

Ingeniería Informática
PROYECTO FINAL DE CARRERA
**Programador de riego
automantenido con control
remoto multiusuario**



VNIVERSITAT
DE VALÈNCIA

AUTOR: Miquel Vento Tormo
TUTOR: Jose Antonio Boluda
Valencia - 2014

Ingeniería Informática
PROYECTO FINAL DE CARRERA
**Programador de riego
automatenido con control
remoto multiusuario**



VNIVERSITAT
DE VALÈNCIA

AUTOR: Miquel Vento Tormo
TUTOR: Jose Antonio Boluda
Valencia - 2014 ©UNIVERSIDAD DE VALENCIA, 2014, Todos
los derechos resevados

Agradecimientos

En primer lugar, quiero hacer un especial agradecimiento a mi director, Jose Antonio Boluda, por el interés, el tiempo y la dedicación prestados durante la realización de este proyecto.

A mi familia, por su apoyo incuestionable y las reconfortantes paellas de los domingos.

A mis amigos, por secundarme en cualquiera de mis propósitos.

A Aina, por su paciencia y amor. Porque sin ella no hubiera sido posible.

Índice general

1. Introducción	3
1.1. Hardware	3
1.2. Software	3
2. Motivación y Objetivos	5
2.1. Motivación	5
2.2. Objetivos	7
3. Estado del Arte	9
3.1. Hardware	9
3.1.1. Programadores	9
3.1.2. Generadores de energía renovable	13
3.1.3. Programación	15
3.1.4. Microcontrolador y entorno de desarrollo	17
3.2. Software	20
3.2.1. Software de control	20
3.2.2. Servidores web	22
3.2.3. Web Frameworks	24
3.2.4. Base de datos	28
3.3. Comunicaciones	32
3.3.1. Conexión	32
3.3.2. Protocolo de comunicaciones	34
4. Especificación	37
4.1. Análisis de requisitos	37
4.1.1. Requisitos del programador	37
4.1.2. Requisitos del servidor	40
4.1.3. Requisitos de las comunicaciones	42
4.2. Conclusiones	42
4.2.1. Programador	42
4.2.2. Servidor	44

4.2.3.	Comunicaciones	45
4.2.4.	Otros	45
4.3.	Especificación del sistema	46
4.3.1.	Especificación programador	47
4.3.2.	Especificación servidor	49
4.3.3.	Especificación comunicaciones	50
4.4.	Planificación temporal	50
4.4.1.	Descomposición en tareas	50
4.4.2.	Descomposición en hitos	52
4.4.3.	Alcance del proyecto	53
4.4.4.	Juicio de expertos	55
4.4.5.	Planificación de las tareas	57
4.5.	Estimación de costes	61
4.5.1.	Costes de personal	63
4.5.2.	Costes software	63
4.5.3.	Coste hardware	65
4.5.4.	Resumen de costes	68
4.6.	Estudio de viabilidad	68
4.6.1.	Viabilidad económica	68
4.6.2.	Viabilidad técnica	71
4.6.3.	Viabilidad legal	71
4.6.4.	Análisis de riesgos	75
4.6.5.	Conclusión	77
5.	Desarrollo del proyecto	79
5.1.	Análisis	79
5.1.1.	Usuarios del sistema	79
5.1.2.	Análisis de casos de uso	81
5.1.3.	Análisis del modelo de datos	109
5.1.4.	Análisis de las interfaces de usuario	114
5.1.5.	Análisis de la fuente de alimentación	118
5.2.	Diseño	125
5.2.1.	Arquitectura del sistema	125
5.2.2.	Diseño de los datos	128
5.2.3.	Diseño de los casos de uso: Diagramas de secuencia	134
5.2.4.	Diseño de las interfaces de usuario	142
5.2.5.	Diseño de la fuente de alimentación	148
5.3.	Implementación	151
5.3.1.	Entorno de desarrollo	151
5.3.2.	Detalles de programación	153
5.3.3.	Detalles de contrucción del hardware	172

6. Pruebas y resultados	175
6.1. Descripción de las fases y pruebas a realizar	175
6.1.1. Descripción de las pruebas independientes	175
6.1.2. Descripción de las pruebas en un entorno real controlado	183
6.2. Análisis de los resultados de las pruebas	185
6.2.1. Resultados en pruebas independientes del programador	185
6.2.2. Resultados en pruebas independientes del servidor . . .	196
6.2.3. Resultados en pruebas sobre entorno real controlado . .	235
6.3. Evaluación de proyecto al cierre	237
7. Conclusiones y trabajo futuro	239
7.1. Conclusiones	239
7.2. Trabajo futuro	241
A. Symfony: la arquitectura	251

Índice de cuadros

3.1. comparativa de los programadores	13
3.2. Comparativa de generadores de energía	15
3.3. Comparativa de los lenguajes de programación	17
3.4. Comparativa técnica de los microcontroladores	19
3.5. Comparativa plataformas	20
3.6. Características ofrecidas por el software	22
3.7. Comparativa servidores web	24
3.8. Comparativa web frameworks	28
3.9. Comparativa de bases de datos	31
3.10. Comparativa conexiones inalámbricas	34
3.11. Comparativa de los protocolos de comunicación.	35
4.1. Porcentaje de evaluación del juicio de expertos	55
4.2. Resultados del juicio de expertos	57
4.3. Sueldo Ingenieros Informáticos	63
4.4. Software utilizado en el desarrollo	64
4.5. Software utilizado en el servicio	65
4.6. Hardware utilizado durante el desarrollo del proyecto y servidor.	66
4.7. Coste del hardware utilizado para la construcción del prototipo del programador.	67
4.8. Coste del servidor en producción	67
4.9. Resumen del coste económico total estimado del proyecto	68
4.10. Resumen del coste económico necesario para afrontar la primera fase del proyecto	69
4.11. Resumen del coste económico necesario para la entrada en producción	69
4.12. Análisis de tipo licencias software	74
4.13. Análisis de licencias software específicas	75
4.14. Posibles riesgos técnicos durante el proyecto	77
5.1. Caso de uso Configuración	82
5.2. Caso de uso gestión de programas	82

5.3. Caso de uso visualizar programas	82
5.4. Caso de uso añadir programa	83
5.5. Caso de uso activar programa	84
5.6. Caso de uso desactivar programa	85
5.7. Caso de uso borrar programa	86
5.8. Caso de uso configuración hora	86
5.9. Caso de uso configurar electro-válvula	87
5.10. Caso de uso autoconfig	87
5.11. Caso de uso mostrar mensajes	88
5.12. Caso de uso gestión manual	88
5.13. Caso de uso accionar electro-válvulas.	89
5.14. Caso de uso comprobarSensores	90
5.15. Caso de uso sincronización servidor	90
5.16. Caso de uso identificarse	91
5.17. Caso de uso recodar contraseña	92
5.18. Caso de uso ayuda	92
5.19. Caso de uso registro	92
5.20. Caso de uso lista de programadores	94
5.21. Caso de uso configurar electroválvulas	94
5.22. Caso de uso mostrar mensajes	95
5.23. Caso de uso lista de programas	95
5.24. Caso de uso añadir programa	96
5.25. Caso de uso activar programa	97
5.26. Caso de uso desactivar programa	97
5.27. Caso de uso borrar programa	98
5.28. Caso de uso configurar programador	98
5.29. Caso de uso generar autoconfig.	99
5.30. Caso de uso histórico	99
5.31. Caso de uso mostrar inactivos	100
5.32. Caso de uso configurar cuenta	101
5.33. Caso de uso cerrar sesión	101
5.34. Caso de uso gestión de programadores	103
5.35. Caso de uso	104
5.36. Caso de uso gestión de backups	105
5.37. Caso de uso gestión de fallos	105
5.38. Caso de uso gestión de bugs.	106
5.39. Caso de uso sincronización	107
5.40. Caso de uso generarErrores	108
5.41. Caso de uso generar backup	108
5.42. Entidades presentes en la estructura de datos del servidor. . .	111

5.43. Diferentes tipos de memoria existentes en el microchip ATmega328.	112
5.44. Información no volátil en el programador.	113
5.45. Atributos de la clase _Programador.	131
5.46. Atributos de la clase Mensajes.	132
5.47. Atributos de la clase _Programas.	132
5.48. Atributos de la clase electro-válvulas.	133
5.49. Atributos de la clase tipo conexión.	133
5.50. Atributos de la clase tipo _APN.	133
5.51. Atributos de la clase tipo Sensores.	134
6.1. Usuarios que realizarán las pruebas de usabilidad en el programador	178
6.2. Usuarios que realizarán las pruebas	180
6.30. Valoración de la usabilidad del producto según los administradores	196
6.77. Valoración de la usabilidad del producto según los usuarios	220
6.78. Valoración de la usabilidad del producto según los administradores	220
6.79. Usuarios concurrentes en el servicio	230
6.80. Resumen del coste económico total real del proyecto	237

Índice de figuras

2.1. Instalación de riego actual.	6
2.2. Programador actual: Irri 2 fabricado por Progres.	7
3.1. Programador actual: Irri 5 fabricado por Progres.	10
3.2. Programador: Agronic 2500 fabricado por Progres.	11
3.3. Programador WVC y controlador inalámbrico WVPE de Hunter Industries	12
3.4. Programador WP de Rainbird	13
3.5. Estación autónoma que se abastece mediante la combinación de un generador eólico y otro solar situada junto al barranco del Carraixet en el término de Vinalsa.	15
3.6. Microgenerador hidroeléctrico fabricado por Seed Studio	16
3.7. Software de control de riego Agronic Net II ofrecido por Progres	21
3.8. Software de control de riego IMMS comercializado por Hunter Industries	22
3.9. Logo de Apache HTTP Project	23
3.10. Logo de Nginx	23
3.11. Logo de Microsoft	24
3.12. Logo de ASP.NET de Microsoft	25
3.13. Logo de Ruby on Rails	26
3.14. Logo de Angular JS de Google	26
3.15. Logo de Django	27
3.16. Logo de Symfony	28
3.17. Logo de Oracle	29
3.18. Logo de MariaDB	29
3.19. Logo de MySQL	30
3.20. Logo de SQL Server de Microsoft	31
3.21. Logo de PostgreSQL	31
3.22. Esquema de comunicaciones para tecnología Wi-Fi	32
3.23. Esquema de comunicaciones para tecnología radio	33
3.24. Esquema de comunicaciones para tecnología GSM	34

4.1. Capas de abstracción del hardware	47
4.2. EDT del sistema	54
4.3. Distribución beta para cálculo de tiempo esperado	58
4.4. Probabilidad de finalizar el proyecto concatenando las tareas .	59
4.5. Días necesarios para una probabilidad del 95 %	60
4.6. Fecha estimada para probabilidad del 95 %	61
4.7. Diagrama de Gantt para la planificación del proyecto sin buffer	62
5.1. Diagrama de caso de uso para un cliente físico.	81
5.2. Diagrama de caso de uso para el tiempo programador.	89
5.3. Diagrama de caso de uso para el usuario anónimo.	91
5.4. Diagrama de caso de uso para el cliente web.	93
5.5. Diagrama de caso de uso para el administrador de servicio. . .	102
5.6. Diagrama de caso de uso para el administrador de sistema. . .	104
5.7. Diagrama de caso de uso para la API.	106
5.8. Diagrama de caso de uso para el tiempo del sistema.	107
5.9. Datos en programador-servidor.	109
5.10. Diagrama de clases del servidor.	111
5.11. Diagrama de clases del programador.	114
5.12. Interfaz programador Rainbird WP.	115
5.13. Diagrama de flujo entre los diferentes menú web del servicio. .	116
5.14. Esquema de la web una vez realizado el proceso de identificación.	117
5.15. Interfaz web de login y carpetas en dropbox.	118
5.16. Esquema de energía del sistema.	119
5.17. Tensión de salida de la batería.	120
5.18. Tensión de salida del generador cuando está funcionado. . . .	121
5.19. Esquema del circuito extraído del generador.	121
5.20. Salida de los generadores en vacío	122
5.21. Salida del generador conectado al Arduino	123
5.22. Salida del generador conectado al Arduino	124
5.23. Salida del generador conectado al Arduino y un condensador de 4200uF	124
5.24. Arquitectura cliente-servidor en una visión global.	126
5.25. Arquitectura cliente-servidor en el servidor.	127
5.26. Arquitectura por capas en el programador.	128
5.27. Diagrama de clases del servidor junto a sus atributos.	129
5.28. Bundles del proyecto	130
5.29. Diagrama de clases del programador junto a sus atributos. . .	131
5.30. Diagrama de secuencia para la interacción física	135
5.31. Diagrama de secuencia para la comprobación de acciones . . .	136

5.32. Diagrama de secuencia interno de symfony para una petición de ejemplo	137
5.33. Diagrama de secuencia para el login del usuario y administrador	138
5.34. Diagrama de secuencia para obtener la lista de programadores de un usuario.	138
5.35. Diagrama de secuencia para obtener la lista de programas de un programador	139
5.36. Diagrama de secuencia para añadir un nuevo programa a un programador	140
5.37. Diagrama de secuencia para el envío de datos mediante el módulo de comunicaciones	141
5.38. Diagrama de secuencia para la carga de todos los programas .	142
5.39. Diagrama de secuencia para la actualización de un programa mediante la API REST	142
5.40. Esquema de la interfaz física.	143
5.41. Esquema de la interfaz del programador tipo lista.	144
5.42. Esquema de la interfaz del programador tipo introducción de datos.	145
5.43. Esquema de la interfaz de identificación.	145
5.44. Lista de programas vía web.	146
5.45. Introducción de nuevo programa vía web.	147
5.46. Contenedor global.	148
5.47. Diseño de los elementos de la fuente de alimentación	149
5.48. Tiempo de carga del condensador de 4200uF dado un amperaje de 30mA	151
5.49. Dependencias en archivo composer.json	153
5.50. Estableciendo permisos en el directorio de log	153
5.51. Información RAW recibida de una petición al servidor	154
5.52. Código para la recepción de datos	155
5.53. Código del método CheckActions definido en la clase Automata	156
5.54. Código referente a la clase StateVir	157
5.55. Esquema de herencia de los estados	158
5.56. Estructura del sistema	159
5.57. Código de el método Init() en la clase communcationInt	160
5.58. Código de el método GetPrograms en la clase communcationInt	161
5.59. Código de la clase padre communicationsVir	161
5.60. Código de la clase GprsLib	162
5.61. Código perteneciente al namespace Resources en resouces.h . .	163
5.62. Código perteneciente al namespace Configuration en resouces.h	164
5.63. Código perteneciente a la clase Program	165

5.64. Texto RAW recibido con verbose y echo activados	166
5.65. Texto RAW recibido con verbose y echo desactivados	167
5.66. Método mergeProgram en clase HardwareProgram	168
5.67. Método NewProgram en la clase ApiBundle:ProgramController	169
5.68. Código para la creación de un formulario con un nuevo programa	170
5.69. Código para generar un nuevo identificador único	171
5.70. Prototipo construido para desarrollo.	172
5.71. Resultado del diseño de la pcb encargada de activar las elec- troválvulas.	173
5.72. Fotolito en la insoladora.	173
5.73. Resultado de la pcb una vez ensamblados los componentes. . .	174
6.1. Mecánica de una electroválvula.	184
6.2. Visualización de la hora del sistema.	186
6.3. Lista de electroválvulas del programador.	187
6.4. Programa activo del programador.	187
6.5. Lista de mensajes del programador.	189
6.6. Mensaje general de confirmación.	190
6.7. Mensaje en pantalla durante la sincronización.	192
6.8. Resultado de log de Apiary después de algunas consultas. . . .	194
6.9. Lista de programadores una vez autenticado como usuario . .	197
6.10. Panel de administración una vez identificado como Aina - ad- ministradora.	198
6.11. Error durante el proceso de identificación	199
6.12. Panel de configuración del programador	201
6.13. Histórico de sensores del programador	203
6.14. Histórico de acciones de las electroválvulas del programador .	204
6.15. Lista de programas activos e inactivos	206
6.16. Lista del histórico de programas	207
6.17. Histórico de acciones de las electroválvulas del programador .	208
6.18. Datos requeridos para el cambio del tipo de usuario	210
6.19. Formulario para añadir una nueva SIM.	212
6.20. Programa utilizado para realizar los test del apirest.	215
6.21. Capturas para el formulario de login(1/3)	221
6.22. Capturas para el formulario de login(2/3)	222
6.23. Capturas para el formulario de login en dispositivos móviles (3/3)	223
6.24. Capturas para el formulario para añadir programa(1/3)	224
6.25. Capturas para el formulario para añadir programa(2/3)	225
6.26. Capturas para el formulario para añadir programa en disposi- tivos móviles (3/3)	226

6.27. Capturas para el formulario para añadir programa(1/3)	227
6.28. Capturas para el formulario para añadir programa(2/3)	228
6.29. Capturas para el formulario para añadir programa en dispositi- vos móviles (3/3)	229
6.30. Resultados temporales para una carga normal	231
6.31. Resultados temporales para una carga media baja	232
6.32. Resultados temporales para una carga media alta	233
6.33. Resultados temporales para una carga alta	234
6.34. Resultados temporales para una carga extrema	235
6.35. Instalación realizada para las pruebas de entorno real controlado	236
6.36. Diagrama de Gantt a partir de los datos reales	238
7.1. Esquema de conexión para un sistema mixto o concentrado. .	242
A.1. Estructura básica de symfony	251

Resumen

Valencia ha sido, desde hace siglos, una población agrícola basada principalmente en minifundios¹ [17, p. 398]. Esta tendencia, no obstante, ha cambiado a lo largo de los últimos años. Los grandes terratenientes, la importación de cítricos desde países del sur, así como el abandono de la administración pública han hecho que muchas de las personas con pequeñas extensiones de tierra tengan que abandonarlas por falta de rentabilidad.

Actualmente, existen multitud de recursos para controlar los sistemas de riego centralizado. Sin embargo, están siempre orientados a grandes extensiones, pues resultan caros y de compleja configuración, ya que es el cliente el que debe comprar, instalar y mantener toda la infraestructura de comunicaciones.

En este proyecto hemos percibido la necesidad de crear un sistema que permita, tanto al pequeño agricultor como a grandes terratenientes², la posibilidad de controlar el sistema de riego y abonado de sus campos también de forma remota, pero esta vez olvidándose de toda la infraestructura más allá del programador y un ordenador con conexión a Internet.

El objetivo principal que perseguimos es que dicho servicio pueda ser instalado por cualquier usuario sin conocimientos técnicos y con pocos recursos. Esto se conseguirá creando un servicio en la nube controlado por nosotros al que se conectarán tanto los programadores como los clientes. Además, siguiendo la filosofía de este tipo de servicios, el cliente solo deberá pagar el alquiler del producto, por lo que no se debe hacer ningún desembolso inicial ni de mantenimiento.

Así pues, el proyecto es el resultado de un análisis minucioso realizado en colaboración tanto con expertos agrícolas como con usuarios finales del sector.

¹Terreno de cultivo de entre una y siete hanegadas

²En un futuro

Resum

València ha sigut, des de fa segles, una població agrícola basada principalment en minifundis³ [17, p. 398]. Aquesta tendència, no obstant això, ha canviat en els darrers anys. Els grans terratinents, la importació de cítrics des de països del sud, així com l'abandonament de l'administració pública han fet que moltes de les persones amb petites extensions de terra hagen d'abandonar-les per falta de rendibilitat.

Actualment, existeixen multitud de recursos per controlar els sistemes de reg centralitzat. Tanmateix, estan sempre orientats a grans extensions, doncs resulten cars i la seua instal·lació és complexa, ja que és el client qui deu comprar, instal·lar i mantindre tota la infraestructura de comunicacions.

En aquest projecte hem percebut la necessitat de crear un sistema que permetia, tant al petit agricultor com a grans terratinents⁴, la possibilitat de controlar el sistema de reg i adobament dels seus camps també de manera remota, però, aquesta vegada, oblidant tota la infraestructura més enllà del programador i un ordinador amb connexió a Internet.

L'objectiu principal que perseguim és que aquest servei puga ser instal·lat per qualsevol usuari sense coneixements tècnics i amb pocs recursos. Açò s'aconseguirà tot creant un servei al núvol controlat per nosaltres, al qual es connectaran tant programadors com clients. A més a més, seguint la filosofia d'aquest tipus de serveis, el client sols haurà de pagar el lloguer del producte, per la qual cosa no s'ha de fer cap desemborsament inicial ni de manteniment.

Així doncs, el projecte és el resultat d'una anàlisi minuciosa realitzada en col·laboració tant amb experts agrícoles com amb usuaris finals del sector.

³Terreny de cultiu d'entre una y set fanecades

⁴En un futur

Abstract

Valencia has been, for centuries, an agricultural population based mainly on smallholdings. This trend, however, has changed over the years. Large landowners, the import of citrus from southern countries, as well as the abandonment of public administration have made that many of those with small acreages have to abandon due to their lack of profitability.

Currently, there are many resources to control centralized irrigation systems. However, they are always focused on large areas, as they are expensive and complex to set up, as it is the customer who must purchase, install and maintain the entire communications infrastructure.

In this project we have realized the need to create a system allowing both small farmers and large landowners, the ability to control the irrigation system and fertilization of their fields also remotely, but this time forgetting all the infrastructure more beyond the programmer and a computer with Internet connection.

The main goal we pursue is that service can be installed by any user without any technical knowledge and with few resources. This will be achieved by creating a cloud service controlled by us to which both developers and clients will connect. In addition, following the philosophy of this type of service, the customer must pay just the rent of the product, so it is not necessary to make any initial or maintenance outlay.

Thus, the project is the result of a detailed analysis in collaboration with both agricultural experts and end users in the sector.

Capítulo 1

Introducción

Dadas las características del proyecto, se ha dividido en dos grandes bloques. El primero constituye la parte de hardware, representada por el programador de riego; el segundo, en el que pondremos más énfasis, constituye la parte software, es decir, el servidor.

1.1. Hardware

Este es el elemento físico que reside en el campo. Debe ser capaz de interactuar con los elementos de control (electroválvulas, sensores, etc.), así como efectuar la conexiones al servidor. Debido a que generalmente los programadores están situados en zonas agrícolas y que carecen de luz eléctrica, debe tener la capacidad de autoabastecerse de energía. Dicho programador, además, debe ser tolerante a fallos, permitiendo su funcionamiento aun fallando algunos de sus módulos; dándose el caso, se tiene que notificar inmediatamente al servidor. El programador, finalmente, ha de ser capaz de configurarse manualmente, así como permitir la apertura y cierre de las electroválvulas sobre el terreno.

