información, para obtener el paquete ESP8266 para Arduino: http://arduino.esp8266.com/stable/package_esp8266com_index.json

Additional Boards Manager URLs:

Figura 22. Campo *Additional Boards Manager URLs*.

Seleccionar el gestor de placas base en *Tool* → *Board* → *Boards Manager...* e instalar el soporte para el módulo ESP8266. Para ello en la barra de búsqueda escribir *Generic ESP8266 Module* e instalar clicando en *Install*. Este documento se ha basado en la versión 2.0.0.

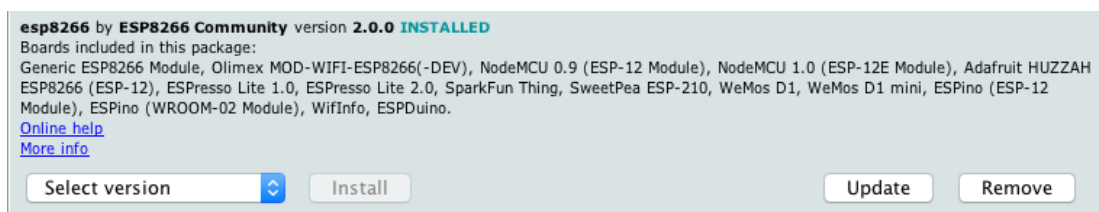


Figura 23. Instalación de la placa base.

A continuación seleccionar la placa instalada en *Tools* → *Board* → *Generic ESP8266 Module*. Configurar con la información que se muestra en la siguiente figura.

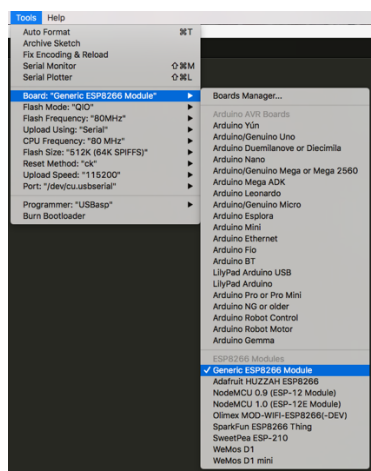


Figura 24. Configuración de la placa base.

Por último, es necesario editar el fichero de configuración de la placa ESP8266 para que *flashee* la aplicación con la librería *esptool*. Este fichero llamado *platform.txt* se encuentra en la ruta:

OS X: /Users/USER/Library/Arduino15/packages/esp8266/hardware/esp8266/2.0.0

Comentar la línea que se parece a:

```
#tools.esptool.upload.pattern="{path}/{cmd}" {upload.verbose} -cd {upload.resetmethod} -cb
{upload.speed} -cp "{serial.port}" -ca 0x00000 -cf "{build.path}/{build.project_name}.bin"
```

Y añadir la ruta donde está instalada la librería *esptool.py*, recogido el proceso de instalación en el apartado 6.2.7. *Flasheo* del módulo ESP8266 desde Unix Por ejemplo:

```
tools.esptool.upload.pattern="/Users/USER/Universidad/PGF - Proyecto Fin De
Grado/PGF.Alternativa6.ESP8266/esptool-master/esptool.py" --port "{serial.port}" write_flash
0x00000 "{build.path}/{build.project_name}.bin"
```

Una vez configurado se dispondrá de todo lo necesario para compilar y flashear programas, como por ejemplo:

- Ver apartado 6.2.8.2. “Hola mundo” en Arduino
- Ver apartado 6.2.8.3. Servidor Web en Arduino

6.2.8.2. “Hola mundo” en Arduino

Abrir Arduino y escribir el siguiente ejemplo:

```
void setup() {
    Serial1.begin(115200);
}
void loop() {
    Serial1.println("Hello! - millis() = " + String(millis()));
}
```

Cargar el programa en el módulo ESP8266 como se indica en el apartado: 6.2.8.4. *Flashear* desde Arduino.

6.2.8.3. Servidor Web en Arduino

Abrir *Arduino* → *File* → *Examples* → *ESP8266WebServer* → *AdvanceWebServer*

Cargar el programa en el módulo ESP8266 como se indica en el apartado: 6.2.8.4. *Flashear* desde Arduino.

6.2.8.4. Flashear desde Arduino

Para *flashear* un programa primero se ha de verificar para compilar el fichero en *Verify* y a continuación para cargar el fichero compilado en el módulo ESP8266 en *Upload*:

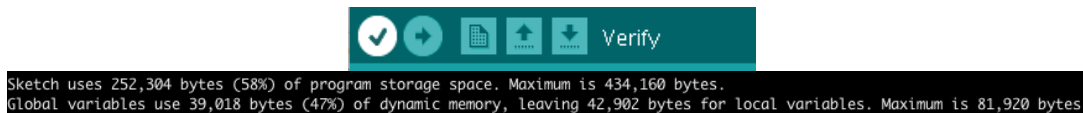


Figura 25. Verificación y compilación de la aplicación.

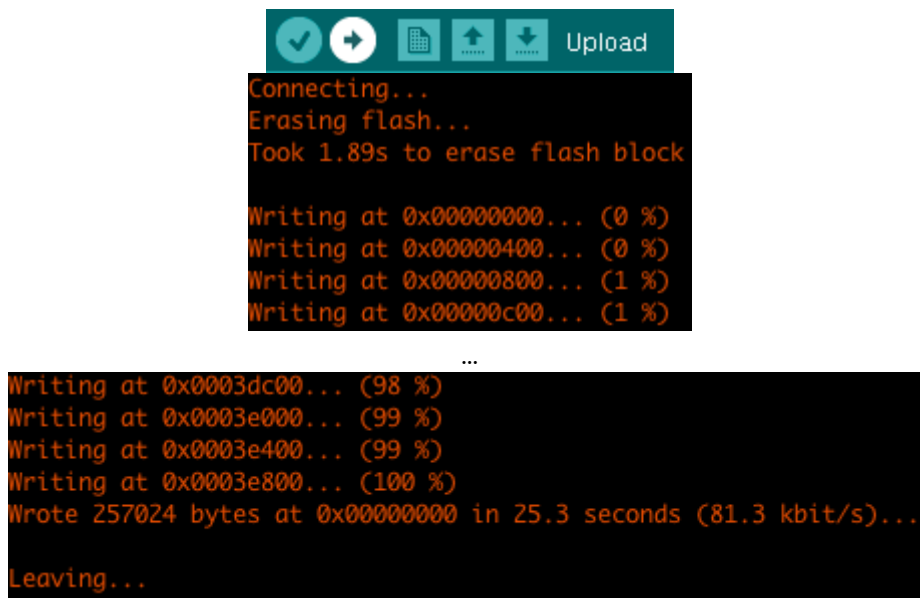


Figura 26. Carga (flasheo) de la aplicación.

6.2.9. Eliminar la memoria flash

Con la librería *esptool*, disponible en el apartado 6.2.1. Herramientas junto con el puerto donde está conectado el módulo ESP8266 ejecutar la función *erase_flash* para eliminar todo el contenido:

- `./esptool-master/esptool.py --port /dev/tty.usbserial erase_flash`

Nota: El modulo tiene ha de estar en modo flash como se indica en el apartado:

6.2.2.2. Modo *flash*.

6.3. Análisis del uso de los diferentes SDKs

A continuación se listan las ventajas y desventajas del uso de los SDK de Espressif en su versión de Non OS SDK y RTOS SDK.

- Non OS SDK utiliza temporizadores y *callbacks* para realizar varias funciones, eventos añadidos, o funciones desencadenadas por varias condiciones.
- La utilización de un Non OS SDK el cual es mono hilo no debe realizar tareas que requieran mucho tiempo de ejecución, está pensado para realizar tareas muy puntuales:
 - Si una tarea requiere el uso de la CPU por mucho tiempo, el módulo ESP no puede realizar otras, esto produce un aviso *watchdog* y se reinicia.
 - Si las interrupciones están deshabilitadas, la CPU sólo puede estar ocupada por nuestra ejecución y el tiempo no debe superar los 10us.
 - Si las interrupciones no están deshabilitadas, se sugiere que la CPU no se ocupe más de 500 ms.
- Se sugiere la utilización de un reloj que realice comprobaciones periódicamente, además si el desarrollador necesita llamar a la función de espera (*delay*) o a una función *while* o *for* no se ha de ocupar la CPU más de 15 ms.
- Usando un sistema **Non OS SDK**, **no se puede llamar** a ninguna función definida con ***ICACHE_FLASH_ATTR*** en el manejo de interrupciones, todas las funciones que se llamen en una interrupción deben de estar declaradas en la RAM.
- Es recomendable el uso de un sistema **RTOS SDK**, para programar **diferentes tareas** si cada una de ellas va a requerir de intervalos muy grandes de tiempo.
- Si se requiere la impresión de *logs*, utilizar la función del API *os_printf* y no añadir demasiadas impresiones en las interrupciones. Si hay demasiadas ocupando la CPU por mucho tiempo, pueden ocurrir errores.
- **RTOS SDK utiliza un sistema FreeRTOS** que es un sistema operativo multitarea. Puede utilizar interfaces estándar para la gestión de recursos FreeRTOS, operaciones de reciclaje, retraso de ejecuciones, mensajes entre tarea y la sincronización, un diseño de proceso orientado a la tarea.
- RTOS utiliza el API *lwIP* para operar en red. En el RTOS SDK el API SD Socket está encapsulado. Los usuarios pueden desarrollar aplicaciones software del mismo modo que lo hacen con el API socket o compilar las aplicaciones

de Socket estándar para ejecutarlas es otras plataformas. Esto ayuda a reducir los costes al cambiar de una plataforma a otra.

- **RTOS SDK** también contiene una librería **JSON**, la cual se recomienda el uso de éstas para una utilización más sencilla del *parseo* de paquetes JSON para facilitar el diseño del protocolo de comunicación.
- **RTOS SDK es compatible con Non OS SDK en la interfaz Wi-Fi**, interfaces de configuración inteligentes, interfaces para esnifar, interfaz del sistema, la interfaz de temporizador, interfaces FOTA y la interfaz de controlador periférico, pero no admite la ejecución de comandos AT.

<<Non OS SDK está diseñado para tareas muy puntuales o simplemente gestión de Wi-Fi (*access point*). Si tiene que haber una app en un nivel superior con un nivel de carga elevado, falla.>> **Iván Piquer.**

6.4. Creación de un proyecto en eclipse

A continuación se indican los pasos necesarios para preparar el entorno de programación en la herramienta Eclipse

Crear un proyecto C en eclipse con el compilador *MinGW GCC* instalado anteriormente en el apartado 6.2.1. Herramientas

File → *New* → *C Project* → establecer un nombre de proyecto, seleccionar *Empty Project* y *MinGW GCC*.

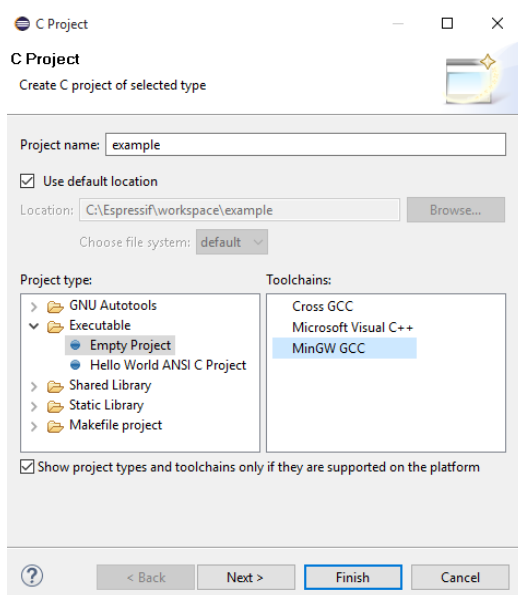


Figura 27. Creación del proyecto C.

A continuación en la raíz del proyecto crear dos carpetas con los siguientes nombres y dos ficheros uno en *user* y otro en *include*:

- user/
 - user_init.c
- include/
 - user_config.h

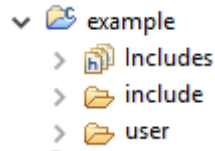


Figura 28. Carpetas necesarias para el proyecto.

Una vez creados, se importan los *includes* y librerías necesarias para compilar el código fuente en C para el módulo ESP8266. Para ello:

- Botón derecho en el proyecto → Properties → C/C++ General → Paths and Symbols → Includes → GNU C.
- Añadir los *includes* necesarios, para la utilización de cada SDK de Espressif (en este caso se añaden las librerías para utilizar un SDK RTOS):
 - C:\Espressif\ESP8266_RTOS_SDK\include
 - C:\Espressif\ESP8266_RTOS_SDK\include\espressif\
 - C:\Espressif\ESP8266_RTOS_SDK\include\freertos\
 - C:\Espressif\ESP8266_RTOS_SDK\include\json\
 - C:\Espressif\ESP8266_RTOS_SDK\include\ssl\
 - C:\Espressif\ESP8266_RTOS_SDK\include\lwip\
 - C:\Espressif\ESP8266_RTOS_SDK\include\lwip\ipv4\
- Añadir las librerías (*Library Paths*) compiladas de Espressif:
 - C:\Espressif\ESP8266_RTOS_SDK\lib

Configurar el constructor para su compilación, *builder*:

- Botón derecho en el proyecto → Properties → C/C++ Build → Tool Chain Editor → Current builder → Escoger: Gnu Make Builder.
- Botón derecho en el proyecto → Properties → C/C++ Build → Builder Settings:
 - Deseleccionar la pestaña *User default build command* y en el campo *Build command* añadir lo siguiente:
 - mingw32-make.exe -f \${ProjDirPath}/Makefile

- Deseleccionar la pestaña *Generate Makefiles automatically* y en el campo *Build directory* añadir lo siguiente:
 - `${workspace_loc:/my_project}`
- Aplicar los cambios, *Apply*.

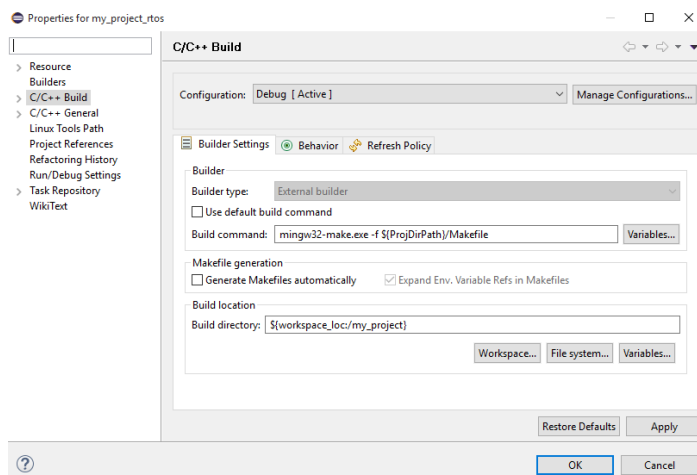


Figura 29. Configuración del builder.

Por último, añadir el fichero *Makefile* siguiente y crear los *Make Target* necesarios del fichero *Makefile*. Este proyecto se ha basado en la versión del SDK RTOS v1.4.0.

Añadir el fichero *Makefile* en la raíz del proyecto:

- Para el SDK RTOS v1.4.0. Fichero:
<https://drive.google.com/file/d/0B3nCzPJqtgWvbFY2U3dTVVAzaWc>
- Para el SDK RTOS v1.3.0. Fichero:
<https://drive.google.com/file/d/0B3nCzPJqtgWvdmtSZU55UVFNaDg>

En la vista de la derecha *Make Target* en el icono de *New Make Target* crear lo siguientes *Make Targets*: *flash* y *rebuild*.

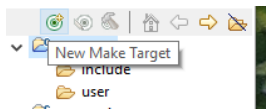


Figura 30. Crear un nuevo Make Target.

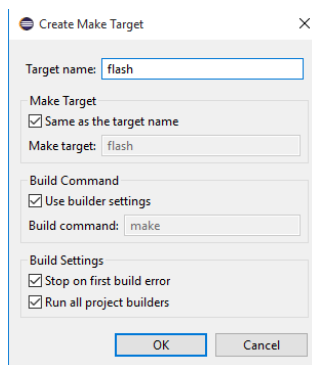


Figura 31. Configuración del Make target a crear.

Una vez finalizado este paso el proyecto estará preparado para compilar y *flashear* el módulo ESP8266, para ello ir a la apartado: 6.2.6.4. *Flashear* en eclipse.

6.5. Identificación de las librerías requeridas

Se ha realizado una búsqueda de información sobre las distintas API que ofrece Espressif del módulo ESP8266 antes de empezar a diseñar e implementar de los casos de uso analizados en el apartado: 7.2.1. Casos de uso.

Entre ellas se han detectado las siguientes:

- Sistema operativo en tiempo real para microcontroladores:
 - FreeRTOS.
- Conectividad WI-FI:
 - WI-FI API.
- Comunicación RES como cliente o servidor:
 - ESPCONN API.
 - ESP-NOW API.
 - MESH API.
- Comunicación TCP-SSL a través de sockets:
 - ESPSSL API
- Parseo de datos de la respuesta del servidor JSON:
 - cJSON API.

A fin de comprender mejor el funcionamiento y llamadas de las APIs listadas anteriormente se han realizado un ejemplo con cada una de ellas que se recogen en el anexo: B. Ejemplos de los APIs del SDK RTOS de Espressif.

Capítulo 7

7. Desarrollo del software

Este capítulo recoge toda la documentación relacionada con el proceso para el desarrollo del software de la aplicación que sustituye el *applet* de Java.

7.1. Especificación y requisitos

En este apartado se listan los requisitos y la especificación obtenida en base a las necesidades que ha planteado el cliente.

El módulo ESP8266 de fábrica tiene que crear una red Wi-Fi mediante la cual poder comunicarse con él para enviarle los parámetros de la configuración inicial que la aplicación requiera en base a las nuevas funcionalidades que se le vayan añadiendo.

Por otro lado, el módulo ha de conectarse a internet a través de una estación Wi-Fi con un protocolo de encriptación como mínimo de WPA2, para poder comunicarse con un servidor a través de una arquitectura REST mediante el protocolo de comunicación TCP SSL. A continuación, se ha de verificar la autenticidad del dispositivo que se conecta al servidor, para ello se establecen un usuario y contraseña.

En cuanto a la lectura de las tarjetas, ésta dispondrá de dos opciones: únicamente la lectura del número de serie de la misma o la lectura del número de serie y la lectura de los datos que contiene. Para poder acceder los datos que contiene cada tarjeta se requiere de la clave que las protege bajo un algoritmo de cifrado, esta clave también tendrá que ser un parámetro más que se indique en la configuración inicial. Además se establecerá un tiempo máximo de espera para posar la tarjeta establecido en la configuración inicial.

El módulo ESP8266 preguntará al servidor por acciones de lectura que tenga pendientes por realizar. Estas peticiones pueden realizarse instantáneamente si acaba de realizar una de ellas o con un tiempo de ventana establecido en la configuración inicial si la petición anterior no tenía ninguna acción disponible para realizar.

La configuración inicial consta de las direcciones IP, la estación Wi-Fi a la que tiene que conectarse, la dirección del servidor al que se realiza las peticiones junto con el usuario y contraseña con los que se autentica. Además, otros parámetros como el tiempo máximo de espera para posar la tarjeta, el tiempo de espera entre petición y petición si no hay acciones pendientes y la clave de las tarjetas para poder leer los datos que contienen.

7.2. Diseño

Una vez identificados y analizados los requisitos obtenidos del cliente se procede al diseño de los casos de uso relacionados con el análisis realizado, así como al diseño de la arquitectura y el protocolo de comunicación.

7.2.1. Casos de uso

A partir del análisis realizado anteriormente se han obtenido los siguientes casos de uso.

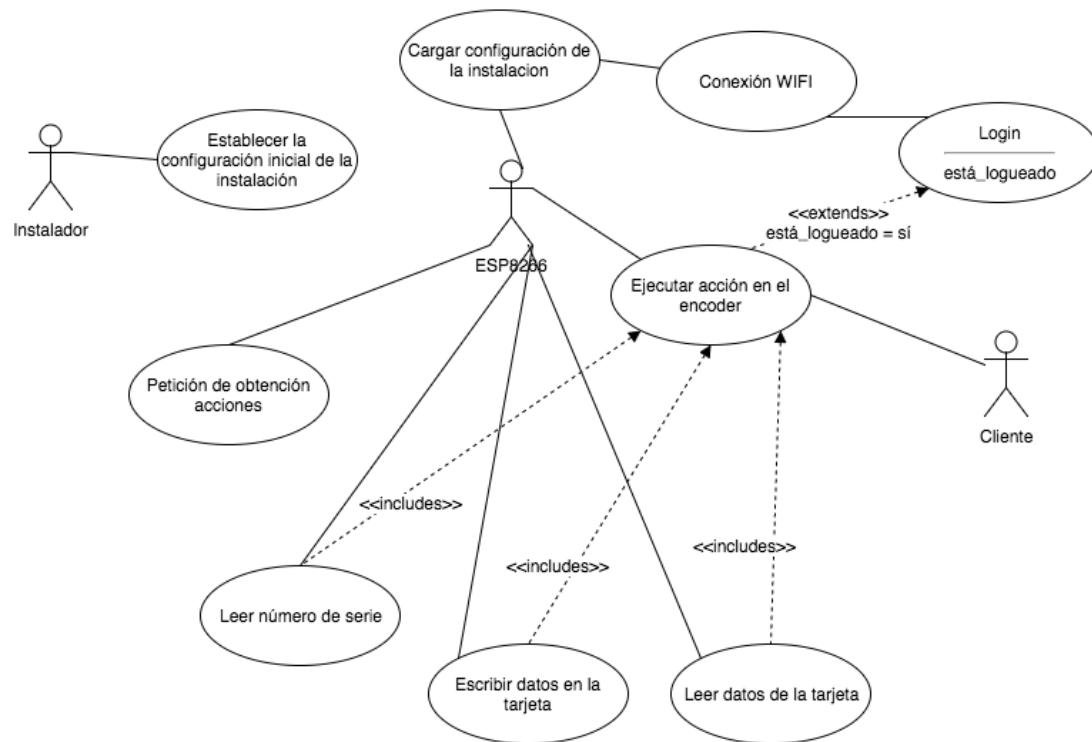


Figura 32. Casos de uso.

7.2.1.1. Caso de uso: Establecer la configuración inicial de la instalación

La configuración del módulo ESP8266 es el caso de uso que se debería realizar en primera instancia ya que es necesaria para que los siguientes casos de uso la utilicen. Tanto para la conexión a internet y al servidor remoto como para la configuración del *encoder* que interacciona con las tarjetas *smart card*.

Nombre	Establecer la configuración inicial de la instalación.
Actores	Instalador y ESP8266.
Descripción	Se establecen los valores que requiere la instalación para la que se va a conectar el módulo ESP8266.
Precondición	El modulo tiene que estar reseteado de fábrica.
Flujo normal	<ol style="list-style-type: none"> 1. El instalador se conecta a la red Wi-Fi que crea el ESP8266 “ESP_MAC”. 2. El instalador realiza una llamada POST con los datos de la instalación. 3. El sistema recoge los datos de la llamada POST y los almacena en la memoria <i>flash</i>. 4. El sistema una vez almacenado los datos se reinicia.
Flujo alternativo	Ninguno.
Postcondición	La configuración cargada tiene que ser correcta.

Tabla 2. Caso de uso establecer la configuración inicial de la instalación.

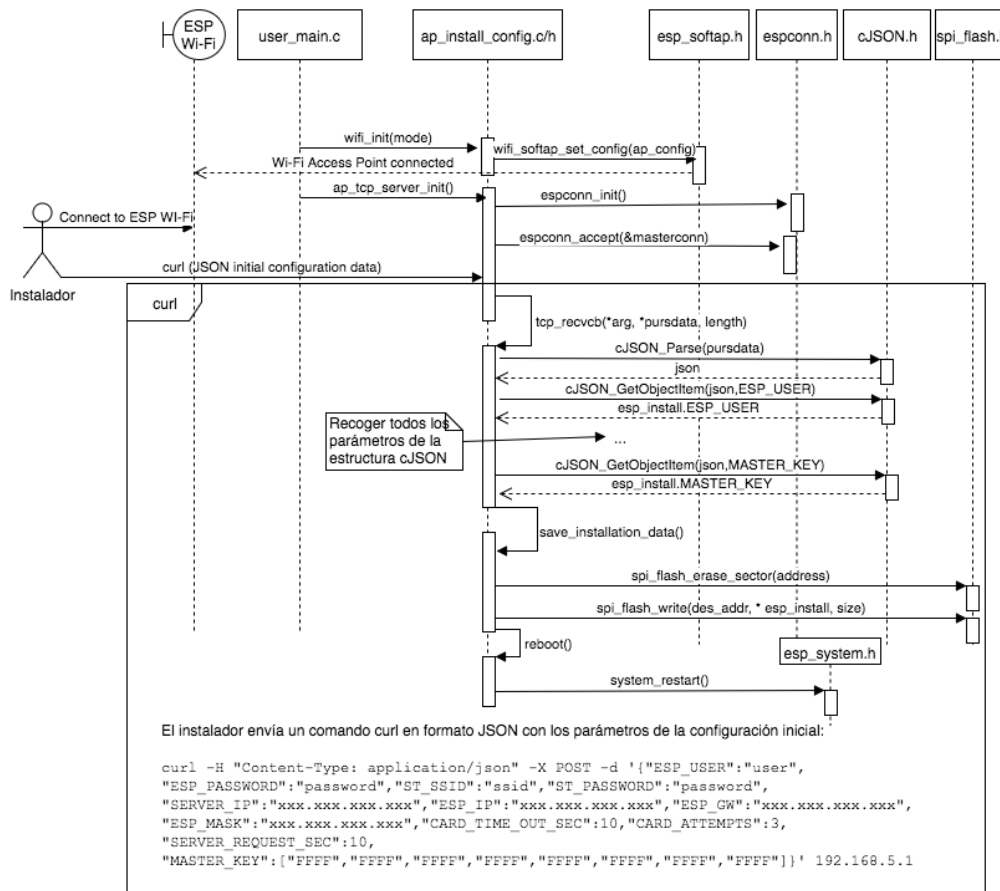


Figura 33. Flujo de eventos establecer la configuración inicial de la instalación.