1.2. Software

A diferencia del resto de sistemas existentes, en este caso la conexión entre el programador y el cliente se realizará a través de nuestro servidor. Dado que el programador está limitado en recursos (tanto energéticos como de rendimiento), será él mismo el que solicitará las conexiones al servidor. A consecuencia de esta característica, recurriremos a un servicio web capaz de gestionar tanto las peticiones de usuario como las de programador. La interfaz de usuario será intuitiva y poco sobrecargada. Se pretende crear un

servicio que permita al cliente poder realizar las mismas operaciones que realizaba físicamente, pero ahora a distancia. También será importante la posibilidad de notificar vía email o SMS cualquier fallo que pueda ocurrir en el sistema.

Será obligación del servicio asegurar la estabilidad o, en caso de fallo, la recuperación, por lo que el servidor deberá estar preparado para gestionar copias de seguridad, fallos de conexión, etc. Los recursos se deberán administrar correctamente, priorizando la conexión de los programadores al servidor y las notificaciones de errores graves sobre la interacción del cliente con el programador.

Capítulo 2

Motivación y Objetivos

2.1. Motivación

La motivación parte de haber pertenecido a una familia agrícola en posesión de diferentes campos de cítricos y de conocer algunos de los problemas existentes una vez realizada la inversión de instalar riego por goteo. Dichos problemas son los que se pretende resolver con el servicio creado, ya no solo para el uso personal sino para que todo aquel que lo desee pueda contratarlo.

Estos son los problemas detectados:

- Actualmente el usuario debe desplazarse hasta el campo para realizar cambios en los programas o hacer comprobaciones. Dichos desplazamientos se producen casi diariamente en las épocas de verano, cuando, con el calor, puede ser desastroso un fallo en el sistema de riego. Estos traslados a los distintos terrenos suponen un coste tanto económico como temporal nada despreciable.
- Los distintos sistemas existentes en el mercado suponen un gran desembolso para el agricultor dado que las conexiones a los sistemas de riego debe realizarlas el mismo cliente. Esto obliga, además de a adquirir costosas licencias y hardware específico, a tener unos conocimientos técnicos avanzados para realizar la configuración del sistema que generalmente no se tienen, dada la escasa formación técnica del cliente.
- Dado que no nos encontramos con macroextensiones, los campos no suelen disponer de toma de corriente, por lo que las fuentes de energía habituales en los programadores son pilas, las cuales tienen una duración limitada. En algunos casos se recurre al uso de placas fotovoltaicas,

las cuales se deben situar en el exterior del refugio por lo que son robadas habitualmente.

- Imposibilidad de realizar un desembolso económico importante.

En la figura 2.1 se muestra la instalación actual de riego, situada en la partida *El Collado* en el término municipal de Manises, Valencia. Esta se encarga del riego de 10 anegadas¹ mediante una única línea de riego más otra de abonado. El agua es proporcionada por un motor de agua de riego, San Pascual Bailón de Ribarroja del Túrria, que proporciona agua las 24 horas del día todos los días de la semana, por lo que no es necesaria ninguna bomba de agua. A la entrada de la línea, la presión del agua es de aproximadamente 4 bares, aunque esta puede variar según el número de regantes activos. Al cierre de las electro-válvulas, la presión puede alcanzar picos de hasta 9 bares, por lo que la instalación deberá estar preparada para ello.



Figura 2.1: Instalación de riego actual.

¹0.83 hectáreas

En la figura 2.2 podemos visualizar el programador utilizado actualmente. La gestión de los programas se realiza mediante los botones presentes, con los que además podremos realizar acciones de apertura manual o paro de emergencia. En el LCD frontal se mostrará la información de los programas, el estado de las electro-válvulas, la hora y el aviso de cambio de batería. La alimentación se realiza mediante 2 pilas de 9 voltios y es el encargado de activar y desactivar las dos electro-válvulas que gestionan tanto la línea de riego como la de abonado. El tiempo de duración de las baterías es de aproximadamente 6 meses, aunque depende mayormente de la frecuencia a la que se activa el riego².



Figura 2.2: Programador actual: Irri 2 fabricado por Progres.

2.2. Objetivos

Para dar solución a los problemas anteriores se establecen los siguientes objetivos:

²Cuanto más cambios de estado se realicen, menor tiempo de duración.

- Crear un programador de riego capaz de automantenerse energéticamente y comunicarse con el servidor central.
- Posibilidad de controlar el programador de riego de una forma remota, reduciendo así los desplazamientos hasta la zona.
- Hacer transparente al cliente la gestión del servidor y las comunicaciones, eliminando así la inversión económica inicial y la necesidad de tener conocimientos técnicos.
- Crear un servicio en la nube al que se puedan conectar los usuarios de una forma sencilla, eliminando al usuario el coste de mantener un servidor.

Consecuencia de los anteriores, surgen los siguientes objetivos secundarios:

- El programador debe disponer de algún recurso de configuración rápida, eludiendo así la necesidad de introducir manualmente los datos de conexión y demás opciones configurables la primera vez que se arranca.
- Crear una API que sea capaz de comunicarse con los programadores de riego. Esto es importante ya que permite que el sistema resulte más versátil si se desea comunicarse con futuros dispositivos.
- Crear una interfaz de usuario sencilla. Esta debe ser lo más parecida posible a la de un programador clásico, debido a la poca adaptabilidad tecnológica característica del cliente.
- Se deben gestionar las copias de seguridad en el servidor y garantizar principalmente la comunicación con los programadores y el envío de notificaciones de problemas graves.

Posteriormente, en la sección Especificación estos objetivos se detallarán en especificaciones más concretas.

Capítulo 3

Estado del Arte

Dada la heterogeneidad del proyecto, se ha hecho necesaria la división de este apartado en tres subbloques. El primero, en el que estudiaremos el hardware situado en el campo; el segundo, en el que estudiaremos las tecnologías existentes para el desarrollo del servidor; y el tercero, que englobará las comunicaciones, es decir, las posibilidades de interacción entre los dos bloques anteriores.

3.1. Hardware

Por lo que respecta al hardware, se ha realizado un estudio de mercado de los distintos programadores existentes y generadores de energía renovable.

3.1.1. Programadores

En el mercado actual existe una gran variedad de programadores. Dadas las exigencias del proyecto, las características de las que deberán disponer serán:

- No deben depender de la red eléctrica para funcionar.
- Deben permitir el accionamiento de al menos dos electro-válvulas.
- Deben permitir su control remoto.

Los programadores existentes en el mercado que más se aproximan a dichas características son:

- **Progres - Irri 5** [67]

El programador Irri 5, es la nueva versión del programador Irri 2¹. Este permite la programación de hasta 5 sectores funcionando con una sola pila de 9 voltios. Su programación se realiza a través de tres pulsadores presentes en el frontal. No permite la conexión de ningún módulo GSM ni de ningún otro tipo de control remoto.

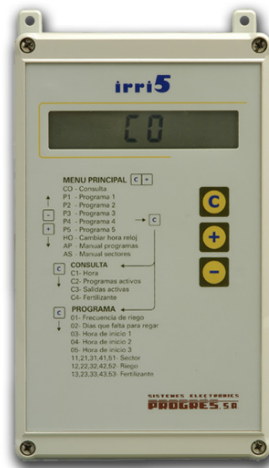


Figura 3.1: Programador actual: Irri 5 fabricado por Progres.

Fuente: <http://www.progres.es/htmls/productos/es/programadores-riego-y-fertirrigacion/ficha/irri-5.html>

- **Progres - Agrónic 2500 + Avisador GSM ó AgroBee** [65, 66, 68]

Este programador 3.2 está preparado para la gestión de hasta 30 sectores de riego, además de la conexión de sensores que permiten medir distintos parámetros. También cuenta con conexiones para programación de motores, fertilizantes, etc., los cuales no hemos examinado con detalle dado que no son relevantes para nuestro propósito.

Se permite la gestión remota mediante radio con el uso de AgroBee, limitando las distancias entre 800 y 20.000 metros (esta última con uso de repetidores) o mediante el uso de un módem GPRS que permite conexiones a largas distancias. La programación se realizará mediante software específico de la empresa (Agronic PC) instalado en un servidor del cliente.

Si hacemos uso del módulo de *Avisador GSM*, contará con la capaci-

¹Es el programador montado actualmente en la zona en la que se desean realizar las pruebas.

dad de envío de notificaciones vía SMS. Estas podrán configurarse hasta con 6 mensajes diferentes. Tanto el programador como los módulos de conexión requieren el uso de corriente eléctrica (220VAC).



Figura 3.2: Programador: Agronic 2500 fabricado por Progres.

Fuente: <http://www.progres.es/htmls/productos/es/programadores-riego-y-fertirrigacion/ficha/agronic-2500.html>

- **Hunter Industries - WVC + WVPE [35]**

Entre los programadores ofrecidos por esta casa se encuentra el WVC 3.3, un programador a pilas capaz de manejar hasta 4 electro-válvulas. Dispondremos además de una entrada para la conexión de un sensor. Dicho programador no cuenta con interfaz física para su programación por lo que, junto a este mismo producto, se deberá adquirir el WVPE. Este accesorio nos permitirá interactuar con los programadores de forma remota, mediante una conexión radio, hasta una distancia máxima de 30 metros. También podremos consultar los programas actuales así como el estado de sus baterías. Un WVPE, podrá ser enlazado con distintos programadores aunque no de forma simultánea.



Figura 3.3: Programador WVC y controlador inalámbrico WVPE de Hunter Industries

Fuente: <http://www.hunterindustries.com/irrigation-product/controllers/wvpwvc>

- **RainBird - WP [70]**

RainBird ofrece este programador 3.4 capaz de controlar hasta dos electro-válvulas. La alimentación se realiza con dos pilas de 9 voltios. Los programas se introducen a través de los 5 botones y es posible su visualización en una pequeña pantalla LCD, prometiendo gran sencillez en su manejo. El programador WP es compatible únicamente con el sensor de lluvia ofrecido por la misma compañía. Este programador no ofrece la posibilidad de ser controlado remotamente.

	Irri 5	Agrónic 2500	WVC	WP
Baterías	✓	✗	✓	✓
Electro-válvulas	5	9-27	1-4	2-8
Control Remoto	✗	✓	✓	✗
Distancia	-	Cobertura GSM	30m	-

Cuadro 3.1: comparativa de los programadores



Figura 3.4: Programador WP de Rainbird

Fuente: http://www.rainbird.eu/5-6558-Fiche-produit.php?id_produits=44

Como se puede observar en la tabla 3.1, ninguno de los programadores se adapta a los requerimientos.

3.1.2. Generadores de energía renovable

Dada la necesidad de crear un sistema automantenido, la energía del sistema debe suponerse infinita o inagotable². Puesto que en la actualidad no disponemos de ninguna batería que cumpla con dichos requisitos, se ha decidido recurrir a generadores de energía renovable que permitan cargar una batería auxiliar, permitiendo esta última mantener la actividad en los periodos en que la energía no pueda ser generada.

■ Generadores fotovoltaicos [15]

Aunque son los más extendidos en el sector, se trata de un producto

²Supondremos inagotable como un periodo muy largo de tiempo de mínimo 5 años dado un uso normal.

caro que debe estar situado en el exterior del recinto. Generalmente cuentan con el problema de que son robados con gran facilidad y obligan al cliente a adquirir junto a ellos un seguro frente a robo. La posibilidad de que días de poca luz no consigan cargar suficientemente las baterías no resulta un problema dado el bajo consumo del producto.

La placa fotovoltaica elegida [15] tiene un precio de 82€ al que se le debería añadir el precio del regulador solar [16] (30€) y la batería. El sistema es capaz de proporcionar un máximo de 50W a 12V en rendimiento máximo por lo que las características eléctricas que ofrece son suficientes como para cubrir las necesidades.

■ Generadores eólicos

Los aerogeneradores están mucho menos extendidos en el sector. Estos son notablemente más caros que los anteriores aunque la producción de energía es mayor. Su instalación viene limitada geográficamente, por lo que se deberá realizar un estudio previo.

El precio del generador estudiado [18] es de 415€ incluido el regulador. Las características eléctricas que ofrece este tipo de generadores son muy superiores a las necesitadas (300w a 12V en su máximo rendimiento), por lo que sería un sistema sobredimensionado. Al precio habría que añadirle el coste de la batería auxiliar. La vida útil está estimada en 20 años. El hecho de estar situado en el exterior los hacen susceptibles al vandalismo o robo.

En la figura 3.5 encontramos un ejemplo real de su uso en estaciones autónomas. En este caso, el riesgo de robo es menor dado que se encuentra muy próximo a un núcleo urbano.

■ Generadores hidroeléctricos

Hoy en día existen pequeños generadores hidráulicos que permiten, a partir de cierto caudal, generar energía limpia. En este caso, el sistema podría ser instalado en el interior del recinto evitando así posibles robos. La energía generada dependerá de las horas de riego del sistema pero, dado el bajo consumo del mismo y la frecuencia de riego, será suficiente para mantenerlo en funcionamiento.

El generador estudiado [19], mostrado en la figura 3.6, tiene un precio de tan solo 23.89€. Este incluye además una batería de 700mAh. Aunque las características eléctricas puedan resultar insuficientes porque la tensión proporcionada es de 3.6 VDC y la corriente máxima de tan solo 1.4 A ³, debido al precio, sería posible la instalación de varios generadores como este.

³Ofreciendo un total de tan solo 5WS



Figura 3.5: Estación autónoma que se abastece mediante la combinación de un generador eólico y otro solar situada junto al barranco del Carraixet en el término de Vinalesa.

En la tabla 3.2 podemos observar una comparativa con las principales características de los generadores de energía.

3.1.3. Programación

La elección del lenguaje de programación es crucial para el desarrollo del producto. Años atrás, entre los candidatos estarían lenguajes de más bajo nivel como ensamblador, pero hoy en día no se ha contemplado siquiera la opción debido a que no se trata de un producto que efectúe operaciones en

	Fotovoltaicos	Eólicos	Hidroeléctricos
Potencia generada	50W	300W	5W
Limitación geográfica	Baja	Alta	Ninguna
Precio	Medio	Alto	Bajo
Riesgo de robo	Muy alto	Medio	Muy bajo

Cuadro 3.2: Comparativa de generadores de energía



Figura 3.6: Microgenerador hidroeléctrico fabricado por Seed Studio

Fuente: http://www.seedstudio.com/depot/36V-Micro-Hydro-Generator-Pro-p-1280.html?cPath=1_2

tiempo real.

El lenguaje deberá permitir:

- Programación por capas que permita variar el hardware y funcionalidades en un futuro con gran facilidad.
- Optimizar los recursos.
- Minimizar el tiempo de desarrollo.

Así pues, los lenguajes que se han estudiado son:

- **C**
Siendo el lenguaje más clásico es, asimismo, el más extendido entre los microcontroladores. Es compatible con la mayoría de microprocesadores y permite una programación a muy bajo nivel, lo que facilita la optimización de los recursos, en especial de la memoria. Se trata de un lenguaje procedural puro.
- **C++** [32, 13]
Derivado del anterior, presenta algunas ventajas e inconvenientes. Aunque su presencia en los microcontroladores es menor, su uso está creciendo en la actualidad, por lo que cada vez encontramos más compiladores

	C	C++	Interpretados
Claridad de código	Baja	Alta	Alta
Optimización de recursos	Muy alta	Alta	Media
Velocidad de desarrollo	Baja	Media	Alta
Presencia	Muy alta	Alta	Muy baja

Cuadro 3.3: Comparativa de los lenguajes de programación

oficiales para los nuevos microcontroladores. Su ventaja principal es su orientación a objetos, que permite una programación mucho más ágil y ordenada. El principal inconveniente es que, al ofrecer opciones de más alto nivel, su mal uso puede repercutir en una peor gestión de los recursos.

■ Lenguajes Interpretados

Aunque no está muy extendido, también es posible la ejecución de lenguajes interpretados como Python, Basic, etc. en microcontroladores [53]. Estos lenguajes presentan las siguientes ventajas principales: no es necesaria su compilación, lo cual no es crítico sobre nuestro sistema debido al tamaño del programa; el reducido tamaño del código, que puede ser beneficioso, aunque como los microcontroladores están diseñados para albergar programas compilados, no suele ser el factor limitante; tipos dinámicos, que nos permitirá facilidad a la hora de realizar la programación. Las desventajas que manifiesta son la baja velocidad de ejecución y la poca expansión que tiene en el mercado [33].

En la tabla 3.3 se ponen de manifiesto las virtudes y deficiencias de los lenguajes.

3.1.4. Microcontrolador y entorno de desarrollo

Siendo el componente más importante en la parte física del proyecto, se deberá realizar un estudio de mercado entre los principales sistemas.

En cuanto al sistema a escoger, deberá ser completo⁴ y de bajo coste. Esto es necesario para asegurar la rentabilidad de los primeros prototipos; el sistema podrá ser sustituido en un futuro para reducir costes en producciones a mayor escala.

Por lo que respecta a las características hardware y en concreto a su rendimiento, el sistema no presentará demasiadas exigencias, pues no se trata

⁴Que no dependa de un programador externo y que ofrezca una IDE de programación profesional

de un sistema en tiempo real⁵.

El número bits del procesador tampoco será crítico en el sistema. No se pretenden realizar operaciones aritméticas de gran precisión ni almacenar valores de grandes contadores, por lo que tampoco resultará crucial.

Sí será decisiva la cantidad de memoria RAM así como la memoria de programa, ya que las operaciones de comunicaciones y almacenamiento externo pueden consumir gran parte de la misma. En cuanto al tamaño para el almacenamiento del programa, debido a que el sistema incluirá multitud de librerías, el tamaño de esta tampoco podrá ser muy bajo. Así pues, el tamaño mínimo estimado de memoria RAM será de 2Kb y la de programa de 32Kb.

La estimación del número de entradas/salidas será de un total de 5 salidas digitales para la interacción con las electroválvulas más 2 entradas analógicas para la lectura de los sensores. El sistema deberá contar con bus SPI e I2C que permitan comunicarse con los distintos periféricos (LCD, RTC, etc.) del sistema. Aunque estos pueden ser los requisitos mínimos, será necesario sobredimensionar el sistema, al menos en la fase de desarrollo, pues algunos de los periféricos podrían precisar de más entradas/salidas.

■ **Arduino Uno** [8]

Este proyecto, nacido en 2005, actualmente cuenta con más de 500000 tarjetas vendidas [45]. Se trata de una placa que pretendía servir para la iniciación a la programación de microcontroladores aunque, debido a su extensión, se han realizado proyectos de carácter avanzado con la misma. La versión más extendida del producto, el Arduino Uno [5], funciona en el microcontrolador ATmega 328 de Atmel [9]. Cuenta con una comunidad que ha crecido mucho en la última década. Se trata de un producto de código abierto-software libre con gran cantidad de librerías tanto oficiales como de terceros. El IDE oficial permite programar la placa bajo el lenguaje C ó C++. Dado que los microprocesadores integrados en la placa vienen con un *bootloader* precargado, no será necesario el uso de un programador externo. Entre sus principales deficiencias se encuentra el alto consumo de la placa respecto a otras de su misma categoría, aunque existen distintas soluciones que permiten resolver el problema [72]. El precio de la placa de estudio es de 24 € en su página oficial [6].

Como alternativa al IDE oficial de Arduino, podremos recurrir al IDE oficial de Atmel. Esta herramienta se distribuye con licencia privativa

⁵El tiempo de reacción mínimo viene dado por la interacción física del usuario con el producto, que puede llegar a ser de incluso varios s

	ATmega 328	TI M430G2553
Bus de datos	8 bit	16 bit
Velocidad	16 MHz	16 MHz
Almacenamiento	32 KB	16 KB
RAM	2 KB	512B
E/S Digitales	14	8
E/S Analógicas	6	8

Cuadro 3.4: Comparativa técnica de los microcontroladores

pero sin coste y además está disponible para todos sus microcontroladores más recientes. Algunas de las ventajas que ofrece esta interfaz respecto de la anterior es la posibilidad de *navegar* entre las clases así como las opciones de autocompletado. La información de los errores se mostrará de una forma mucho más amigable.

■ TI MSP430 [37]

Es la placa de desarrollo creada por Texas Instruments para competir contra Arduino. Ofrece compatibilidad con el código escrito en Arduino a través de la interfaz desarrollada por la comunidad *opensource* [25], Energia. Texas Instruments también permite la programación de este microcontrolador bajo el Code Composer Studio (CCStudio), que permite funciones más avanzadas no disponibles en la anterior interfaz. Podremos utilizar los lenguajes C o C++ para la programación del dispositivo. Esta placa ofrece como cualidad principal que se trata de un microcontrolador de ultrabajo consumo, lo cual posibilita alargar la vida de la batería. El gran interés de la compañía por ganar cuota de mercado se refleja en la completa documentación así como en su bajo precio.

La plataforma permite la compatibilidad con diversos microcontroladores. Su edición más barata y extendida es el LaunchPad con el microcontrolador TI M430G2553 y tiene un precio de 7 € en su web oficial [36].

Como se puede observar en la tabla 3.5, cada producto presenta sus ventajas e inconvenientes, siendo Arduino y TI los que mejor ratio presentan en cuanto a calidad precio. En la tabla 3.4 podemos observar la comparación técnica de los microcontroladores de cada plataforma.

	Arduino Uno	TI MSP430
Lenguajes	C/C++	C/C++
Precio	Bajo	Muy Bajo
Interfaz oficial para depuración	✓	✓
Consumo	Medio	Muy bajo

Cuadro 3.5: Comparativa plataformas

3.2. Software

El servidor debe consistir en un servicio en la nube capaz de gestionar tanto las conexiones de los clientes como las de los programadores, actuando además de nexo entre ellas de una forma transparente para el usuario.

Dado que el servidor no realizará complejos cálculos para ofrecer resultados al cliente, la mayor parte de la potencia de cálculo se concentrará en la gestión de conexiones y bases de datos.

También serán objeto de estudio las posibles soluciones de servidores completos ofrecidas en el mercado.

El número de clientes estimados durante la primera fase del producto será de 50 con 50 programadores. El sistema deberá ser capaz de soportar la interacción concurrente de todos ellos.

3.2.1. Software de control

El software de control es el encargado de realizar la comunicación con el programador. Este le facilita al usuario un control remoto de los programas así como la posibilidad de recibir notificaciones sobre problemas. Generalmente están orientados a grandes latifundios o empresas.

Las funcionalidades básicas que debe ofrecer dicho software son:

- Posibilidad de controlar remotamente los programadores
- Interfaz clara y con funcionalidades orientadas al pequeño agricultor
- Servicio en servidor externo, que permita recibir alertas sin que sea necesario mantener un ordenador encendido.

Las opciones que ofrecen las principales empresas son:

- **Agronic Net II** [69]

Es un software que se ejecuta en el servidor del cliente desde el que podremos modificar los programas y monitorizar el estado de las válvulas

de riego. Ofrece también la posibilidad de control mediante el teléfono móvil así como la recepción de notificaciones. Es posible la monitorización de sensores conectados en la instalación de campo. La conexión con el programador se puede realizar de forma tanto alámbrica como inalámbrica. Está orientado a comunidades de regantes o personas con grandes latifundios por lo que ofrece una visualización de sectores a través de un mapa 3.7.

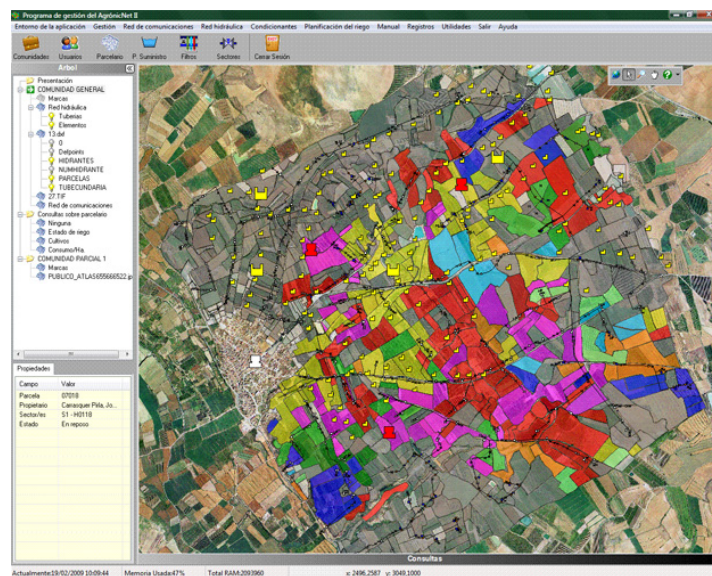


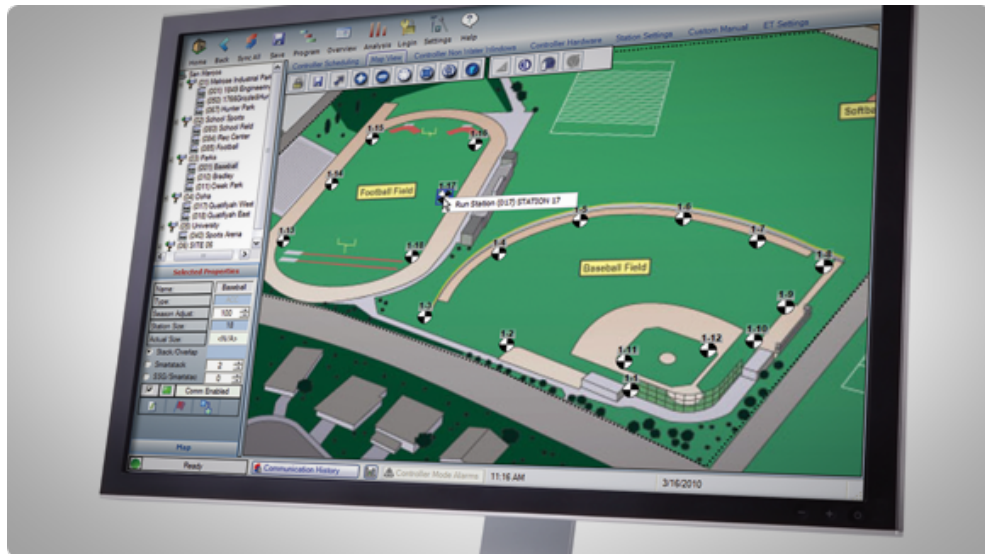
Figura 3.7: Software de control de riego Agronic Net II ofrecido por Progres

Fuente: <http://www.progres.es/htmls/productos/es/telegestion-del-riego/ficha/agronic-net-ii.html>

- **Hunter Industries - IMMS [34]**

Hunder Industries nos ofrece un software capaz de monitorizar y programar sus programadores. Este software se ejecutará sobre un sistema Windows en el ordenador del cliente. La comunicación se realizará de forma inalámbrica o por cable. Una interfaz basada en mapa nos permitirá visualizar nuestras instalaciones. Es compatible con la mayoría de los sensores ofrecidos por la compañía, lo que permitirá recibir avisos en caso de fallos o situaciones extremas. La visualización, al igual que el anterior, se realiza a través de un mapa 3.8 que permitirá visualizar los sectores en grandes extensiones.

Como podemos ver en la tabla 3.6, el software ofrecido por las empresas no encaja con todas las características necesarias para nuestro producto ya



los tres sistemas operativos más importantes [30]. En su instalación más básica, se trata de un servidor estático que implementa el protocolo HTTP/1.1 [14] y la posibilidad de hacer uso de sus sitios virtuales⁷. Gracias a la posibilidad de instalación de paquetes resulta realmente sencilla la configuración de un servidor LAMP⁸ en el que correr aplicaciones dinámicas. La posibilidad de activación de módulos facilitará al usuario configurar a su medida el servidor.

El servidor cuenta con una amplia comunidad además de estar presente en muchos de los servidores más importantes y con más afluencia.



Figura 3.9: Logo de Apache HTTP Project

Fuente: <http://httpd.apache.org/>

■ Nginx 3.10

Se presenta como un servidor web de alto rendimiento [55]. En los últimos años ha presentado el mayor crecimiento y actualmente está presente en un 20 % de los servidores. Ha sido adoptado por plataformas importantes como WordPress o GitHub. Es un proyecto de software libre y código abierto que cuenta con una comunidad menor que la de Apache pero que ha aumentado junto a su cuota de mercado.

Su diferencia principal respecto a Apache es que Nginx es un servidor basado en eventos. Esto le permite un consumo de memoria mucho menor, llegando a soportar más de 10000 conexiones simultáneas [56].



Figura 3.10: Logo de Nginx

Fuente: <http://wiki.nginx.org/Main>

⁷La posibilidad de utilizar el mismo servidor para distintos dominios

⁸Linux, Apache, MySQL, PHP. Combinación de tecnologías más utilizadas en servidores web.

	Apache	Nginx	IIS
Software Libre	✓	✓	✗
Precio	Gratuito	Gratuito	Gratuito ¹⁰
Personalización	Alta	Alta	Baja
Rendimiento	Medio	Muy alto	Medio
Nivel administrador	Medio	Medio	Bajo

Cuadro 3.7: Comparativa servidores web

■ Microsoft IIS 3.11

Es la solución ofrecida por Microsoft. Está disponible gratuitamente para todas las versiones de su sistema operativo destinadas a ofrecer servicios⁹. Como principal característica nos ofrece la posibilidad de gestionar nuestro servidor de una manera gráfica, que ayudará al usuario con pocos conocimientos a una gestión más sencilla del servidor [48]. Dispone de soporte nativo para el uso de ASP así como PHP o Perl [51]. Como es característico en la empresa, se trata de software privativo de código cerrado. Este sistema cuenta con un 14 % de cuota de mercado, aunque su elevado precio de licencia y sus restricciones como software privativo lo están haciendo retroceder.



Figura 3.11: Logo de Microsoft

Fuente: <http://www.microsoft.com/en-us/news/imagetail.aspx?id=4DABA54CBB4D25A9E9905BC59E4A6D44E33694EA>

3.2.3. Web Frameworks

Los frameworks nos permiten un desarrollo más ágil a la hora de realizar webs de gran tamaño. Los frameworks son librerías de funciones que evitan tener que reescribir operaciones comunes cada vez que empezamos un nuevo proyecto web.

⁹Server, Professional e Interprise

¹⁰Incluido en licencias de windows server

- **ASP.NET 3.12**

De la familia .NET, está desarrollado por Microsoft. Cuenta con la mayor cuota de mercado en el sector. Podremos desarrollar en cualquier lenguaje soportado por .NET (C++, Python, Perl, C#...) permitiendo incluso desarrollar un mismo producto con distintos lenguajes [29, p 3].

Como es común en este desarrollador, estamos delante de un software de carácter privativo aunque, en este caso, su licencia será gratuita. No deberemos olvidar, no obstante, el coste asociado al producto, pues solo se nos permitirá ejecutar ASP.NET sobre Microsoft IIS, servidor web gratuito también pero que solo podremos ejecutar sobre las versiones más completas de Windows.

Este producto presenta importantes ventajas. Para comenzar, ofrece la posibilidad programar en distintos lenguajes, permitiendo al desarrollador escoger el que más le convenga y facilitando además el trabajo de un grupo de desarrollo más heterogéneo. Seguidamente, presenta una completa integración con el resto de productos de la compañía como IIS y SQL Server, optimizando el tiempo de ejecución y desarrollo [50]. Finalmente, al tratarse de un lenguaje compilado, el tiempo de ejecución es menor que competidores directos escritos sobre lenguajes interpretados. El uso de ASP.NET promete un nivel mayor de seguridad que el resto de sus competidores.



Figura 3.12: Logo de ASP.NET de Microsoft

Fuente: <http://www.asp.net/>

- **Ruby on Rails 3.13**

Se trata del framework open source más utilizado. Programado en ruby,

se trata de un framework *full-stack* por lo que nos facilitará el desarrollo tanto del *front-end* como del *back-end* pasando por la gestión de conexión a bases de datos [27, p VII]. Algunas de sus principales ventajas son: su gran comunidad de soporte, gracias a que posee la máxima cuota de mercado en proyectos framework de código abierto; soporte multiplataforma, que nos permitirá portar dicho software a los principales sistemas operativos Unix y Windows [26].



Figura 3.13: Logo de Ruby on Rails

Fuente: http://en.wikipedia.org/wiki/Ruby_on_Rails

■ Angular JS 3.14

Framework de código abierto en JavaScript. Actualmente mantenido por Google, es el más importante escrito en dicho lenguaje. Dada la naturaleza de este lenguaje, este framework se ejecuta sobre la parte del cliente [28], siendo perfecto para aplicar un patrón MVC¹¹. Así pues, diremos que Angular JS es un framework que se encarga del *front-end* o, lo que es lo mismo, de la Vista. Como es obvio, este framework es multiplataforma ya que dependerá exclusivamente del navegador y el motor javascript que este ejecute [41].



Figura 3.14: Logo de Angular JS de Google

Fuente: <https://github.com/angular/angular.js/tree/master/images/logo/AngularJS.exports>

¹¹Model View Controller es una filosofía de Ingeniería de Software que separa la lógica de negocio y datos de la interfaz de usuario.

■ Django 3.15

Django es un framework de código abierto nacido en 2005 [39, p 2]. Escrito en Python, este framework permite un desarrollo completo tanto de la lógica de negocio como de la vista. Está basado en una filosofía MVC, aunque esta no se cumple estrictamente pues, como pasa en los desarrolladores de Python, prefieren hacer *lo que les parece correcto*.

Algunas de las ventajas que presenta Django derivan del propio lenguaje, Python, que obliga al desarrollador a utilizar una estructura muy limpia y organizada [23], cosa que facilitará el mantenimiento de las aplicaciones así como la modificación de código de terceros. Ofrece una gran integración de bases de datos así como de Javascript y Ajax. Django proporciona un entorno de programación completo, lo cual facilitará la detección y depuración de errores [40].



Figura 3.15: Logo de Django

Fuente: <https://www.djangoproject.com/community/logos/>

■ Symfony 2 3.16

Es el framework que mayor crecimiento ha experimentado en los últimos años. Symfony está programado en PHP, lenguaje por excelencia en el desarrollo de webs dinámicas y desarrollado por la compañía francesa SensioLabs. De nuevo, nos encontramos ante un framework completo que cumple con las directivas de diseño MVC [24]. Como ventajas presenta una gran flexibilidad, permitiendo a los desarrolladores del proyecto construir una versión del framework solo con las funcionalidades que deseen. El soporte, ofrecido tanto por SensioLabs como por la gran comunidad, permite una comunicación rápida a través de listas de correo y IRC [24].

¹²Excluyendo precio de sistema operativo base y herramientas de desarrollo



Figura 3.16: Logo de Symfony

Fuente: <http://symfony.com/logo>

	ASP.NET	Ruby on Rails	Angular JS	Django	Symfony
Licencia	EULA	MIT	MIT	BSD	MIT
Precio	Gratuito ¹²	Gratuito	Gratuito	Gratuito	Gratuito
Lenguaje	Multilenguaje	Ruby	JavaScript	Python	PHP

Cuadro 3.8: Comparativa web frameworks

3.2.4. Base de datos

Dado que el almacenamiento de información es un factor importante en el proyecto, se estudiarán las bases de datos con más cuota de mercado [21] así como otras de reciente creación que, aunque no gozan de la misma cuota, sí han recibido el apoyo de grandes marcas y han experimentado un importante crecimiento.

■ Oracle 3.17

Es la base de datos de tipo objeto-relacional más extendida en el mercado. Se remonta a 1977, por lo que ha estado presente en todo momento en la expansión de Internet contando así con una gran experiencia en el mercado. Se trata de un software privativo teniendo un coste en todas sus versiones¹³.

Aun siendo un software privativo, Oracle ofrece un gran abanico de productos que se adaptan a las diferentes necesidades del mercado, siendo además cada uno de ellos altamente configurable [12].

El precio a pagar por el producto dependerá no solo de la versión utilizada sino que algunas de sus versiones ofrecen una modalidad de pago diferente según las características del hardware utilizado [57].

Desde su versión 8i (1999) se centra en trabajar más próximo a la web,

¹³Exceptuando la versión Express Edition que permite un uso muy restringido del producto

permitiendo hacer consultas desde internet. JSP es su buque insignia para su integración, aunque ofrece soporte también a las mayores plataformas como PHP [58].



Figura 3.17: Logo de Oracle

Fuente: <http://www.oracle.com/us/corporate/press/016236>

■ MySQL y MariaDB 3.19

Base de datos de tipo relacional. Comprada en 2008 por Sun Microsystems y esta misma a su vez en 2009 por Oracle, forma la segunda fuerza en el sector. Consiguió una gran adopción durante la expansión de la web dado que se trataba de un producto de software libre y sin coste. Actualmente, sigue siendo un proyecto de software libre gratuito aunque Oracle ofrece diferentes versiones del producto que incluyen otras características [54] de pago.



Figura 3.18: Logo de MariaDB

Fuente: <https://mariadb.org/en/logos/>

Con la compra de MySQL por parte de Oracle (muy mal vista por las comunidades de software libre dado que todo sus productos resultan privativos) nació una bifurcación¹⁴ de MySQL, MariaDB [44] 3.18. Este proyecto está ganando popularidad en el mercado y cuenta con la adopción de grandes empresas como Google que ya han migrado sus productos en MySQL a MariaDB.

¹⁴ *fork*



Figura 3.19: Logo de MySQL

Fuente: <http://www.mysql.com/about/legal/logos.html>

- **Microsoft SQL Server 3.20**

Esta base de datos de tipo relacional ocupa la tercera fuerza en el mercado. Esta opción ofrecida por Microsoft es de código cerrado y privativo. El precio de explotación del producto variará según dos factores principales: la versión del mismo y el tipo de licencia escogida. Existen distintos tipos de licencias que dependen del número de cores utilizados, licencias por tipo de servidor, licencias por volumen, etc. [47] La instalación del servidor SQL solo se podrá realizar sobre Windows, por lo que al precio de la licencia se le deberá sumar el precio de este último. Microsoft sí ofrece driver de comunicación para Linux, por lo que será posible la conexión desde un servidor externo a la base de datos con dicho sistema operativo. Microsoft ofrece además una perfecta integración con Windows Server, por lo que podría resultar una opción óptima en empresas en las que ya dispongan de un servidor con un sistema operativo Windows [46].

Como principales ventajas cabe destacar su optimizador de consultas, que resulta notablemente mejor que el ofrecido por su competidora más directa, Oracle. Asimismo, presenta una mayor integración con su producto de desarrollo .NET, resultando mucho más sencillo y eficaz el trabajo con estas dos tecnologías. Finalmente, su sencillez de configuración y mantenimiento acorde con Windows Server permite que un mismo administrador gestione los mismos servicios [11, p 2].

- **PostgreSQL 3.21**

Es el proyecto software libre de base de datos objeto-relacional más importante en el mercado. Nacido hace menos de dos décadas, ha conseguido hacerse con una cuota importante de mercado. Su instalación se puede realizar tanto en sistemas Unix como Windows [74]. Sus ventajas predominantes son: una fuerte comunidad, que ofrece soporte del



Figura 3.20: Logo de SQL Server de Microsoft

Fuente: <http://www.microsoft.com/en-us/server-cloud/products/sql-server/#fbid=5hXq3BP7EJ8>

	Oracle	MySQL	SQL Server	PostgreSQL
Tipo	Objeto Relacional	Relacional	Relacional	Objeto Relacional
Software libre	×	✓	×	✓
Gratuito	×	✓	×	✓
Multiplataforma	✓	✓	×	✓

Cuadro 3.9: Comparativa de bases de datos

producto todos los días a todas horas y a coste cero; una gran cantidad de extensiones, producidas por terceros que han decidido implementar dicha base de datos [63].



Figura 3.21: Logo de PostgreSQL

Fuente: <http://wiki.postgresql.org/wiki/Logo>

3.3. Comunicaciones

3.3.1. Conexión

Las comunicaciones serán realizadas a petición del programador. Debe quedar presente que la potencia de cálculo en el programador será limitada. [75]

Recordemos que la tecnología deberá permitir:

- Realizar conexiones de forma inalámbrica.
- Realizar conexiones en zonas remotas sin acceso a la red eléctrica.

Algunas de las opciones hardware estudiadas fueron:

■ Conexiones Wi-Fi

Está demostrado que con antenas unidireccionales y con contacto visual entre ellas se puede conseguir un alcance de varios kilómetros [78]. Partiendo de este hipotético caso, el uso de estas antenas obligaría al cliente a disponer de conexión a Internet en la otra parte donde está situado el receptor para realizar la conexión con el servidor. Estas antenas, además, deberían estar situadas en el exterior del habitáculo y resultaría complicada su instalación. La ventaja principal que presenta el uso de esta tecnología, dadas las restricciones de proximidad y bajo el supuesto de que el usuario ya dispusiese de conexión a Internet, es que el coste de conexión sería gratuito puesto que se utilizaría el ya contratado por el usuario.

El módulo Wi-Fi de estudio [7] tiene un precio de 80€ y cuenta con la capacidad de conexión de una antena externa, lo que nos permitiría lograr mayores distancias que en el caso contrario. Éste módulo se comercializa para la plataforma Arduino, por lo que también será compatible con la MSP430.

La estructura de comunicaciones de esta tecnología se muestra en la figura 3.22.



Figura 3.22: Esquema de comunicaciones para tecnología Wi-Fi

■ Transmisiones de radio UHF

Esta tecnología utiliza las bandas de frecuencias libres disponibles para realizar la conexión [3]. De nuevo se verá limitada la distancia a la que podemos transmitir datos. Se debe hacer un despliegue de estaciones por parte de la empresa o del cliente que permitan realizar la conexión con el servidor. Además, el servidor deberá disponer de un receptor radio, que permita recibir las conexiones de los diferentes dispositivos. Además de la dificultad de instalación a grandes distancias, las transmisiones de radio podrían suponer un cuello de botella, por lo que podría limitar la escalabilidad del sistema. Al igual que en el caso anterior la transmisión de datos sería gratuita.

Los módulos estudiados [31] permiten el intercambio de información mediante radio. Estos módulos deberían estar presentes tanto en el programador como en el servidor, así como en las posibles estaciones intermedias repetidoras, las cuales deberían de disponer además de alimentación. La conexión con el microprocesador se realiza mediante I2C, por lo que será compatible con todos los estudiados en el apartado anterior.

En la figura 3.23, se representa a modo de esquema la estructura de comunicaciones para esta tecnología.



Figura 3.23: Esquema de comunicaciones para tecnología radio

■ Conexiones GSM

En este caso la distancia es muy grande a causa de la externalización de la gestión de la conexión. Sin embargo, se deberá contar con cobertura en la zona para su instalación. En este caso, la conexión no será gratuita aunque, dada la gran extensión realizada en los últimos años, tampoco resultará excesivamente cara. La instalación en el campo será sencilla, pudiendo situar la antena en el interior del recinto a fin de evitar posibles robos.

El módulo GPRS estudiado [42] permite realizar llamadas, envío de sms y conexiones a Internet. Aunque su *layout* cuenta con una distribución para conexión a placas Arduino, la conexión se realizará mediante conexión serie. El precio del producto es de 63 €. La comunicación con el microcontrolador se realizará mediante conexión serie. Esto nos asegu-

	Wifi	UHF	GSM
Distancia	Reducida	Reducida	Muy grande ¹⁵
Limitación geográfica	Alta	Muy baja	Media
Coste instalación	Medio	Alto	Muy Bajo
Coste por conexión	Medio	Gratuito	Medio

Cuadro 3.10: Comparativa conexiones inalámbricas

ra la compatibilidad con todos los microcontroladores estudiados. Como podemos observar en la figura 3.24, esta tecnología es la que menos elementos requiere para realizar la conexión cliente-servidor.

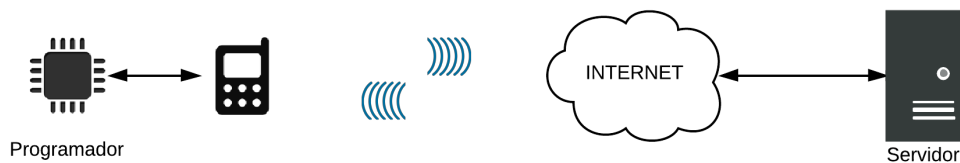


Figura 3.24: Esquema de comunicaciones para tecnología GSM

En la tabla 3.10 podemos observar las características principales de las comunicaciones.