7.2.1.2. Caso de uso: Cargar configuración inicial de la instalación

Este caso de uso se encarga de cargar la configuración de la instalación que previamente el instalador ha cargado en la memoria flash del módulo ESP8266.

Nombre	Cargar configuración inicial de la instalación.
Actores	ESP8266.
Descripción	En cada inicio del módulo ESP8266 carga la configuración almacenada en la memoria <i>flash</i> .
Precondición	La configuración inicial tiene que estar almacenada en la memoria flash una vez realizado el caso de uso: 7.2.1.1. Caso de uso: Establecer la configuración inicial de la instalación
Flujo normal	1. Una vez conectado a una fuente de alimentación (USB), el módulo ESP8266 se inicia cargando los datos almacenado en la memoria flash. Continuar.
Flujo alternativo	Ninguno.
Postcondición	La configuración se carga exitosamente.

Tabla 3. Caso de uso cargar configuración inicial de la instalación.

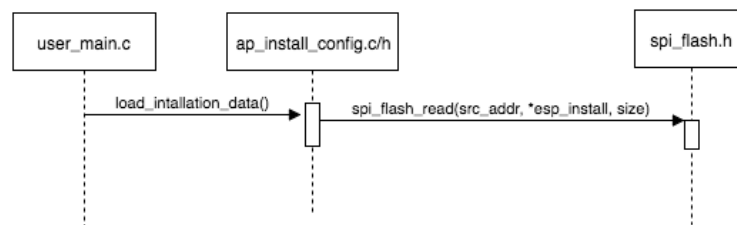


Figura 34. Flujo de eventos cargar configuración inicial de la instalación.

7.2.1.3. Caso de uso: Conexión Wi-Fi y puesta en marcha del cliente de peticiones

Este caso de uso se encarga de conectarse a una infraestructura de red Wi-Fi. Una vez obtenida la dirección IP crea el servidor de peticiones e intenta comunicarse con el servidor remoto.

Nombre	Conexión Wi-Fi y puesta en marcha del cliente de peticiones.
Actores	ESP8266.
Descripción	El módulo ESP8266 se conecta a la red Wi-Fi con los parámetros de la configuración inicial y una vez obtenida dirección IP pone el marcha el cliente de peticiones.
Precondición	La configuración de la conexión Wi-Fi ha de ser correcta y ha de tener conexión a internet. El servidor al que realiza las peticiones tiene que estar conectado y cumplir con el protocolo de comunicación.
Flujo normal	1. Una vez obtenidos los datos de la configuración el módulo intenta conectarse a la red Wi-Fi establecida en la configuración. Continuar.
Flujo alternativo	1. Si la configuración de red es correcta y se establece la conexión a la infraestructura de red, crea el cliente de SSL para realizar el caso de uso: 7.2.1.4. Caso de uso: Login Continuar. 2. Si la configuración no es correcta, intenta conectarse hasta conseguirlo.
Postcondición	La configuración se carga exitosamente.

Tabla 4. Caso de uso Conexión Wi-Fi y puesta en marcha del cliente de peticiones.

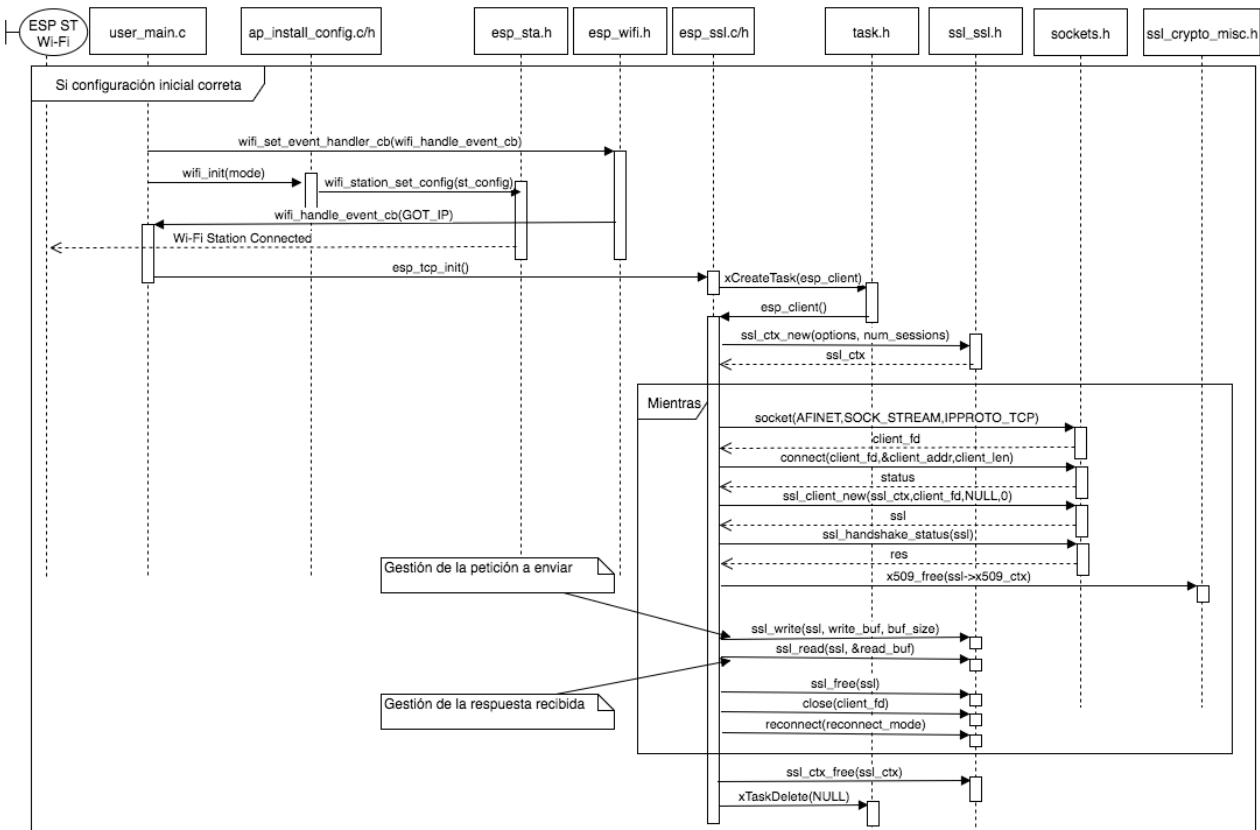


Figura 35. Flujo de eventos Conexión Wi-Fi y puesta en marcha del cliente de peticiones.

7.2.1.4. Caso de uso: Login

Este caso de uso hace referencia a la autenticación del módulo ESP8266 con el servidor remoto. De esta forma el servidor proveerá acciones al módulo ESP8266 y no a cualquier otro que realice peticiones intentando suplantar su identidad.

Nombre	Login.
Actores	ESP8266.
Descripción	Logueo del módulo ESP8266 en el servidor remoto.
Precondición	El módulo ESP8266 ha de haber creado el cliente de peticiones y estar dispuesto para hacer la petición de <i>login</i> .
Flujo normal	<ol style="list-style-type: none"> 1. El módulo ESP8266 realiza una petición al servidor remoto con el usuario y contraseña de la configuración de la instalación. 2. Si los datos son incorrectos, módulo ESP8266 volverá a intentar <i>loguearse</i> en el servidor indefinidamente hasta lograrlo.
Flujo alternativo	<ol style="list-style-type: none"> 1. Si el usuario y contraseña son correctos, el módulo ESP8266 envía la petición de obtención de acciones. 2. Si se produce una caída del servidor, el módulo ESP8266 intentará conectarse a él indefinidamente hasta que se conecte.
Postcondición	Se ha de <i>loguear</i> en el servidor.

Tabla 5. Caso de uso login.

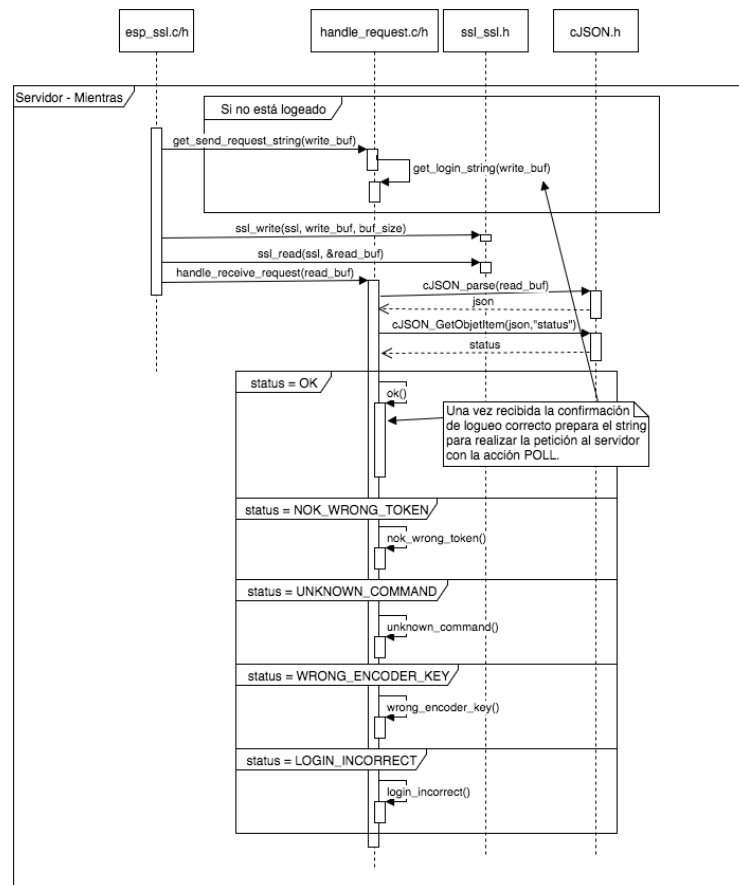


Figura 36. Flujo de eventos login.

7.2.1.5. Caso de uso: Obtener acciones

Una vez que el módulo ESP8266 se ha *logueado* en el servidor realiza peticiones en busca de acciones pendientes para realizar y no obtiene ninguna. Por lo tanto éste espera el tiempo de ventana establecido en la configuración inicial antes de realizar la siguiente petición.

Nombre	Obtener acciones.
Actores	ESP8266.
Descripción	El módulo ESP8266 pide acciones y no hay acciones para realizar.
Precondición	El módulo ESP8266 ha de estar <i>logueado</i> en el servidor y no tener acciones disponibles.
Flujo normal	<ol style="list-style-type: none"> 1. El módulo ESP8266 recibe una respuesta de con ninguna acción a realizar. 2. Esperar para la siguiente conexión. Continuar.
Flujo alternativo	Ninguno
Postcondición	El <i>string</i> de la siguiente petición es correcto.

Tabla 6. Caso de uso obtener acciones.

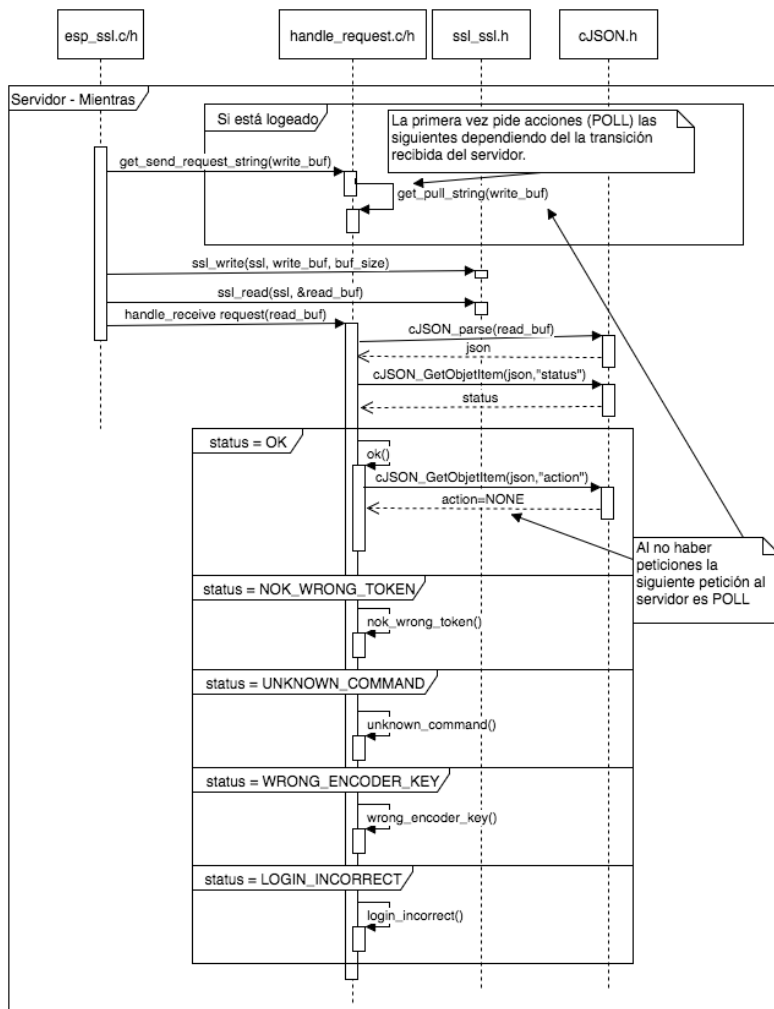


Figura 37. Flujo de eventos obtener acciones.

7.2.1.6. Caso de uso: Leer datos o leer número de serie de la tarjeta

Una vez que el módulo ESP8266 se ha *logueado* en el servidor realiza peticiones en busca de acciones pendientes para realizar y obtiene acciones para realizar.

Nombre	Leer datos o leer número de serie de la tarjeta.
Actores	ESP8266.
Descripción	El módulo ESP8266 pide acciones y hay acciones disponibles para realizar.
Precondición	El módulo ESP8266 ha de estar <i>logueado</i> en el servidor y tener acciones disponibles.
Flujo normal	<ol style="list-style-type: none"> 1. El módulo ESP8266 recibe una petición de la acción a realizar. 2. El usuario tiene un tiempo de ventana establecido en la configuración de la instalación para poner la tarjeta en el <i>encoder</i> para realizar la acción. 3. Si la lectura es correcta el módulo realiza una petición al servidor los datos de la tarjeta y el estado de la transacción. 4. El servidor le contesta con la siguiente acción y vuelve al 2.
Flujo alternativo	<ol style="list-style-type: none"> 1. Si se ha producido un error en la lectura de la tarjeta vuelve a realizar a partir del flujo de eventos normal 2 tantas veces como intentos de lectura/escritura se hayan establecido en la configuración de la instalación. 2. Si el tiempo de posado de tarjeta establecido en la configuración de la instalación expira, realiza una petición al servidor indicando en el estado de la transacción el <i>time out</i> producido. 3. Si no hay más acciones pendientes, el módulo enviará peticiones de obtención de acciones en periodos del tiempo, establecido en la configuración inicial de la instalación.
Postcondición	El <i>string</i> de la siguiente petición es correcto.

Tabla 7. Caso de uso leer datos y/o leer número de serie de la tarjeta.

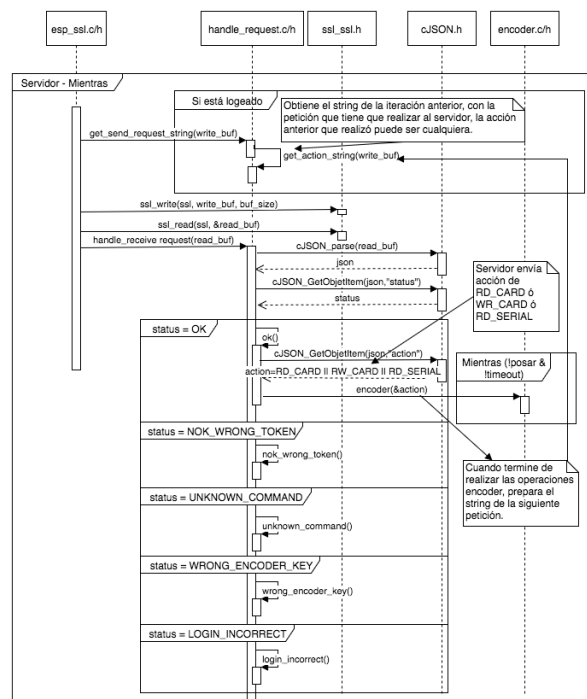


Figura 38. Flujo de eventos leer datos y/o leer número de serie de la tarjeta.

7.2.1.7. Caso de uso: Ejecutar acción en el *encoder*

Este caso de uso se encarga habilitar el *encoder* y esperar hasta que el usuario ponga la tarjeta o se produzca un *time out*.

Nombre	Ejecutar acción en el <i>encoder</i> .
Actores	ESP8266, <i>encoder</i> y cliente.
Descripción	El módulo ESP8266 habilita el <i>encoder</i> para que el cliente pose la tarjeta en él.
Precondición	El <i>encoder</i> tiene que tener una acción esperando que realice el <i>encoder</i> .
Flujo normal	<ol style="list-style-type: none"> 1. El módulo cada vez que requiera de un posado de tarjeta para realizar la acción, indicará al cliente que está listo. 2. El cliente posa la tarjeta, y el <i>encoder</i> intenta leer.
Flujo alternativo	<ol style="list-style-type: none"> 1. Si la lectura es incorrecta, vuelve a leer tantas veces como se indique en la configuración inicial.
Postcondición	La comunicación con el <i>encoder</i> termina en un determinado estado.

Tabla 8. Caso de uso ejecutar acción en el *encoder*.

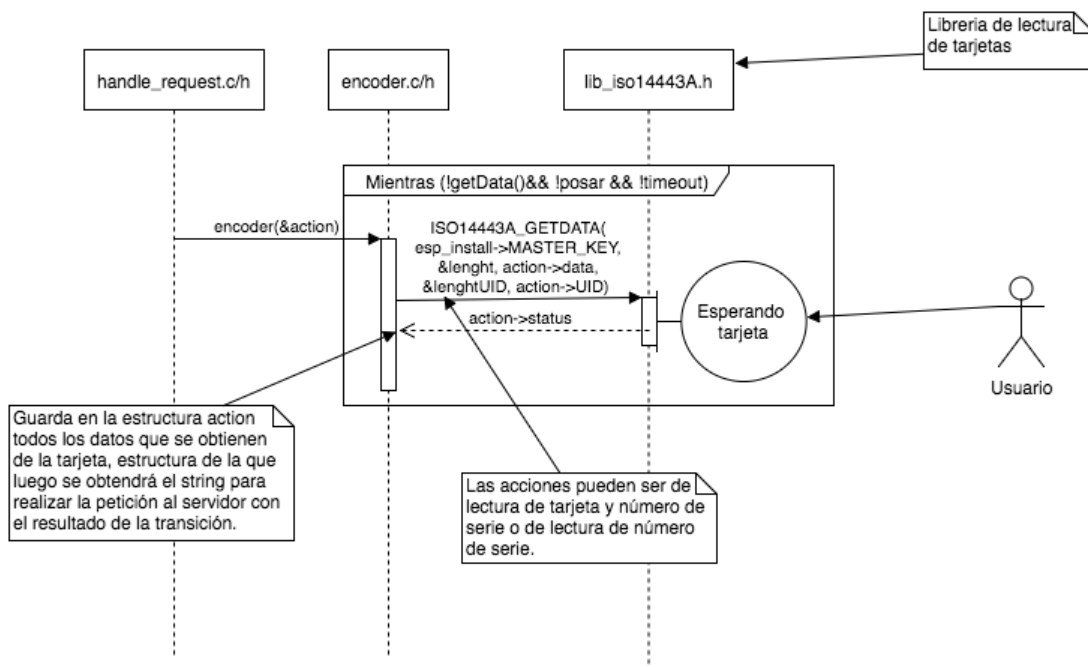


Figura 39. Flujo de eventos ejecutar acción en el *encoder*.

7.2.2. Diagrama de estados

Los casos de uso descritos anteriormente siguen este flujo y pasan por los siguientes estados:

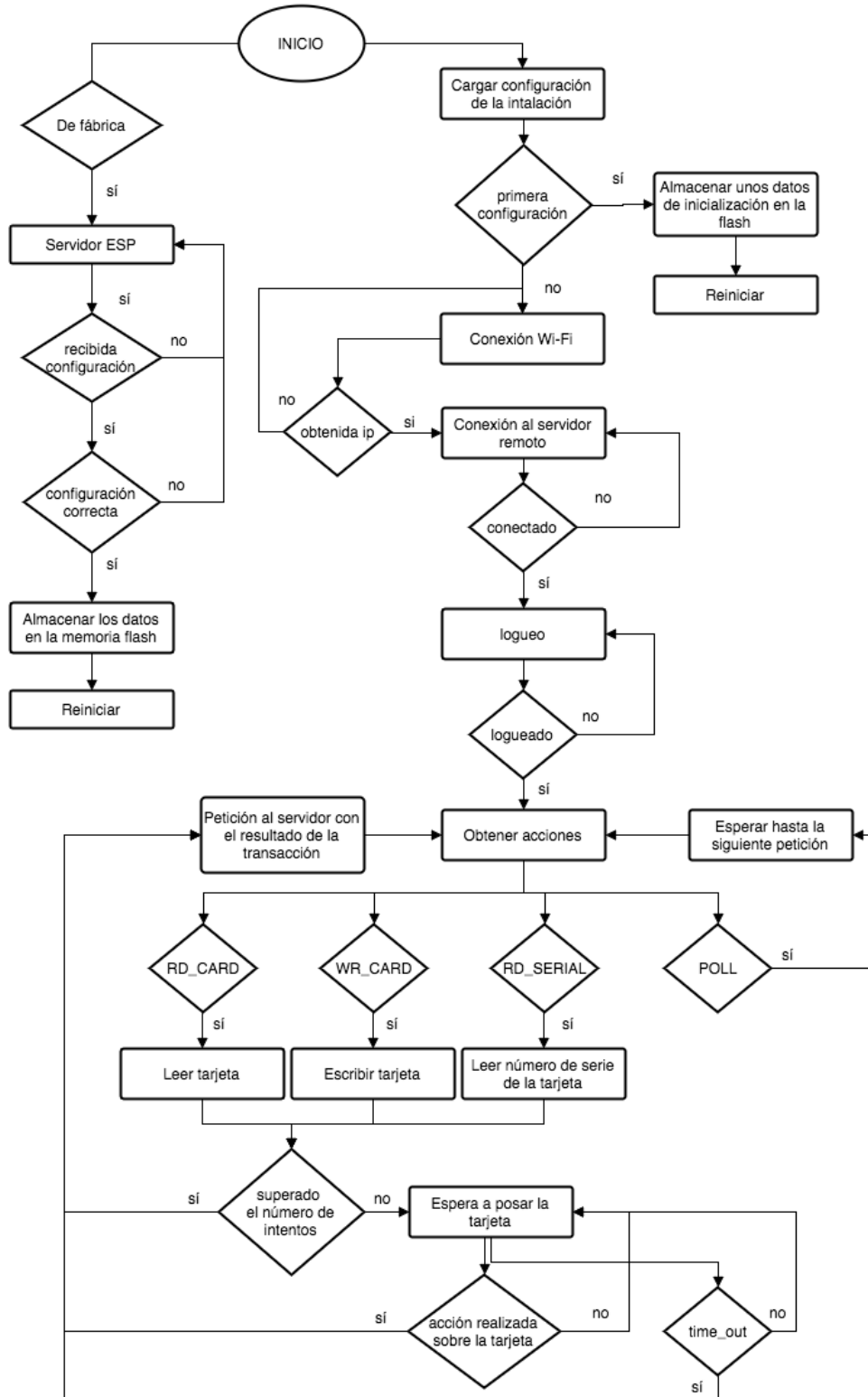


Figura 40. Diagrama de estados de la aplicación.

Comienza en un estado inicial:

- Si el módulo está configurado de fábrica pasa al estado Servidor ESP que esperará a que reciba la configuración de la instalación.
 - En el momento que se reciba la configuración y sea correcta se pasa al estado de almacenar los datos en la memoria *flash*.
 - Se almacenan los datos y el módulo ESP8266 se reinicia.
- Una vez pasado el estado de inicio, carga la configuración de la instalación.
 - Si está recién *flasheado* el módulo ESP8266, prepara los datos de inicialización y se reinicia.
 - Si ya tenía los datos cargados, los obtiene y se intenta conectar a la estación Wi-Fi .
 - Una vez se ha obtenido la dirección IP, crea la conexión con el servidor.
 - Establecida la conexión, realiza el *logueo*.
 - Cuando se *loguea* comienza a realizar el flujo de obtener acciones.
 - RD_CARD: Leer el número de serie de la tarjeta y los datos que contiene.
 - WR_CARD: No implementada la escritura.
 - RD_SERIAL: Leer número de serie de la tarjeta.
 - POLL: Petición de acciones y la acción no tenía acciones pendientes.
 - Se posa la tarjeta sobre la antena del *transceiver*, puede haber error o no.
 - Si no se ha superado el número de intentos definidos (en la configuración de la instalación), espera un tiempo también definido (en la configuración de la instalación) en el que se tiene que posar la tarjeta, si no se posa en ese tiempo, se realiza una petición con el estado de la transacción y recibe una respuesta con las acciones pendientes.

7.2.3. Arquitectura

De acuerdo a los requisitos y a los casos de uso especificados la arquitectura que se origina es una Arquitectura API REST.

El esquema de la arquitectura es el siguiente:

- Un módulo ESP8266 con un *encoder* o *transceiver* que interacciona con las tarjetas.
- Una estación Wi-Fi que dispone de conexión a internet.
- Un dispositivo que dispone de conexión a internet y de un navegador web.
- Acceso a un servidor en la nube que interpreta y se comunica a través del protocolo de comunicación descrito.