3.3.2. Protocolo de comunicaciones

Por otra parte deberemos escoger un protocolo para realizar dichas comunicaciones. Este especifica la forma de comunicación cliente-servidor una vez realizada la conexión.

En el estudio se deberá tener en cuenta

- La capacidad de cómputo y almacenamiento en el programador.
- Que sea lo más versátil posible para futuros productos.

A continuación se exponen los protocolos estudiados:

- **Gestión de conexiones a través de sockets**

Permite minimizar la transmisión de información no útil. Posibilita crear sesiones que identifiquen al usuario. Se debe crear un programa que permita realizar conexiones simultáneas desde distintos clientes.

¹⁵Infinita siempre que se cuente con cobertura GSM

- **Gestión de conexiones a través de una arquitectura REST**

Favorece una gran reutilización, facilitando gestionar las conexiones tanto del programador como de la página web. Delegaremos la administración de las conexiones en el servidor http, eliminando posibles *bugs* en nuestro código. Permite una gran simplicidad en la interacción.

- **Gestión de conexiones serie**

Este tipo de conexión estaría pensado en caso de que se decidiese realizar una conexión radio. El método de conexión sería similar al de sockets pero, en este caso, el número de conexiones simultáneas sería mucho más limitado, tantas como multiplexaciones permitiese la señal.

	Sockets	APIREST	Serie
Tiempo desarrollo	Medio	Bajo	Alto
Versatilidad	Media	Alta	Baja
Rendimiento	Alto	Medio	Bajo

Cuadro 3.11: Comparativa de los protocolos de comunicación.

Capítulo 4

Especificación

En este capítulo se van a definir las especificaciones del proyecto. Para ello se hará primero un análisis de requisitos. Una vez establecidos estos requisitos, se deberán escoger las tecnologías a utilizar estudiadas en el apartado de Estado del Arte. Finalmente se realizará la planificación temporal y de costes para abordar el proyecto.

4.1. Análisis de requisitos

Para abordar correctamente el análisis de requisitos, este apartado deberá dividirse en tres subapartados: requisitos del programador 4.1.1, requisitos del servidor 4.1.2 y requisitos de las comunicaciones 4.1.3

4.1.1. Requisitos del programador

Los requisitos técnicos del programador vienen dadas por el estudio de los programadores actuales existentes en el mercado, realizado en el apartado anterior. No se debe perder la perspectiva del producto: un programador para el control de riego en minifundios, no latifundios o grandes extensiones.

Para mayor claridad, se diferenciará entre los requisitos físicos o hardware de los software o de programa.

4.1.1.1. Requisitos hardware del programador

- **Presión máxima de línea**

El sistema debe soportar una presión máxima de 8 bares en la línea principal.

- **Autonomía**

La autonomía debe ser ilimitada.

- **Electroválvulas**

El programador debe permitir el control de electro-válvulas de dos hilos tipo *latch*.

- **Salidas para 2 sectores**

El programador debe ser capaz de gestionar hasta dos electroválvulas independientes que permitan el control tanto del riego como de fertilizante.

- **Entradas para la conexión de hasta 4 sensores**

Dos asignadas a los sensores de presión y otras dos para futuras actualizaciones del sistema.

- **Conexión al servidor**

Se deberá permitir la conexión con el servidor central de forma inalámbrica. Dicha conexión la establecerá siempre el programador con el servidor y no a la inversa.

4.1.1.2. Requisitos software del programador

- **Configuración inicial**

Se debe permitir la configuración de los programas de riego sin ningún tipo de dispositivo externo ni conexión a Internet.

- **Configuración inicial mediante tarjeta SD**

Se podrá realizar la carga de una configuración inicial mediante una configuración guardada en una tarjeta de almacenamiento SD que contenga dicha configuración previamente guardada por el usuario.

- **Arranque y paro manual**

Se debe permitir el arranque y paro manual de las electro-válvulas sobre el terreno. Este estará limitado a un número máximo de horas predefinido por el usuario a fin de evitar posibles descuidos.

- **Programas**

El programador debe ser capaz de almacenar hasta 10 programas, tanto activos como inactivos.

Los programas vendrán definidos por:

- Periodo de tiempo
Indicará desde y hasta qué hora debe permanecer activo el programa.
- Días de la semana
Días de la semana en que deberá activarse el programa.
- Estado
Cada programa puede ser activado o desactivado por el cliente. Un programa inactivo no se ejecutará llegada la hora configurada.
- Tipo e identificación de electro-válvula
Cada programa va ligado a una electro válvula, esta a su vez será de tipo riego o abonado.

■ **Prevención de errores en programa**

Existen operaciones no deseadas que deben ser prevenidas y notificadas. En los siguientes casos se deberá actuar en consecuencia:

- Tiempo de abonado no contenido dentro de un programa de riego
Debido a que el abonado circula a través del agua de riego, no siendo posible si así no ocurriese, se deberá impedir y advertir la programación de horas de abonado si no está contenida dentro de otro ciclo de riego activo.
En caso de desactivar un programa de riego que contuviese a otro de abonado, se deberá advertir al usuario y desactivar ambos.
En caso de borrar un programa de riego que contuviese a otro de abonado, se notificará al usuario, se borrará el programa de riego y se desactivará el programa de abonado.
- Solapamientos de programas
No se debe permitir la creación o activación de programas que accionen una misma electro-válvula a la misma hora.

■ **Notificación de errores**

En caso de fallo de algún módulo del sistema, se deberá notificar al usuario el fallo y gravedad del mismo. En caso de resolverse se marcará como resuelta.

Los errores a notificar serán:

- Fallos dados en el apartado anterior.
- Fallo en el suministro de agua.
- Fallo en apertura electro-válvula.
- Fallo en cierre de electro-válvula.

- Fallo en línea de riego.

- **Notificación de advertencias**

Se deberá avisar al usuario de posibles riesgos o fallos inminentes en el sistema. Esto permitirá al cliente actuar en consecuencia si lo considerase oportuno antes de que fuese demasiado tarde.

Las advertencias a notificar serán:

- Riesgo por alta presión
Se debe notificar al cliente en caso de que la presión a la entrada de línea exceda el máximo permitido.
- Riesgo por baja presión
El cliente debe ser notificado en caso de que la presión en la entrada de línea sea menor de la deseada.
- Nivel de carga de la batería
En caso de detectarse un nivel bajo de autonomía se deberá notificar al cliente.

- **Conexiones al servidor**

Las conexiones al servidor se deberán realizar de forma periódica. El intervalo de las mismas dependerá del estado de las baterías. Se efectuarán conexiones al servidor en caso de:

- Configuración manual
Se deberá conectar con el servidor antes y después de realizar cambios de programación en el programador. De esta forma se limitarán posibles conflictos por falta de sincronización.
- Errores
En caso de producirse alguno de los errores anteriormente citados tanto al inicio como a su fin.
- Advertencias
Solo se realizará la notificación de las advertencias en caso de que el estado de las baterías no sea muy bajo.

4.1.2. Requisitos del servidor

Las especificaciones del servidor deben ser compatibles con las del programador.

- **Acceso al servicio**

El cliente podrá acceder a todos los servicios mediante un portal web.

Para ello deberá facilitar un usuario y contraseña ambos creados por el mismo durante el proceso de inscripción.

- **Concurrente**

El servicio debe permitir a los usuarios acceder al servicio de forma concurrente. También se debe permitir la comunicación simultánea de distintos programadores a la vez. Se deben poder soportar hasta 50 ¹ programadores más clientes.

- **Múltiples programadores por cliente**

Un usuario podrá acceder a todos sus programadores desde una única cuenta de usuario.

- **Interfaz clara y poco sobrecargada**

La interfaz de usuario debe contener solo los elementos presentes en un programador físico, ya que se trata de un público sin grandes conocimientos técnicos.

- **Historial de riego**

Se debe facilitar al usuario acceder y descargar el historial de datos: horas de riego, presión en línea, etc.

- **Conflictos durante la sincronización**

En caso de haber algún conflicto durante la sincronización de los programas, se dará prioridad a la información proporcionada por el programador.

- **Fallos**

En caso de recibir un fallo de un programador, deberá ser notificado al cliente inmediatamente mediante un correo electrónico o un sms. Además, se deberá mostrar un aviso destacado en la interfaz.

En caso de resolverse la incidencia, se notificará de nuevo al cliente y aviso en la interfaz cambiará de color.

- **Advertencias**

Las advertencias se mostrarán siempre en la interfaz de usuario. Además, el cliente será notificado por correo electrónico.

- **Información meteorológica**

Se deberá mostrar la previsión meteorológica de la semana así como

¹Durante una primera fase de expansión, este número ha sido extraído del porcentaje de campos con riego centralizado en la comunidad de regantes de San Pascual Bailón de Ribarroja del Túrria.

otros datos de interés facilitados por institutos de investigaciones como el IVIA [38]

- **Localización**

Para una correcta información meteorológica, el cliente deberá configurar la ubicación del programador. Esta ubicación no podrá ser calculada por el programador de forma automática.

- **Gestión copias de seguridad**

El servidor debe de ser capaz de recuperar los datos tras un fallo en el sistema. Se deberá evitar en todo momento la posible pérdida de datos.

4.1.3. Requisitos de las comunicaciones

Los requisitos para las comunicaciones serán los siguientes:

- Dado el nivel de conocimiento tecnológico de usuario final, las comunicaciones deberán resultar totalmente transparentes para el usuario.
- La configuración de las mismas deberá realizarla el administrador del servicio, evitando así complicadas configuraciones para el usuario.
- Las comunicaciones de cliente-servidor deberán realizarse inalámbricamente y su consumo de datos no debe superar los 25 MB mensuales. La posibilidad de conexión no debe depender de la distancia real entre el servidor y el programador.

4.2. Conclusiones

La óptima elección de las tecnologías será un punto clave para poder finalizar el proyecto satisfactoriamente. Es por ello que, después del estudio realizado en el capítulo 3 (estado del arte, página 9), deberemos escoger las tecnologías con las que se llevará a cabo el diseño del producto y que nos permitirán cumplir con las características anteriormente definidas especificaciones.

Finalmente en el subapartado Otros 4.2.4, se citarán otras tecnologías escogidas que no han sido evaluadas durante el estado el arte.

4.2.1. Programador

Para la elección de las tecnologías utilizadas en el programador, partiremos de aquellas con mayor importancia y que confeccionarán la elección del

resto de ellas.

- **Lenguaje de programación - C++**

Finalmente se ha optado por realizar la programación en C++ frente a C, dado que no nos encontramos en un entorno en tiempo real donde el tiempo de ejecución es crítico. Además, como se ha estudiado anteriormente, los rendimientos no son tan dispares si se programa correctamente. Por otra parte, ganaremos con un código más ordenado, lo que permitirá aumentar la productividad y disminuir el tiempo de desarrollo.

- **Microcontrolador y entorno de desarrollo - Arduino + Atmel Studio**

Quedando fuera la oferta de PIC por no ser compatible de forma oficial con C++, la elección final ha sido escoger Arduino como microcontrolador. El IDE escogido no ha sido el propuesto por la compañía sino el ofrecido por el fabricante, Atmel Studio. Dicha elección viene motivada por la experiencia del usuario con dicha plataforma e interfaz, lo cual permitirá reducir la curva de aprendizaje y por tanto, tiempo de desarrollo.

Cabe dejar presente que la oferta de Texas Instruments es compatible con el código y las librerías ofrecidas por Arduino, lo que permitirá en un futuro migrar a dicho entorno con gran facilidad, beneficiándose de las ventajas ofrecidas por el producto.

Para que el sistema sea más portable a otras ofertas que puedan surgir en el futuro, la programación se deberá realizar de forma modular, creando una capa intermedia que haga de puente entre el núcleo del sistema y las llamadas específicas del microcontrolador.

- **Conexión - GSM**

La tecnología GSM ha sido la escogida para realizar las conexiones al servidor. La razón principal de la elección es la eliminación del límite de distancia. Aunque con esta tecnología contamos con la limitación geográfica, que no permitirá instalar el sistema en zonas donde no exista cobertura, el problema será menor dado que la gran mayoría de las zonas cuentan con este servicio.

En un futuro, se plantea poder utilizar instalaciones híbridas que permitan, en combinación con tecnologías de menor alcance como la radio, llegar a aquellos sitios donde no exista cobertura móvil.

Para cumplir con el objetivo de eliminar la configuración de las comunicaciones por parte del cliente, la gestión de elementos como la

tarjeta SIM correrán por parte de la empresa, que se encargará de su configuración.

- **Generador de energía renovable - Hidroeléctrico**

Entre las expuestas anteriormente, se ha optado por el generador hidroeléctrico, ya que era el único que ofrecía la posibilidad de permanecer en el interior del habitáculo, dificultando posibles robos. Además, aunque solo permite la carga de la batería durante los tiempos de riego, estos se suponen lo suficientemente amplios como para que no implique un problema.

Cabe decir, no obstante, que esta elección no impide que en productos futuros se pudiesen utilizar otras de las tecnologías, o incluso la red eléctrica en caso de estar disponible, por lo que el programador deberá ser diseñado para afrontar estos cambios en un futuro.

4.2.2. Servidor

Las tecnologías escogidas para desarrollar el servidor han sido seleccionadas no solo para poder cumplir con las especificaciones actuales, sino para poder afrontar futuras necesidades que surjan en etapas posteriores de desarrollo.

Como principales directivas para la selección de las tecnologías hemos optado por el software libre y gratuito con gran presencia o proyección en el mercado, que nos asegurará facilidades a la hora de resolver posibles problemas durante el desarrollo.

La elección de las tecnologías se ha hecho, al igual que en el caso del programador, empezando por aquellas que condicionarán al resto, en este caso, el framework escogido.

- **Web framework - Symfony 2**

Para poder afrontar el reto de escalabilidad, hoy en día deberemos utilizar un framework. Aunque sería totalmente viable el desarrollo del sistema sin el uso del mismo, en un futuro permitirá una mayor facilidad en su mantenimiento a la vez que evitaremos posibles fallos de seguridad o *bugs* en las operaciones más básicas. Es por ello que hemos elegido el desarrollo de la aplicación mediante el uso de un framework, a pesar de que no se tienen conocimientos en ninguno de los frameworks estudiados, lo que aumentará la curva de aprendizaje y por tanto el tiempo de desarrollo de la aplicación inicial.

Entre los frameworks estudiados, se ha apostado por Symfony 2 debido al gran crecimiento que ha experimentado en el sector además de que permitirá reaprovechar los conocimientos que ya se tienen sobre PHP.

- **Servidor web - Apache**

El framework escogido es compatible con toda la gama de servidores web estudiados. Debido a la apuesta por el uso de software libre, el producto de Microsoft ha sido descartado desde un principio. La elección entre Nginx y Apache ha sido una tarea difícil ya que, aunque años atrás Apache hubiera sido la apuesta indiscutible, hoy en día Nginx ha experimentado un gran crecimiento y promete rendimientos mucho mayores que el anterior. Aun siendo así, se ha optado por el sistema tradicional de Apache, ya que se ha trabajado con él anteriormente.

- **Base de datos - MariaDB**

Quedando fuera las ofertas de Oracle y Microsoft, se ha optado por Maria DB. En segunda instancia, quedó descartada MySQL ya que, si actualmente sigue siendo software libre, la compra de Oracle deja dudas sobre su futuro. Quedando la elección entre PostgreSQL y Maria DB, la elección favorable ha sido la segunda por motivos de rendimiento [59].

4.2.3. Comunicaciones

Este apartado es de especial interés debido a que será la forma de entenderse e intercambiar datos entre el programador y el servidor. De nuevo, la elección deberá permitir una escalabilidad permitiendo compatibilidad y adaptabilidad para futuros nuevos productos.

- **Arquitectura REST**

Habiendo ganado mucho terreno en los últimos años, esta arquitectura casa perfectamente con nuestras necesidades ya que es el programador el que realizará las peticiones al servidor cuando lo desee. REST ofrece, además, la posibilidad de utilizar nuestra API para peticiones en la misma web de una forma sencilla y asegura una gran compatibilidad con la aparición de futuros productos. La posibilidad de gestionar conexiones https nos permitirá también olvidarnos de la encriptación en las peticiones².

4.2.4. Otros

A continuación se citarán ciertas tecnologías tanto hardware como software que serán utilizadas en el proyecto. La elección de estas no será detallada

²Si el producto lo permitiese, no siendo el caso del actual programador.

igual que las anteriores debido a que no ha resultado interesante su comparación con rivales en el mercado, ya sea porque no afectan de forma importante al desarrollo del proyecto o debido a que ya se disponía de ellas.

- **Trello**

Esta herramienta gratuita online sirve para administrar proyectos. Permite la distribución de tareas en distintas listas, TO DO, DOING, DONE inicialmente, aunque es totalmente personalizable.

- **Ubuntu 12.04 LTS**

El sistema operativo Linux será utilizado en el servidor bajo una máquina virtual.

- **Hardware servidor**

Durante el desarrollo del proyecto recurriremos a un servidor personal. Las características hardware del mismo será:

- Asus E45M1-M PRO
- 16GB RAM
- SSD 128GB, HDD 3TB
- Conexión Upload 200Mbps
- Conexión Download 200Mbps
- Windows server 2012 v2

Estas deberán ser suficientes para pasar las pruebas de rendimiento expuestas en el capítulo 6.1.1.2.

- **Control de versiones**

Para realizar el control de versiones del producto se ha recurrido al tradicional Subversion por ser el que ya utilizamos en otros proyectos.

4.3. Especificación del sistema

Definidos ya los requisitos en la sección 4.1 y las tecnologías a utilizar para llevar a término el proyecto (sección 4.2), se va a proceder a proponer una solución al sistema.

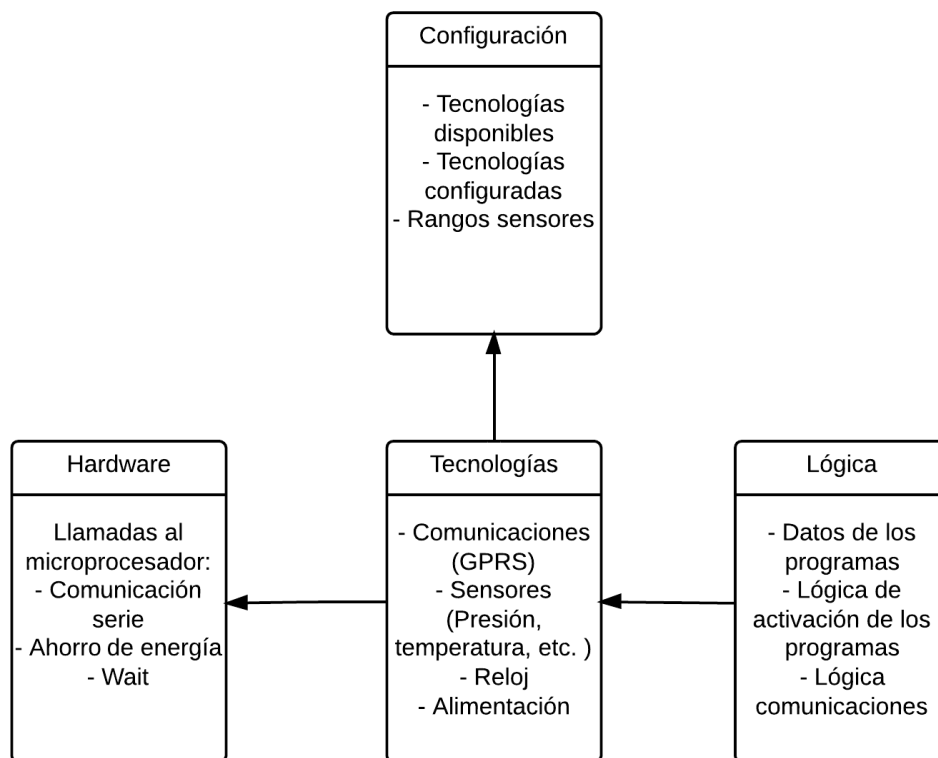


Figura 4.1: Capas de abstracción del hardware

4.3.1. Especificación programador

Como se ha definido en el apartado 4.1.1 el sistema debe estar diseñado con vistas a futuras ampliaciones o migraciones de hardware. Para ello, se ha realizado el diseño expuesto en la figura 4.1.

■ Lógica

Este bloque contendrá toda la lógica del programador junto el almacenamiento de datos. Será el bloque encargado de almacenar los datos de los programas, realizar la configuración inicial, hacer peticiones de apertura y cierre de válvulas, peticiones de comunicación, etc.

La lógica se comunica únicamente con el bloque de tecnologías. Además, este bloque, *no conoce* qué tecnologías se utilizan para realizar las acciones.

■ Tecnologías

El bloque de tecnologías se encarga de gestionar los periféricos del sistema. Se encargará así del funcionamiento de los mismos, estableciendo

configuraciones iniciales, tiempos de de comunicación, etc.

Este bloque se comunica con todos, recibiendo peticiones de la Lógica, consultando datos del bloque de Configuración y haciendo peticiones al Hardware para recoger los datos necesarios.

- **Configuración**

Este módulo almacena todas las tecnologías que están activas en el dispositivo además de sus configuraciones particulares.

Únicamente recibe peticiones del bloque de Tecnologías.

- **Hardware**

El módulo hardware es el de más bajo nivel y se encarga de gestionar todas las llamadas directas al hardware. Permite así que, solo modificando este módulo, podamos cambiar de microprocesador y sistema.

Cada programador se identifica mediante un número identificativo único de 32 bits el cual no podrá ser cambiado por el cliente. Este código permitirá identificar inequívocamente cada programador a la hora de realizar comunicaciones.

El número de programas viene limitado por los requisitos, pero la estructura de datos interna debe permitir ampliar este rango.

La interacción física con el programador se realiza mediante pulsadores que permiten navegar a lo largo del menú que se muestra en un pequeño LCD .

Las funcionalidades del programador, a las que se podrá acceder a través del menú anterior, serán las siguientes:

- **Mostrar mensajes**

En caso de existir mensajes de error o advertencia, se mostrará una lista de ellos.

- **Programas**

Muestra un submenú que permite listar todos los programas por categorías: Activos, Inactivos o Todos.

- **Añadir programa**

Esta opción permite añadir un nuevo programa. Menú a menú iremos introduciendo los datos. Una vez introducido, el programa se establecerá como activo.

- **Activar programa**

Muestra una lista de los programas inactivos permitiendo activarlos.

- **Desactivar programa**
Muestra una lista con los programas activos permitiendo desactivarlos.
- **Borrar programa**
Muestra una lista con todos los programas (activos e inactivos) permitiendo borrarlos. Esta acción será irrecuperable desde el programador.
- **Activar riego manual**
Permite el accionamiento de electro-válvulas manualmente por un periodo limitado de tiempo.
- **Desactivar riego manual**
Permite el cierre de las electro-válvulas activadas manualmente.

4.3.2. Especificación servidor

El servidor estará formado por una interfaz web, que permitirá acceder al usuario y una API, que gestionará el intercambio de información con los programadores.

Existen dos tipos de roles para realizar el acceso a la web: usuario y administrador. Ambos acceden desde la web principal con su usuario y contraseña.

Cuando un cliente se identifica en el sistema, se mostrará una lista los programadores contratados. Dicha lista se mostrará siguiendo un criterio de colores que permitirá identificar claramente aquellos en estado de error o advertencia. Cada programador se identifica por un alias definido por el usuario. La lista debe permitir acceder a cada programador.

Una vez accedido a un programador, se visualizarán todos los programas existentes en el programador. Esta lista podrá ser filtrada por programas activos, inactivos o en ejecución. Se mostrará también los mensajes, siguiendo el criterio de colores anteriores, que muestren los errores existentes.

Cada programa ofrecerá además la posibilidad de ser activado, desactivado o borrado. Además se debe permitir añadir nuevas programaciones, consultar el historial de programas borrados, consultar el histórico de riego o modificar la configuración del programador.

Si el usuario accede como administrador, podrá dar de alta nuevos usuarios, vincular nuevos programadores a un usuario o dar de baja aquellos que hayan dejado de utilizarse.

Un cliente deberá contactar con la empresa si desea realizar alguna de las siguientes acciones:

- Contratar un nuevo programador.
- Dar de alta un nuevo programador.

- Dar de baja un programador

4.3.3. Especificación comunicaciones

Las especificaciones para las comunicaciones deberán ser comunes a ambos, por lo que se ha definido un apartado diferente que ambos deben cumplir.

Las comunicaciones se efectuarán por el programador en un tiempo máximo de 120 minutos. Dicho periodo de tiempo es conocido también por el servidor, lo que permitirá detectar el posible fallo del programador en caso de no efectuarse la comunicación.

Las comunicaciones se realizan al dominio `http://api.regaigua.com`.

4.4. Planificación temporal

Es de vital importancia realizar una buena planificación temporal ya que nos permitirá calcular el tiempo necesario en horas para finalizar el proyecto. Gracias a la estimación temporal, podremos establecer una fecha de finalización del proyecto, que nos permitirá a su vez calcular el coste del proyecto (sección 4.5). Cuanto mayor sea el periodo de finalización del proyecto, mayores recursos tendremos que destinar a su desarrollo y por tanto mayor será el coste.

Para estimar el tiempo en que se realizan las tareas, se ha decidido utilizar la técnica del juicio de expertos. Esta técnica consiste en realizar una encuesta entre profesionales del sector, para que realicen una estimación sobre la duración temporal del proyecto. A dichos expertos se les facilitarán las tareas expuestas en las tareas que se van a realizar en el proyecto además de las especificaciones expuestas en el apartado 4.3. Cada experto deberá realizar, a partir de su experiencia con proyectos similares y sus conocimientos en la materia, la estimación del tiempo (pesimista, optimista y esperado). La valoración de cada experto puede ser ponderada según sus conocimientos en la materia.

4.4.1. Descomposición en tareas

En este apartado de tratará de dividir el proyecto en tareas. Una tarea trata de una serie de cometidos altamente relacionados. Estas tareas deben ser, además, independientes entre si.

Las tareas que definiremos tanto para el proyecto hardware como software serán las mismas pero duplicadas, por lo que podrán ser paralelizables entre

ellas a excepción de la tarea de pruebas dependientes del sistema que se explicará posteriormente.

Procedamos así a definir las tareas:

- **Inicio**

Es la etapa inicial del proyecto. En ella nos reuniremos con gente próxima al sector agrario. De estas reuniones, se asentará la idea inicial, definiendo los principales objetivos.

- **Estudio documental previo**

Se procederá al análisis de productos existentes en el mercado con los que poder cumplir las especificaciones.

También se procederá al estudio de tecnologías del marco tecnológico actual con las que desarrollar el sistema. Sobre cada una de ellas deberemos evaluar las ventajas e inconvenientes que presentan.

Una vez realizado dicho estudio, se deberán escoger aquellas tecnologías que encajan y favorecen más al desarrollo del proyecto.

- **Análisis del sistema**

Se trata de la fase previa al desarrollo. En esta tarea se tratará de definir los objetivos del proyecto así como sus funcionalidades. En esta fase deberá especificarse cuáles serán los elementos a desarrollar a lo largo del proyecto.

- **Desarrollo del sistema**

Tras un análisis del sistema y habiendo definido ya las especificaciones, se procederá a su desarrollo. Se trata de la fase más larga y costosa del proyecto. Como se ha explicado anteriormente, el desarrollo del hardware y software es independiente y por tanto totalmente paralelizable. El desarrollo de ninguna de las dos tecnologías deberá estar ligado a la otra por lo que se deberán crear sistemas de prueba si fuese necesario.

- **Pruebas y resultados del sistema**

En esta etapa se realizarán las pruebas para asegurar un correcto funcionamiento. Se deberá probar inicialmente los sistemas de forma independiente. Estas pruebas se realizarán mediante tests creados a medida que testeen las características de ambos.

Una vez realizadas dichas pruebas, se procederá a la unión del sistema y su instalación en un medio controlado. De esta forma, se evaluará su correcto funcionamiento en conjunto y sobre medios reales o cuasi reales. Las pruebas a realizar serán las de un funcionamiento normal además de forzar situaciones de posibles fallos.

Terminado el periodo de las pruebas anteriores, se instalará sobre un sistema real, donde se verificará diariamente su correcto funcionamiento y apoyará, si se considerase necesario, de un programador comercial previamente utilizado.

- **Documentación del proyecto**

Durante las tareas anteriores se deberá ir generando la documentación intrínseca a ella. En la etapa de desarrollo, se deberá generar la documentación en el código utilizando doxygen o tecnologías similares. En la etapa final de esta tarea, se deberá recopilar y estructurar toda la documentación generada para ser presentada a posteriori.

4.4.2. Descomposición en hitos

Ya definidas las tareas del proyecto, se procederá a definir las etapas del desarrollo del proyecto. Los hitos se definen con el objetivo de chequear si la planificación realizada se está cumpliendo conforme avanzamos en el proyecto. Gracias a ello, podremos detectar posibles desviaciones temporales.

Los hitos establecidos para el proyecto serán:

- **Admisión del análisis del sistema**

Se debe dar una vez finalizadas las tareas de estudio documental y análisis del sistema. Es el punto previo al comienzo del desarrollo del proyecto y donde se debe decidir sobre la viabilidad del mismo.

- **Admisión del desarrollo del sistema**

Esta etapa se da una vez finalizado el desarrollo del prototipo. El producto deberá estar finalizado, por lo que deberá cumplir con los objetivos y especificaciones definidos anteriormente. Se trata de una versión primaria que puede contener fallos.

- **Admisión de las pruebas**

Este hito se da una vez superados todos los tests, tanto independientes como en conjunto y sobre el terreno. El producto deberá encontrarse en estado finalizado a falta de la documentación.

- **Admisión de la documentación**

Una vez recopilada y estructurada toda la documentación del sistema se dará este hito.

Se tratará del último hito del proyecto y con él se procederá a la finalización y cierre del mismo.

4.4.3. Alcance del proyecto

Ya realizada la Descomposición en tareas podremos definir el alcance del proyecto.

El alcance de un proyecto trata de determinar el trabajo que debe realizarse para entregar el producto con todas las funciones y características [64, p 90].

El alcance del proyecto se definirá mediante su representación en el diagrama EDT³ mostrado en la figura 4.2.

³Estructura de Descomposición de Trabajo

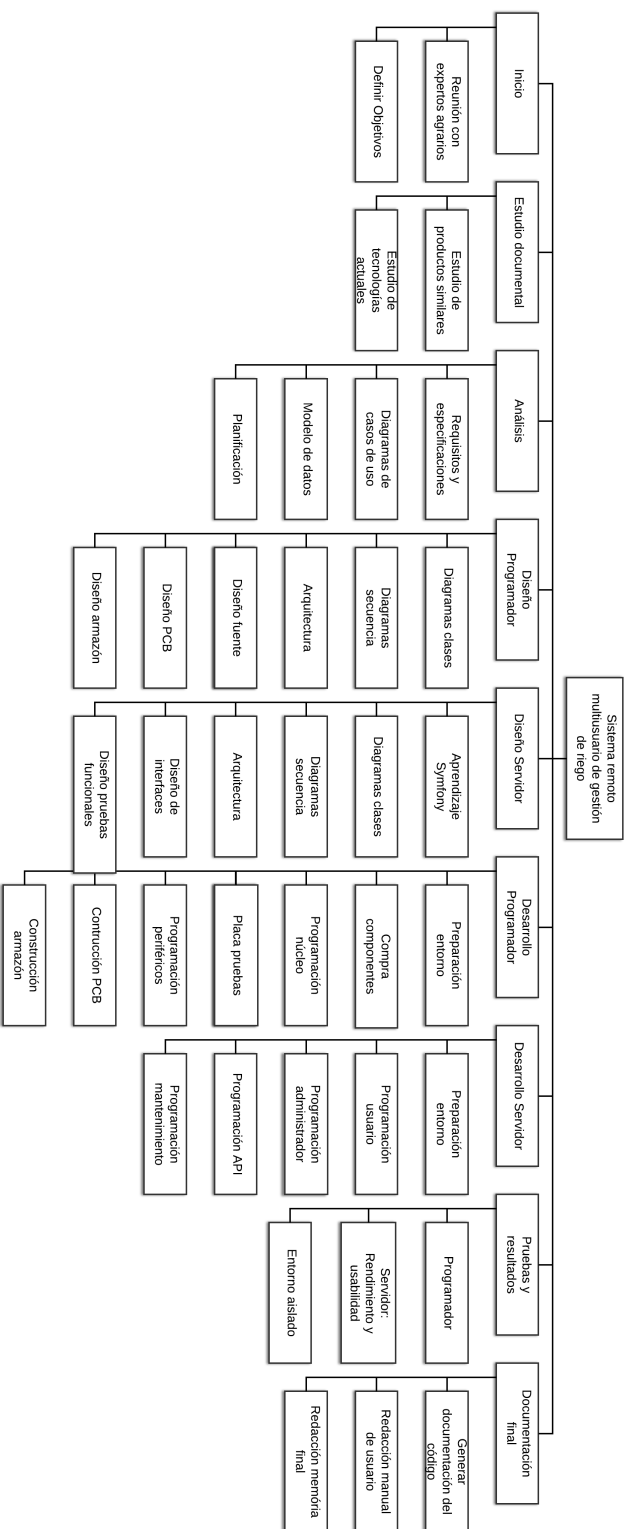


Figura 4.2: EDT del sistema

4.4.4. Juicio de expertos

Definido ya el alcance del proyecto, procederemos a realizar el juicio de expertos.

A continuación expondremos una breve biografía de cada uno de los expertos seleccionados:

- **Jose Antonio Vázquez:**

Ingeniero Industrial por la Universidad Politécnica de Valencia, ha desarrollado su actividad profesional a lo largo de más de 30 años, ejerciendo como director del Centro de Cálculo de la UPV, Centro de Cálculo de la Conselleria de Hacienda de la Generalitat Valenciana y como director del servicio de informática de la Universitat de València.

- **Julio Martos:**

Ingeniero en Electrónica (1997) y Doctor Ingeniero en Electrónica (2003) por la Universitat de València es actualmente profesor de la misma en el departamento de Electrónica. Forma parte del grupo de investigación de Diseño de Sistemas Digitales y Comunicaciones DSDC⁴ donde es especialista en sistemas de adquisición, redes de sensores inalámbricos, diseño de ultra bajo consumo y recolección de energía.

- **Rafael Martínez:**

Profesor titular en la Universidad de València en el Departamento de Informática, es actualmente director del Instituto Irtic y del grupo de investigación LSyM⁵. Con más de una década como director del grupo LSyM ha planificado y dirigido una gran cantidad de proyectos de un gran rango económico.

Dada la diferente experiencia de cada uno de los profesionales, se ha decidido ponderar de manera distinta sus calificaciones. En la figura 4.1 reflejamos cuáles serán los porcentajes aplicados a cada valoración.

Alias	Nombre	Peso
A	Jose Antonio Vázquez	40 %
B	Julio Martos	30 %
C	Rafael Martínez	30 %

Cuadro 4.1: Porcentaje de evaluación del juicio de expertos

⁴<http://dsdc.uv.es>

⁵<http://www.uv.es/lsym>

Los resultados de dicha encuesta se reflejan en la tabla 4.2 ya aplicadas las ponderaciones⁶. En ella además se ha reflejado el tiempo esperado real. Este es calculado mediante la fórmula 4.1, siendo a el tiempo optimista, b el pesimista y m el esperado.

$$te = \frac{a + 4m + b}{6} \quad (4.1)$$

Tarea	Estimado	Optimista	Pesimista	Tiempo medio
Inicio				
Reunión con expertos agrários	1	1	2	1,2
Definir objetivos	1	1	3	1,3
Estudio documental				
Revisión de productos similares	2	1	5	2,3
Estudio de tecnologías actuales	2	1	5	2,3
Análisis				
Requisitos y especificaciones	10	8	14	10,3
Diagramas de casos de uso	3	2	6	3,3
Modelo de datos	3	2	6	3,3
Planificación	5	4	8	5,3
Diseño programador				
Diagramas de clases	3	2	6	3,3
Diagramas de secuencia	3	2	6	3,3
Diseño de interfaces	1	1	3	1,3
Diseño de las pruebas	2	2	4	2,3
Diseño de la fuente de alimentación	14	10	19	14,2
Diseño de PCB	6	4	10	6,3
Diseño armazón	1	1	2	1,2
Diseño servidor				
Aprendizaje de Symfony	12	10	20	13,0
Diagramas de secuencia	5	4	7	5,2
Diseño de interfaces	1	1	2	1,2
Diseño de las pruebas funcionales	2	1	4	2,2
Desarrollo programador				
Compra de componentes	30	15	40	29,2
Preparación del entorno	2	1	3	2,0
Programación del núcleo	14	12	20	14,7
Programación de las comunicaciones	23	18	27	22,8

⁶Para visualizar los datos individuales con más precisión, consultar el CD adjunto

Construcción de placa de pruebas	2	1	4	2,2
Programación periféricos	8	7	10	8,2
Construcción PCB	3	2	5	3,2
Construcción armazón	1	1	2	1,2
Desarrollo servidor				
Preparación del entorno	1	1	2	1,2
Programación usuario	18	16	23	18,5
Programación administrador	12	10	16	12,3
Programación API	11	9	13	11,0
Programación mantenimiento	2	1	3	2,0
Pruebas y resultados				
Programador	5	4	6	5,0
Rendimiento y usabilidad servidor	5	4	6	5,0
Entorno real controlado	13	7	20	13,2
Documentación final				
Generar documentación del código	3	2	5	3,2
Redacción manual de usuario	6	5	8	6,2
Redacción memoria final	28	26	32	28,3
TOTAL	264	200	377	272.2

Cuadro 4.2: Resultados del juicio de expertos

4.4.5. Planificación de las tareas

Nos valdremos de un diagrama de Gantt que nos ayudará a representar gráficamente el tiempo dedicado a la realización de las actividades definidas en el apartado Descomposición en tareas. El diagrama de Gantt utiliza un sistema de coordenadas: en el eje vertical se indicarán las tareas que se extenderán a lo largo del eje horizontal, representando así su duración temporal. Para su representación será necesaria la realización previa de unos cálculos:

- **Camino crítico**

El camino crítico se define como *la cadena más larga de tareas basada solo en las dependencias de las tareas*. Esto nos permitiría la planificación de las tareas dada una fuente de recursos ilimitada o muy grande. Al no ser así deberemos calcular la cadena crítica.

- **Cadena crítica**

La cadena crítica es definida como *la cadena más larga de tareas que considera, tanto las dependencias entre las tareas como las dependencias con los recursos*. En nuestro caso la fuente de recursos es muy

limitada pues solo dispondremos del proyectando para realizar todas las tareas. Es por ello que la cadena crítica resultante será la concatenación de todas las tareas en el tiempo. El tiempo de realización del proyecto no será sino la suma de estos tiempos, pues vendrá dictado por la ley de los grandes números, que añade una incertidumbre dada la sucesión de muchas actividades de tiempo no definido.

■ Cálculos probabilísticos

Para la realización del diagrama nos valdremos de la tabla 4.2. Para obtener una planificación lo más fidedigna posible, realizaremos una aproximación estadística mediante el método de Monte Carlo. Dicho método consiste en simular muchas veces el sistema aplicando valores estadísticos hasta encontrar la convergencia. Para realizar dicha simulación, aplicaremos una distribución de tipo beta 4.3. Estas simulaciones se han realizado mediante el software RiskyProject ⁷.

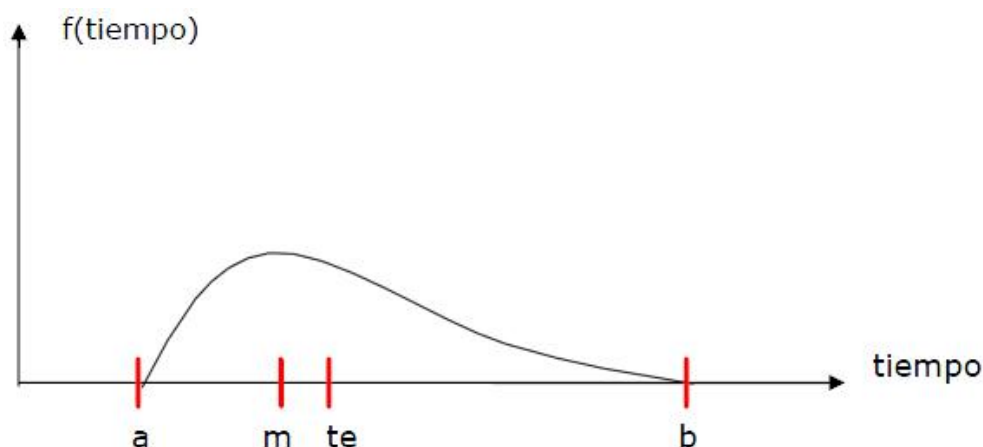


Figura 4.3: Distribución beta para cálculo de tiempo esperado

Tras la simulación será importante remarcar cierta información para realizar una correcta planificación:

■ Días de duración del proyecto:

En la figura 4.4 representamos la probabilidad de finalizar el proyecto si enlazamos todas las tareas una tras otra. Así pues, solo podemos afirmar con un 51 % de probabilidad que finalizaremos el proyecto en 259 días. Este porcentaje es muy bajo por lo que diremos que la estimación no es correcta.

⁷<http://www.intaver.com/riskyprojectprof.html>

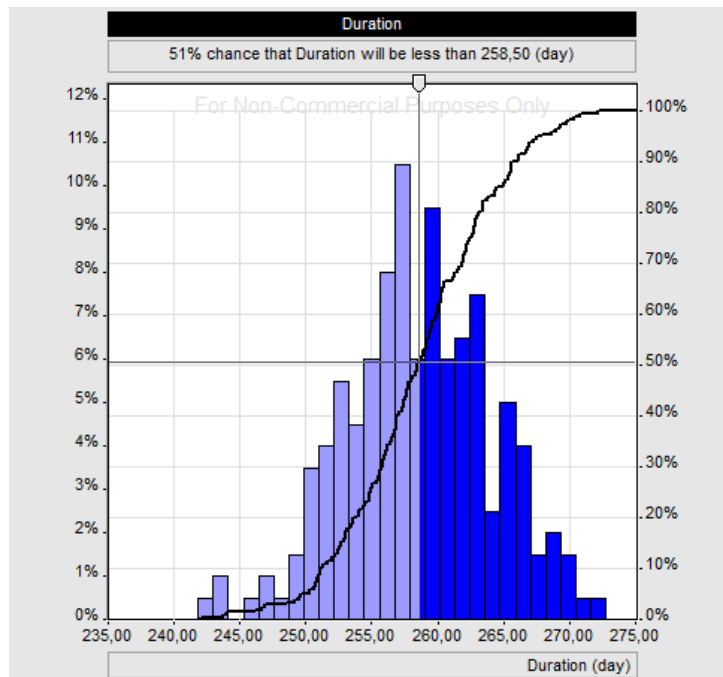


Figura 4.4: Probabilidad de finalizar el proyecto concatenando las tareas

En la figura 4.5 se representa el histograma de tiempos, en el que se estima el número de días necesarios para finalizar el proyecto con una probabilidad del 95 %. Este será el valor en relación al cual se estimará el tiempo de coste del proyecto. Podremos afirmar así con una alta probabilidad que los recursos humanos destinados al proyecto serán de 268 días.