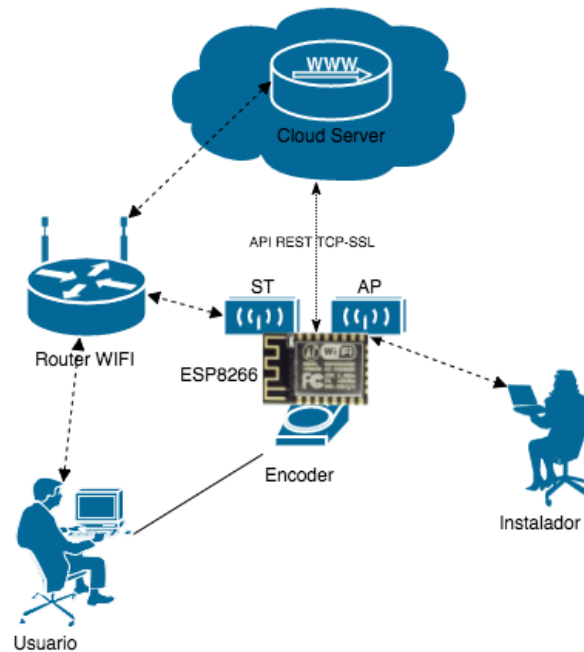


Figura 41. Arquitectura de la instalación.

7.2.3. Estructuras de datos

La estructura de datos que seguirá el módulo para almacenar los datos de cada transacción realizada con el servidor será la siguiente:

- Un variable para almacenar el identificador de la acción: *id_request*.
- Una variable que indique el tipo de acción: *action_type*.
- Una variable para almacene los datos de la acción: *data*.
- Una variable que indique la longitud de los datos de la tarjeta: *data_lenght*.
- Una variable para almacene el identificador único de la tarjeta: *uid*.

- Una variable que indique la longitud del número de serie de la tarjeta: *uid_lenght*.
- Una variable que guarde el estado de la transacción.

De modo que la estructura creada es la siguiente:

```
typedef struct _action{
    int id_request;
    uint8_t action_type;
    char *data;
    uint16_t data_lenght;
    char *uid;
    uint16_t uid_lenght;
    uint8_t status;
}action;
```

La estructura de datos de la configuración que se almacena en la memoria *flash* y con la que continuamente trabaja la aplicación para obtener los datos de configuración es la siguiente:

PARAMETRO	DESCRIPCIÓN
ESP_USER	Usuario con el que se <i>loguea</i> en el servidor remoto.
ESP_PASSWORD	Contraseña con la que se <i>loguea</i> en el servidor remoto.
ST_SSID	Nombre o SSID de la infraestructura de red Wi-Fi a la que se conecta el módulo ESP8266 para obtener acceso a Internet.
ST_PASSWORD	Contraseña de la infraestructura de red Wi-Fi a la que se conecta el módulo ESP8266 para obtener acceso a Internet.
SERVER_IP	Dirección IP del servidor remoto al que se va a realizar peticiones el módulo ESP8266.
ESP_IP	Dirección IP estática del módulo ESP8266 en la red a la que está conectado.
ESP_GW	Dirección IP de la puerta de enlace del módulo ESP8266 en la red a la que está conectado.
ESP_MASK	Máscara de red del módulo ESP8266 en la red a la que está conectado.
CARD_ATTEMPTS	Intentos de lectura de la tarjeta que se posa.
CARD_TIME_OUT_SEC	Tiempo establecido para posar la tarjeta en cada uno de los intentos de posado de la tarjeta sobre el <i>transceiver</i> .
SERVER_REQUEST_SEC	Periodo de tiempo entre una petición por acciones y otra en el caso de que no hayan acciones pendientes en el servidor remoto.
MASTER_KEY	Clave maestra de las tarjetas, para poder leer y escribir datos en las mismas.
IS_INITIALIZED	Indica que la configuración ha sido inicializada.

Tabla 9. Parámetros de configuración de la instalación.

Los tamaños de cada una de las cadenas de caracteres contenidas en la estructura *esp_intallation* se indican en las siguientes definiciones:

```
#define ESP_USER_SIZE          30
#define ESP_PASSWORD_SIZE     50
#define ST_SSID_SIZE          32
#define ST_PASSWORD_SIZE     50
#define SERVER_IP_SIZE        15
#define ESP_IP_SIZE           15
#define ESP_GW_SIZE           15
#define ESP_MASK_SIZE         15
#define MASTER_KEY_SIZE       8
typedef struct _esp_installation{
    char ESP_USER[ESP_USER_SIZE];
    char ESP_PASSWORD[ESP_PASSWORD_SIZE];
    char ST_SSID[ST_SSID_SIZE];
    char ST_PASSWORD[ST_PASSWORD_SIZE];
    char SERVER_IP[SERVER_IP_SIZE];
    char ESP_IP[ESP_IP_SIZE];
    char ESP_GW[ESP_GW_SIZE];
    char ESP_MASK[ESP_MASK_SIZE];
    uint8_t SERVER_REQUEST_SEC;
    uint16_t MASTER_KEY[MASTER_KEY_SIZE];
    uint8_t CARD_TIME_OUT_SEC;
    uint8_t CARD_ATTEMPTS;
    uint8_t IS_INITIALIZED;
} esp_installation;
```

7.3. Implementación

La implementación se ha realizado en base al estudio previo realizado sobre el módulo ESP8266. Este módulo ofrece infinidad de funciones que han servido para seguir diferentes estrategias.

7.3.1. Implementación de los casos de uso Establecer la configuración inicial de la instalación y carga de los datos

En primer lugar, para la configuración externa de los datos de una instalación se ha visto oportuno crear un servidor en la red *Access Point* del módulo ESP8266. Este servidor se ha colocado en la puerta de enlace, concretamente en la dirección IP 192.168.5.1 con la ayuda de la librería *espconn.h*. Un ejemplo del uso de la librería puede verse en el anexo B en el apartado: B.6. Creación de un cliente y servidor *HTTP Request*. Para comunicarse con ese este servidor el protocolo a seguir es; enviar

una petición *http request* POST con contenido de tipo JSON (para las pruebas se ha utilizado un comando CURL desde terminal). Cuando el módulo ESP8266 recibe el mensaje JSON lo parsea con la librería *cJSON.h* y obtiene cada uno de los objetos. Un ejemplo del uso de esta librería se puede encontrar en apartado: B.5. Uso de la librería *cJSON* del Anexo B.

Una vez obtenido todos los datos, los almacena en una estructura de datos (*esp_install*) que es la estructura que se almacena en la dirección de memoria `x7b000` del módulo ESP8266. Se ha escogido esa dirección de memoria puesto que son posiciones de memoria que no están *mapeadas* por la gestión de memoria que realiza el SDK de RTOS y, por lo tanto, están libres.

El atributo `IS_INITIALIZED` se utiliza para comprobar si en esas posiciones de memoria hay algún dato escrito, de no haberlo se inicializan y se guarda la estructura de la configuración con todos datos vacíos para en que cada reinicio no se bloquee y se reinicie el modulo automáticamente por intentar leer un dato de una dirección de memoria que no contiene nada.

7.3.2. Implementación del caso de uso conexión Wi-Fi y cliente de peticiones

Después del reinicio y carga de los datos de la configuración de la instalación se intenta conectar la red Wi-Fi especificada para poder acceder a internet. Esta configuración se ha realizado gracias a la librería *esp_st.h*, un ejemplo de su uso puede verse en el Anexo B en el apartado: B.2. Configuración WI-FI.

Cuando éste se conecta a la red se ejecuta el servidor creado con la librería *esp_ssl.h*. Esta librería permite conexiones seguras de modo que todos los datos que se envían entre cliente y servidor van cifrados, aspecto importante para brindar seguridad a la aplicación. Un ejemplo de uso de esta librería aparece en el apartado: B.7. Ejemplo TCP - SSL del Anexo B.

La peticiones *http request* barajadas que puede realizar al servidor son las peticiones GET y POST. Pero se ha escogido el uso de GET por su sencillez frente a POST que requiere de cabeceras adicionales. A continuación se muestra un ejemplo de uso de cada una de las llamadas *request*:

- GET:
 - `char data[500];`
 - `sprintf(data, "GET / HTTP/1.1\nHOST: %s:%d \n\n0", server_ip, server_port);`

- POST:
 - `char data[500];`
 - `sprintf(data, "POST / HTTP/1.1\nHOST: %s \ncontent-type: application/x-www-form-urlencoded \ncontent-length: 22 \n\nndata=This+is+the+data\n\n", server_ip, server_port);`

La conexión que se crea con el servidor se hace mediante *sockets* y el protocolo TCP.

Los *Socket* son un punto final de la comunicación entre los dos sistemas en una red. Para ser un poco más precisos, un *socket* es una combinación de la dirección IP y el puerto en un sistema. Así que en cada sistema existe un *socket* para interactuar con el *socket* en otro sistema de la red.

En primer lugar se crea el descriptor del fichero o *socket* por el que se va a comunicar (en este caso el cliente módulo ESP8266), para ello se indica el dominio que especifica la familia de protocolos en este caso se usa el protocolo IPv4 (AF_INET), el tipo de comunicación que es SOCK_STREAM para TCP y el protocolo por defecto para el tipo de comunicación definido (TCP IPPROTO_TCP).

Acto siguiente se intenta realizar la conexión (*connect*) al servidor, con el descriptor de fichero creado anteriormente, la estructura del *socket* con la información del servidor al que se va a conectar (tipo de familia AF_INET, puerto, y la dirección IP del servidor). Una vez se ha conectado, en este caso, se crea el cliente de SSL con el un contexto definido para aceptar que el servidor le mande el certificado y el *socket*. Una vez verificada que la conexión SSL es correcta, se procede a realizar la comunicación. Primero se envían los datos, para ello se escriben, en el cliente de SSL que contiene la información del *socket*, el *string* de los datos (*data*) y la longitud de los mismos. Una vez enviados, se lee la respuesta que da el servidor con el cliente SSL y los datos que contiene.

A continuación, con los datos recibidos en el formato establecido en el protocolo de comunicación entre cliente y servidor (JSON), comienza la lógica de la aplicación. Para obtener los datos encapsulados en el formato JSON, al igual que se hace en la configuración de la instalación, se parsean con la librería cJSON.h y se obtienen cada uno de los objetos que contiene. Un ejemplo del uso de esta librería se puede encontrar en apartado: B.5. Uso de la librería cJSON del Anexo B.

7.3.3. Implementación de los casos de uso *login*, obtener acciones y manejo de acciones

Obtenida cada respuesta del servidor, se interpretan los datos que contiene, para ello, primero se envía una petición de *logueo*. Si el usuario y contraseña enviados son correctos, el módulo ESP8266 recibe un *token* (identificador de la sesión) para que verifique la autenticidad en cada una de las peticiones posteriores que realice el cliente.

La siguiente petición que se envía es la de obtención de acciones. Si existen acciones, el servidor envía la acción y sino, el cliente se queda esperando el tiempo de ventana indicado en la configuración inicial hasta volver a preguntar por más acciones. En caso de haber acciones, el módulo recibe la respuesta con la acción junto con más información de la misma para que luego el *encoder* la realice. Cuando el *encoder* ha terminado de realizar la acción, éste devuelve los datos del resultado de la transacción y prepara el *string* para realizar la petición *http request* al servidor con el resultado. El servidor le contesta con la siguiente acción si hay y se vuelve a realizar el mismo proceso.

7.3.4. Implementación del caso de uso de la ejecución de la acción en el *encoder*

Cuando se dispone a realizar la acción recibida, el *encoder* se habilita y se queda en espera a que un usuario pose la tarjeta sobre el mismo. La acción se repetirá tantas veces como intentos se hayan establecido en la configuración inicial de la instalación hasta que el resultado obtenido por el *encoder* sea satisfactorio o se llegue al número máximo de intentos. También existe la posibilidad de indicar un tiempo máximo de espera para el posado de la tarjeta que se indica en la configuración inicial de la instalación. Una vez terminada la transacción se recoge el resultado y se procesa para crear el *string* de la siguiente petición.

7.3.5. Problemas en la implementación

7.3.5.1. Desbordamiento de la IRAM

A medida de que la aplicación ha ido creciendo ha surgido un problema de espacio de la memoria RAM a la hora de compilar la aplicación. Para solventar el problema de desbordamiento de la memoria RAM han de almacenarse las funciones en la memoria flash en lugar de en la memoria RAM, para ello hay que crearlas de la siguiente manera:

```
LOCAL void ICACHE_FLASH_ATTR my_function(void *arg) {...}
```

7.3.5.2. Gestión del RTOS

Cada una de las tareas que se crean en el RTOS contienen una prioridad de ejecución. La prioridad máxima reside en la tarea principal la función *user_init()* donde ha de ir la configuración de la conexión Wi-Fi para que ésta no falle. A continuación la tarea con más prioridad es la del cliente TCP para la gestión de envío y recepción de los datos. De acuerdo a lo anterior se quería añadir una nueva tarea que gestionara las acciones recibidas. Pero surgió la problemática de que una vez creada la tarea el *encoder* no respondía bien, fallando en las lecturas y bloqueando el servidor. El servidor en este caso estaba realizando peticiones en busca de transacciones para cancelar la transacción que anteriormente había enviado, para eliminarla. Era una

posible mejora (indicada en el capítulo: 10. Propuestas de mejora) que no ha sido posible llevar a cabo.

7.4. Verificación y pruebas

Las pruebas realizadas se han realizado en paralelo con la implementación de los casos de uso, depurando el código continuamente mediante el puerto serie del módulo ESP8266. Se ha tenido en cuenta las precondiciones de los casos de uso especificados en el diseño.

7.4.1. Verificación del caso de uso Establecer configuración inicial de la instalación

Cuando se quiere cargar la configuración, una vez conectados a la red Wi-Fi que crea el módulo ESP8266, se comprueban que los datos que se reciben están en formato JSON, además de que exista cada uno de los parámetros indicados, de modo que si no se cumplen, no se realiza la carga de los datos en la flash. En cuanto al contenido de los datos, han de seguir la precondición y han de ser datos válidos, cumpliendo con el tamaño máximo asignado para cada cadena de caracteres. No se comprueba el desbordamiento de las variables, de modo que si no se cumple la precondición el módulo ESP8266 se reinicia.

```
➔ ~ curl -H "Content-Type: application/json" -X POST -d '{"ESP_USER":"admin","ESP_PASSWORD":"admin", "ST_SSID":"SSID","ST_PASSWORD":"*****","SERVER_IP":"192.168.0.227","ESP_IP":"192.168.0.245","ESP_GW":"192.168.0.1","ESP_MASK":"255.255.255.0","CARD_TIME_OUT_SEC":30,"CARD_ATTEMPTS":5,"SERVER_REQUEST_SEC":2,"MASTER_KEY":["FFFF","FFFF","FFFF","FFFF","FFFF","FFFF","FFFF","FFFF","FFFF"]}' 192.168.5.1
```

Figura 42. Curl desde la terminal al módulo ESP8266.

7.4.2. Verificación del caso de uso cargar configuración

Una vez que los datos han sido cargados en la memoria flash, se procede a lectura de los datos. No se controla la validez de los datos cargados de modo que los siguientes casos de uso se van a ver afectados si estos no son correctos, siguiendo los tamaños especificados en las precondiciones y los formatos de cada uno de ellos.

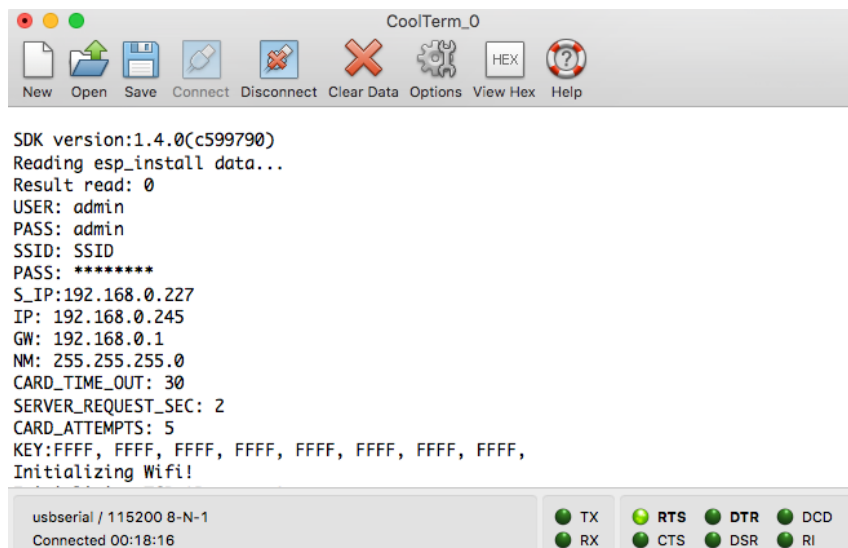


Figura 43. Carga de datos almacenados en la flash en la estructura `esp_install`.

7.4.3. Verificación del caso de uso Conexión Wi-Fi y puesta en marcha del cliente de peticiones

La verificación de los parámetros de conexión WI-FI los realiza el API del módulo internamente, los errores que pueden darse son los siguientes:

- El SSID de la infraestructura de red a la que se intenta conectar no existe o la contraseña es incorrecta. En este caso se realizan intentos de conexión continuamente.
- El SSID y contraseña son correctos pero la red WI-FI está desactivada. Al igual que en el caso anterior hasta que no se active la red se queda intentando la conexión.
- El SSID y contraseña son correctos pero los valores introducidos de las direcciones IP no lo son. Queda en el estado de reconexión.

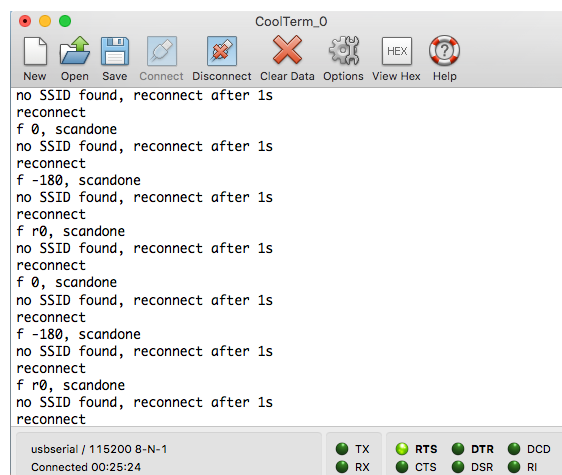


Figura 44. Intento de conexión a la red Wi-Fi.

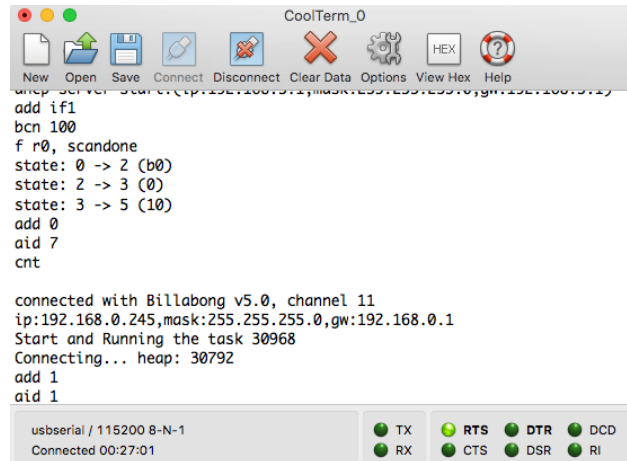


Figura 45. Conexión establecida a la red Wi-Fi.

Una vez conectado y obtenida la dirección IP se intenta crear el servidor, pudiendo darse las siguientes circunstancias:

- Los parámetros de la configuración del servidor son correctos pero no hay acceso al mismo, el servidor está caído o la dirección IP donde se encuentra no es alcanzable desde la red en la que se encuentra el módulo ESP8266. Queda esperando la reconexión con el servidor hasta que se conecta.
- Hay conexión con el servidor pero esté se cae, la transacción de la petición que se estaba realizando se queda almacenada hasta que se reestablezca la conexión y se envíe.
- El cliente recibe una respuesta del servidor no esperada o que no cumple con el protocolo, la próxima petición del cliente le indica que el comando anterior no se reconoce (UNKNOWN_COMMAND).
- En el caso de que se desconecte de la red WI-FI por una pérdida de señal se elimina la tarea del servidor y se realiza el mismo proceso una vez se obtenga la dirección IP.
- El mensaje que envía el cliente al servidor tiene que corresponderse con el protocolo https. Hay que tener una especial atención en la construcción de la cabecera. Un carácter erróneo o en un orden incorrecto puede repercutir en la interpretación por parte del servidor. El mensaje a enviar se contempla en el apartado: 7.3.2. Implementación del caso de uso conexión Wi-Fi y cliente de peticiones.

117	3.449946	192.168.0.245	192.168.0.23	TLSv1...	110 Client Hello
120	3.455656	192.168.0.245	192.168.0.23	TCP	54 231 → 443 [ACK] Seq=57 Ack=583 Win=5258 Len=0
122	3.477634	192.168.0.245	192.168.0.23	TLSv1...	193 Client Key Exchange
124	3.519624	192.168.0.245	192.168.0.23	TLSv1...	129 Change Cipher Spec, Encrypted Handshake Message
127	3.528424	192.168.0.245	192.168.0.23	TLSv1...	299 Application Data
129	3.573116	192.168.0.245	192.168.0.23	TCP	54 231 → 443 [ACK] Seq=516 Ack=1036 Win=4805 Len=0
183	5.736425	192.168.0.245	192.168.0.23	TLSv1...	107 Encrypted Alert
185	5.737191	192.168.0.245	192.168.0.23	TCP	54 231 → 443 [FIN, ACK] Seq=569 Ack=1036 Win=4805 Len=0
186	5.737194	192.168.0.245	192.168.0.23	TCP	58 4849 → 443 [SYN] Seq=0 Win=5840 Len=0 MSS=1440
189	5.740634	192.168.0.245	192.168.0.23	TCP	54 231 → 443 [ACK] Seq=570 Ack=1037 Win=4804 Len=0
190	5.741504	192.168.0.245	192.168.0.23	TCP	54 4849 → 443 [ACK] Seq=1 Ack=1 Win=5840 Len=0

Figura 46. Captura mediante Wireshark de mensajes enviados y recibidos entre cliente-servidor.

Se han realizado las siguientes *pruebas de estrés* contra el servidor en las que el módulo ESP8266 seguía en completo funcionamiento:

- Intento de *logueo* continuo.
- Enviar peticiones de obtención de acciones durante un día.
- Monitorización de la memoria RAM. Comprobado el envío de 50.000 peticiones con datos de tarjetas *hard-codeados* sin que varíe el tamaño de la memoria RAM disponible.

7.4.4. Verificación del caso de uso Login.

Para *logueo* del módulo ESP8266 se envían los datos usuario y contraseña que ha de tener el servidor registrados. Se repite este proceso continuamente hasta que se recibe una respuesta con el estado *OK* y un *TOKEN* de la sesión.

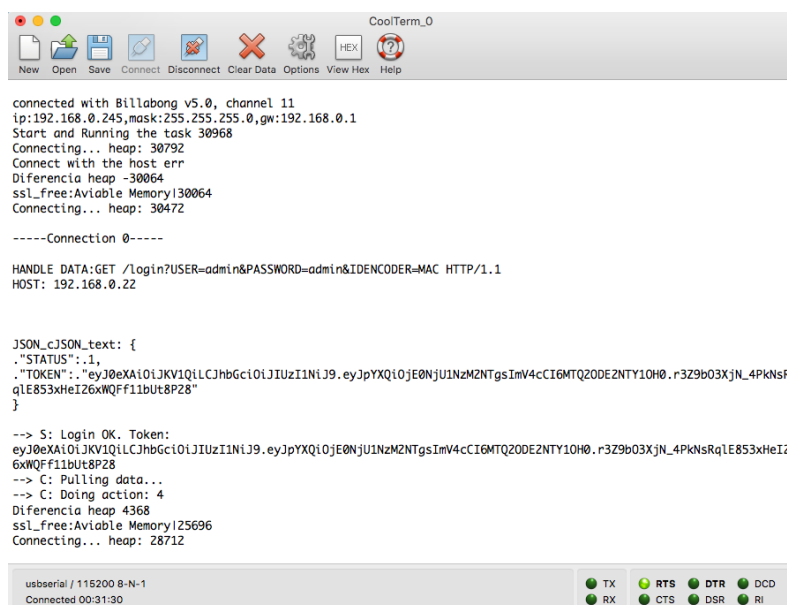


Figura 47. Logueo del módulo ESP8266 en el servidor (visto desde el cliente).

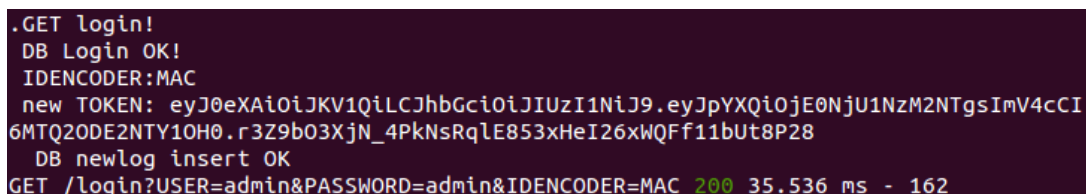


Figura 48. Logueo del módulo ESP8266 en el servidor (visto desde el servidor).

7.4.5. Verificación del caso de uso Obtener acciones

Continuamente está realizando peticiones por acciones hasta que recibe una y la trata.