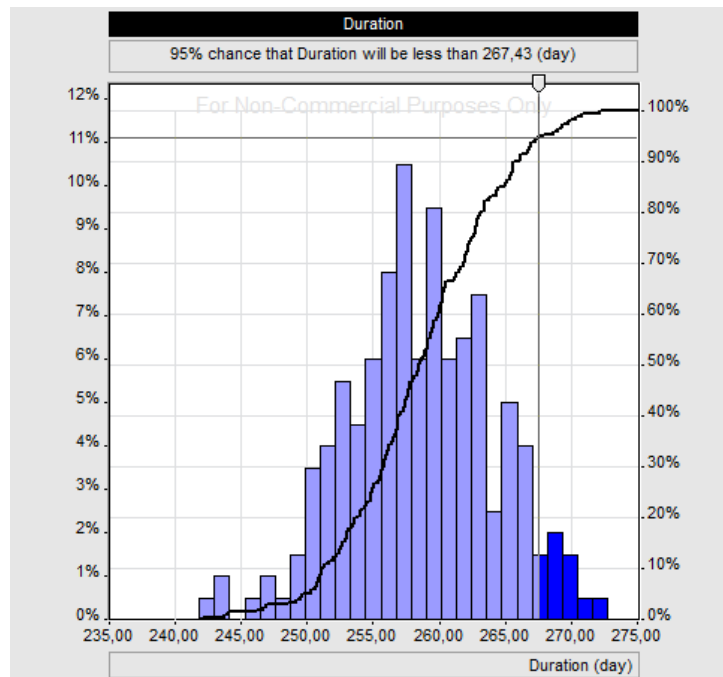


Figura 4.5: Días necesarios para una probabilidad del 95 %

La diferencia de días entre ambos casos se añadirá a la planificación como *buffer*. Dicho *buffer* se añadirá al final de la última tarea como colchón para posibles retrasos de las tareas. La ventaja de incluir solo un *buffer* en el proyecto y no uno individual a cada uno de los hitos, radica en que el síndrome del estudiante o ley Parkinson es mucho menor ya que el desarrollador de cada tarea es consciente de que el proyecto en global se está retrasando debido a que su tarea no se ha realizado en el tiempo estimado.

Así pues, en la figura 4.6 se estima la fecha de finalización del proyecto dado un calendario de trabajo de 8 horas diarias en 5 días a la semana.

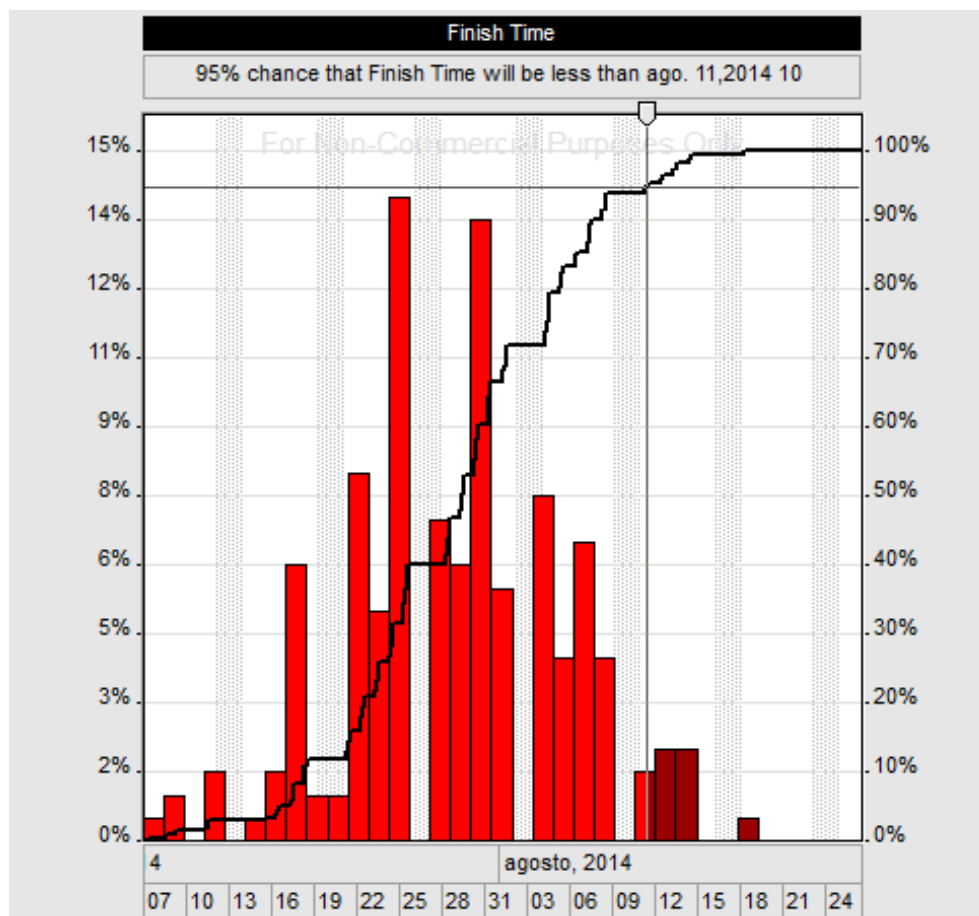


Figura 4.6: Fecha estimada para probabilidad del 95 %

En la figura 4.7 se observa el diagrama de Gantt resultante para la planificación de nuestro proyecto⁸

4.5. Estimación de costes

Ya obtenidos los costes temporales, procederemos a realizar los cálculos del coste económico. Estos vendrán dados por el tiempo total del ingeniero cobrando según la función que esté realizando en dicho momento, además del coste de las herramientas de trabajo así como del hardware necesario para realizar el proyecto.

⁸Para una mayor resolución, consultar el CD adjunto.

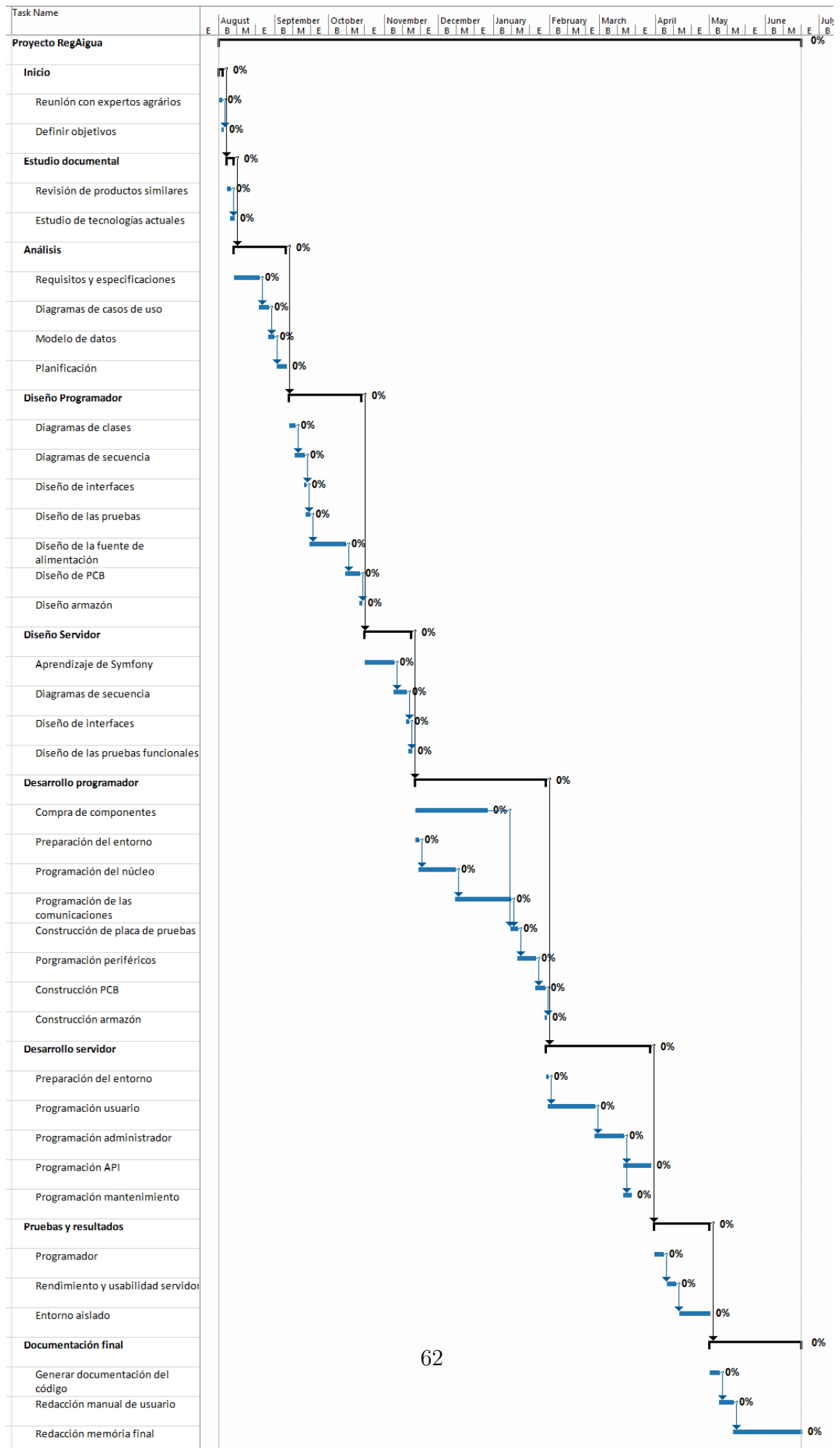


Figura 4.7: Diagrama de Gantt para la planificación del proyecto sin buffer

4.5.1. Costes de personal

Definiremos aquí el coste de los recursos humanos. Para realizar dicho cálculo, se deberá tener en cuenta el sueldo de un ingeniero informático a fecha actual.

En la tabla 4.3, se representa el sueldo como ingeniero Informático⁹. Se ha calculado además el sueldo por día si se cuenta con un total de 252 días laborales al año.

	Sueldo anual	Días laborables	Precio día
Ingeniero Informático	23.430,82 €	252	93 €

Cuadro 4.3: Sueldo Ingenieros Informáticos

Si aplicamos el coste por días calculado en la tabla 4.3 al tiempo total estimado, podremos obtener el valor económico destinado a los recursos humanos:

$$Costesrecursoshumanos = 93\text{€} * 268\text{días} = 24.924\text{€} \quad (4.2)$$

En la ecuación 4.2, se establece que el coste total asociado a los recursos humanos para desarrollar el proyecto será de 24.924€.

4.5.2. Costes software

En este punto se mostrará todo el software necesario para el desarrollo y ejecución del proyecto indicando además el precio de licencia del mismo. También se deberá reflejar la amortización que se ha realizado durante la duración del proyecto. La amortización en el software se supondrá de 3 años. El precio amortizado se calculará según la fórmula 4.3 para cada programa utilizado.

$$Amortizado = Precio \cdot \frac{Meses\ de\ uso}{Meses\ AmortizaciónTotal} \quad (4.3)$$

Dividiremos en dos tablas el software a utilizar durante el proyecto 4.4 en las que se listará aquel que ha sido utilizado durante todas las etapas de desarrollo del proyecto y 4.5 donde se mostrará el coste de licencias asociadas al servidor. En las tablas se mostrará, asimismo, la fecha de compra, el precio de licencia y el coste amortizado.

⁹El sueldo ha sido establecido conforme a la legislación presente en el Estado Español a fecha de publicación de esta memoria, expuestas en el BOE [4]

Software	Descripción	Licencia	Fecha compra	Precio	Amort.
Windows 8	Ordenador desarrollo	Estudiante	Septiembre 2012	0 €	0 €
PHPStorm	IDE desarrollo PHP	Privativa - pago por suscripción	Septiembre 2013	26 €	26 €
Internet Explorer	Navegador web	Microsoft	-	0 €	0 €
Chrome	Navegador web	Freeware	-	0 €	0 €
Firefox	Navegador web	MPL	-	0 €	0 €
Atmel Studio 6	IDE desarrollo Atmega	Freeware	-	0 €	0 €
MiKTeX	Compilador LaTeX	Freeware	-	0 €	0 €
SublimeText 2	Editor texto	Privativo	-	0 €	0 €
Visual Paradigm	Editor UML	Universidad	-	0 €	0 €
Lucid chart	Editor de diagramas online	Privativa - pago por suscripción	-	0 €	0 €
Kicad	EDA software. Edición de PCB.	GNU GPL v2	-	0 €	0 €
Subversion	Control de versiones utilizado durante el desarrollo	Apache license	-	0 €	0 €
Trello	Gestión de tareas pendientes	Freeware web	-	0 €	0 €
RiskyProject 6	Cálculo estadístico para gestión de proyectos	Privativa (versión prueba)	Enero 2014	0 €	0 €
Apiary	Desarrollo de API y documentación	Freeware web	-	0 €	0 €
TOTAL				26 €	26 €

Cuadro 4.4: Software utilizado en el desarrollo

Software	Descripción	Licencia	Fecha compra	Precio	Amort.
Ubuntu	SO Servidor	GPL	-	0€	0€
Apache	Servidor Web	Apache License 2.0	-	0€	0€
Symfony	Framework PHP	MIT	-	0€	0€
MariaDB	Base de datos servidor	GNU	-	0€	0€
Twitter Bootstrap	Framework para visualización del frontend	MIT	-	0€	0€
Sandstone	Tema para bootstrap	MIT	-	0€	0€
TOTAL				0€	0€

Cuadro 4.5: Software utilizado en el servicio

Si realizamos la suma del coste del software en el servidor y desarrollo obtendremos un total de 26€.

4.5.3. Coste hardware

En este apartado se listará todo el hardware utilizado para llevar a cabo el proyecto. En él se mostrará el hardware utilizado para desarrollar, los componentes electrónicos que constituyen el programador y el hardware que da soporte al servidor. De nuevo se calculará el precio amortizado mediante la fórmula 4.3 estableciendo en este caso, de acuerdo al Real Decreto 1777/2004 [2], un periodo de amortización de 4 años.

Además de la diferenciación entre programador y servidor llevada a cabo, para determinar el precio del hardware, será interesante diferenciar entre la etapa de desarrollo y producción.

4.5.3.1. Coste hardware en desarrollo

Durante la fase de desarrollo del producto, es natural que el coste asociado al programador sea mayor, pues se suele recurrir a módulos externos ya ensamblados, además de que la compra es individual. En caso del coste asociado al servidor, durante la fase de desarrollo el coste será menor ya que el volumen de peticiones será reducido.

En la tabla 4.6 se muestra el hardware que se ha utilizado para realizar tanto la programación como la documentación durante todo el proyecto, el

ordenador utilizado para ser usado como servidor y el hardware utilizado para desarrollar el programador.

Dado que el equipo de desarrollo fue adquirido en Julio de 2009 ya ha sido totalmente amortizado, por lo que no supondrá ningún coste para el proyecto.

Hardware	Descripción	Fecha compra	Precio	Amort
AMD Phenom II 3GHz, 8GBB RAM DDR2, 500GB HDD	Equipo desarrollo	Julio 2009	635.70 €	0 €
Asus E45M1-M PRO, 16GB RAM, SSD 128GB, HDD 3TB	Equipo Servidor	Febrero 2013	486.42 €	121 €
Estación de soldadura	ST 201	Abril 2008	90 €	0 €
Fuente alimentación	Fuente de alimentación DC Netzgerät NG-1620BL	Septiembre 2013	50 €	21.5 €
Osciloscopio	Rigol DS1052E	Septiembre 2013	312 €	78 €
Herramienta multi-función	Dremel 395	Septiembre 2008	130 €	0 €
TOTAL			1704.12 €	220.5 €

Cuadro 4.6: Hardware utilizado durante el desarrollo del proyecto y servidor.

En la tabla 4.7 se especifica el material adquirido para la construcción del prototipo y realización de pruebas.

Hardware	Descripción	Precio unidad	Cantidad
Arduino Uno	Placa de desarrollo	20 €	1
ATmega328	Procesador suelto	4.4 €	1
LCD	I2c	5.74 €	1
Módulo GPRS	Módulo GPRS con chip hilo	93.12 €	1
Módulo RTC	Con conexión I2C	2.19 €	1
Módulo SD	Con regulador incorporado	1.81 €	1
Hidrogenerador	Con batería incorporada	20.46 €	1
Hidrogenerador	Sin batería incorporada	11.4 €	1
Bateria	LiPo 1000mAh	5.04 €	1

PCB	Insolable positiva 1 cara	5.3 €	1
PCB	Insolable positiva 2 caras	19.93 €	2
Sensor presión	De alta precisión	25.7 €	1
Componentes electrónicos	Resistencias, Condensadores, etc.	23 €	1
Electroválvulas	de dos cables	43 €	2
Caja estanca	Para construcción del prototipo	5 €	1
TOTAL		369.62 €	

Cuadro 4.7: Coste del hardware utilizado para la construcción del prototipo del programador.

4.5.3.2. Coste hardware en producción

No es posible estimar el coste en producción del hardware del programador ya que el producto ha sido diseñado como prototipo. Esto implica que los componentes utilizados son, en su mayoría, de tipo *through-hole*, que facilitan la construcción de prototipos pero encarecen el producto para tiradas en producción (ya que son más grandes y utilizan las dos capas de la pcb). Además, para poder minimizar el coste de las placas, se deberán redistribuir los componentes para minimizar el espacio ocupado y así reducir su coste. También se deberá integrar en la placa principal periféricos como el reloj RTC o la placa GPRS (la cual reduciría notablemente el coste del producto).

Respecto el servidor en producción, se contratará un servidor virtual dedicado a ello. A modo de estimación, se ha escogido la empresa Linode¹⁰, las características hardware y su coste serán las mostradas en la tabla 4.8. Gracias a que está alojada en una máquina virtual, podremos cambiar el plan dinámicamente si fuese necesario.

Hardware	Descripción
RAM	2GB
CPU	2 core
Disco duro	48GB SSD
Transferencia entrada	40Gbps
Transferencia salida	250 Mbps
TOTAL(mensual)	14.67 €

Cuadro 4.8: Coste del servidor en producción

¹⁰<https://www.linode.com>

4.5.4. Resumen de costes

El coste económico total del proyecto será la suma del total obtenido en cada uno de los apartados anteriores. Como se representa en la tabla 4.9, el mayor coste del proyecto viene dado por la remuneración de las horas de trabajo, suponiendo más de un 97 % por ciento del coste. Al total de esta suma, se le deberá añadir el *overhead*, es decir, los gastos indirectos que no se pueden cuantificar directamente. El *overhead* estimado para dicho proyecto es de un 20 % de los costes directos.

Tipo	Coste
Recursos humanos	24.924 €
Software	26 €
Hardware	220,5+369,62 €
Total (directo)	25.540,12 €
Total (<i>overhead</i>)	30.648,14 €

Cuadro 4.9: Resumen del coste económico total estimado del proyecto

4.6. Estudio de viabilidad

Este apartado es de crucial importancia pues nos indicará si es posible, viable, llevar a cabo el proyecto. Solo se llevará a término el proyecto si las conclusiones de este apartado resultan favorables. En este apartado estudiaremos la viabilidad económica, técnica y legal.

4.6.1. Viabilidad económica

En este apartado se realizará el estudio de la viabilidad económica o de costes. En él se evaluará si es posible hacer frente a los costes obtenidos en el apartado Estimación de costes.

La financiación se afrontará mediante el modelo de *Family, Friends and Fools* ¹¹. En todo momento se tratará de evitar recurrir a la financiación de agencias privadas.

¹¹Este modelo trata de captar la financiación necesaria para llevar a cabo el proyecto mediante donaciones o préstamos realizados por los familiares, los amigos y *locos*. Cabe mencionar que los préstamos realizados por los familiares generalmente resultan a fondo perdido mientras que aquellos realizados tanto por los amigos como por los *fools* son devueltos como participaciones en la empresa. [73]

Será necesario analizar la viabilidad económica desde distintas perspectivas¹²:

■ Realización de prototipo

El proyectando deberá conseguir la suficiente financiación para afrontar la primera fase de el proyecto: la realización del primer prototipo.

En este primer apartado se tendrán en cuenta el coste del hardware y software, además del *overhead* de los mismos. En la tabla 4.10 se muestra el total de capital necesario para afrontar esta primera fase.

Tipo	Coste
Software	26 €
Hardware	220,5+369,62 €
Total (overhead)	708,144 €

Cuadro 4.10: Resumen del coste económico necesario para afrontar la primera fase del proyecto

■ Entrada en producción

La entrada en producción representa al capital necesario para producir un número significativo de réplicas del producto listas para vender, afrontando previamente una fase de rediseño. En nuestro caso, el volumen de la primera tirada del producto será de 50 unidades.

Para que resulte viable esta fase, se deberá contar con un capital igual o superior al expuesto en la tabla 4.11. Donde se estima que el coste unitario del hardware del producto se encontrará por debajo de los 50 €.

Tipo	Coste
Hardware programador (50ud)	50 €*50
Ampliación servidor (anual)	176 €
Total (overhead)	3.211,2 €

Cuadro 4.11: Resumen del coste económico necesario para la entrada en producción

■ Modelo de negocio

Para que el producto resulte económicamente viable, se deberá encontrar un modelo de negocio que permita generar ingresos con el producto.

¹²Todos los datos referentes a costes, así como la fórmulas utilizadas que se mostrarán en los siguientes puntos, han sido extraídos del apartado anterior 4.5.

El modelo de negocio del que se valdrá el proyecto será el de alquiler del producto: cuando un cliente contrata el servicio, no adquirirá el hardware ni una licencia vitalicia sino que deberá abonar periódicamente una cantidad que le facilitará el acceso al mismo. Esto permitirá al cliente evitar realizar un gran desembolso económico inicial además de tener acceso a las actualizaciones más recientes del producto.

■ **Amortización en producción**

Otro de los requisitos para que proyecto resulte viable económicamente es que el producto resulte competitivo en el mercado. Para que así sea, el precio del producto no deberá resultar caro, pues el cliente potencial procede de una economía débil, pero sin olvidar que debe generar ingresos.

Este apartado estará ligado a los dos anteriores. Para calcular el coste de cada unidad en ejecución, primero deberemos establecer un tiempo máximo de amortización del hardware. Este tiempo ha sido estimado en 12 meses.

Para realizar este cálculo primero deberemos obtener el precio aproximado por programador. Este viene dado por la ecuación 4.4.

$$Coste_{unidad} = \frac{\text{Precio total}}{\text{Número programadores}} = \frac{3.211,2\text{€}}{50} = 64,22\text{€} \quad (4.4)$$

Como el modelo de negocio es por alquiler, deberemos calcular la amortización mensual del programador a la que le deberemos añadir el coste de línea GSM. Este último se ha extraído tras la consulta de precios como empresa a Movistar.

$$Ud.mensual = \frac{\text{total unidad}}{1 \text{ año}} + \text{línea móvil} = \frac{64,22\text{€}}{12} + 2,99\text{€} \approx 8\text{€} \quad (4.5)$$

Así pues deberíamos establecer un precio mensual mayor a (€10) para generar beneficios.