Las acciones que recibe han de corresponderse con el protocolo de comunicación definido, en caso contrario la petición que a continuación realiza es la de comando desconocido.

```

CoolTerm_0
-----Connection 1-----
HANDLE DATA:GET /execute?
IDENCODER=MAC&TOKEN=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpYXQiOiJlbnZMNTgsImV4cCI6MTQ2ODE2NTY1OH0.r3Z9b03XjN_4PkNsRqLE853xHeI26xWQFf11but8P28&ACTION=4 HTTP/1.1
HOST: 192.168.0.22

JSON_cJSON_text: {
  "STATUS":1,
  "IDREQUEST":0,
  "ACTION":0,
  "DATA":,
  "UID":,
}

--> S: There are not pending actions.
--> C: Doing action: 4
Diferencia heap 312
1 0
ssl_free:Aviable MemoryI25344
Connecting... heap: 28360

-----Connection 2-----
HANDLE DATA:GET /execute?
IDENCODER=MAC&TOKEN=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpYXQiOiJlbnZMNTgsImV4cCI6MTQ2ODE2NTY1OH0.r3Z9b03XjN_4PkNsRqLE853xHeI26xWQFf11but8P28&ACTION=4 HTTP/1.1
HOST: 192.168.0.22

JSON_cJSON_text: {
  "STATUS":1,
  "IDREQUEST":0,
  "ACTION":0,
  "DATA":,
  "UID":,
}

--> S: There are not pending actions.
--> C: Doing action: 4
Diferencia heap 336
1 0
ssl_free:Aviable MemoryI25048
Connecting... heap: 28096

usbserial / 115200 8-N-1
Connected 00:32:08
TX RTS DTR DCD
RX CTS DSR RI

```

Figura 49. Obtención de acciones – No hay acciones (visto desde el cliente).

```

.GET execute
req.query.TOKEN: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpYXQiOiJlbnZMNTgsImV4cCI6MTQ2ODE2NTY1OH0.r3Z9b03XjN_4PkNsRqLE853xHeI26xWQFf11but8P28
req.query.IDENCODER: MAC
req.query.ACTION: 4
Authentication OK
NO hay tareas pendientes
JSON--> { STATUS: 1,IDREQUEST:0, ACTION:0, DATA: , UID: }
GET /execute?IDENCODER=MAC&TOKEN=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpYXQiOiJlbnZMNTgsImV4cCI6MTQ2ODE2NTY1OH0.r3Z9b03XjN_4PkNsRqLE853xHeI26xWQFf11but8P28&ACTION=4 200 17.531 ms - 82
.GET execute
req.query.TOKEN: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpYXQiOiJlbnZMNTgsImV4cCI6MTQ2ODE2NTY1OH0.r3Z9b03XjN_4PkNsRqLE853xHeI26xWQFf11but8P28
req.query.IDENCODER: MAC
req.query.ACTION: 4
Authentication OK
NO hay tareas pendientes
JSON--> { STATUS: 1,IDREQUEST:0, ACTION:0, DATA: , UID: }
GET /execute?IDENCODER=MAC&TOKEN=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpYXQiOiJlbnZMNTgsImV4cCI6MTQ2ODE2NTY1OH0.r3Z9b03XjN_4PkNsRqLE853xHeI26xWQFf11but8P28&ACTION=4 200 3.985 ms - 82
.GET execute
req.query.TOKEN: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpYXQiOiJlbnZMNTgsImV4cCI6MTQ2ODE2NTY1OH0.r3Z9b03XjN_4PkNsRqLE853xHeI26xWQFf11but8P28
req.query.IDENCODER: MAC
req.query.ACTION: 4
Authentication OK
NO hay tareas pendientes
JSON--> { STATUS: 1,IDREQUEST:0, ACTION:0, DATA: , UID: }
GET /execute?IDENCODER=MAC&TOKEN=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpYXQiOiJlbnZMNTgsImV4cCI6MTQ2ODE2NTY1OH0.r3Z9b03XjN_4PkNsRqLE853xHeI26xWQFf11but8P28&ACTION=4 200 6.816 ms - 82

```

Figura 50. Obtención de acciones – No hay acciones (visto desde el servidor).

7.4.6. Verificación del caso de uso Lectura de datos y número de serie y Ejecución en el *encoder*

En la realización de acciones de lectura, se dan han realizado las siguientes comprobaciones:

- Lectura correcta. Tras acercar la tarjeta a la antena lectora el tiempo suficiente hasta que termine de leer.

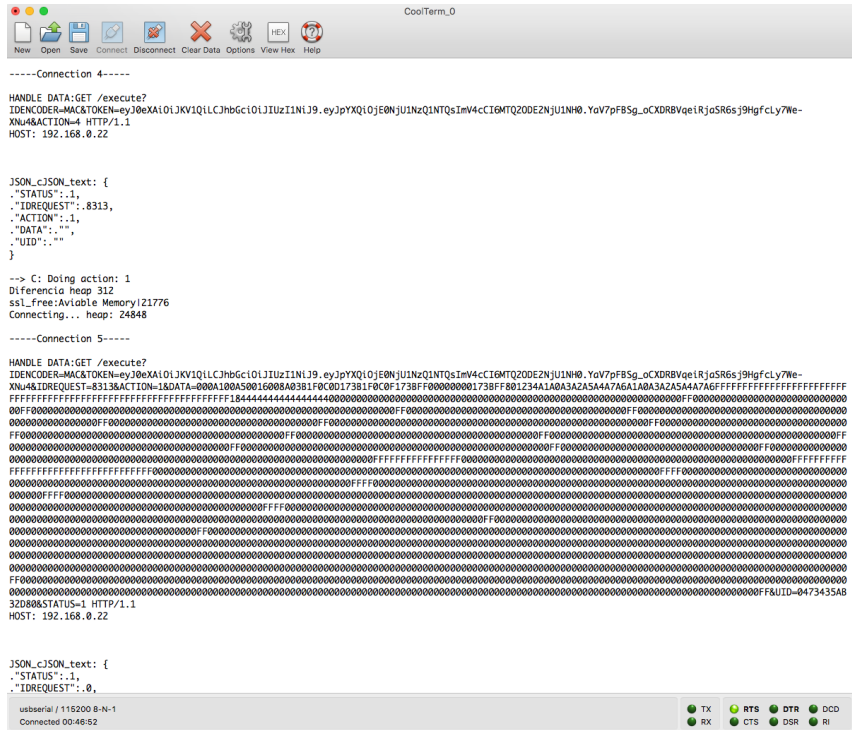


Figura 51. Lectura correcta de datos y número de serie de la tarjeta (visto desde el cliente).

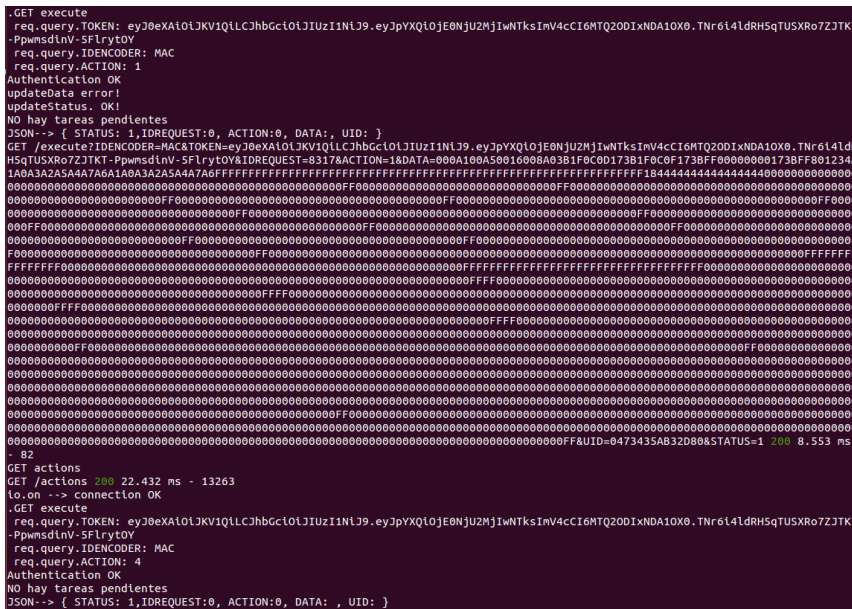


Figura 52. Lectura correcta de datos y número de serie de la tarjeta (visto desde el servidor).

- Lectura fallida de la tarjeta. Tras retirar la tarjeta de la antena lectora antes de que finalice la lectura completa.
- Número de intentos de lectura superado. Tras realizar el número máximo de lecturas fallidas indicado en la configuración.

```

CoolTerm_0
New Open Save Connect Disconnect Clear Data Options View Hex Help

----Connection 4----
HANDLE DATA:GET /execute?
IDENCODER=MAC&TOKEN=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpYXQiOiJlbnV4cC16MTQ2ODZlZnI2Nm0.4WY0L1TQ2vKFDcbCvkjQGEExockuEQ
5H3i0AKpqr76A&ACTION=4 HTTP/1.1
HOST: 192.168.0.23

JSON_cJSON_text: {
  "STATUS":.1,
  "IDREQUEST":.8315,
  "ACTION":.1,
  "DATA":. ,
  "UID":. "
}

--> C: Doing action: 1
Diferencia heap 296
ssl_free:Aviable MemoryI24496
Connecting... heap: 27552

----Connection 5----
HANDLE DATA:GET /execute?
IDENCODER=MAC&TOKEN=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpYXQiOiJlbnV4cC16MTQ2ODZlZnI2Nm0.4WY0L1TQ2vKFDcbCvkjQGEExockuEQ
5H3i0AKpqr76A&IDREQUEST=8315&ACTION=1&DATA=&UID=&STATUS=7 HTTP/1.1
HOST: 192.168.0.23

JSON_cJSON_text: {
  "STATUS":.1,
  "IDREQUEST":.0,
  "ACTION":.0,
  "DATA":. " ,
  "UID":. "
}

--> S: There are not pending actions.
--> C: Doing action: 4
Diferencia heap 288
1 0
ssl_free:Aviable MemoryI24208
Connecting... heap: 27264

usbserial / 115200 8-N-1
Connected 00:58:13
TX RTS DTR DCD
RX CTS DSR RI

```

Figura 53. Acción de lectura fallida (visto desde el cliente).

```

req.query.TOKEN: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpYXQiOiJlbnV4cC16MTQ2ODZlZnI2Nm0.4WY0L1TQ2vKFDcbCvkjQGEExockuEQ5H3i0AKpqr76A
req.query.IDENCODER: MAC
req.query.ACTION: 4
Authentication OK
Hay tareas pendientes
JSON-> { STATUS: 1, IDREQUEST: 8315, ACTION: 1, DATA: , UID: }
GET /execute?IDENCODER=MAC&TOKEN=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpYXQiOiJlbnV4cC16MTQ2ODZlZnI2Nm0.4WY0L1TQ2vKFDcbCvkjQGEExockuEQ5H3i0AKpqr76A&ACTION=4 200 4.113 ms - 85
updateStatus. OK!
updateAtemps. OK!
updateStatus. OK!
.GET execute
req.query.TOKEN: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpYXQiOiJlbnV4cC16MTQ2ODZlZnI2Nm0.4WY0L1TQ2vKFDcbCvkjQGEExockuEQ5H3i0AKpqr76A
req.query.IDENCODER: MAC
req.query.ACTION: 1
Authentication OK
STATUS == 1-7
idreqtrans. OK!
idreq status:NO_OK
NO hay tareas pendientes
JSON-> { STATUS: 1, IDREQUEST: 0, ACTION: 0, DATA: , UID: }
GET /execute?IDENCODER=MAC&TOKEN=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpYXQiOiJlbnV4cC16MTQ2ODZlZnI2Nm0.4WY0L1TQ2vKFDcbCvkjQGEExockuEQ5H3i0AKpqr76A&IDREQUEST=8315&ACTION=1&DATA=&UID=&STATUS=7 200 5.808 ms - 82
.GET execute
req.query.TOKEN: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpYXQiOiJlbnV4cC16MTQ2ODZlZnI2Nm0.4WY0L1TQ2vKFDcbCvkjQGEExockuEQ5H3i0AKpqr76A
req.query.IDENCODER: MAC
req.query.ACTION: 4
Authentication OK
NO hay tareas pendientes
JSON-> { STATUS: 1, IDREQUEST: 0, ACTION: 0, DATA: , UID: }
NO hay tareas pendientes
JSON-> { STATUS: 1, IDREQUEST: 0, ACTION: 0, DATA: , UID: }

```

Figura 54. Acción de lectura fallida (visto desde el servidor).

- Espera por el posado de la tarjeta superada. Tras superar el tiempo máximo establecido en la configuración.

```

-----Connection 4-----
HANDLE DATA:GET /execute?
IDENCODER=MAC&TOKEN=eyJ0eXA101KV1Q1LLC3hbGc101JiUzI1NlJ9.eyJpYXQ10jE0NjU1NzU0NzAsImV4cCI6MTQ2ODE2NzQ3MH0.6buRwEHM4dZuo3XWMD0EgDWZlFKqb
qBL9j83-Eudc4E8&ACTION=4 HTTP/1.1
HOST: 192.168.0.23

JSON_cJSON_text: {
  "STATUS":1,
  "IDREQUEST":8316,
  "ACTION":1,
  "DATA":",",
  "UID":",",
}

--> C: Doing action: 1
Diferencia heap 312
ssl_Free:Aviable Memory124480
Connecting... heap: 27552

-----Connection 5-----
HANDLE DATA:GET /execute?
IDENCODER=MAC&TOKEN=eyJ0eXA101KV1Q1LLC3hbGc101JiUzI1NlJ9.eyJpYXQ10jE0NjU1NzU0NzAsImV4cCI6MTQ2ODE2NzQ3MH0.6buRwEHM4dZuo3XWMD0EgDWZlFKqb
qBL9j83-Eudc4E8&IDREQUEST=8316&ACTION=1&DATA=&UID=&STATUS=5 HTTP/1.1
HOST: 192.168.0.23

JSON_cJSON_text: {
  "STATUS":1,
  "IDREQUEST":0,
  "ACTION":0,
  "DATA":",",
  "UID":",",
}

--> S: There are not pending actions.
--> C: Doing action: 4
Diferencia heap 272
10
ssl_Free:Aviable Memory124208
Connecting... heap: 27264
    
```

Figura 55. Acción de lectura tiempo de espera superado (visto desde el cliente).

```

.GET execute
req.query.TOKEN: eyJ0eXA101KV1Q1LLC3hbGc101JiUzI1NlJ9.eyJpYXQ10jE0NjU1NzU0NzAsImV4cCI6MTQ2ODE2NzQ3MH0.6buRwEHM4dZuo3XWMD0EgDWZlFKqbqBL9j83-Eudc4E
lFKqbqBL9j83-Eudc4E
req.query.IDENCODER: MAC
req.query.ACTION: 4
Authentication OK
Hay tareas pendientes
JSON-> { STATUS: 1, IDREQUEST: 8316, ACTION: 1, DATA: , UID: }
GET /execute?IDENCODER=MAC&TOKEN=eyJ0eXA101KV1Q1LLC3hbGc101JiUzI1NlJ9.eyJpYXQ10jE0NjU1NzU0NzAsImV4cCI6MTQ2ODE2NzQ3MH0.6buRwEHM4
dZuo3XWMD0EgDWZlFKqbqBL9j83-Eudc4E&ACTION=4 200 5.040 ms - 85
updateStatus. OK!
updateAtemps. OK!
updateStatus. OK!
GET actions
GET /actions 200 12.127 ms - 13263
Io.on -> connection OK
.GET execute
req.query.TOKEN: eyJ0eXA101KV1Q1LLC3hbGc101JiUzI1NlJ9.eyJpYXQ10jE0NjU1NzU0NzAsImV4cCI6MTQ2ODE2NzQ3MH0.6buRwEHM4dZuo3XWMD0EgDWZlFKqbqBL9j83-Eudc4E
lFKqbqBL9j83-Eudc4E
req.query.IDENCODER: MAC
req.query.ACTION: 1
Authentication OK
STATUS === 1-5
ldreqtrans. OK!
ldreq status:NO_OK
NO hay tareas pendientes
JSON-> { STATUS: 1, IDREQUEST: 0, ACTION: 0, DATA: , UID: }
GET /execute?IDENCODER=MAC&TOKEN=eyJ0eXA101KV1Q1LLC3hbGc101JiUzI1NlJ9.eyJpYXQ10jE0NjU1NzU0NzAsImV4cCI6MTQ2ODE2NzQ3MH0.6buRwEHM4
dZuo3XWMD0EgDWZlFKqbqBL9j83-Eudc4E&IDREQUEST=8316&ACTION=1&DATA=&UID=&STATUS=3 200 23.432 ms - 82
.GET execute
req.query.TOKEN: eyJ0eXA101KV1Q1LLC3hbGc101JiUzI1NlJ9.eyJpYXQ10jE0NjU1NzU0NzAsImV4cCI6MTQ2ODE2NzQ3MH0.6buRwEHM4dZuo3XWMD0EgDWZlFKqbqBL9j83-Eudc4E
lFKqbqBL9j83-Eudc4E
req.query.IDENCODER: MAC
req.query.ACTION: 4
Authentication OK
NO hay tareas pendientes
JSON-> { STATUS: 1, IDREQUEST: 0, ACTION: 0, DATA: , UID: }
    
```

Figura 56. Acción de lectura tiempo de espera superado (visto desde el servidor).

Las pruebas de datos recogidos de las tarjetas en ningún momento ha superado los 1440 Bytes de datos que es tope máximo de las tarjetas *smart card* (1KBytes, 720KB para datos) con las que se han realizado las pruebas y el establecido para esta primera versión.

7.4.7. Verificación y pruebas globales del módulo ESP8266

El consumo de memoria ha sido un punto importante a tener en cuenta, puesto que ha llevado más de problema no percatarse de qué variables estaban realizando un uso incremental de la memoria RAM. Ha sido necesario liberar el espacio de las mismas una vez acabada su utilización, para solucionar el inconveniente.

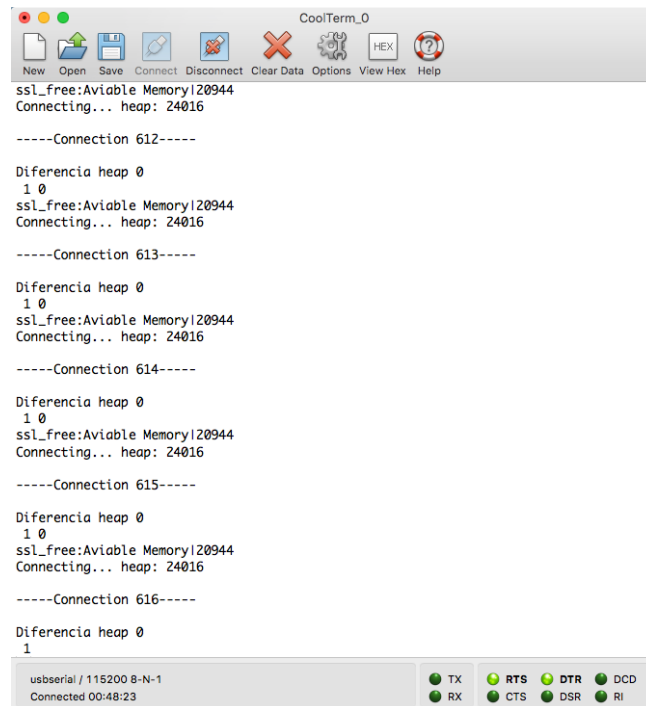


Figura 57. Memoria RAM (heap) del módulo ESP8266 estable en cada petición.

En el vídeo de la demostración del funcionamiento es accesible por medio de este enlace: http://tiny.cc/esp8266_demo

Capítulo 8

8. Gestión del proyecto

A lo largo del ciclo de vida del proyecto se ha requerido adaptar el alcance, en razón a los resultados que se iban obteniendo en el estudio de las alternativas descritas anteriormente. También ha sido necesario adaptar el alcance a causa del desconocimiento del funcionamiento del módulo ESP8266 y las funciones que podía emplear. A partir de ello se han tenido que definir un orden de tareas prioritarias y descartar las que no estuvieran dentro de la fecha fin del desarrollo de la aplicación. A medida que se obtenían unas funcionalidades se estimaban las siguientes, lo que

permitía complementar los casos de uso que se iban implementando durante el ciclo de desarrollo en espiral.

8.1. Gestión del alcance

El alcance ha sufrido varias adaptaciones durante el transcurso del proyecto, las más significativas han sido dos. Una durante el estudio de las cinco alternativas barajadas en primera instancia y la segunda propiciada por una dudosa documentación sobre los distintos SDKs de la empresa distribuidora del módulo ESP8266 utilizado a la hora de implementa.

La primera de ellas surgió mientras se evaluaban las cinco alternativas, dónde apareció una sexta (la utilización del módulo ESP8266) que resultó ser la alternativa más adecuada y la que daría un aliciente extra de cara al mercado, la cual se podía distribuir independientemente como lector de tarjetas.

El alcance inicial constaba de los siguientes puntos:

- Diseño del protocolo de comunicación entre cliente-servidor.
- Diseño de la estructura a compartir entre el ESP8266 y la obtención de datos del *transceiver* que lee las tarjetas.
- Conexión Wireless del módulo.
- Gestión de los procesos de la aplicación mediante un sistema embebido en tiempo real RTOS.
- Creación de un cliente TCP que siga el protocolo descrito.
- Lectura de tarjetas y lectura del número de serie.

Más adelante, en base de las nuevas necesidades del cliente se añadieron las siguientes características:

- Mejora del cliente TCP a un cliente TCP con encriptación SSL.
- Carga de los datos de la instalación de forma externa sin que estén *hard-codeados*.
- Almacenar de los parámetros de configuración de la instalación en la memoria flash del módulo ESP8266.
- Añadir un tiempo máximo de espera para que se cancele la transacción si no se posa la tarjeta.
- Añadir un número máximo de intentos de lectura de la tarjeta.

Cabe mencionar que esta aplicación se encuentra aún en una versión experimental y requiere mejoras para su completo funcionamiento.

8.2. Estudio de la viabilidad

A continuación se detalla el estudio en el que se identifican los diferentes aspectos del proyecto que podrían derivar en el no cumplimiento de los objetivos y como consecuente la insatisfacción de los resultados esperados. Junto al estudio se elabora un plan orientado a evitar esos posibles sucesos que puedan concluir el proyecto en fracaso o que impidan su desarrollo.

8.2.1. DAFO

	Puntos fuertes	Puntos débiles
De origen interno	FORTALEZAS Proyecto motivador Experiencia previa en la empresa Dedicación total	DEBILIDADES Desconocimiento de las alternativas Dificultades con el lenguaje de programación
De origen externo	OPORTUNIDADES Experiencia laboral Puesta en mercado Proyecto multidisciplinar Poca competencia	AMENAZAS Soporte muy limitado y confuso Adaptación al cliente

Tabla 10. Análisis DAFO del proyecto.

8.2.1.1. Fortalezas

- **Proyecto motivador:** Requiere de un estudio previo frente a unas necesidades que han de ser cubiertas con la mejor de las posibles soluciones. Además de estar orientado a un aprendizaje continuo y generalizado de todas las asignaturas cursadas en el grado.
- **Experiencia previa en la empresa:** Al conocer el funcionamiento de la empresa y a cada integrante de la misma por una previa realización de prácticas en empresa, la adaptación y conocimiento del dominio facilita ese previo aprendizaje.
- **Dedicación total:** No existe el riesgo de no llegar a los objetivos puesto que se va a invertir el tiempo necesario para satisfacerlos.

8.2.1.2. Debilidades

- **Desconocimiento de las alternativas:** Puesto que se parte desde cero se desconoce si actualmente existe alguna alternativa a la problemática surgida.

- **Dificultades con el lenguaje:** El lenguaje de programación C ha sido una espina un tanto atravesada durante el grado, distinta a la programación orientada a objetos, por una pobre dedicación al mismo.

8.2.1.3. Oportunidades

- **Puesta en mercado:** Actualmente el uso de tarjetas *smart cards* está aumentado a gran velocidad y este producto podría ofrecer muchas posibilidades en diferentes sectores:
 - En el sector de la hostelería: Para hoteles en sustitución a dispositivos cableados o que requieran de un ordenador físico cuando no se dispone del mismo (en el caso de los media centers).
 - Mecánica: Control de uso de piezas mediante una aplicación que lleve el seguimiento de las mismas.
 - Educación: Con carácter didáctico para que futuros técnicos o ingenieros estudien el funcionamiento interno de las tarjetas.
 - Cualquier otro sector que quiera gestionar accesos, como escuelas, universidades o cualquier edificio de acceso público o restringido.
- **Proyecto multidisciplinar:** Proporciona motivación aprender sobre otras especialidades que puedan proporcionar un añadido a los conocimientos propios.
- **Poca competencia:** Las características del producto y su portabilidad lo hacen un producto único y de fácil integración en cualquier instalación.

8.2.1.4. Amenazas

- **Soporte muy limitado y confuso:** La documentación generada por la empresa fabricante del módulo ESP8266 es parcial y no está del todo clara.
- **Adaptación al cliente:** El desarrollo de la aplicación puede verse alterado por la integración de nuevas funcionalidades o aspectos que alteren la duración del mismo.

8.2.2. Identificación de los riesgos

Los riesgos que podrían afectar negativamente en el desarrollo y satisfacción de los objetivos del proyecto se listan a continuación:

- **Estimación muy optimista:** Una falta de experiencia del uso de la tecnología como en la gestión de proyectos puede derivar en una estimación que afecte a la duración del mismo.
- **Soporte parcial:** La ausencia de una documentación clara y precisa puede repercutir en el tiempo de dedicación a la comprensión de la misma.

- **Problemas de salud:** Realización de actividades que derivan en problemas físicos.
- **Falta de disponibilidad:** Poca disponibilidad de personas involucradas en el proyecto.
- **Integración de todas las partes del proyecto:** Problemas de comunicación y de la tecnología a emplear puede derivar en interbloqueos de las diferentes partes.

8.2.3. Cuantificación de los riesgos

El gráfico que se muestra a continuación se evalúan los diferentes riesgos definidos en el apartado anterior. Donde se valoran las probabilidades de que sucedan y el impacto que supondrían en la cumplimiento de los objetivos.

Riesgo	Probabilidad	Impacto
Estimación muy optimista	Alta	Media
Soporte parcial	Media	Alto
Problemas de salud	Baja	Medio
Falta de disponibilidad	Alta	Bajo
Integración de cada parte	Media	Alta

Tabla 11. Tabla de cuantificación de riesgos.

8.2.4. Plan de contingencia

- **Estimación muy optimista:** A medida de que se van realizando las diferentes tareas del proyecto, serán necesarias replanificaciones y/o reducir el alcance del mismo si se ve oportuno, con el fin de cumplir la mayoría de los objetivos. Además se ha dejado un margen extra al final del desarrollo para posibles modificaciones.
- **Soporte parcial:** La empresa distribuidora del módulo ESP8266 tiene un foro en el que se pueden plantear dudas que no están o no se comprenden en la documentación que tienen disponible. Foro en el cual tardan en responder varios días y no siempre consiguen aclarar las dudas surgidas. De forma que habría que tomar un camino distinto e intentar dar con la solución de otra manera.
- **Problemas de salud:** Realización de labores de repartidor en moto con condiciones climatológicas adversas, que han propiciado algún accidente con la consiguiente falta de dedicación al proyecto. En este caso, sería conveniente aumentar las medidas de seguridad y protección.

- **Falta de disponibilidad de los involucrados:**
 - En el caso de la indisponibilidad de terceros para el desarrollo de las aplicaciones necesarias en combinación con las del proyecto, necesitarán una inversión mayor de horas para cumplir su cometido. En el caso más extremo habría que realizar ese desarrollo en su lugar.
 - En el caso de la indisponibilidad por parte del alumno, al igual que en el caso anterior, habría que realizar una replanificación y ajustar el tiempo de dedicación diario.
- **Integración de todas las partes del proyecto:** Puesto que se necesitan varias piezas para unir todo el puzzle. La unión de ellas puede repercutir en otras dejando de funcionar o que estén incompletas, en todo caso habría que volver a planificar y reducir el alcance si fuese necesario.

8.3. Gestión de cambios

Como todo proyecto de empresa, las necesidades de un proyecto se ven alteradas por distintos factores que pueden repercutir positivamente o negativamente en la ejecución del mismo. Estos cambios pueden darse en resultado a una investigación previa realizada o por satisfacer las nuevas necesidades que se le plantean al cliente.

La primera alteración surgió en la fase de estudio de las cinco alternativas software seleccionadas para dar solución a la problemática de desaparición de los *applets* de Java. Cada una de esas alternativas tal y como se describen en su documento de estudio tenía sus pros y sus contras. Si bien en unas requerían un navegador en las otras requerían el otro, lo que no permitía que la aplicación fuera multiplataforma además de necesitar de un ordenador físico, tendencia que empieza a desaparecer y sustituirse por *tablets* o *mediacenters*. Los resultados que se iban dando y las necesidades del cliente dieron lugar a la aparición de otra alternativa más ventajosa. Esta alternativa utilizaba tanto software como hardware que unidos podrían facilitar y dar un aliciente extra al desarrollo del proyecto.

La segunda alteración apareció a la hora de cambiar el protocolo de comunicación entre el cliente-servidor TCP a un protocolo TCP seguro bajo el cifrado SSL. Hubo que reestructurar y modificar sustancialmente la implementación porque la documentación sobre los SDKs del módulo ESP8266 no estaba muy clara. Esto dio confusión al uso de unas funciones (para establecer conexiones seguras) que el SDK en la versión Non-OS sí disponía y que el SDK en la versión RTOS no. De modo que se preguntó en el foro del fabricante y recomendaron el uso de la librería SSL que tienen ellos disponible, la cual implementa conexiones seguras.

Por otro lado, en medida que se desarrollaba la aplicación se tuvo que descartar la escritura de las tarjetas para esta primera versión por la compleja gestión de los procesos que hace el módulo ESP8266 en su SDK de RTOS.

Otro factor condicionante que puso en riesgo el proyecto fueron las correcciones y mejoras de esta primera versión por petición del cliente/empresa. Que tuvieron que ser realizadas fuera del periodo del desarrollo una vez comenzada la redacción de la memoria, que se solventó con la máxima dedicación diaria al proyecto.

8.4. Gestión de costes

En este apartado se detalla la distribución de horas dedicadas a cada una de las tareas realizadas del proyecto. Estas tareas son cinco: Estudio de las alternativas, Estudio del SDK, Desarrollo del proyecto, Gestión del proyecto y Trabajo Académico.

Tarea	Subtarea	Dedicación
Estudio de las alternativas previas	Resumen de las 5 alternativas	6
	Alternativa 1	31
	Alternativa 2	1
	Alternativa 3	1
	Alternativa 4	1
	Alternativa 5	1
Estudio Alternativa 6	Análisis del módulo	3
	Lectura documentación	15
	Comandos AT	4
	Eclipse SDK	20
	Arduino SDK	7
	NoN-OS SDK	5
	RTOS SDK	45
Desarrollo del proyecto	Especificación	5
	Análisis	10
	Diseño	15
	Implementación	105
	Pruebas	75
Gestión del proyecto	Gestión del alcance	10
	Gestión de costes	5
	Reuniones empresa	20
	Reuniones director	9
Trabajo académico	Redacción memoria	120
	Preparación de la defensa	25*
TOTAL		539

Tabla 12. Gestión de costes del proyecto.

8.5. Planificación

El proyecto comenzó el 25 de Enero del 2016 y se previó que se finalizara para el 22 de Junio del mismo año, fecha límite acordada para la entrega y subida de la memoria a la plataforma digital ADDI.

En cuanto al tiempo diario que se indicó para invertir en el proyecto, en los primeros tres meses hasta una vez acabadas las clases no sería inferior a 4 horas. Una vez terminadas las clases como mínimo se dedicarían 5 horas diarias. Cabe mencionar que han existido dos periodos vacacionales uno de 8 días que dio comienzo el 28 de Marzo y concluyó el 3 de Abril y el otro de 5 días que empezó el 18 de Abril y finalizó el 22 del mismo mes.

8.5.1. Gráfico Gantt

A continuación se muestra un gráfico Gantt que incluye las tareas principales del proyecto y las dependencias entre ellas, de haberlas.



Figura 58. Gantt general del proyecto.

A fin de que resulte más fácil la comprensión del gráfico anterior se mostrará un gráfico para cada una de las tareas del proyecto.

- **Estudio de las alternativas:** Se muestran cada unas de las alternativas junto. Cabe destacar que las alternativas entre la segunda y la quinta no tuvieron prácticamente dedicación por el giro que tuvo que darse en los acontecimientos como se indica en el apartado de gestión de cambios.

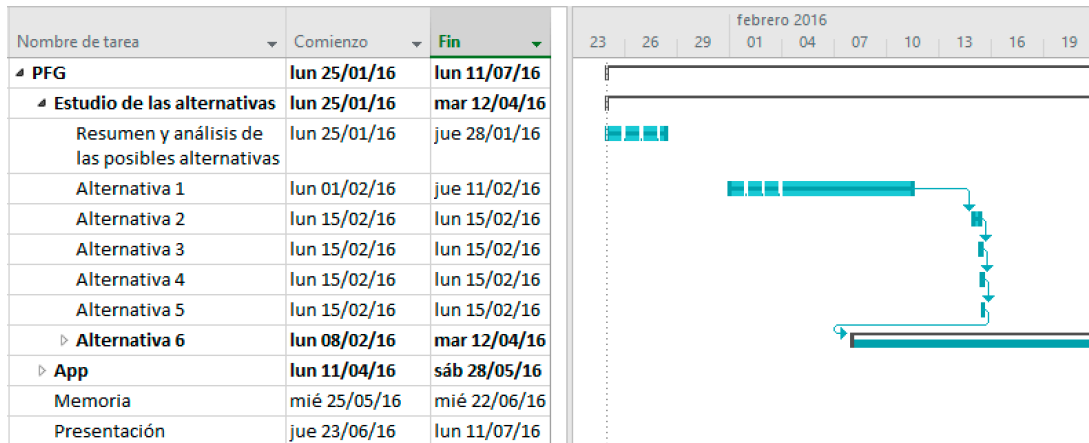


Figura 59. Gráfico Gantt del estudio de las alternativas.

- **Estudio y aprendizaje de la alternativa 6:** Ésta es la alternativa que se ha llevado a cabo, compuesta por las siguientes subtareas; el análisis del módulo ESP8266 y el estudio individualizado de las diferentes plataformas o herramientas en las que puede desarrollar.

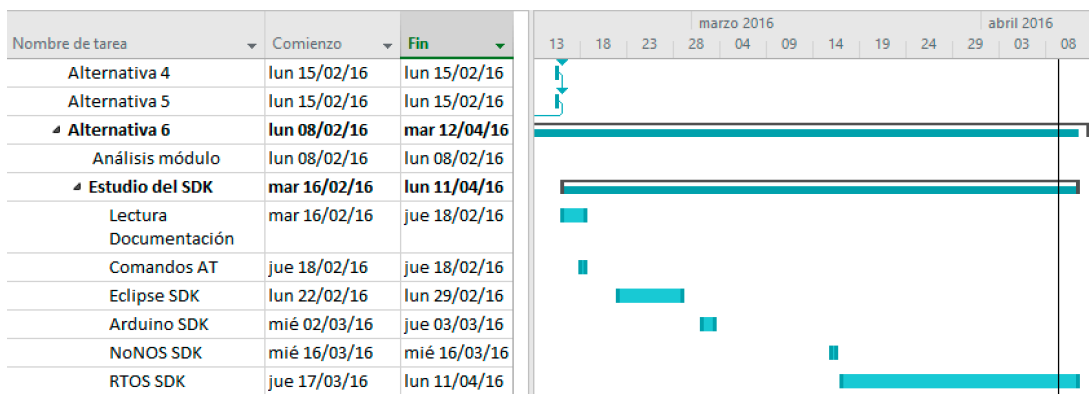


Figura 60. Gráfico Gantt etapa de aprendizaje del módulo ESP8266.

- **Especificación:** Se muestra la tarea de especificación de la aplicación que será de utilidad para saber que herramientas o funciones del módulo ESP8266 serán necesarias aprender.

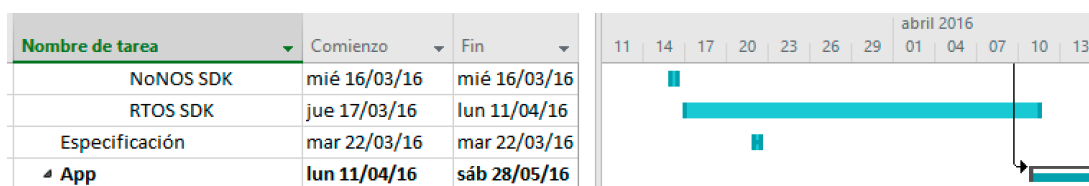


Figura 61. Gráfico Gantt de la especificación del funcionamiento esperado de la aplicación.

- **Ciclo de desarrollo de la aplicación:** Aquí se detalla cada una de las partes del ciclo de vida de desarrollo software de la aplicación.

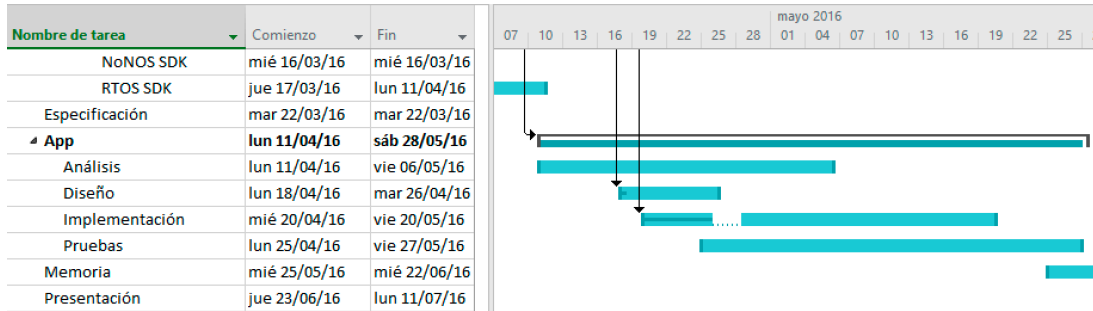


Figura 62. Gráfico Gantt del ciclo de vida de desarrollo software de la aplicación.

- **Memoria y presentación:** Para la redacción de la memoria se han ido creando pequeños informes de los estudios pero no ha sido hasta el periodo indicado en el gráfico siguiente cuando ha comenzado su redacción. En cuanto la presentación comenzara su preparación el día siguiente a la subida del proyecto a la plataforma digital ADDI.

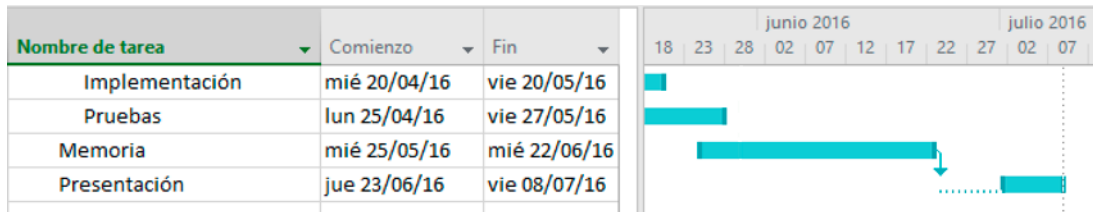


Figura 63. Gráfico Gantt de la redacción de la memoria y preparación de la presentación.

8.5.2. Diagrama de Hitos

En la tabla que se muestra a continuación se detallan los días con eventos importantes que de alguna manera han sido relevantes para el desarrollo del proyecto. Entre ellos están la fechas de inicio y fin del proyecto y reuniones de suma importancia. También ha de detallarse la fecha de la defensa del proyecto, que aunque esté fuera del periodo de entrega de la memoria es un acontecimiento muy importante a tener en cuenta para este proyecto.

Ciclo de vida del proyecto. Desde el 25/01/2016 al 11/07/2016																							
Hitos del proyecto	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15	S16	S17	S18	S19	S20	S21	S22	S23
Inicio proyecto	◆																						
Reunión inicial director	◆																						
Puesta en común ESP8266								◆															
Reunión seguimiento director											◆												
Puesta en común protocolo											◆												
Inicio memoria																		◆					
Matricular proyecto																		◆					
Borrador avanzado director																					◆		
Revisión de la memoria																					◆		
Depositar memoria ADDI																						◆	
Inicio de la presentación																						◆	
Defensa del proyecto																							◆

Tabla 13. Diagrama de hitos.

A continuación se listan las fechas exactas de cada acontecimiento descrito en el diagrama anterior:

- 25 de Enero del 2016: Inicio del proyecto.
- 29 de Enero del 2016: Reunión inicial con el director del proyecto.
- 22 de Marzo del 2016: Puesta en común de las funcionalidades del módulo ESP8266, división de las funcionalidades más significativas para luego exponerlas al resto de trabajadores.
- 11 de Abril del 2016: Reunión de seguimiento con el director.
- 15 de Abril del 2016: Puesta en común del protocolo de comunicación entre el cliente (el módulo ESP8266) y un servidor en la nube.

- 25 de Mayo del 2016: Inicio de la memoria.
- 1 de Junio del 2016: Matricular proyecto.
- 13 de Junio del 2016: Enviar un borrador avanzado de la memoria al director del proyecto.
- 17 de Junio del 2016: Revisión de la memoria.
- 22 de Junio del 2016: Depositar la memoria en la plataforma digital ADDI.
- 24 de junio del 2016: Inicio de la presentación de la defensa.
- 8-12 de Julio del 2016: Fechas para la defensa del proyecto.

Además de los hitos anteriores, semanalmente se han realizado reuniones de seguimiento con la empresa, para controlar los progresos y planificar las siguientes tareas.

Capítulo 9

9. Conclusiones

En este capítulo se realiza un análisis de los objetivos establecidos en el apartado 3.1. Alcance del proyecto y la metodología aplicada para el desarrollo de los mismos que se detalla en el capítulo 4. Metodología. Además de un segundo subapartado dedicado a la conclusiones obtenidas en la gestión del proyecto y las decisiones tomadas en el transcurso del mismo. Por último, se concluye con una opinión personal del autor y la experiencia personal y laboral adquirida durante todo el grado.

9.1. Conclusiones sobre los objetivos

En primer lugar, los análisis realizados sobre las alternativas no tienen nada que ver con cualquier otro análisis realizado en la universidad. El estudio de las alternativas sobre qué tecnología era la más precisa y cual era la que cumplía con todos los objetivos, ha sido más riguroso. Este tipo de análisis requieren más tiempo, esfuerzo y dedicación completa, para no fallar en la elección del camino a tomar.

A continuación, se describen los beneficios obtenidos de la metodología utilizada y las recomendaciones para el uso de la misma en proyectos, en los que se requiera de un estudio previo y de un desarrollo de éste. El uso de esta metodología ha sido una decisión muy acertada, ya que los estudios de las alternativas y el estudio más avanzado sobre el módulo ESP8266 se han visto comprometidos, por modificaciones inmediatas, que han podido realizarse gracias a esta metodología, reduciendo así los riesgos del proyecto. También ha facilitado el diseño, la implementación y las pruebas de cada uno de los casos de uso, recogidos a partir de la especificación, ya que se ha comenzado con los casos de uso básicos, hasta realizar los más complejos. Todo ello mediante los ejemplos realizados de cada unas de las librerías requeridas.

Se han satisfecho y cumplido todos los objetivos que se especificaron en el alcance inicial y las nuevas funcionalidades pedidas por el cliente. Sin embargo, hay que puntualizar que una de las nuevas funcionalidades solicitadas por el cliente, no ha podido realizarse, la cual se recoge en el capítulo 10. Propuestas de mejora. Se trata de la posibilidad de cancelar una acción (de lectura) en el módulo ESP8266, su causa se ha visto comprometida por las dificultades encontradas en el manejo de tareas del SDK RTOS.

9.2. Conclusiones sobre la gestión

Para que todo lo anterior no se convirtiera en un fracaso, se lleva un control y anotación de los siguientes puntos a tomar en cada una de las iteraciones, esto ha sido una de las claves para el éxito del proyecto, tal y como se ha realizado en cada una de las reuniones, se han agrupado los *action ítems* identificados al final de la misma, con las tareas a realizar en la siguiente iteración.

Gracias al seguimiento llevado, cada reincorporación realizada después de los periodos vacacionales, ha resultado más sencilla e inmediata, facilitando así la continuación de las tareas previstas.

Por otro lado, el estudio referido al módulo ESP8266 ha llevado más tiempo del previsto, por la escasa y dudosa documentación proporcionada por Espressif, que ha repercutido en el desarrollo, siendo necesario invertir más horas diarias en su realización. Concretamente a la hora de crear el cliente TCP SSL, que se creía que

una librería (*espconn.h*) realizaba el envío de peticiones de manera segura pero no se entendió bien y al no ser para el SDK RTOS hubo que cambiar sustancialmente el código. Pero gracias al soporte que la empresa Espressif ofrece en el foro referente al módulo ESP8266 se ha podido solucionar ese inconveniente.

9.3. Conclusiones personales

En una empresa es importante compartir información con el resto de compañeros, como puede ser el estudio sobre una tecnología, a fin de que ese esfuerzo realizado por uno de ellos (A) reduzca el coste que va a necesitar el otro (B), para aprenderla. A largo plazo, esto es beneficioso puesto que esta otra persona (B) podría añadirse al proyecto que realiza (A), para agilizar el desarrollo. De esta manera se reduce el coste, no teniendo que realizar todo ese estudio la otra persona (B). Así mismo esta última persona (B), también podría encargarse del estudio de otra tecnología. Estudio que pondría a disposición de la primera persona (A) y así entre las dos (A y B) sabrían lo mismo con menor coste para el proyecto.

Un gran aliciente durante desarrollo de una tecnología específica, es la presencia de un experto en la misma. Esto puede ser de gran ayuda, para reducir el coste que lleva el problema en cuestión a solucionar, de modo que el aprendiz invierte menos tiempo en la solución y puede beneficiar al experto para el que está ofreciendo el servicio.

En cuanto a la experiencia con el módulo, es un producto altamente recomendable puesto que es capaz de realizar muchas funciones, tanto aplicables al sector de la industria como personales. Por ejemplo, la monitorización de cualquier tipo de actividad mediante sensores y su publicación en una aplicación web, todo ello beneficiado por su bajo consumo energético.

Ahora se deja el relevo a la empresa para continuar con su desarrollo, aunque no se descarta que en un futuro se vuelva a utilizar el módulo ESP8266 para dotar a otro sistema de sus capacidades y funcionalidades, vistos los resultados y la experiencia gratificante obtenida.

Haber realizado el proyecto de fin de grado en empresa, te da una perspectiva adicional de lo que es un proyecto de verdad, un proyecto que esté definido por una necesidad, y con los inconvenientes de las cambiantes necesidades del cliente. Una visión más real de la vida, que no se puede apreciar cuando se realiza exclusivamente como un trabajo académico.

En cuanto a la experiencia propia vivida en el grado. En los primeros cursos me encontraba bastante perdido y no sabía qué es lo que me aportarían en un futuro las asignaturas que cursaba, y para qué me servirían. No fue hasta tercero cuando me empezó a gustar desarrollar lo aprendido y buscar otras herramientas alternativas para mejorar ese desarrollo. Además me di cuenta de que la mayor de las

satisfacciones las conseguía a través de la creación de nuevas e innovadoras aplicaciones.

Por otro lado, he sido de esas personas que únicamente se centraban en una especialidad de la informática, sin interesarle el resto, hasta una vez llegado al cuarto curso del grado, que he visto que con la unión de las tres especialidades puede lograrse aplicaciones muy completas e interesantes.

Como última reflexión, diría que no hay que cerrarse puertas porque una asignatura de una especialidad no sea del agrado de uno mismo, para que no se curse ninguna otra de la misma especialidad. Dentro del grado, al igual que dentro de cada especialidad, hay cientos y miles de aplicaciones diferentes de cada una de las asignaturas. Hay que tener en cuenta que el software sin el hardware y viceversa no son nada, por ello hay que saber de ambos, ya que unidos engloban la informática. La cual es un mundo de alternativas que continuamente está evolucionando y para el que hay que seguir un proceso de aprendizaje continuo.

Capítulo 10

10. Propuestas de mejora

Este proyecto desde un inicio ha ido evolucionando, ya que se le han ido añadiendo mejoras en el transcurso del mismo. No obstante, ha tenido que llegar a su fin, y en el cierre de éste se indican muchas ideas que no han podido realizarse y se han ido acumulando.

Todas esas ideas se recogen en el siguiente listado de propuestas de mejora, que han sido aportadas por la empresa, además de las origen propio que según se realizaba el desarrollo surgían. Todo ello con fin de que se dé continuidad al trabajo realizado.

1. El módulo como servidor y no como cliente de peticiones. Habría que modificar la arquitectura para que un cliente, por ejemplo un navegador web u otro servidor, realizara peticiones al módulo ESP8266 con las acciones a realizar. De esta manera se reduciría el tráfico de la red, además de poder poner en modo reposo el módulo para consumir menos energía y aumentar su autonomía si se conectase a baterías.
2. Añadir a una tarea (*task*) las funciones del *encoder*. Creando esta tarea podría gestionarse la tarea actual del servidor por una y la tarea del *encoder* por otra, a fin de que si el *encoder* está a la espera del posado de una tarjeta no repercuta en el servidor de peticiones. De esta manera se podría añadir un caso el caso de uso de cancelar acciones por si el usuario ha enviado una acción equivocada.
3. Añadir una validación para los datos que se envían en formato JSON de la configuración inicial de la instalación y el correspondiente *feedback* con los datos erróneos insertados. Aunque lo ideal sería construir una página web dentro módulo desde donde se configurara.
4. Actualmente los datos se almacenan en una dirección de memoria del módulo en bruto. La mejora consistiría en el uso de la librería *Spiffs* que crea un sistema de ficheros donde se pueden almacenar esos datos más ordenados.
5. Añadir soporte para más tarjetas *smart card* a parte de la MIFARE Plus.
6. Añadir más sensores y tecnologías como Bluetooth y NFC para la comunicación entre el módulo y los datos que contiene la tarjeta.
7. Añadir más indicadores en el módulo como un *display* que muestre mensajes o diodos led que muestren más estados.
8. Crear un cajero específico para ensamblar el módulo y el *transceiver* en él. Además, también que éste sea motorizado para coger la tarjeta y devolverla, como hace el cajero de un banco.