■ **Recuperación del capital**

Finalmente, para que el proyecto resulte viable económicamente, se debe prever una recuperación del capital invertido¹³ a corto o medio plazo.

¹³Incluyendo el coste en recursos humanos del propio educando.

Dados los datos ofrecidos en los apartados anteriores y conocida la gran aceptación que podría tener entre los potenciales clientes ¹⁴, diremos que será posible una entrada en el mercado del producto que nos permita recuperar este capital incluso generar ingresos en un futuro.

4.6.2. Viabilidad técnica

En este apartado estudiaremos si es posible y viable, mediante la tecnología actual, llevar a cabo el proyecto. También se tratará la viabilidad según el conocimiento que tiene el equipo de desarrollo sobre las tecnologías con las que se va a tratar, así como la documentación existente de la misma.

Pese a que el hardware del proyecto está creado por nosotros mismos, la mayoría de sus bloques están formados por tecnología de carácter comercial y extendida, de modo que no se debería tener ningún problema a la hora de realizar el proyecto.

Por lo que respecta al conocimiento de las tecnologías, a excepción de una, todas ellas son conocidas por el equipo de desarrollo, el cual, pese a que se deberá programar APIs que gestionen los módulos a bajo nivel, será capaz de realizarse.

Así pues, el proyecto resultará viable técnicamente por lo que no resultará un obstáculo para su realización.

4.6.3. Viabilidad legal

Dado que en un futuro el proyecto podría ser comercializado, deberemos analizar la viabilidad desde un punto legal. Para llevar a cabo dicho análisis, deberemos contemplar todas aquellas posibles infracciones, responsabilidades legales, etc. a las cuales debamos estar obligados según la legislación española y europea.

4.6.3.1. Ley de protección de datos

Para la realización de nuestro proyecto, deberemos poner especial hincapié en cumplir la ley de protección de datos, por lo que deberemos analizar si los datos almacenados sobre los clientes son sensibles y pueden verse afectados por dicha ley. Todo el proyecto debe cumplir con la normativa vigente de la LOPD¹⁵.

La LOPD obliga a regular los datos almacenados de carácter personal referentes a las personas físicas. La ley define tres niveles distintos de datos

¹⁴Conocida gracias a la proximidad al sector.

¹⁵Ley Orgánica de Protección de Datos [1].

que requerirán unas medidas a aplicar distintas según el nivel de privacidad que requieran¹⁶:

- Alto
Comprende datos de carácter muy personal como ideología, religión, salud o vida sexual. También requerirán un nivel alto de seguridad los datos recabados con fines policiales sin consentimiento de las personas afectadas y derivados de actos de violencia de género.
- Medio
Se requerirá un nivel de seguridad medio para aquellos datos relacionados con administraciones tributarias, mutuas de accidentes de trabajo, enfermedades profesionales, definición de la personalidad en los que se permita evaluar los aspectos o comportamiento de la persona y en operadores de comunicaciones electrónicas, respecto de los datos de tráfico y localización.
- Básico
Engloba aquellos datos que almacenan información de carácter personal. También los recogidos en el apartado de nivel alto cuando se trate de ficheros no automatizados.

Nuestro proyecto se encuentra englobado dentro de un nivel básico pues solo almacenaremos la información asociada a una persona que permita identificar al usuario. Además, el proyecto estará separado en dos bases de datos, aquella en la que están recogidos los datos de los clientes y aquella que almacena datos relativos a los programadores.

Para proteger los datos referentes a los clientes, nivel básico, deberemos adoptar las siguientes acciones:

- Registro de la base de datos en la AEPD.
- Desarrollo de documento de seguridad: en el que se reflejarán todas las personas que pueden acceder a los datos y sus perfiles.
- Informar a los usuarios: El usuario deberá ser informado de que sus datos van a ser almacenados durante el proceso de registro.
- Control de acceso: Deberemos limitar el acceso a los mismos mediante una identificación previa.

¹⁶La lista de datos que se muestran a continuación no está completa. Para obtener la lista completa, consultar la guía de seguridad de datos ofrecida por la Agencia Española de Protección de Datos [20]

- Protección: Los datos personales almacenados en la base de datos deben permanecer encriptados evitando su posible acceso a personas no autorizadas.

Dado que se trata de un proyecto académico, durante la fase de desarrollo nos limitaremos a cumplir exclusivamente con aquellas normas que estén a nuestro alcance, como la incorporación de los datos del cliente, aplicar un control de acceso e informar al usuario durante el periodo de registro. Quedará como trabajo futuro el registro de la base de datos así como el desarrollo del documento de seguridad.

Respecto al uso de *cookies* en la web, se deberá notificar al usuario del uso de las mismas desde el primer momento en el que accede a la web.

4.6.3.2. Uso del espectro de radiofrecuencia

Otro punto sensible en el proyecto puede ser el de las comunicaciones. Estas deben estar dentro del marco de las leyes tanto españolas como europeas. En este caso, aunque algunas de las tecnologías estudiadas en el capítulo de estado del arte se verían afectadas por las mismas, para la realización del proyecto se ha decidido externalizar las comunicaciones además de adquirir hardware ya aprobado por su uso en la Unión Europea.

4.6.3.3. Legalidad en el software utilizado

Para determinar la viabilidad de un proyecto también se deberán revisar el tipo de licencias del software utilizado en el desarrollo del proyecto.

A lo largo de este subapartado se tratará de estudiar que las licencias del software utilizado permiten el uso comercial del producto. Para realizar dicho estudio del software utilizado, analizaremos si las licencias del software listado en el apartado de Costes software (Página 63) permite su uso comercial (tabla 4.12), entrando en detalle en caso de que se trate una licencia de tipo estudiante o *freeware* (tabla 4.13).

Licencia	Comentarios	Uso comercial
GNU GPL v2	No permite añadir restricciones adicionales sobre su uso. Se deberá publicar el código en caso de hacer modificaciones sobre la fuente original.	✓

Apache License 2.0	En este tipo de licencia no se diferencia entre uso personal, interno o comercial.	✓
MIT	Se caracteriza por ser una de las licencias con menos restricciones.	✓

Cuadro 4.12: Análisis de tipo licencias software

Software	Licencia	Comentarios	Uso comercial
Windows 7	Privativa - Estudiante	Esta licencia es proporcionada por Microsoft para todos los estudiantes superiores a través de su plataforma DreamSpark ¹⁷ su uso estará vigente hasta finalizar los mismos. No permite su uso comercial. [49]	✗
MikTex	<i>Freeware</i>	Esta licencia permite el uso del producto en proyectos de carácter comercial. [52]	✓
PHPStorm	Privativa - Estudiante	Esta licencia no permite el desarrollo de carácter comercial, por lo que solo sería válida en esta primera fase del proyecto. En caso de comercialización debería adquirirse una licencia de carácter personal. [61]	✗

¹⁷<https://www.dreamspark.com/>

Atmel Studio	Privativa	Se permite su uso con fines comerciales. [10]	✓
Visual Paradigm	Privativa - Estudiante	Esta licencia especial para centros educativos no permite su uso en proyectos comerciales [60]	✗
Lucidchart	<i>Freeware</i> Web - Pago por suscripción	Se permite el uso del producto en proyectos con fines comerciales en cualquiera de sus versiones. [43]	✓
Trello	<i>Freeware</i> Web	Se permite su uso en proyectos comerciales. [76]	✓

Cuadro 4.13: Análisis de licencias software específicas

Como hemos podido observar en la tabla 4.13, algunas de las licencias utilizadas solo serán válidas para el uso en proyectos no comerciales de carácter educativo. Estas licencias deberían ser revisadas en caso de comercialización del producto, lo que podría acarrear un coste adicional en el proceso de desarrollo.

4.6.3.4. Conclusión

Podemos concluir así que, aunque quedan puntos pendientes en un futuro, el proyecto puede coexistir con la legalidad vigente.

4.6.4. Análisis de riesgos

En este apartado trataremos de anticiparnos a posibles problemas que puedan surgir durante la realización del proyecto. Deberemos prever los riesgos existentes para poder valorar la viabilidad del proyecto, analizando cuál es la probabilidad de que ocurra, cuál sería el impacto, qué deberemos hacer para evitar que ocurra y cuál será el plan de contingencia si ocurriese.

En la tabla 4.14 se presentan los resultados de dicho análisis.

Riesgo	Prob	Impa	Prevención y contingencia
Incremento de la curva de aprendizaje	Baja	Medio	Aunque el equipo de desarrollo conoce la mayoría de tecnologías a utilizar, no ocurre así con el framework Symfony. Si se contemplara que el tiempo de aprendizaje está aumentando mucho, se puede cambiar por otra tecnología que si se conozca y que permita cumplir con las especificaciones.
Avería en el hardware de desarrollo	Baja	Medio	Tanto el código como la memoria estarán almacenados en el equipo local como en la nube además del svn almacenado en el servidor. Esto permitirá la recuperación de todos los datos. El hardware debería ser reemplazado incrementando el precio final del proyecto.
Avería en el hardware servidor	Baja	Baja	Se deberán realizar copias automáticas de la máquina virtual en una unidad externa al servidor, permitiendo restaurarla en el ordenador de desarrollo minimizando la pérdida de tiempo. Dado que el hardware se encuentra en garantía, es altamente probable que no supusiese un coste económico para el proyecto.

Continúa en la página siguiente...

Cuadro 4.14 – Continúa en la página anterior

Riesgo	Prob	Impa	Prevención y contingencia
Retraso en la ejecución del proyecto	Media	Medio	Durante la ejecución de cada fase se deberá establecer una fecha de finalización. En caso de producirse un retraso, el equipo de desarrollo deberá invertir más horas diarias en él. Esto producirá un aumento en el coste de los recursos humanos, pero estos no se verán afectados dado que se trata de un proyecto final de carrera.
Avería en el hardware del programador	Media	Medio	Aquel material que no sea excesivamente caro deberá estar duplicado, reduciendo así el impacto temporal.
El tiempo de carga de las baterías no es suficiente	Media	Alta	De darse el caso, deberemos incluir dos o más hidrogeneradores que permitan una menor descarga de sus baterías. Como plan futuro, se debería crear un hidrogenerador más potente.

Cuadro 4.14: Posibles riesgos técnicos durante el proyecto

4.6.5. Conclusión

Como hemos podido observar a lo largo de los subapartados anteriores, existen algunos riesgos a la hora de realizar el proyecto, aunque estos resultarán mínimos, por lo que se considera que el proyecto es viable y podrá ser realizado por el equipo de desarrollo.

Capítulo 5

Desarrollo del proyecto

En este capítulo trataremos de hacer el diseño del proyecto. Este capítulo estará dividido en tres subapartados: Análisis, Diseño e Implementación.

5.1. Análisis

Este primer apartado es en el que se basarán los dos siguientes, y quizás es por ello por lo que resulta uno de los más importantes. Realizar un buen análisis del sistema resultará crucial para poder realizar correctamente los siguientes apartados.

Durante el proceso análisis, trataremos de pulir los requisitos y se establecerán los diferentes roles del sistema así como todos los casos de uso que pudiesen ejercer. Durante este proceso se deben producir distintas reuniones con los profesionales del sector y agricultores pues su opinión es fundamental ya que serán ellos los usuarios finales.

5.1.1. Usuarios del sistema

En este apartado detallaremos los roles de usuario que podrán interactuar con el sistema. Dado que se trata de dos sistemas diferentes, cabe diferenciar de los roles existentes en el programador físico y aquellos que lo hacen en el servidor.

Roles en el programador:

- **Cliente físico**

Comprende al usuario final del producto cuando realiza la interacción con el programador de manera física. Este se encargará de la gestión

de los programas y electro-válvulas desde el campo además de la monitorización actual del sistema. El cliente físico solo permite la gestión del programador con el que se está interaccionando físicamente.

- **Tiempo del programador**

Este rol se encarga de gestionar los eventos temporales como lo activación de electro-válvulas, comunicaciones, comprobación de sensores y activación de errores en el programador.

Roles existentes en el servidor:

- **Usuario anónimo**

Comprende a todo usuario que interactúa con la web sin haber realizado un proceso de autenticación. Este rol permitirá las acciones de identificación en la página así como el alta en el servicio.

- **Cliente Web**

Trata al usuario final del producto cuando está realizando la gestión de los programadores físicamente. Una vez identificado en la interfaz, podrá gestionar y monitorizar todos los programadores desde una única cuenta.

- **Administrador del servicio**

Este rol comprende a aquellas personas que realizan la gestión del producto. Como gestión del producto entenderemos la realización de altas y bajas de programadores, control de pagos, etc. Estos accederán mediante la misma interfaz web que el Cliente web.

- **Administrador de sistemas**

El administrador de sistemas se encargará de la gestión del servidor. Es el responsable de que el producto esté siempre operativo, solución de bugs, aplicación de nuevas versiones además de la gestión de las copias de seguridad. El administrador de sistemas accederá directamente al servidor mediante una cuenta ssh.

- **API**

Este rol se ocupa de la interacción de los programadores con el servidor. Mediante este podremos sincronizar los datos del servidor con el programador.

- **Tiempo del sistema**

Gestiona los tiempos de conexión para detectar posibles fallos en los programadores.

5.1.2. Análisis de casos de uso

En este apartado definiremos los casos de uso existentes en el sistema. Un caso de uso trata de *describir en forma de acciones y reacciones el comportamiento del sistema, estudiado desde el punto de vista del usuario* [22]. Así pues, procederemos al análisis de los casos de uso a partir de cada rol existente en el sistema. Para cada rol se mostrará el diagrama de casos de uso asociados y la descripción para cada uno de ellos.

Cliente físico:

Visual Paradigm for UML Standard Edition (Universitat de Valencia)

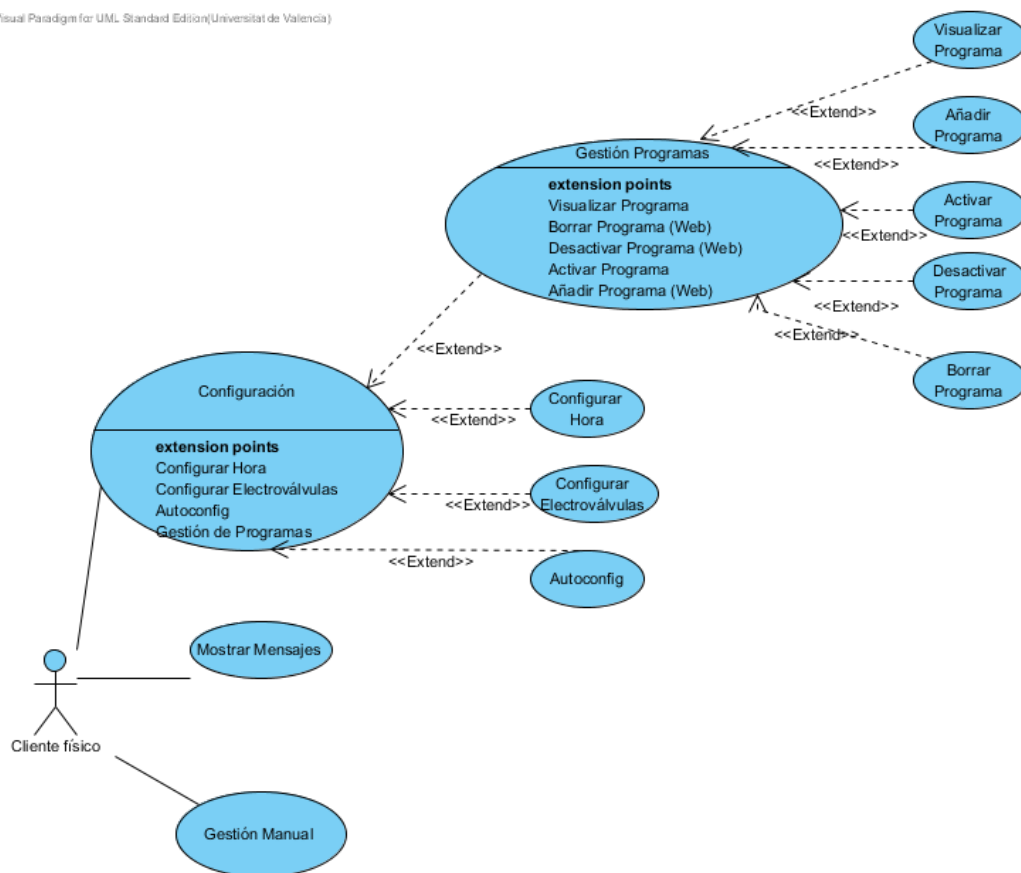


Figura 5.1: Diagrama de caso de uso para un cliente físico.

Configuración

Actor	Cliente físico
Descripción	Permite al usuario acceder a todas las configuraciones del programador
Precondición	Ninguna

Flujo normal	A través de los pulsadores físicos, el usuario podrá navegar entre las distintas opciones de configuración, seleccionado la deseada.
Flujo alternativo	El usuario pulsa el botón de volver, donde accede de nuevo al menú principal.

Cuadro 5.1: Caso de uso Configuración

Gestión de programas

Actor	Cliente físico
Descripción	Muestra al cliente un menú que engloba todas las posibles configuraciones a realizar en los programas
Precondición	Ninguna
Flujo Normal	El usuario navega mediante los pulsadores a la acción que desea realizar y pulsa ok para acceder a la misma.
Flujo Alternativo	El usuario pulsa el botón de volver, donde accede de nuevo al menú de configuración.

Cuadro 5.2: Caso de uso gestión de programas

Visualizar programas

Actor	Cliente físico
Descripción	Permite al cliente visualizar todos los programas existentes en el programador.
Precondición	Ninguna.
Flujo normal	El usuario navega entre los programas existentes mediante los pulsadores físicos del programador y acciona volver cuando desee finalizar el proceso.
Flujo alternativo	Si no existe ningún programador, se debe mostrar un mensaje de notificación.

Cuadro 5.3: Caso de uso visualizar programas

Añadir programa

Actor	Cliente físico
Descripción	Permite al cliente añadir un programa de riego o abonado.
Precondición	Existe al menos una electro-válvula configurada en el sistema.
Postcondición	Se ha añadido un nuevo programa activo en el que la válvula a accionar no coincide con la de otro programa. Si la válvula a accionar es de abonado, está contenida dentro de un programa de riego.
Flujo normal	El usuario introduce todos los datos solicitados mediante los pulsadores físicos.
Flujo alternativo	■ El usuario pulsa retroceder: se navegará hacia atrás entre las opciones pedidas hasta llegar al menú de configuración. ■ Ya existe un programa que acciona esa electro-válvula durante el mismo periodo de tiempo: se debe mostrar un mensaje de error. Se añade el programa como inactivo. ■ Se ha intentado configurar un programa de abonado que no está contenido en uno de riego: se mostrará un mensaje de error. Se añade el programa como inactivo.

Cuadro 5.4: Caso de uso añadir programa

Activar programa

Actor	Cliente físico
Descripción	Muestra al usuario una lista de programas inactivos, permitiendo navegar entre ellos activando aquellos que desee.
Precondición	Ninguna.
Postcondición	Se ha añadido el programa a la lista de programas activos en el que la válvula a accionar no coincide con la de otro programa. Si la válvula a accionar es de abonado, está contenida dentro de un programa de riego.
Flujo normal	El usuario selecciona el programa que desea activar y pulsa el botón de ok para finalizar la operación.
Flujo alternativo	■ El usuario acciona el botón de volver: se regresará al caso de uso de gestión de programas (caso de uso 5.2) ■ No existe ningún programa inactivo: se muestra un mensaje de advertencia. ■ Ya existe un programa que acciona esa electro-válvula durante el mismo periodo de tiempo: Se debe mostrar un mensaje de error. Se añade el programa como inactivo. ■ Se ha intentado configurar un programa de abonado que no está contenido en uno de riego: se mostrará un mensaje de error. Se añade el programa como inactivo.

Cuadro 5.5: Caso de uso activar programa

Desactivar programa

Actor	Cliente físico
Descripción	Muestra al usuario una lista de programas activos, permitiendo navegar entre ellos desactivando aquellos que desee.
Precondición	Ninguna
Postcondición	El programa seleccionado pasa a la lista de inactivos.
Flujo normal	El usuario selecciona el programa que desea desactivar y pulsa el botón de ok para finalizar la operación.
Flujo alternativo	■ El usuario acciona el botón de volver: se regresará al caso de uso de gestión de programas (caso de uso 5.2) ■ No existe ningún programa inactivo: se muestra un mensaje de advertencia. ■ El programa a desactivar es de riego y contiene a otro de abonado: tanto el programa de riego como el de abonado pasarán a la lista de inactivos. Se mostrará una notificación por pantalla.

Cuadro 5.6: Caso de uso desactivar programa

Borrar programa

Actor	Cliente físico
Descripción	Muestra al usuario una lista de programas, permitiendo navegar entre ellos borrar aquellos que desee.
Precondición	Ninguna.
Postcondición	El programa seleccionado ha sido eliminado del sistema.

Flujo normal	El usuario selecciona el programa que desea borrar y pulsa el botón de ok para finalizar la operación.
Flujo alternativo	■ El usuario acciona el botón de volver: se regresará al caso de uso de gestión de programas (caso de uso 5.2) ■ No existe ningún programa: se muestra un mensaje de advertencia. ■ El programa a eliminar es de riego y contiene a otro de abonado: se mostrará una notificación por pantalla.

Cuadro 5.7: Caso de uso borrar programa

Configuración hora

Actor	Cliente físico
Descripción	Permite al usuario configurar la hora del programador manualmente.
Precondición	Ninguna.
Postcondición	Se ha establecido la nueva hora en el programador.
Flujo normal	El usuario introduce los datos horarios según se le van solicitando mediante los pulsadores físicos.
Flujo alternativo	El usuario acciona el botón de volver: se regresará al caso de uso de gestión de programas (caso de uso 5.1)

Cuadro 5.8: Caso de uso configuración hora

Configurar electroválvula

Actor	Cliente físico
Descripción	Muestra una lista de todas las salidas de electro-válvulas, permitiendo configurarlas.
Precondición	Ninguna.