Capítulo 11

11. Bibliografía

- [1] Java web Start: *¿Qué es java Web Start y como se ejecuta?*
https://www.java.com/es/download/faq/java_webstart.xml
- [2] *Kolban's Book on ESP8266*. Neil Kolban, Enero 2016.
<https://drive.google.com/file/d/0B3nCzPJqtgWvbUFoNTd4anNVN0E>
- [3] *ESP8266 Espressif AT instruction set*, v. 24. Espressif Systems IOT Team
Copyright © 2015. <https://drive.google.com/file/d/0B3nCzPJqtgWvN2NrWTJQQ3JTbnM>

- [4] Vídeo tutorial error UART PL2303: Fix PL2303 Error Code 10 for windows 8/8.1 x86 and x64. <https://www.youtube.com/watch?v=H394FnMzeFo>
- [5] Windows UART DRIVER y pasos de instalación: *FAKE PL2303 – HOW TO INSTALL ON WINDOWS 8.1*. <http://wp.brodzinski.net/hardware/fake-pl2303-how-to-install/>
- [6] Windows Development Kit: *Everything ESP8266 - My Espressif DevKit for Windows + Eclipse IDE*. <http://www.esp8266.com/viewtopic.php?f=9&t=820>
- [7] OS X Development Kit: *ESP8266 Development Kit on Mac Os Yosemite and Eclipse IDE*. <http://tuanpm.net/esp8266-development-kit-on-mac-os-yosemite-and-eclipse-ide/>
- [8] Linux Developmet Kit: *Setting up the ESP8266 Open SDK*. <http://www.penninkhof.com/2015/03/esp8266-open-sdk/>
- [9] Uso del puerto serie en OS X: *Serial Terminal Basics*. <https://learn.sparkfun.com/tutorials/terminal-basics/connecting-to-your-device/>
- [10] Guía rápida de iniciación en el módulo ESP8266: *ESP8266_WiFi_Module_Quick_Start_Guide_v_1.0.4.pdf*
http://rancidbacon.com/files/kiwicon8/ESP8266_WiFi_Module_Quick_Start_Guide_v_1.0.4.pdf
- [11] *Tutorial*. <http://randomnerdtutorials.com/esp8266-web-server/>
- [12] Web server con comandos AT + C: *Tutorial: Web Server with the ESP8266 WiFi Module | MCU on Eclipse* . <http://mcuoneclipse.com/2014/11/30/tutorial-web-server-with-the-esp8266-wifi-module/>
- [13] Web server control de temperatura: *ESP8266 powered web server + LED control + DHT22 temperature/humidity sensor reading | Martin's corner on the web*. <https://harizanov.com/2014/11/esp8266-powered-web-server-led-control-dht22-temperaturehumidity-sensor-reading/>
- [14] Conexión del módulo y creación de un servidor web con Arduino: *Simple Arduino Web Server on ESP-07/ESP-12 Tutorial*. <https://www.youtube.com/watch?v=8J7zflVO8K0>
- [15] Configuración de Arduino la librería “esptool”: *ESP8266 Arduino IDE on Mac OS X Yosemite 10.10.3*. <http://hpclab.blogspot.com.es/2015/06/esp8266-arduino-ide-on-mac-os-x.html>
- [16] Documentación oficial de Espressif: *Recopilación de todos los SDKs y diferentes funciones que puede realizar el módulo ESP8266*. <https://drive.google.com/folderview?id=0B3nCzPJqtgWvTFZNNFhsTmdhcEk>
- [17] Uso de ESP-NOW: *About new ESP-NOW functions on SDK 1.2.0*. <http://bbs.espressif.com/viewtopic.php?t=689#p2663>
- [18] Red malla o Mesh: *Mesh networking*. https://en.wikipedia.org/wiki/Mesh_networking
- [19] ESP8266 ESPCONN server: *Accepting incoming tcp connections on the esp8266 (trivial example)*. <http://41j.com/blog/2015/01/accepting-incoming-tcp-connections-esp8266-trivial-example/>

- [20] Tarjetas MIFARE Plus: *Familia MIFARE Plus*.
<https://www.mifare.net/es/productos/ics-de-tarjetas-con-chip/familia-mifare-plus/>
- [21] Desaparece java de los navegadores: *El plugin de Java para navegadores tiene los días contados*. <http://www.xataka.com/servicios/el-plugin-de-java-para-navegadores-tiene-los-dias-contados>
- [22] El plugin de Java se adapta: *Ya hay fecha para la muerte del plugin de Java*.
<http://www.adslzone.net/2016/01/28/ya-hay-fecha-para-la-muerte-del-plugin-de-java/>
- [23] No hay soporte para el *plugin* en el navegador Chrome: *Plugin-based content doesn't work on Chrome*. <https://support.google.com/chrome/answer/6213033>

Capítulo 12

12. Glosario y Acrónimos

Ahead-of-time (*compilation*): Es el acto de compilar código de alto nivel como C o C++ en *bytecode* de Java o .NET, en código máquina con la intención de ejecutar el fichero binario nativamente. Éste es beneficioso cuando el intérprete es muy lento.

AP: (*Access Point*) Punto de acceso inalámbrico, que interconecta equipos de comunicación inalámbricos para formar una red inalámbrica.

Applet: Es un componente de una aplicación que se ejecuta en el contexto de otro programa, por ejemplo en un navegador web. El *applet* debe ejecutarse en un

contenedor, que le proporciona un programa anfitrión, mediante un *plugin*, o en aplicaciones como teléfonos móviles que soportan el modelo de programación por *applets*.

Applet de Java: Programa escrito en Java que forma parte de los componentes de una página de Internet. Han sido usados para proporcionar funcionalidad a páginas de Internet que no pueden ser satisfechas usando únicamente HTML. El uso de *applets* llegó a ser muy conveniente ya que el código es independiente del sistema operativo que se esté corriendo el navegador. El código del *applet* es ejecutado por el mismo navegador a través de una máquina virtual java (JVM). Con la llegada del HTML5, el uso de *applets* se ha visto disminuido y tiende a desaparecer, además de que navegadores como Edge o Chrome ya no tienen habilitada la posibilidad de ejecutar *applets* por defecto o han quitado el soporte de los mismos.

API: (*Application Programming Interface*) Interfaz de programación de aplicaciones como conjunto de subrutinas, funciones y procedimientos que ofrece cierta biblioteca para ser utilizada por otro software.

Bytecode: Es un código intermedio más abstracto que el código máquina. Habitualmente es tratado como un archivo binario que contiene un programa ejecutable similar a un módulo objeto, que es un archivo binario producido por el compilador cuyo contenido es el código objeto o código máquina.

CDT: (*C/C++ Development Tool*) Es un IDE de potencia industrial para los lenguajes de programación C y C++ que también sirve como una plataforma para proporcionar herramientas de valor añadido para los desarrolladores de estos lenguajes.

CoolTerm: Es una aplicación basada en *HyperTerminal* para Mac OS X, al igual que RealTerm de Windows.

Cross-plataform: Programas que son implementados e interoperan en múltiples plataformas informáticas, pueden dividirse en dos tipos; los que requieren una compilación individual para cada plataforma o los que se pueden ejecutar directamente en cualquier plataforma sin ninguna preparación, como cualquier software en un lenguaje interpretado.

Eclipse IDE: Es una plataforma de software compuesto por un conjunto de herramientas de programación de código abierto multiplataforma para desarrollar aplicaciones.

Encoder: En este contexto se refiere al dispositivo lector de la información contenida en las tarjetas.

Hard-code: Es una mala práctica en el desarrollo de software que consiste en incrustar datos directamente (a fuego) en el código fuente del programa, en lugar de obtener esos datos de una fuente externa como un fichero de configuración.

HTTP: (*Hypertext Transfer Protocol*) Protocolo de transferencia de hipertexto, es el protocolo de comunicación que permite las transferencias de información en la World Wide Web (WWW).

IDE: (*Integrated Drive Enviroment*) Entorno de desarrollo integrado, es una aplicación informática que proporciona servicios integrales para facilitar al desarrollador o programador el desarrollo de software.

GET: Método de peticiones HTTP. Envía datos para que sean procesador por el recurso identificado, datos que se incluyen en la cabecera de la petición.

Git: Es un software de control de versiones pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente.

GPIO: (*General Purpose Input/output*) Entrada/Salida de propósito general, es un pin genérico en un chip, cuyo comportamiento, ya sea de entrada o salida, se puede controlar por el usuario en tiempo de ejecución.

JSON: (*JavaScript Object Notation*) Formato de texto ligero para el intercambio de datos.

JWS o JAWAS: (*Java Web Start*) Es la implementación de Java Network Launching Protocol (JNLP), desarrollada por Oracle, mediante la cual permite arrancar aplicaciones Java que están en un servidor web de aplicaciones comprobando previamente si el cliente tiene la versión actualizada de dicha aplicación.

NPAPI: (*Netscape Plugin Application Programming Interface*) Es una interfaz de programación de aplicaciones (API) que permite a los *plugins* (extensiones del navegador) ser desarrollados para los navegadores web. Fue desarrollado por primera vez para los navegadores Netscape, a partir de 1995 con Netscape Navigator 2.0, pero fue adoptado posteriormente por otros navegadores. Cuando el navegador se encuentra con un tipo de contenido que no puede manejar de forma nativa, carga el *plugin* adecuado, reserva un espacio dentro del contexto del navegador para el *plugin* para realizar la función que tenga que desempeñar y luego transmitir los datos de vuelta al navegador.

NFC: (*Near Field Communication*) Comunicación de campo cercano, es una tecnología de comunicación inalámbrica, de corto alcance y alta frecuencia que permite el intercambio de datos entre dispositivos.

Non OS: (*Non Operating System*) Sistema operativo mono hilo.

P2P: (*Peer-to-peer*) Red de pares, es una red de ordenadores en la que todos o la mayoría funcionan sin clientes ni servidores fijos, sino una serie de nodos que se comportan como iguales entre sí.

POST: Método de peticiones HTTP. Envía datos para que sean procesador por el recurso identificado, datos que se incluyen en el cuerpo de la petición.

Plugin: Es un complemento que se relaciona con una aplicación para agregarle una funcionalidad más específica. Esta aplicación adicional es ejecutada por la aplicación principal a través de una interfaz de programación de aplicaciones.

QSPI: (*Queued serial peripheral interface*) es un tipo de controlador SPI que utiliza una cola de datos para transferir datos a través del bus de datos SPI.

RAM: (*Random Access Memory*) Memoria de acceso aleatorio, memoria de trabajo de computadoras para el sistema operativo, los programas y la mayor parte del software.

RealTerm: Es una aplicación basada en *HyperTerminal* para Windows, un programa que se puede utilizar para conectarse a otros equipos, sitios Telnet, sistemas de anuncios (BBS), servicios en línea y equipos host, utilizando un módem o una conexión Ethernet.

REST: (*Representational State Transfer*) Transferencia de Estado Representacional, es un estilo de arquitectura software para sistemas hipermedia distribuidos como la *World Wide Web*.

RFID: Es un sistema de almacenamiento y recuperación de datos remoto que usa dispositivos denominados etiquetas, tarjetas, transpondedores o tags RFID.

RTOS: (*Real Time Operating System*) sistema operativo de tiempo real es un sistema operativo que ha sido desarrollado para aplicaciones de tiempo real.

Sandbox: Caja de arena, un entorno de ejecución distinto al que se ejecuta la aplicación inicial. Mecanismo seguro para separar la ejecución de programas. Usado normalmente para ejecutar código no testado o programas de terceros de origen desconocido, como de páginas web u otros usuarios. Permite ejecutar programas a conjunto de recursos controlados, como un espacio temporal en memoria y disco.

SIMD: (*Single Instruction, Multiple Data*) es una técnica empleada para conseguir paralelismo a nivel de datos.

Silverlight: Es una herramienta de desarrollo para la Web y aplicaciones móviles. Silverlight es un *plugin* gratuito, impulsado por el marco .NET y compatible con múltiples navegadores, dispositivos y sistemas operativos.

Slack: Es una herramienta de comunicación en equipo que ofrece salas de chat organizadas por temas, así como grupos privados y mensajes directos.

Smart card: Es cualquier tarjeta del tamaño del bolsillo con circuitos integrados, que permite la ejecución de cierta lógica programada. Aunque existe un diverso rango de aplicaciones, hay dos categorías principales de TCI. Las tarjetas de memoria contienen sólo componentes de memoria no volátil y posiblemente alguna lógica de seguridad. Las tarjetas microprocesadoras contienen memoria y microprocesadores.

Smart card reader: Es un dispositivo de entrada que lee datos de tarjetas *smart card*. Son de diferentes formas:

- Teclados que incorporan *smart card readers*.
- Dispositivos externos e internos.
- Algunos ordenadores portátiles que incorporan *smart card readers*.

Dispositivos externos que pueden leer un PIN o otra información que también pueden estar conectados a un teclado, llamados *card readers* con *PIN pad*.

SSL: (*Secure Sockets Layer*) Capa de puertos seguros, es un protocolo criptográfico que proporciona comunicaciones seguras por una red.

SPI: (*Serial Peripheral Interface*) Interfaz periférica serie, es un bus de comunicación serie síncrona utilizada para comunicación a corta distancia, sobre todo en sistemas embebidos.

STA: (*Station*) Estación, dispositivo que tiene capacidad de usar en protocolo de comunicación 802.11. Puede ser un ordenador portátil, un ordenador de sobremesa, PDA punto de acceso o un teléfono Wi-Fi.

Swing: es una biblioteca gráfica para Java. Incluye *widgets* para interfaz gráfica de usuario tales como cajas de texto, botones, desplegados y tablas.

TCP: (*Transmission Control Protocol*) Protocolo de Control de Transmisión, es uno de los protocolos fundamentales en Internet para que puedan comunicarse de forma segura.

Transceiver: Es un dispositivo que cuenta con un transmisor y un receptor que comparten parte de la circuitería o se encuentran dentro de la misma caja.

UART: (*Universal Asynchronous Receiver-Transmitter*) Transmisor-Receptor Asíncrono Universal, es el dispositivo que controla los puertos y dispositivos serie. El **UART** toma bytes de datos y transmite los bits individuales de forma secuencial.

USB: (*Universal Serial Bus*) Bus Universal en Serie, es un bus estándar industrial que define los cables, conectores y protocolos usados en un bus para conectar, comunicar y proveer de alimentación eléctrica entre computadoras, periféricos y dispositivos electrónicos.

Web View: Es un componente del sistema con la tecnología de Chrome que permite a las aplicaciones mostrar contenido Web.

Wi-Fi: (*Wireless Fidelity*) Fidelidad Inalámbrica, se trata de un conjunto de redes que no necesitan cables y que funcionan en base a ciertos protocolos previamente establecidos.

ZIP: ZIP es un formato de compresión sin pérdida, muy utilizado para la compresión de datos como documentos, imágenes o programas.

Anexo A

A. Uso de Java Web Start

Éste es un ejemplo de creación y despliegue de una aplicación Java Web Start. Por un lado se crea una simple aplicación JWS y a continuación se crea un chat entre un cliente brindado por un servidor apache y un cliente brindado por un servidor Node Js dentro de la aplicación Java a fin de que sirva de ejemplo de comunicación entre dos servidores. Estos servidores podrían simular una prueba en la vida real en la que estuviese ejecutándose una aplicación en uno de ellos y que otra aplicación necesitase datos de la aplicación anterior.

A.0. Herramientas utilizadas

- OS X o Linux.
- Desarrollar de la aplicación java:
 - Netbeans 8.1 → <https://netbeans.org/downloads/>
- Servidores de aplicaciones:
 - Apache (XAMP) → <https://www.apachefriends.org/es/index.html>
 - Node Js v4.2.6 → <https://nodejs.org/en/>
- Librería para comunicación entre puertos para Node Js:
 - Socket.io v1.4.5 → <http://socket.io/download/>

A.1. Ejemplos aplicación JWS

A.1.1. Pasos de creación

A continuación se listan los pasos para la creación, desarrollo y producción de una aplicación Java Web Start.

1. Creación de un proyecto Java en eclipse: alternative1_webstart

2. Posibilidades para la aplicación Java:

a. Opción 1. Crear una clase `Alternative1_webstart` que extienda.

`JFrame`:

```
package alternative1_webstart;

import javax.swing.JFrame;
import javax.swing.JLabel;

/**
 * @author AndoniSanchez
 */

public class Alternative1_webstart extends JFrame{
    private JLabel label = new JLabel("Hello!");
    public Alternative1_webstart() {
        super("Java Web Start Example");
        this.setSize(350, 200);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLayout(null);
    }
    public void addButtons() {
        label.setSize(200, 30);
        label.setLocation(80, 50);
        this.getContentPane().add(label);
    }
    public static void main(String[] args) {
        // TODO code application logic here
        Alternative1_webstart alt1 = new Alternative1_webstart();
        alt1.addButtons();
        alt1.setVisible(true);
    }
}
```

b. Opción 2. Web View (simular un navegador web mediante java).

i. Crear la clase siguiente `Alternative1_webstart` que extienda `Application`:

```
package alternative1_webstart;

import javafx.application.Application;
import javafx.geometry.HPos;
import javafx.geometry.VPos;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.layout.HBox;
import javafx.scene.layout.Priority;
import javafx.scene.layout.Region;
import javafx.scene.paint.Color;
import javafx.scene.web.WebEngine;
```

```
import javafx.scene.web.WebView;
import javafx.stage.Stage;
public class Alternative1_webstart extends Application {
    private Scene scene;
    @Override public void start(Stage stage) {
        // create the scene
        stage.setTitle("Alternative 1 - Java Web Start - With Web
View");
        scene = new Scene(new Browser(),750,500,
Color.web("#666970"));
        stage.setScene(scene);
        //adding CSS
        //scene.getStylesheets().add("webviewsample/BrowserTool
bar.css");
        stage.show();
    }
    public static void main(String[] args){
        launch(args);
    }
}
class Browser extends Region {
    final WebView browser = new WebView();
    final WebEngine webEngine = browser.getEngine();
    public Browser() {
        //apply the styles
        getStyleClass().add("browser");
        // load the web page
        webEngine.load("http://localhost:8080/webview.html");
        //add the web view to the scene
        getChildren().add(browser);
    }
    private Node createSpacer() {
        Region spacer = new Region();
        HBox.setHgrow(spacer, Priority.ALWAYS);
        return spacer;
    }
    @Override protected void layoutChildren() {
        double w = getWidth();
        double h = getHeight();
        layoutInArea(browser,0,0,w,h,0, HPos.CENTER, VPos.CENTER);
    }
    @Override protected double computePrefWidth(double height) {
        return 750;
    }
    @Override protected double computePrefHeight(double width) {
        return 500;
    }
}
```

```
    }  
}
```

ii. Crear el fichero `webview.html`:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  
<html lang="en-US">  
  <head>  
    <title>Testing Alternative 1 - Java Web Start</title>  
    <meta http-equiv="Content-Type" content="text/html;  
charset=windows-1252">  
  </head>  
  <body>  
    <h1>Testing Alternative 1 - Java Web Start</h1>  
    <p>Testing Alternative 1 with Java web view </p>  
    <p><a href="http://localhost:8080/webview_test_page.php">Go  
to test page</a></p>  
  </body>  
</html>
```

iii. Crear el fichero `webview_test_page.php`:

```
<?php  
    echo '<p>This is a test page</p>';  
    echo '<p><a href="http://localhost:8080/webview.html">Return back</a></p>';  
?>
```

3. Exportación la aplicación en un fichero `.jar`:
 - a. Click derecho en el proyecto → *Properties* → *Build* → *Packaging*, seleccionar *Compress JAR File* y *OK*.
 - b. Click derecho en el proyecto → *Clean and build*.
 - c. El fichero `.jar` se encuentra en la carpeta *Dist* de la carpeta donde está almacenado el proyecto.
4. Copiar el fichero a la raíz del servidor Apache: `./XAMP/htdocs`
5. Ejecutar el siguiente comando en la terminal, para empaquetar la aplicación en un ejecutable: `jar -cf alternative1_webstart.jar *.*`
6. Crear una firma y firmar el fichero `.jar`; responder a las preguntas que propone el primer comando:

```
keytool -genkey -keystore testkeys -alias myKey  
jarsigner -keystore testkeys alternative1_webstart.jar myKey
```

7. Crear el fichero `alternativel_webstart.jnlp` de la aplicación en la raíz del servidor Apache:

```
<?xml version="1.0" encoding="utf-8"?>
<jnlp spec="1.0+" codebase="http://localhost:8080/"
href="alternativel_webstart.jnlp">
  <information>
    <title>JNLP Example</title>
    <vendor>Andoni Sánchez</vendor>
    <homepage href="http://localhost:8080/" />
    <description>JNLP Testing</description>
  </information>
  <security>
    <all-permissions/>
  </security>
  <resources>
    <j2se version="1.6+" />
    <jar href="alternativel_webstart.jar" />
  </resources>
  <application-desc main-
class="alternivel_webstart.Alternativel_webstart" />
</jnlp>
```

8. Crear un fichero `index.html` en la raíz del servidor Apache desde el cual llamaremos a la aplicación java:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html lang="en-US">
  <head>
    <title>Testing Alternative 1 - Java Web Start</title>
    <meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
  </head>
  <body>
    <h1>Testing Alternative 1 - Java Web Start</h1>
    <p>Testing Alternative 1 <a
href="http://localhost:8080/alternativel_webstart.jnlp"> here </a> or:</p>
    <script src="https://www.java.com/js/deployJava.js"></script>
    <script>
      var url = "http://localhost:8080/alternativel_webstart.jnlp";
      deployJava.createWebStartLaunchButton(url, '1.6.0');
    </script>
  </body>
</html>
```

A.1.2. Configuración y ejecución del servidor Apache

Arrancar el servidor apache en el puerto 8080:

Abrir *XAMP* con privilegios de administrador → *Manage Servers* → Seleccionar *Apache Web Server* → *Configure* → *Port* y escribir *8080* → *OK* → *Start*.

Acceder a la URL: <http://localhost:8080/>

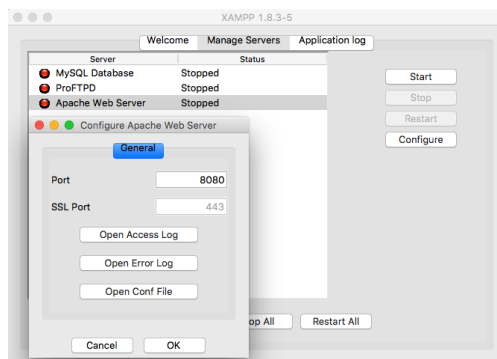


Figura 64. Diagrama de flujo de la aplicación JWS.

A.1.3. Diagrama de flujo

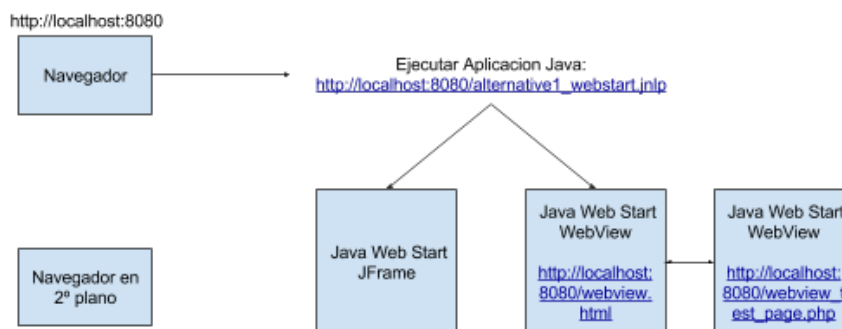


Figura 65. Diagrama de flujo de la aplicación JWS.

A.1.4. Uso de la aplicación java mediante Java Web Start

1. Dirigirse a la URL donde está el servidor Apache ejecutado (<http://localhost:8080>).
2. Clicar en cualquiera de los enlaces.

Testing Alternative 1 - Java Web Start

Testing Alternative 1 [here](#) or:



Figura 66. Imagen de la página web que ofrece la aplicación JWS.

3. Una vez descargada la aplicación, puesto que la firma no es de una fuente segura hay que permitir a Java Web Start que ejecute aplicaciones de las urls de “confianza”:
 - a. Abrir el panel de control de Java → *Security* → *Edit Site List* → *Add* y añadimos la URL de la cual se provee la aplicación.



Figura 67. Ejemplo para permitir la ejecución de una aplicación JWS.

- b. Ejecutar la aplicación *alternative1_webstart.jnlp*. Aparece una ventana advirtiéndole que la aplicación a ejecutar es de una fuente desconocida, aceptar el riesgo de uso de la misma y *Run*.
4. Aplicación en funcionamiento:
 - a. Opción 1: JFrame.

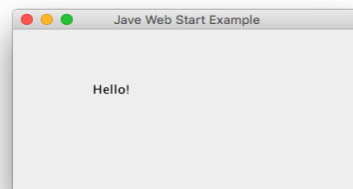


Figura 68. Imagen de ejecución de un etiqueta dentro de un JFrame en una aplicación JWS.

- b. Opción 2: WebView.

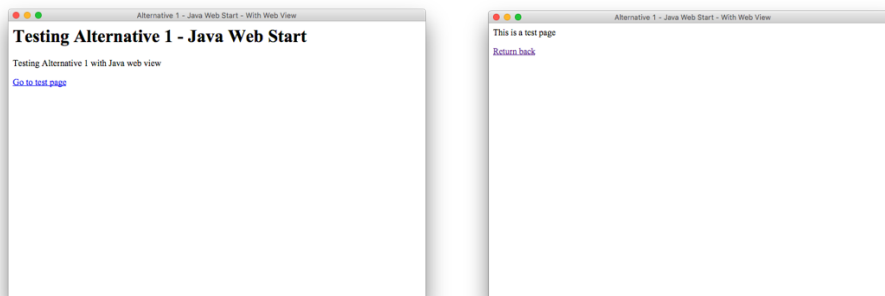


Figura 69. Imagen de ejecución de una página HTML dentro de un Web View en una aplicación JWS.

A.1.5. Ficheros creados

Opción 1. JFrame:

- Proyecto de NetBeans:
 - <https://drive.google.com/file/d/0B3nCzPJqtgWvUE4wVVJJeUhqdTQ>
- Ficheros para el servidor Apache XAMP:
 - <https://drive.google.com/folderview?id=0B3nCzPJqtgWvZy1GaXZpdm5hQWc>

Opción 2. WebView:

- Proyecto de NetBeans:
 - <https://drive.google.com/file/d/0B3nCzPJqtgWvYjl6TjZjc2x4U00>
- Ficheros para el servidor Apache XAMP:
 - <https://drive.google.com/folderview?id=0B3nCzPJqtgWvM3ZYTHI1bGNabXM>

A.2. Ejemplo JWS con Node Js

A.2.1. Instalación y configuración del servidor Node Js + Express

Instalación de Node Js:

- Instalar Node Js a partir desde el enlace del apartado: A.0. Herramientas utilizadas.

Instalación de paquetes necesarios para Node Js:

1. Express: *npm install express@4.9.0*
2. Librería Socket.io: *npm install socket.io*
3. Librería para parsear parámetros: *npm install body-parser*
4. Librería para realizar llamadas http request: *npm install request*

Para la instalación de cada paquete en primer lugar crear una carpeta para el servidor, por ejemplo: *alternative1_node*. Para la instalación de los paquetes necesarios, dentro de la carpeta *alternative1_node* ejecutar los comandos anteriores en el orden indicado.

Se crea una carpeta *node_modules* que contiene los módulos instalados y los paquetes necesarios para ejecutar el servidor.

A.2.2. Comunicación cliente-servidor Node Js mediante *sockets*

A.2.2.1. Creación de ficheros

Crear la siguiente fichero del servidor Node Js en la raíz de la carpeta `alternative1_node`:

main.js

```
var express = require('express');
app = express();
server = require('http').createServer(app);
io = require('socket.io')(server);

//server root directory
app.use(express.static('public'));

//Every page create a socket connection.
io.on('connection', function (socket) {
    //welcome message
    message = 'Node, new connection!';
    console.log('Node, new connection');
    socket.emit('wellcome', {message: message});
    //disconnect message
    socket.on('disconnect', function(){
        console.log('Node, disconnection');
    });
    //sent message
    socket.on('node_message', function(data){
        //reply the message to io --> everyone connected
        io.emit('node_message', data);
        //reply the message to connected socket
        //socket.emit('message', data);
        console.log("Node message: " + data.message);
    });
});

//server listen port 8081.
server.listen(8081, 'localhost', function(){
    console.log('listening on *:8081');
});
```

A continuación, en la raíz de la carpeta `alternative1_node` crear la carpeta *public* (carpeta raíz del servidor que se ha establecido como estática para contener los ficheros del servidor). Dentro de esa carpeta crear la siguiente fichero `index.html` para

el envío de mensajes. Estos envíos se realizan entre el cliente y servidor Node Js y viceversa por medio de sockets:

index.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html lang="en">
  <head>
    <title>Testing Alternative 1 - Java Web Start</title>
    <meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
  </head>
  <body>
    <h1>Testing Alternative 1 - Java Web Start (NODE)</h1>
    <p>Testing Alternative 1 with Java web view and NodeJs.</p>
    <h3>Socket message: </h3>
    <ul id="messages"></ul>
    <br>
    Message: <input type="text" id="messageText"/><input type="button"
onclick="send()" value="Send"/> <br>
    <!-- Socket.io -->
    <script src="/socket.io/socket.io.js"></script>
    <!-- jQuery -->
    <script src="http://code.jquery.com/jquery-latest.min.js"
type="text/javascript"></script>

    <script>
var socket = io.connect('http://localhost:8081', { 'forceNew': true });
$('#messageText').focus();
//writen messages from node to node
socket.on('node_message', function(data){
  console.log(data);
  $('#messages').append('<li>').text('Node message: ' + data.message));
});
//writen messages from php to node
socket.on('php_message', function(data){
  console.log(data);
  $('#messages').append('<li>').text('PHP message: ' + data.message));
});

function send(){
  //send to node socket
  socket.emit('node_message', {message:$('#messageText').val()});
  $('#messageText').val('');
  $('#messageText').focus();
}
    </script>
  </body>
```

A.2.2.2. Ejecución del servidor

Por último situarse en la carpeta de la raíz del servidor Node Js y ejecutarlo arrancar Node Js a través de la terminal:

```
cd /Users/andonisanchezantolin/Downloads/alternative1_node/node main.js
```

Acceder a la URL <http://localhost:8081/> y empezar el envío de mensajes.

A.2.2.3. Diagrama de flujo

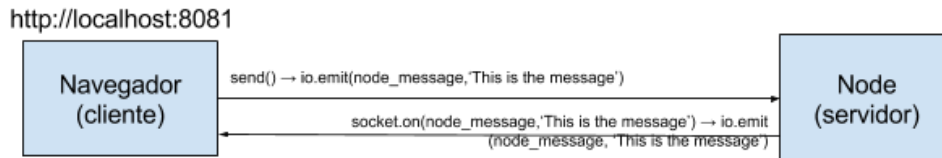


Figura 70. Diagrama de flujo de comunicación entre el cliente y el servidor de Node JS.

A.2.2.4. Ejemplo en funcionamiento

Navegador:

<p>Testing Alternative 1 - Java Web Start (NODE)</p> <p>Testing Alternative 1 with Java web view and NodeJs.</p> <p>Socket message:</p> <ul style="list-style-type: none"> • Node message: Hello! <p>Message: <input type="text" value="How are you?"/> <input type="button" value="Send"/></p>	<p>Testing Alternative 1 - Java Web Start (NODE)</p> <p>Testing Alternative 1 with Java web view and NodeJs.</p> <p>Socket message:</p> <ul style="list-style-type: none"> • Node message: Hello! • Node message: How are you? <p>Message: <input type="text"/> <input type="button" value="Send"/></p>
--	---

Figura 71. Chat de mensajes entre los servidores Apache y Node JS.

Mensajes enviados y recibidos mostrados en la consola del servidor Node Js:

```

→ alternative_node node main.js
listening on *:8081
Node, new connection
Node message: Hello!
Node message: How are you?
  
```

Figura 72. Consola de Node JS, conexión de un cliente y mensajes escritos en el socket.

A.2.3. Comunicación desde un servidor apache a un servidor node Js y al cliente (Node Js)

A.2.3.1. Creación de ficheros

Para realizar una comunicación entre un servidor apache y un servidor Node Js ha de realizarse una llamada *http request* desde el servidor apache al servidor Node Js. Para ello desde un cliente se realiza una llamada POST a un fichero PHP que se encarga de crear una url y realizar una llamada al servidor Node JS. Los ficheros creados son los siguientes: *index.html* y *send_message.php*.

index.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html lang="en">
  <head>
    <title>Testing Alternative 1 - Java Web Start</title>
    <meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
  </head>
  <body>
    <h1>Testing Alternative 1 - Java Web Start (PHP)</h1>
    <p>Testing Alternative 1 <a href="http://localhost:8080/alternative1_webstart.jnlp">
here</a>.</p>
    <h3>Sent and reveived messages: </h3>
    <ul id="messages"></ul>
    <br>
    Message: <input type="text" id="messageText"/><input type="button"
onclick="send()" value="Send to Node"/> <br>
    <script src="https://www.java.com/js/deployJava.js"></script>
    <!-- Socket.io -->
    <script src="http://localhost:8081/socket.io/socket.io.js"></script>
    <!-- jQuery -->
    <script src="http://code.jquery.com/jquery-latest.min.js"
type="text/javascript"></script>
    <script>
      function send(){
        var data={message:${'#messageText'}.val()};
        $.ajax({
          url: "http://localhost:8080/send_message.php",
          type: "POST",
          data: data,
          success: function(result){
            console.log(result);
          }
        });
        $('#messageText').val('');
        $('#messageText').focus();
        $('#messages').append($('- ').text('PHP message: ' + data.message));
      }
    </script>
  </body>
</html>

```

send_message.php

```
<?php
function notifyNode($data) {
    //$host = 'http://localhost:8081';
    $json = json_encode($data);

    $ch = curl_init();
    curl_setopt($ch, CURLOPT_URL, 'http://localhost:8081/');
    curl_setopt($ch, CURLOPT_HEADER, 1);
    curl_setopt($ch, CURLOPT_PORT, 8081);
    curl_setopt($ch, CURLOPT_CONNECTTIMEOUT, 1);
    curl_setopt($ch, CURLOPT_TIMEOUT, 1);
    curl_setopt($ch, CURLOPT_POST, 1);
    curl_setopt($ch, CURLOPT_POSTFIELDS, $json);
    curl_setopt($ch, CURLOPT_HTTPHEADER, array(
        'Content-Type: application/json',
        'Content-Length: ' . strlen($json)
    ));
    curl_exec($ch);
    curl_close($ch);
}
$data->message = $_POST['message'];
notifyNode($data);
?>
```

[El código completo creado de la comunicación entre los servidores se dispone en el apartado: A.2.6. Ficheros creados para la comunicación.]

A.2.3.2. Modificación de ficheros del servidor Node Js anterior

Añadir las siguientes líneas al fichero main.js de la carpeta raíz del servidor Node Js (alternative1_node).

main.js

```
...
var bodyParser = require('body-parser');
app.use(bodyParser.json());

//allow external Post request to home page.
app.all('/', function(req, res, next) {
    res.header('Access-Control-Allow-Origin', '*');
    res.header('Access-Control-Allow-Methods', 'POST');
    res.header('Access-Control-Allow-Headers', 'X-Requested-With, X-HTTP-Method-Override, Content-Type');
    next();
});
```

```
//get php sent message and emit to socket
app.post('/', function(req, res) {
    msg = {message:req.body.message};
    console.log("PHP message: " + msg.message);
    //send to everione who is connected to the server
    io.emit('php_message', msg);
});
```

[El código completo creado de la comunicación entre los servidores se dispone en el apartado: A.2.6. Ficheros creados para la comunicación.]

A.2.3.3. Diagrama de flujo

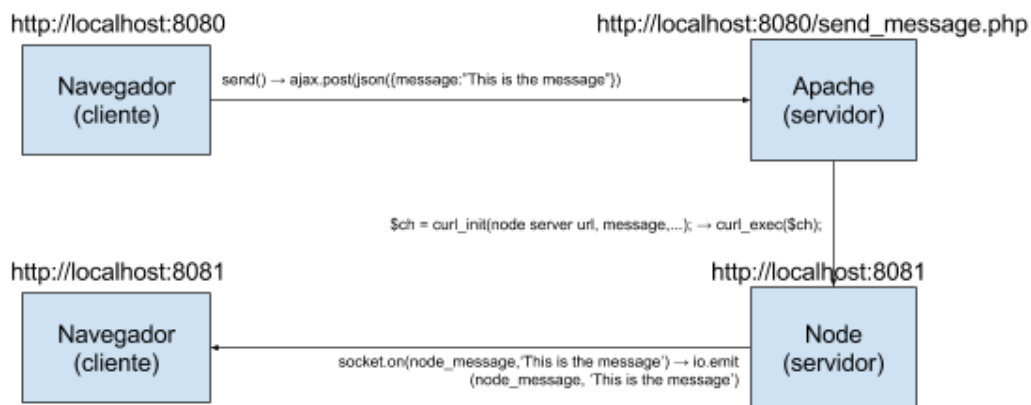


Figura 73. Diagrama de flujo de comunicación entre el cliente y el servidor de Apache con Node JS y su cliente.

A.2.3.4. Ejemplo en funcionamiento

Ejecutar el servidor apache y el servidor Node Js y acceder a las urls: <http://localhost:8080/> y <http://localhost:8081/>

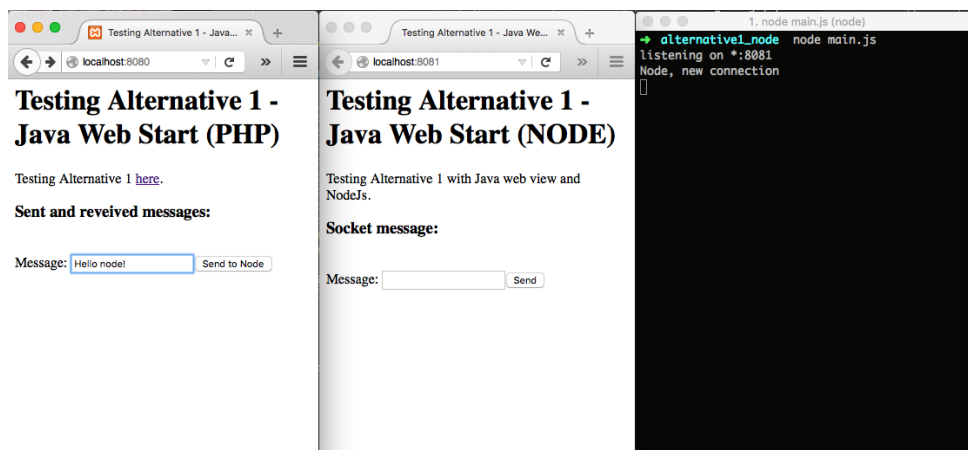


Figura 74. Servidor Apache, Servidor Node Js y consola con mensajes del servidor Node Js. Mensaje sin enviar.

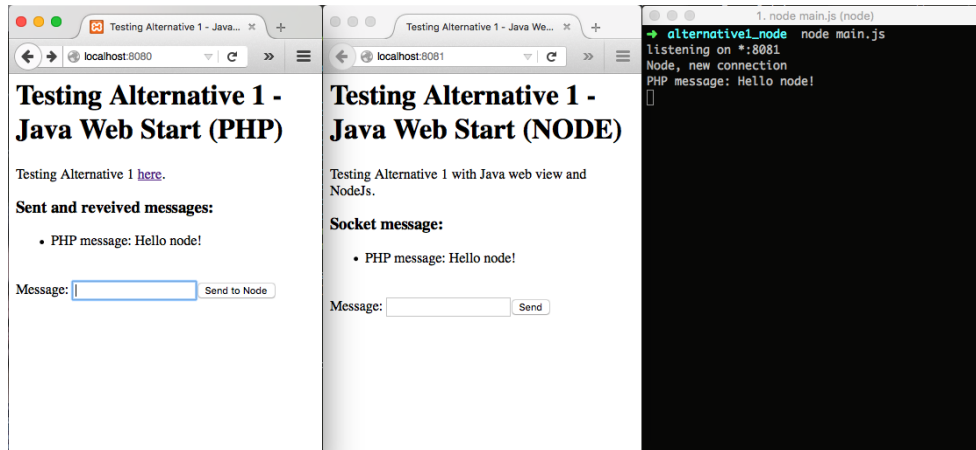


Figura 75. Servidor Apache, Servidor Node Js y consola con mensajes del servidor Node Js. Mensaje enviado.

A.2.4. Comunicación desde un cliente Node Js al servidor Node Js y éste a un servidor apache

A.2.4.1. Modificación de ficheros del servidor Node Js anterior

Para que el servidor Node Js realice una comunicación con un servidor Apache ha de realizarse una llamada *request* desde Node Js a Apache. La llamada se realiza a partir de de la llamada (POST por ejemplo) realizada desde un cliente a Node Js. Los cambios realizados para la comunicación son los siguientes:

main.js

```
...
var request = require("request");
...
//Every page create a socket connection.
io.on('connection', function (socket) {
    //welcome message
    message = 'Node, new connection!';
    console.log('Node, new connection');
    socket.emit('wellcome', {message: message});
    //disconnect message
    socket.on('disconnect', function(){
        console.log('Node, disconnection');
    });
    //sent message
    socket.on('node_message', function(data){
        //reply the message to io --> everyone connected
        io.emit('node_message', data);
        //reply the message to connected socket
        //socket.emit('message', data);
        console.log("Node message: " + data.message);
    });
});
```

```
//HTTP Request in order to send message to PHP
request({
  method:'GET',
  uri:
'http://localhost:8080/post_from_node.php?message='+data.message
}, function (error, response, body) {
  if (!error && response.statusCode == 200) {
    console.log("Node message from PHP: "+ body); // Show
the HTML for the PHP
    io.emit('php_message_from_node', {message:body});
  }
});
});
...

```

Esta llamada *request* escribe en consola la información que se recibe como respuesta de la llamada al servidor PHP.

index.html

```
...
send(){...}

//forward message from php to node client
socket.on('php_message_from_node', function(data){
  console.log(data);
  $('#messages').append($('- ').text('PHP message: ' + data.message));
});
...

```

A.2.4.2. Ficheros creados en el servidor apache

Se ha creado un fichero simple *post_from_node.php* para realizar un echo del mensaje recibido por POST que reenvía el servidor Node Js a través de su cliente.

post_from_node.php

```
<?php echo $_REQUEST['message'];
```

A.2.4.3. Diagrama de flujo

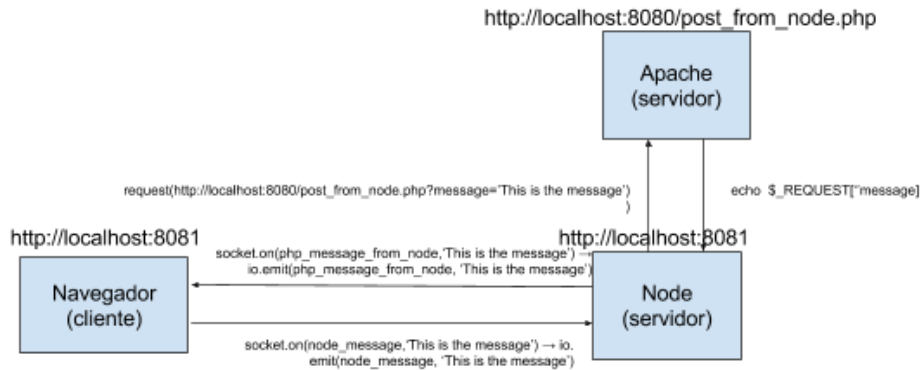


Figura 76. Diagrama de flujo, Cliente Node Js escribe en socket y el servidor Node Js petición a servidor Apache.

A.2.4.4. Ejemplo en funcionamiento



Figura 77. Servidor Node Js en funcionamiento.

A.2.5. Aplicación Java

Se requiere adaptar el puerto de la aplicación java para que abra en el navegador (*web view*) en la página brindada por el servidor Node Js.

Establecer la página inicial del navegador (*web view*) de java a la página inicial del servidor Node Js ha de modificarse la función carga del motor web:

```
webEngine.load("http://localhost:8081/");
```

A.2.6. Ficheros creados para la comunicación

- Servidor Node Js:
 - <https://drive.google.com/folderview?id=0B3nCzPJqtgWvb19GSklFTGN3bUE>
- Servidor apache:
 - <https://drive.google.com/folderview?id=0B3nCzPJqtgWvV2lQNDJCQWxt2M>

A.3. Bibliografía consultada

Configuración del servidor Apache Tomcat con Netbeans:

- <https://technology.amis.nl/2012/01/02/installing-tomcat-7-and-configuring-as-server-in-netbeans/>

Desarrollando aplicaciones Java Web Start:

- <http://examples.javacodegeeks.com/java-basics/web-start/java-web-start-getting-started/>
- <http://www.mkyong.com/java/java-web-start-jnlp-tutorial-unofficial-guide/>

Java, extraer el fichero *.jar de Netbeans:

- <http://stackoverflow.com/questions/9681876/how-to-create-a-jar-file-in-netbeans>

Java Swing ejemplos:

- <http://math.hws.edu/javanotes/c6/s1.html>

Java Web View:

- <http://docs.oracle.com/javafx/2/webview/jfxpub-webview.htm#CEGDIBBI>

Configurar servidor Apache Tomcat en Netbeans:

- <http://tomee.apache.org/tomee-and-netbeans.html>

Apache modificar el fichero de configuración *httpd* para adaptarlo a Java Web Start (no es necesario):

- <https://www.foundationdms.com/support/developer-software/webserver/apache-configuration-for-java-webstart>
- <https://docs.oracle.com/javase/8/docs/technotes/guides/javaws/developersguide/setup.html>

Parar Apache instalado desde terminal:

- <http://www.macuarium.com/foro/index.php?showtopic=149104>

Enviar argumentos mediante la url a un fichero *.jar*:

- <http://stackoverflow.com/questions/14116640/how-to-use-jnlp-to-pass-command-line-arguments-to-the-application>
- <http://stackoverflow.com/questions/13721587/pass-dynamic-params-via-jnlp>
- <http://portal.kryphonas.de/2010/10/11/passing-dynamically-parameters-to-a-java-web-start-app-jnlp/>
- <http://stackoverflow.com/questions/11038773/generate-jnlp-dynamically>

JSP:

- Ejemplo:
 - <https://pubs.vmware.com/vfabric52/index.jsp?topic=/com.vmware.vfabric.tc-server.2.8/getting-started/tutwebapp-jsp-source.html>
- Definición:
 - https://es.wikipedia.org/wiki/JavaServer_Pages

JNLP con jsf:

- <http://justobjects.org/cowcatcher/browse/advjava/slides/java-dev-env/core/deploy/slide.0.20.html>

Diferencia entre Apache HTTP y Apache Tomcat:

- <http://stackoverflow.com/questions/30632/difference-between-the-apache-http-server-and-apache-tomcat>

Ejemplo chat socket.io:

- <http://socket.io/get-started/chat/>
- <http://socket.io/docs/>

Comunicación entre PHP y Node Js:

- *From PHP to Node Js:*
 - <http://stackoverflow.com/questions/10048978/sending-messages-from-php-to-node-js>
 - <http://php.net/manual/es/book.curl.php>
 - <http://stackoverflow.com/questions/11181546/how-to-enable-cross-origin-resource-sharing-cors-in-the-express-js-framework-o>
 - <http://stackoverflow.com/questions/7067966/how-to-allow-cors-in-express-node-js>
 - <https://scotch.io/tutorials/use-expressjs-to-get-url-and-post-parameters>
 - <http://stackoverflow.com/questions/27776129/php-curl-curl-opts-connecttimeout-vs-curl-opts-timeout>
- *From Node Js to PHP:*
 - <https://www.npmjs.com/package/request#examples>
 - <http://stackoverflow.com/questions/21441786/post-to-php-from-node-js>
 - <http://stackoverflow.com/questions/24169391/node-js-to-php-cannot-get-it-working>
 - <http://stackoverflow.com/questions/4295782/how-do-you-extract-post-data-in-node-js>

Anexo B

B. Ejemplos de los APIs del SDK RTOS de Espressif

En este anexo se recogen distintos ejemplos de uso de cada una de las librerías del SDK RTOS v1.4.0 de Espressif necesarias para satisfacer el alcance del proyecto.

A partir de la configuración realizada en el apartado 6.4. se han realizado los siguientes ejemplos. Estos ejemplos han sido creados a partir de la documentación oficial de los diferentes API que ofrece Espressif^[16].

A la hora de compilar los diferentes ejemplos tener en cuenta que en el fichero *Makefile* han de estar las librerías de cada unas de las APIs *linkadas* como aparecen en la siguiente imagen.

```
LIBS = gcc hal phy pp net80211 wpa crypto main freertos lwip minic espconn cirom mirom json ssl pwm spiiffs phy
```

B.1. Manejo de tareas 1

Éste es un primer ejemplo de manejo de tareas utilizando la API FreeRTOS, donde se están ejecutando dos tareas al mismo tiempo mediante la función *xTaskCreate(...)*.

user_main.c

```
//esp libraries
#include "esp_common.h"
//configuration libraries
#include "user_config.h"
//RTOS libraries
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"

void task1(void *pvParameters) {
    vTaskDelay(1000); //10 seconds
    os_printf("\n\n*****\n\n");
    while(1){
        vTaskDelay(100); //1 second
```

```
        os_printf("Hello, welcome.\n");
    }
}

void task2(void *pvParameters) {
    int i = 0;
    vTaskDelay(1050); //10.5 seconds
    while (1){
        i++;
        vTaskDelay(100); //1 second
        os_printf("Goodbye %d.\n", i); //integer print
    }
}

void user_init(void)
{
    uart_div_modify(0, UART_CLK_FREQ / 115200);
    os_printf("\n\r\nSDK version:%s\n", system_get_sdk_version());

    xTaskCreate(task1, "tsk1", 256, NULL, 2, NULL);
    xTaskCreate(task2, "tsk2", 256, NULL, 2, NULL);
}
```

Proyecto creado con todos los ficheros utilizados, importar desde:

- *Eclipse* → *Import* → *Existing Projects into Workspace* → *Select archive file* → *Browse* → *my_project_rtos.zip*
- Proyecto: <https://drive.google.com/file/d/0B3nCzPJqtgWvd2tUZDR0TnpKY1U>

B.2. Configuración WI-FI

En este ejemplo se detallan los dos modos de conexión Wi-Fi que dispone el modulo ESP8266 que son el modo estación (*station*) y el modo punto de acceso (*access point*). Uno para tener acceso a internet y el otro para crear una red propia.

user_main.c

```
//ESP LIBRARIES
#include "esp_common.h"
//Configuration libraries
#include "user_config.h"
//RTOS LIBRARIES
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
```



```
void set_up_wifi(void){
    //set wifi opmode before setting config
    wifi_set_opmode(STATIONAP_MODE); //STATIONAP_MODE

    //AP CONFIGURATION
    struct softap_config *ap_config = (struct softap_config *) zalloc(sizeof(struct
softap_config));
    wifi_softap_dhcps_stop();
    sprintf(ap_config->ssid, "Test");
    sprintf(ap_config->password, "password");
    ap_config->authmode = AUTH_WPA_WPA2_PSK;
    ap_config->channel = 7;
    ap_config->max_connection = 5;
    wifi_softap_set_config(ap_config);
    free(ap_config);

    //AP STATIC IP CONFIGURATION
    struct ip_info *ap_ipinfo = (struct ip_info *) zalloc(sizeof(struct ip_info));
    IP4_ADDR(&ap_ipinfo->ip, 192, 168, 5, 1);
    IP4_ADDR(&ap_ipinfo->gw, 192, 168, 5, 1);
    IP4_ADDR(&ap_ipinfo->netmask, 255, 255, 255, 0);
    wifi_set_ip_info(SOFTAP_IF, ap_ipinfo);
    //free(ap_ipinfo);
    wifi_softap_dhcps_start();
    wifi_get_ip_info(SOFTAP_IF, ap_ipinfo);
    os_printf("\n\n*****\n%d.%d.%d.%d\n*****\n\n",
IP2STR(&ap_ipinfo->ip));

    //STATION CONFIGURATION
    struct station_config *st_config = (struct station_config *)
zalloc(sizeof(struct station_config));
    sprintf(st_config->ssid, "My_Station");
    sprintf(st_config->password, "*****");
    // make sure st_mode otherwise it fails
    //if(wifi_get_opmode()==STATION_MODE || wifi_get_opmode()==STATIONAP_MODE)
    wifi_station_set_config(st_config);
    free(st_config);

    //STATION STATIC IP CONFIGURATION
    struct ip_info *st_ipinfo = (struct ip_info *) zalloc(sizeof(struct ip_info));
    wifi_station_disconnect();
    wifi_station_dhcpc_stop();
    IP4_ADDR(&st_ipinfo->ip, 192, 168, 1, 230);
    IP4_ADDR(&st_ipinfo->gw, 192, 168, 1, 1);
    IP4_ADDR(&st_ipinfo->netmask, 255, 255, 255, 0);
```

```
wifi_set_ip_info(STATION_IF, st_ipinfo);
free(st_ipinfo);

wifi_station_connect();
//With static ip does not use --> wifi_station_dhcpc_start();
wifi_station_set_auto_connect(1);

}

void user_init(void) {
    uart_div_modify(0, UART_CLK_FREQ / 115200);

    set_up_wifi();
}
```

Proyecto creado con todos los ficheros utilizados, importar desde:

- *Import* → *Existing Projects into Workspace* → *Select archive file* → *Browse* → *my_project_rtos+wifi.zip*
- Proyecto: <https://drive.google.com/file/d/0B3nCzPJqtgWvQWhXcTRxUXRtZ0E>

B.3. Manejo de tareas 2

Éste es un ejemplo de manejo de tareas utilizando la API de FreeRTOS más exhaustivo con creación, pausa y eliminación de las mismas.

user_main.c

```
//ESP LIBRARIES
#include "esp_common.h"
//RTOS LIBRARIES
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"

xTaskHandle xHandle_task1;
xTaskHandle xHandle_task2;
xTaskHandle xHandle_task3;

bool deleted = true;
int i = 0;

LOCAL void ICACHE_FLASH_ATTR task1(void *pvParameters) {
    deleted = false;
    vTaskDelay(10000 / portTICK_RATE_MS); //10 seconds
    printf("\n\n*****\n\n");
```

```
while (1) {
    vTaskDelay(1000 / portTICK_RATE_MS); //1 second
    printf("Task 1: Hello, welcome.\n");
    if (i++ == 3) {
        vTaskResume(xHandle_task2);
        printf("Task 1: Resume task2.\n");
    }
}
}
LOCAL void ICACHE_FLASH_ATTR task2(void *pvParameters) {
    vTaskDelay(10000 / portTICK_RATE_MS); //10 seconds
    printf("Task 2: Suspended task 2.\n");
    vTaskSuspend(xHandle_task2);
    while (1) {
        vTaskDelay(1000 / portTICK_RATE_MS); //1 second
        printf("Task 2: Goodbye %d.\n", i); //integer print
        //os_printf("GPIO: %s", gpio_input_get());
    }
}
LOCAL void ICACHE_FLASH_ATTR task3(void *pvParameters) {
    vTaskDelay(10000 / portTICK_RATE_MS); //10seconds
    while (1) {
        vTaskDelay(2000 / portTICK_RATE_MS); //2 second
        printf("Task 3: 2 seconds.\n"); //integer print
        if (!deleted) {
            printf("Task 3: Deteted task 1.\n");
            vTaskDelete(xHandle_task1);
            deleted = true;
        }
    }
}
LOCAL void ICACHE_FLASH_ATTR task_example(void) {
    xTaskCreate(task1, "tsk1", 256, NULL, 2, &xHandle_task1);
    xTaskCreate(task2, "tsk2", 256, NULL, 2, &xHandle_task2);
    xTaskCreate(task3, "tsk3", 256, NULL, 2, NULL);
}
void user_init(void) {
    /* OUTPUT GPIO */
    uart_init_new();
    UART_SetBaudrate(UART0, BIT_RATE_115200);

    task_example();
}
```

B.4. Uso de *callbacks* dentro de una tarea

Los *callbacks* son de gran ayuda para que una vez dado un evento como la finalización de una tarea se realice otra función. Un ejemplo del uso de las mismas se detalla a continuación.

user_main.c

```
//ESP LIBRARIES
#include "esp_common.h"
//RTOS LIBRARIES
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"

/** A callback prototype */
typedef void (* my_callback)(char *texto);

/* Struct for data and callback function*/
struct my_struct {
    my_callback cb;
    char *texto;
    char *action;
};

/* Task which is going to call callback function*/
void my_task(void *arg) {
    struct my_struct *my_struct_task = (struct my_struct *) arg;
    int k;
    int m=0;
    for (k=0; k<10; k++){
        vTaskDelay(200 / portTICK_RATE_MS);
        printf("Task: my_task - %d\n",k);
    }
    my_struct_task->cb(my_struct_task->texto);
    vTaskDelete(NULL);
}

/*Callback function*/
LOCAL void ICACHE_FLASH_ATTR
my_callback_function(char *texto) {
    printf("\n-----\nHas entrado en el callback: %s\n-----
-----\n",texto);
}

LOCAL void ICACHE_FLASH_ATTR
get_data() {
```

```

        my_callback cb = my_callback_function; /* defining the callback function*/
        struct my_struct *my_struct = (struct my_struct *) zalloc(sizeof(struct
my_struct));
        my_struct->cb=my_callback_function; /* adding the calback function to the
struct*/
        my_struct->texto="Este es mi texto";
        my_struct->action=NULL;
        xTaskCreate(my_task, "mytsk", 256, my_struct, 2, NULL);
    }

void user_init(void) {
    /* OUTPUT GPIO */
    uart_init_new();
    UART_SetBaudrate(UART0, BIT_RATE_115200);

    callback_example();
}

```

B.5. Uso de la librería cJSON

Ejemplo de manejo de la librería cJSON. Requiere que en el fichero *makefile* estén linkadas las librerías “*cirom, mirom y json*” en la variable LIBS.

```
LIBS = ... cirom mirom json ...
```

user_main.c

```

//ESP LIBRARIES
#include "esp_common.h"
//cJSON LIBRARIES
#include "cJSON.h"
#include "math.h"

LOCAL void ICACHE_FLASH_ATTR
json_example() {
    char *json_string =
"{\n\"status\":\n\"0\", \n\"token\":\n\"fui1dh47yo347y592o9nrsoy72dbyln2sbdot2nms2sdn7t\n\
}";
    printf("\n\nJSON_string: %s\n\n", json_string);
    cJSON *json;
    json= cJSON_Parse(json_string);
    printf("\n\nJSON_cJSON_text: %s\n\n", cJSON_Print(json));
    printf("--> JSON size: %d\n", cJSON_GetArraySize(json));
    //PRINT ALL JSON ARRAY
    for (i = 0 ; i < cJSON_GetArraySize(json) ; i++){
        printf("--> %s: %s\n", cJSON_GetArrayItem(json,i)->string,
cJSON_GetArrayItem(json,i)->valuelstring);
    }
}

```

```
}

void user_init(void) {
    /* OUTPUT GPIO */
    uart_init_new();
    UART_SetBaudrate(UART0, BIT_RATE_115200);

    json_example();
}
```

B.6. Creación de un cliente y servidor *HTTP Request*

Primero se han estudiado los siguientes APIs para dar con el más apropiado para la arquitectura especificada en el apartado 7.2.3. Arquitectura

B.6.1. Espconn API

El API espconn^[16] es un api utilizado para crear conexiones TCP y UDP entre servidores y clientes.

Entre sus características, destacar las necesarias para el proyecto. Como que consta de funciones para dejar escuchando el módulo como un servidor TCP^[19] o para conectarse como un cliente TCP^[20].

Pasos de creación de un cliente TCP:

- Primero se inicializa la estructura de acuerdo al tipo de protocolo (*ESPCONN_TCP*)
- Se registra las funciones *connect callback* y *recv callback* de conexión y para cuando se reciben datos.
- Para empezar la comunicación se ha de conectar al *host* indicado en la estructura ESPCONN.
- Una vez conectado se puede empezar a enviar datos.

El API del SDK de red está basado en eventos. Estos eventos se activan cuando son registrados en el API, como son las funciones “*connected, disconnected, data receiver, etc*”.

B.6.2. Esp-now API

El API esp-now^[16] se encarga de realizar *callbacks* a otros ESP8266 de la infraestructura de red a la que está conectado. Únicamente se pueden conectar nodos formados por módulos ESP8266^[17].

B.6.3. Mesh API^[16]

Mesh^[18]^[18] es un tipo de topología de red en la cual cada nodo está conectado al resto de nodos. Todos los nodos *mesh* cooperan en la distribución de los datos en una red, llevando los mensajes de un nodo a otro por distintos caminos. Si un AP (nodo) está fuera del alcance del rango de cobertura de la estación (nodo principal), éste se comunicará con las tarjetas de red (otros nodos) que estén dentro de esa cobertura para que reenvíe los datos (salto de los datos de un nodo a otro).

B.6.4. Ejemplo

Ejemplo realizado a partir de la configuración Wi-Fi realizada en el ejemplo del apartado: B.2. Configuración WI-FI.

user_main.c

```
//ESP LIBRARIES
#include "esp_common.h"
#include "espconn.h"

LOCAL void ICACHE_FLASH_ATTR
wifi_init(void) {...}

LOCAL void ICACHE_FLASH_ATTR
shell_tcp_disconcb() {
    printf("Client disconnected!\n");
}

LOCAL void ICACHE_FLASH_ATTR
shell_tcp_recvcb(void *arg, char *pusrdata, unsigned short length) {
    printf("%s", pusrdata);
    //HANDLE RECEIVE DATA (pusrdata)
}

LOCAL void ICACHE_FLASH_ATTR
shell_tcp_connectcb(void *arg) {
    struct espconn *pespconn = (struct espconn *) arg;
    printf("Client connected!\n");
    char data[500];
    int i;
    //IMPORTANT: To send \n\n at the end of the string.
    //IMPORTANT: Remove any spaces between HTTP/1.1 and HOST
    sprintf(data, "GET /%s?user=user1&pass=1234 HTTP/1.1\nHOST: %s\n\n0", "url",
server_ip);
    for(i=0; data[i]!='\0'; ++i);
    printf("\n%s*****\nData_size=%i\n", data, i);
}
```

```
    printf("[espconn_send]=%d\n", espconn_send(espconn, data, i));
}

void ICACHE_FLASH_ATTR
shell_init(void) {
    printf("Initializing TCP server!\n");
    espconn_init();
    masterconn.type = ESPCONN_TCP;
    masterconn.state = ESPCONN_NONE;
    masterconn.proto.tcp = (esp_tcp *) zalloc(sizeof(esp_tcp));
    /*Server PORT. Only for TCP Server */
    //masterconn.proto.tcp->local_port = espconn_port();
    /*Remote Server PORT. Only for TCP Client */
    masterconn.proto.tcp->remote_port = server_port;
    /*Server IP. Only for TCP client */
    masterconn.proto.tcp->remote_ip[0] = 192;
    masterconn.proto.tcp->remote_ip[1] = 168;
    masterconn.proto.tcp->remote_ip[2] = 1;
    masterconn.proto.tcp->remote_ip[3] = 100;

    //REGIST CALLBACK
    masterconn.recv_callback=shell_tcp_recvcb;
    masterconn.proto.tcp->connect_callback=shell_tcp_connectcb;
    masterconn.proto.tcp->disconnect_callback=shell_tcp_disconcb;
    /*Conect to remote server. Only for TCP Client */
    espconn_connect(&masterconn);
    /*Accept connection and timeout. Only for TCP server */
    //espconn_accept(&masterconn);
    //espconn_regist_time(&masterconn, server_timeover, 0);
}

void user_init(void) {
    /* OUTPUT GPIO */
    uart_init_new();
    UART_SetBaudrate(UART0, BIT_RATE_115200);

    wifi_init();
    shell_init();
}
```


B.7. Ejemplo TCP - SSL

Ejemplo realizado a partir de la configuración Wi-Fi realizada en el ejemplo del apartado B.2. Configuración WI-FI

user_main.c

```
//ESP LIBRARIES
#include "esp_common.h"

//SOCKETS LIBRARIES AND SSL LIBRARIES
#include "lwip/sockets.h"
#include "lwip/dns.h"
#include "lwip/netdb.h"
#include "ssl/ssl_os_port.h"
#include "ssl/ssl_ssl.h"

LOCAL int client_fd;
LOCAL struct sockaddr_in client_addr;
LOCAL uint32 sin_addr;
LOCAL uint16 port = 0;
LOCAL SSL *ssl;
LOCAL SSL_CTX *ssl_ctx = NULL;
LOCAL uint8_t *read_buf = NULL;
LOCAL int recbytes;
LOCAL uint8_t write_buf[500]="";

static void ICACHE_FLASH_ATTR esp_client(void *pTaskserver_connr)
{
uint32 options = SSL_SERVER_VERIFY_LATER|SSL_DISPLAY_CERTS|SSL_NO_DEFAULT_KEY;
while (ssl_ctx == NULL){
    ssl_ctx = ssl_ctx_new(options, SSL_DEFAULT_CLNT_SESS);
}
while(1){
    ssl = NULL;
    read_buf = NULL;

    port = 443;
    sin_addr = "192.168.1.100";

    client_fd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (client_fd < 0){
        printf("Create the socket err\n");
    }else{
        memset(&client_addr, 0, sizeof(client_addr));
```

```
    client_addr.sin_family = AF_INET;
    client_addr.sin_port = htons(port);
    client_addr.sin_addr.s_addr = sin_addr;
    if (connect(client_fd, (struct sockaddr *)&client_addr,
                printf("Connect with the host err\n"));
        )else if ((ssl = ssl_client_new(ssl_ctx, client_fd, NULL,
0))==NULL){
        )else if ((res = ssl_handshake_status(ssl)) != SSL_OK){
        )else{
            x509_free(ssl->x509_ctx);
            ssl->x509_ctx=NULL;
            // Request string
            bzero(write_buf, sizeof(&write_buf));
sprintf(write_buf, "GET / HTTP/1.1\nHOST: %s:%d \n\n\0", server_ip, server_port);

            if ((res=ssl_write(ssl, write_buf, strlen(write_buf)) >
0)){
                while((recbytes = ssl_read(ssl, &read_buf) >= 0)
                {
                    if (recbytes != 0){
                        // Return the pointer of the JSON
                        if(strchr(read_buf, '{'))
                            break;
                    }
                }
                //USE RECEIVED DATA
            }
        }
        }
        ssl_free(ssl);
        close(client_fd);
    }
    ssl_ctx_free(ssl_ctx);
    vTaskDelete(NULL);
}

// TCP communication task.
int ICACHE_FLASH_ATTR esp_tcp_main() {
    xTaskCreate(esp_client, "esp_client", 4 * 256, NULL, 2,
                NULL);
    return 0;
}

void ICACHE_FLASH_ATTR user_init(void) {
    uart_init_new();
    UART_SetBaudrate(UART0, BIT_RATE_115200);
    esp_tcp_main();
}
```

B.8. Almacenar datos en la memoria flash

En este ejemplo se realiza la escritura del contenido de una variable en la memoria flash y lectura de la flash de la misma.

user_main.c

```
void ICACHE_FLASH_ATTR user_init(void) {
    // OUTPUT GPIO
    uart_init_new();
    UART_SetBaudrate(UART0, BIT_RATE_115200);
    char buff[64] = {"2342"};
    char buff2[64] = {0};
    printf("Erase.\n");
    SpiFlashOpResult result_spi = spi_flash_erase_sector(0x7B);
    printf("Result erase: %d\n", result_spi);
    printf("Writing...\n");
    result_spi = spi_flash_write(0x7B000, (uint32*)buff, sizeof(buff));
    printf("Result write: %d\n", result_spi);
    result_spi = spi_flash_read(0x7B000, (uint32*)buff2, sizeof(buff2));
    printf("Result read: %d\n", result_spi);
    printf("Prueba: %s\n", buff2);
}
```

Anexo C

C. Comandos AT

Este anexo recoge las pruebas de comandos AT realizada con el módulo ESP8266 a través de la *HyperTerminal*, RealTerm.

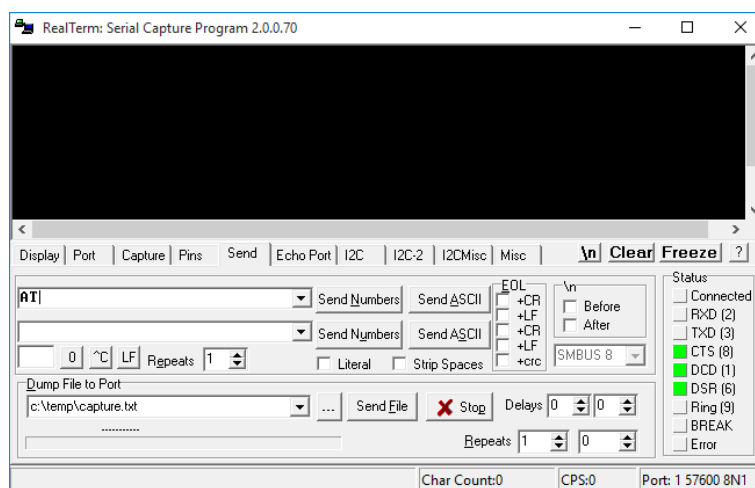


Figura 78. Envío de comandos AT con RealTerm.

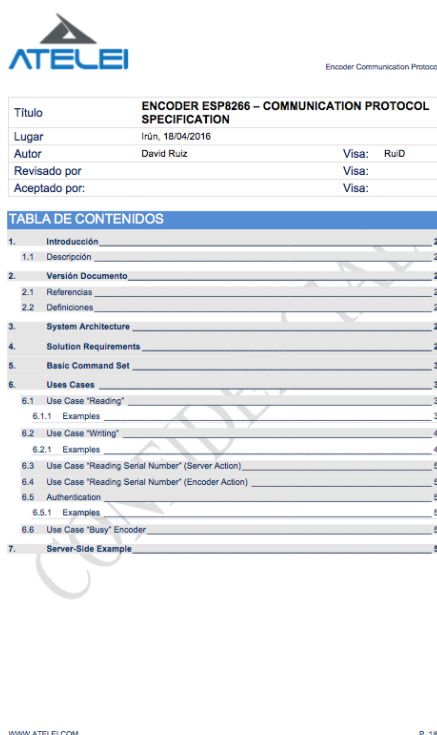
```
#Connect to wifi
AT+CWJAP_DEF="Billabong v5.0","PcAm*1962*"
#Get mode information
AT+CWMODE_DEF?
#Set the default mode : 1 Station, 2 AP, 3 AP + Station
AT+CWMODE_DEF=3
#Get IP
AT+CIPSTA_CUR?
#Set IP
AT+CIPSTA_CUR="192.168.0.202"
#Get IP DEF
AT+CIPSTA_DEF?
#Set IP DEF
AT+CIPSTA_DEF="192.168.0.202","192.168.0.1","255.255.255.0"
#Reset
AT+RST
#Access point information
AT+CWSAP_CUR?
#Configure access point network
```

```
AT+CWSAP_DEF="Test","testtest",5,3
#Reset
AT+RST
#Set softAP and set enable DHCP
AT+CWDHCP_DEF=0,1
#Multi Connection 1 enable 0 disable.
AT+CIPMUX=0
#Create server on port 8085
AT+CIPSERVER=1,8085
#Local AP router IP address and Router public address with their MACs
AT+CIFSR
#Local AP router IP address
AT+CIPAP_CUR?
AT+CIPAP_DEF?
#Set local AP router IP address
AT+CIPAP_DEF="192.168.4.1","192.168.4.1","255.255.255.0"
#Network status
AT+CIPSTATUS
#Enable multi connections
AT+CIPMUX=1
#Create TCP server
AT+CIPSERVER=1,8008
AT+CIPSTART="TCP","192.168.4.200",8085
AT+CIPSTART=0,"TCP","192.168.0.202",8008
#Reset
AT+RST
```

Anexo D

D. Protocolo de comunicación

Este anexo hace referencia al documento técnico, propietario de la empresa, *Protocolo de comunicación* y sus elementos más característicos.



ENCODER ESP8266 - COMMUNICATION PROTOCOL SPECIFICATION	
Título	ENCODER ESP8266 - COMMUNICATION PROTOCOL SPECIFICATION
Lugar	Irún, 18/04/2016
Autor	David Ruiz
Revisado por	Visa: RuiD
Aceptado por:	Visa:

TABLA DE CONTENIDOS		
1.	Introducción	2
1.1	Descripción	2
2.	Versión Documento	2
2.1	Referencias	2
2.2	Definiciones	2
3.	System Architecture	2
4.	Solution Requirements	2
5.	Basic Command Set	3
6.	Uses Cases	3
6.1	Use Case "Reading"	3
6.1.1	Examples	3
6.2	Use Case "Writing"	4
6.2.1	Examples	4
6.3	Use Case "Reading Serial Number" (Server Action)	5
6.4	Use Case "Reading Serial Number" (Encoder Action)	5
6.5	Authentication	5
6.5.1	Examples	5
6.6	Use Case "Busy" Encoder	5
7.	Server-Side Example	5

WWW.ATELEI.COM P. 16

Figura 79. Instantánea del índice del protocolo de comunicación.

En primer lugar se describe el propósito para el que está creado el documento y los elementos necesarios para llevar a cabo la comunicación.

A continuación, se especifica la arquitectura del sistema junto con el funcionamiento principal del cliente el ESP8266 y el servidor.

Acto seguido, se identifica la estructura de datos con la que se va a realizar la comunicación, estructura con datos en formato JSON. También se listan los diferentes estados que se pueden dar en cada transacción.

Por último, se listan los casos de uso que recogen la actividad completa entre los elementos de la arquitectura.

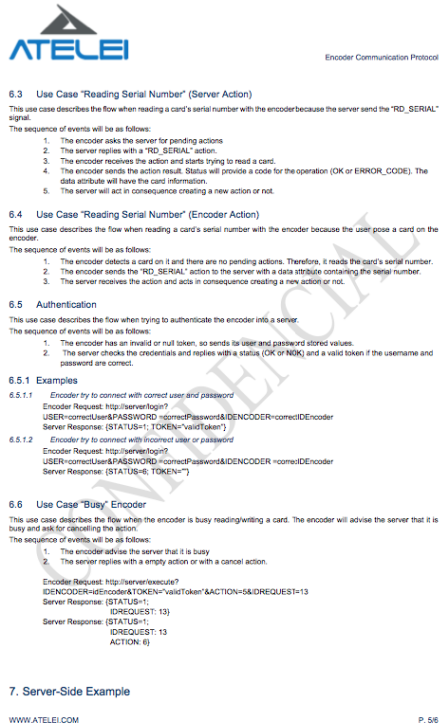


Figura 80. Instantánea de los casos de uso del protocolo de comunicación.

Anexo E

E. Informe de la empresa

En este anexo se recoge el informe del instructor sobre el Proyecto de Fin de Grado.



Grado en Ingeniería Informática

Plantilla para Informe del instructor de la empresa sobre Proyecto Fin de Grado

Empresa	ATELEI Engineering SLU
Instructor	David Ruiz Osés
Alumno	Andoni Sánchez Antolín
Título Proyecto	ESTUDIO Y COMPARATIVA DE LAS DIFERENTES ALTERNATIVAS PARA EL USO DE SMART CARDS DESDE APLICACIONES WEB
Fechas estancia	01.11.2015 – 21.06.2016

		Insuficiente	Regular	Bueno	Muy bueno	Excelente
Actitud	Autonomía					X
	Adaptabilidad					X
	Creatividad					X
	Capacidad de escuchar					X
	Capacidad de trabajar en equipo					X
	Motivación					X
	Eficiencia					X
Competencias generales	Metodología					X
	Conocimiento teórico					X
	Capacidad de aprendizaje					X
	Organización personal					X
Trabajo realizado	Utilización de herramientas					X
	Corrección de la solución					X
	Justificación de las decisiones técnicas					X
	Cumplimiento de especificaciones					X
	Alcance de los objetivos iniciales					X
	Calidad de la documentación					X

Insuficiente = resultados muy por debajo de lo esperado
Regular = resultados por debajo de lo esperado
Bueno = resultados de acuerdo con lo esperado
Muy bueno = resultados por encima de lo esperado
Excelente = resultados muy por encima de lo esperado

Marcar con una X la casilla que corresponda

Figura 81. Informe empresa. Parte 1/2.

Comentarios	<p>El proyecto fin de carrera propuesto consistía en un estudio de alternativas para solucionar un problema al que la empresa debería hacer frente más bien pronto que tarde: la prohibición por parte de Google Chrome y Mozilla Firefox para ejecutar applets escritos en JAVA. Lo que imposibilita la ejecución de código desde el servidor que pueda acceder a periféricos del cliente (en nuestro caso Smart-readers USB).</p> <p>Como empresa de base tecnológica enfocada al desarrollo I+D, sabíamos dónde estaba el problema, pero no sabíamos de la viabilidad de las diferentes alternativas, por lo que el alumno que fuera a desarrollar el Proyecto, debería de realizar un estudio del estado del arte, un pequeño ejemplo y ser capaz de reconocer pros y contras de cada alternativa.</p> <p>En este punto, Andoni ha demostrado las siguientes capacidades:</p> <ul style="list-style-type: none"> - Gran capacidad analítica a la hora de enfrentarse a un problema cuya solución no se sabe a priori si es viable. - Organización a la hora de comenzar a desarrollar las alternativas. - Visión global del problema (viabilidad tanto del desarrollo como dificultad de ejecución por parte del cliente). - Capacidad tanto para trabajar de forma autónoma, como para trabajar en equipo multidisciplinar (compuesto por ingenieros electrónicos e ingenieros informáticos). <p>Cuando la empresa dedujo que de todas las alternativas propuestas, la de mayor viabilidad era la de crear nuestro propio lector de tarjetas (diseño del hardware, creación de las rutinas que ejecutarían la inicialización, lectura y escritura de tarjetas), Andoni tuvo que adaptarse a un cambio grande en el proyecto. En ese punto ha ayudado al desarrollo de las rutinas y ha tenido que trabajar mano a mano con los ingenieros electrónicos para diseñar el hardware que soportara las especificaciones necesarias para las rutinas.</p> <p>Con toda la documentación aportada por Andoni en estos meses, la empresa ha ganado un background muy importante sobre el momento actual y futuro del desarrollo de aplicaciones web que necesiten acceder a periféricos del lado del cliente. Y con todo el código desarrollado por él, tenemos los primeros prototipos de lectores de tarjetas que en breves estarán funcionales para nuestras instalaciones.</p> <p>A lo largo del proyecto, la empresa ha visto una posibilidad comercial con el lector de tarjetas, por lo que además de ser incluido en nuestros sistemas, creemos que podría ser un producto a comercializar para que terceras empresas puedan desarrollar su propio software interactuando con el lector de tarjetas mediante un API/REST definido con un protocolo en el que Andoni también ha ayudado a definir (y que se incluye en un apartado de su memoria).</p> <p>Como resumen, la empresa ATELEI ha intentado que Andoni haga un proyecto fin de carrera que añadiera experiencia en casos reales a su currículo, y que le ofreciera enfrentarse a problemas reales y comunes en una empresa I+D. Por su parte, Andoni ha respondido muy por encima de nuestras expectativas, dándole tiempo a meterse de lleno en un desarrollo que no teníamos identificado a priori. Es por ello, que valoramos todos los puntos de este informe como excelentes.</p>
-------------	--

Fecha	22.06.2016
Firma	

Figura 82. Informe empresa. Parte 2/2.