Postcondición	Se ha configurado la electro-válvula como riego, abonado o inactiva.
Flujo normal	El usuario selecciona el número de electro-válvula e introduce mediante los pulsadores físicos de qué se trata.
Flujo alternativo	El usuario acciona el botón de volver: se regresará al caso de uso de gestión de programas (caso de uso 5.1)

Cuadro 5.9: Caso de uso configurar electro-válvula

Autoconfig

Actor	Cliente físico
Descripción	Permite realizar la configuración del programador automáticamente desde una tarjeta SD.
Precondición	Se ha introducido una tarjeta SD en la ranura.
Postcondición	Se ha cargado la configuración en el programador.
Flujo normal	Se muestra un mensaje advirtiendo de que se va a cargar la configuración eliminando cualquier información anterior. El usuario pulsa OK cargando la información.
Flujo alternativo	■ El usuario acciona el botón de volver: se regresará al caso de uso de gestión de programas (caso de uso 5.1) ■ Se produce un error durante la carga de la información: se mostrará un mensaje de error y se reiniciará el programador.

Cuadro 5.10: Caso de uso autoconfig

Mostrar mensajes

Actor	Cliente físico
Descripción	Permite visualizar una lista con todos los mensajes de error y advertencia activos.
Precondición	Ninguna.
Postcondición	Se ha mostrado la lista con todos los mensajes activos.
Flujo normal	El usuario entra en el menú de mostrar mensajes, se muestran los mensajes activos.
Flujo alternativo	El usuario acciona el botón de volver: se regresará al menú principal

Cuadro 5.11: Caso de uso mostrar mensajes

Gestión manual

Actor	Cliente físico
Descripción	Permite abrir y cerrar las electro-válvulas manualmente.
Precondición	Existe presión en la línea.
Postcondición	Se ha abierto o cerrado una electro-válvula manualmente.
Flujo normal	Se muestra una lista con las electro-válvulas configuradas en las que el usuario podrá navegar mediante los pulsadores físicos, permitiendo abrir o cerrar cada una de ellas.
Flujo alternativo	■ El usuario acciona el botón de volver: se regresará al menú principal ■ No existe ninguna válvula configurada: se mostrará un mensaje de advertencia. ■ La válvula a accionar es de abonado: se solicitará la confirmación de activación de la línea de riego.

Cuadro 5.12: Caso de uso gestión manual

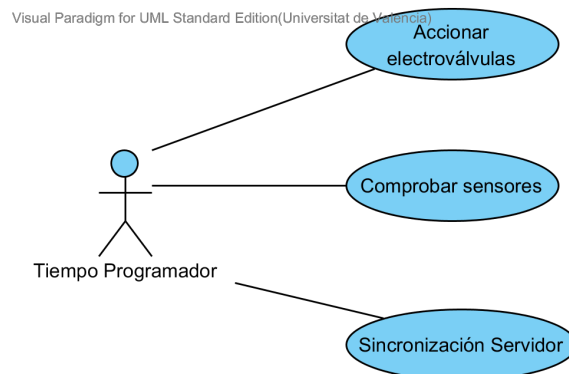


Figura 5.2: Diagrama de caso de uso para el tiempo programador.

Accionar electro-válvulas

Actor	Tiempo del programador
Descripción	Permite el accionamiento de las electro-válvulas según los programas introducidos por el usuario.
Precondición	La hora actual coincide con la hora de encendido o apagado de un programa. No existe ningún fallo de presión en la línea.
Postcondición	La electro-válvula seleccionada ha sido accionada.
Flujo normal	La hora actual coincide con la de apertura o cierre de un programa activo. La electro-válvula es abierta o cerrada.
Flujo alternativo	Existe un error de presión en la línea. La electro-válvula no se acciona y se genera un mensaje de error.

Cuadro 5.13: Caso de uso accionar electro-válvulas.

Comprobar sensores

Actor	Tiempo del programador
Descripción	Permite la revisión periódica de los sensores.
Precondición	Existe algún sensor configurado.
Postcondición	Se ha guardado el estado del sensor.

Flujo normal	Se ha cumplido el periodo de tiempo de comprobación por lo que se revisará y guardará el estado de los sensores, comprobando si están dentro de los parámetros preestablecidos.
Flujo alternativo	Los valores obtenidos exceden los límites: se debe generar un mensaje de error.

Cuadro 5.14: Caso de uso comprobarSensores

Sincronización servidor

Actor	Tiempo del programador
Descripción	Abre una comunicación con el servidor que permite la descarga de datos para su sincronización tanto de programas como de mensajes.
Precondición	El bloque de comunicaciones ha sido configurado.
Postcondición	Se han enviado los mensajes y programas modificados localmente y se han recogido aquellos modificados de forma remota.
Flujo normal	Se cumple el periodo de tiempo en el que se debe efectuar la operación. Se envían los programas y mensajes modificados localmente. Se reciben los programas modificados en el programador.
Flujo alternativo	No se ha podido establecer la conexión con el servidor: Se reintentará hasta 3 veces. Si no se consigue realizar la conexión se genera un mensaje local de error.

Cuadro 5.15: Caso de uso sincronización servidor

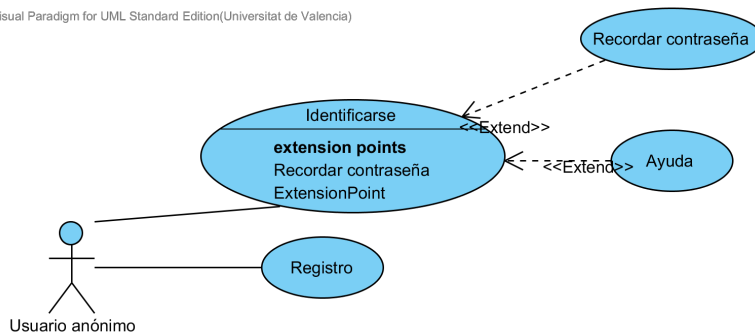


Figura 5.3: Diagrama de caso de uso para el usuario anónimo.

Identificarse

Actor	Usuario anónimo
Descripción	Permite a un usuario identificarse como cliente en el servicio.
Precondición	El usuario está dado de alta en el servicio.
Postcondición	Se ha iniciado una sesión en el servidor.
Flujo normal	El usuario introduce su nombre de usuario o email y contraseña. Se verifican los datos abriéndose una sesión en el servidor.
Flujo alternativo	Los datos introducidos no son correctos: se permitirán hasta 10 intentos cada 10 minutos por nombre de usuario e IP. Se sugerirá al usuario que utilice la opción de recordar contraseña (caso de uso 5.17) o la ayuda al usuario (caso de uso 5.18).

Cuadro 5.16: Caso de uso identificarse

Recordar contraseña

Actor	Usuario anónimo
Descripción	Permite a un usuario existente cambiar la contraseña olvidada.
Precondición	El usuario está registrado en el servicio.
Postcondición	El usuario recibe un enlace en el que podrá efectuar el cambio de su contraseña.

Flujo normal	El usuario introduce su nombre de usuario o email y pulsa el botón de continuar. El sistema verifica que se trata de un usuario registrado. Se envía un email con un enlace mediante el cual se puede cambiar la contraseña.
--------------	--

Cuadro 5.17: Caso de uso recodar contraseña

Ayuda

Actor	Usuario anónimo
Descripción	Permite la solicitud de ayuda si se están experimentando problemas con el servicio o se tienen dudas.
Precondición	Ninguna.
Postcondición	El usuario es atendido.
Flujo normal	El usuario pulsa el botón de ayuda, en el que se mostrará una lista de preguntas frecuentes y un número de teléfono.

Cuadro 5.18: Caso de uso ayuda

Registro

Actor	Usuario anónimo
Descripción	Permite a un usuario anónimo darse de alta en el servicio.
Precondición	Ninguna
Postcondición	Se ha creado un nuevo usuario en el sistema.
Flujo normal	El usuario introduce los datos requeridos. El usuario pulsa el botón de finalizar.
Flujo alternativo	Los datos introducidos no son correctos: Se indicará al usuario cuáles son.

Cuadro 5.19: Caso de uso registro

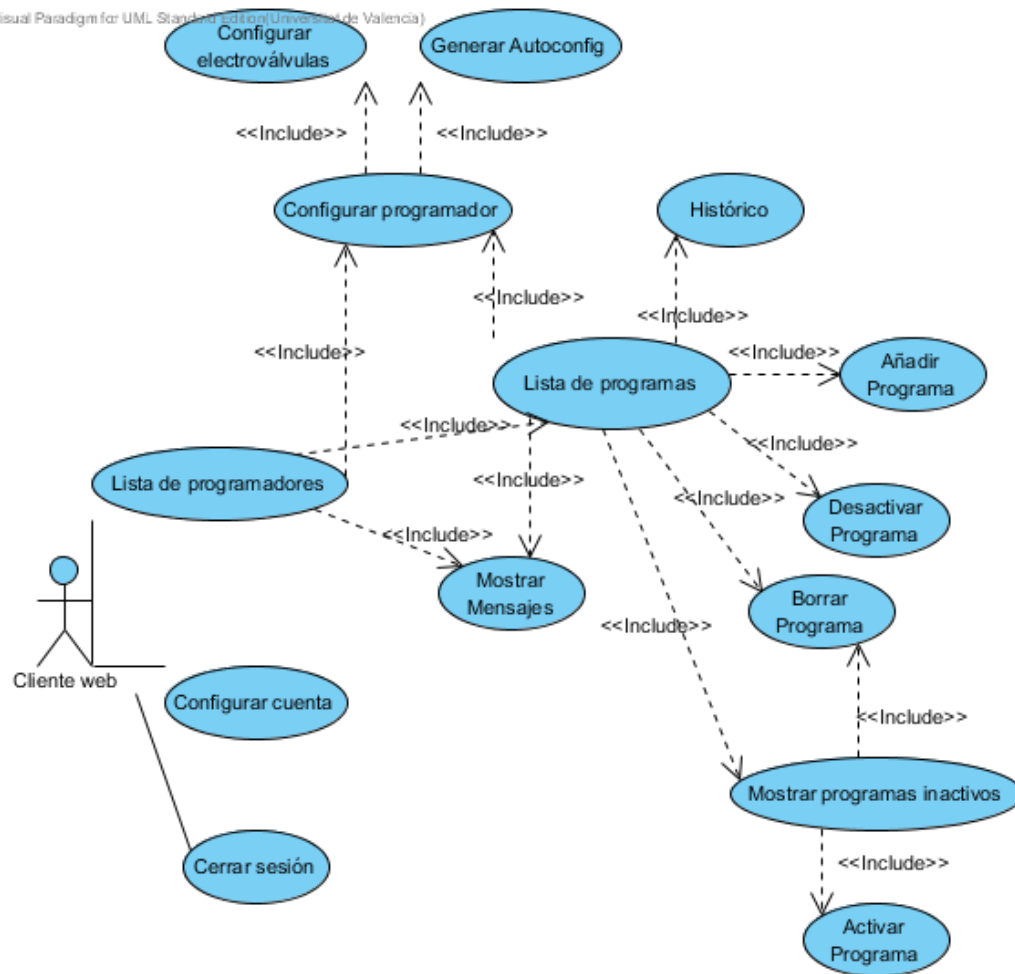


Figura 5.4: Diagrama de caso de uso para el cliente web.

Lista de programadores

Actor	Cliente web
Descripción	Muestra la lista de todos los programadores registrados en el cliente permitiendo acceder a cada uno de ellos.
Precondición	El usuario tiene al menos un programador dado de alta en el sistema.

Flujo normal	Se muestra una lista con todos los programadores. El usuario podrá acceder a los siguientes casos de uso: ■ 5.23 Lista de programas ■ 5.9 Configuración de electroválvulas ■ 5.28 Configuración programador. El sistema pasa al caso de uso seleccionado.
Flujo alternativo	No existe ningún programador: El sistema muestra una advertencia en el sistema.

Cuadro 5.20: Caso de uso lista de programadores

Configurar electroválvulas

Actor	Cliente web
Descripción	Muestra un menú que permite asociar y desasociar las electroválvulas a los puertos.
Precondición	Ninguna.
Postcondición	Se ha configurado la electro-válvula como riego, abonado o inactiva.
Flujo normal	El usuario selecciona el puerto de la electroválvula y si se trata de riego o abonado.

Cuadro 5.21: Caso de uso configurar electroválvulas

Mostrar mensajes

Actor	Cliente web
Descripción	Muestra en un recuadro destacado los mensajes de error en los sistemas de riego.
Precondición	Existe algún error o advertencia.
Postcondición	Se ha mostrado la lista con todos los mensajes activos.

Flujo normal	Se muestra una lista con los errores en el sistema.
Flujo alternativo	Si se encuentra dentro de un programador, solo se muestran los errores referentes a ese programador.

Cuadro 5.22: Caso de uso mostrar mensajes

Lista de programas

Actor	Cliente web.
Descripción	Muestra al usuario una lista con todos los programas configurados en el programador seleccionado en el caso de uso 5.20.
Precondición	Ninguna.
Flujo normal	El usuario accede a cualquiera de los siguientes casos de uso: ■ 5.26 Desactivar programa ■ 5.27 Borrar programa ■ 5.24 Añadir programa ■ 5.30 Histórico de programas ■ 5.31 Mostrar programas inactivos

Cuadro 5.23: Caso de uso lista de programas

Añadir programa

Actor	Cliente web
Descripción	Permite al cliente añadir un programa de riego o abonado.
Precondición	Existe al menos una electro-válvula configurada en el sistema.

Postcondición	Se ha añadido un nuevo programa activo en el que la válvula a accionar no coincide con la de otro programa. Si la válvula a accionar es de abonado, está contenida dentro de un programa de riego.
Flujo normal	El usuario introduce todos los datos solicitados mediante el formulario mostrado.
Flujo alternativo	■ El usuario pulsa retroceder: se navegará hacia atrás entre las opciones pedidas hasta llegar al menú de configuración. ■ Ya existe un programa que acciona esa electro-válvula durante el mismo periodo de tiempo: se debe mostrar un mensaje de error. Se añade el programa como inactivo. ■ Se ha intentado configurar un programa de abonado que no está contenido en uno de riego: se mostrará un mensaje de error. Se añade el programa como inactivo.

Cuadro 5.24: Caso de uso añadir programa

Activar programa

Actor	Cliente web
Descripción	Permite al usuario activar un programa actualmente inactivo.
Precondición	Existe al menos un programa inactivo.
Postcondición	Se ha añadido el programa a la lista de programas activos en el que la válvula a accionar no coincide con la de otro programa. Si la válvula a accionar es de abonado, está contenida dentro de un programa de riego.

Flujo normal	El usuario selecciona el programa que desea activar.
Flujo alternativo	■ Ya existe un programa que acciona esa electro-válvula durante el mismo periodo de tiempo: se debe mostrar un mensaje de error. Se añade el programa como inactivo. ■ Se ha intentado configurar un programa de abonado que no está contenido en uno de riego: se mostrará un mensaje de error. Se añade el programa como inactivo.

Cuadro 5.25: Caso de uso activar programa

Desactivar programa

Actor	Cliente web
Descripción	Permite al usuario desactivar un programa actualmente activo.
Precondición	Existe al menos un programa activo.
Postcondición	El programa seleccionado pasa a la lista de inactivos.
Flujo normal	El usuario selecciona el programa que desea desactivar.
Flujo alternativo	■ No existe ningún programa inactivo: se muestra un mensaje de advertencia. ■ El programa a desactivar es de riego y contiene a otro de abonado: tanto el programa de riego como el de abonado pasarán a la lista de inactivos. Se mostrará una notificación por pantalla.

Cuadro 5.26: Caso de uso desactivar programa

Borrar programa

Actor	Cliente web
Descripción	Permite al usuario eliminar un programa.
Precondición	Existe al menos un programa en el sistema.
Postcondición	El programa seleccionado ha sido eliminado del sistema.
Flujo normal	El usuario selecciona el programa que desea borrar.
Flujo alternativo	■ El programa a eliminar es de riego y contiene a otro de abonado: se mostrará una notificación por pantalla.

Cuadro 5.27: Caso de uso borrar programa

Configurar programador

Actor	Cliente web.
Descripción	Permite acceder a la configuración menos variable del programador.
Precondición	Existe un programador configurado.
Flujo normal	El usuario puede acceder a los siguientes casos de uso ■ Modificar el alias del programador ■ 5.29 Generar autoconfig .
Flujo alternativo	■ Algún campo no es válido: se indicará cuál es con un color distintivo. ■ El alias introducido ya existe en otro programador: se indicará mediante un mensaje de error.

Cuadro 5.28: Caso de uso configurar programador

Generar autoconfig

Actor	Cliente web.
Descripción	Permite al usuario crear y descargar el archivo de autoconfiguración para programador físico.
Precondición	Ninguna.
Postcondición	Se ha descargado en el ordenador del cliente el archivo para introducir en la memoria flash.
Flujo normal	■ El cliente pulsa sobre el botón de generar autoconfiguración. ■ El servidor crea un archivo de configuración que recopila toda la información del cliente. ■ El fichero es descargado en el ordenador del cliente.

Cuadro 5.29: Caso de uso generar autoconfig.

Histórico

Actor	Cliente web
Descripción	Muestra el histórico de programas ordenados por fecha.
Precondición	Ninguna.
Flujo normal	El cliente pulsa el botón de histórico. Se muestra una lista con todos los programas activos los últimos días.

Cuadro 5.30: Caso de uso histórico

Mostrar programas inactivos

Actor	Cliente web.
Descripción	Muestra la lista de programas inactivos.
Precondición	Existe al menos un programa inactivo. Hay un programador seleccionado.

Postcondición	se muestra una lista de programas inactivos.
Flujo normal	El cliente pulsa la opción de mostrar programas inactivos, se muestra la lista de programas inactivos. El usuario acceder a los siguientes casos de uso: ■ 5.26 Desactivar programa ■ 5.27 Borrar programa ■ 5.24 Añadir programa ■ 5.30 Histórico de programas ■ 5.23 Mostrar programas activos ■ 5.5 Activar un programa
Flujo alternativo	

Cuadro 5.31: Caso de uso mostrar inactivos

Configurar cuenta

Actor	Cliente web.
Descripción	Permite realizar cambios sobre la cuenta de usuario tales como la contraseña, los datos de contacto, etc.
Precondición	Ninguna.
Postcondición	Los cambios se han guardado en la base de datos.

Flujo normal	■ El cliente pulsa sobre la configuración de la cuenta. ■ Se realizan los cambios sobre los parámetros que se desee. ■ Se pulsa sobre el botón guardar. ■ Los datos han sido guardados en la base de datos.
Flujo alternativo	Los datos introducidos no son correctos: se mostrará un mensaje advirtiéndolo qué campos no son los correctos.

Cuadro 5.32: Caso de uso configurar cuenta

Cerrar sesión

Actor	Cliente web.
Descripción	Permite a un cliente cerrar la sesión web activa.
Precondición	Ninguna.
Postcondición	La sesión ha sido cerrada y no se puede acceder a los datos del cliente.
Flujo normal	■ El cliente pulsa el botón de cerrar sesión. ■ La sesión activa se cierra.

Cuadro 5.33: Caso de uso cerrar sesión

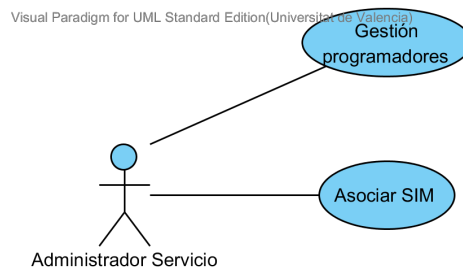


Figura 5.5: Diagrama de caso de uso para el administrador de servicio.

Gestión de programadores

Actor	Administrador Servicio.
Descripción	Permite al gestor del servicio administrar los programadores de los clientes.
Precondición	Ninguna.
Postcondición	Se ha asignado o eliminado una relación entre un programador y un cliente.
Flujo normal	■ El administrador introduce el número de identificador del programador. ■ Se introduce el DNI de un cliente. ■ Se selecciona la operación a realizar. ■ El programador queda asignado o desasignado.

Flujo alternativo	■ El número identificador no existe: se mostrará un error indicándose. ■ Se está intentando desvincular un programador no perteneciente al cliente: se mostrará un mensaje de error notificando el fallo. ■ Se está intentando vincular un programador ya asignado a otro cliente: se mostrará un mensaje de error así como el cliente al que está asignado.
-------------------	--

Cuadro 5.34: Caso de uso gestión de programadores

Asociar SIM

Actor	Administrador de servicio.
Descripción	Permite a un administrador de servicio asociar una tarjeta SIM a un programador.
Precondición	Existe un programador y una tarjeta SIM.
Postcondición	Se ha asociado una SIM a un programador.
Flujo normal	■ El administrador de servicio solicita asociar una tarjeta SIM. ■ El administrador introduce el número de teléfono asociado, su número pin y el operador al que pertenece. ■ La información se guarda en la base de datos.

Flujo alternativo	■ El número de teléfono ha sido asociado a otra tarjeta: el administrador deberá revisar los datos. ■ Existe algún error en los datos introducidos: la aplicación indicará cuál es.
-------------------	--

Cuadro 5.35: Caso de uso

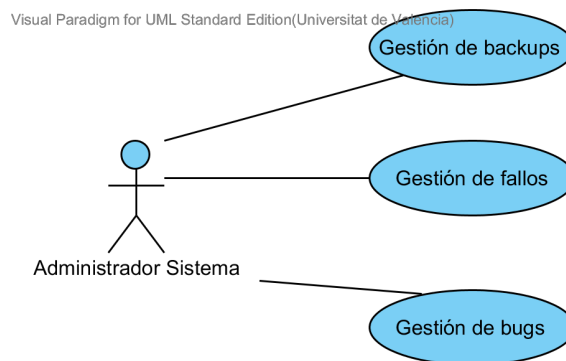


Figura 5.6: Diagrama de caso de uso para el administrador de sistema.

Gestión de backups

Actor	Administrador de sistema.
Descripción	Permite al administrador de sistemas gestionar los backup del sistema.
Precondición	Ninguna.
Postcondición	Se ha realizado o restaurado una copia de seguridad.
Flujo normal	■ El administrador introduce los parámetros de backup. ■ El backup se realiza en la fecha prevista (caso de uso 5.41).

Flujo alternativo	El sistema ha fallado: el administrador deberá seleccionar una copia de seguridad y restaurarla.
-------------------	--

Cuadro 5.36: Caso de uso gestión de backups

Gestión de fallos

Actor	Administrador de Sistema.
Descripción	Si se ha producido un fallo en el sistema, el administrador deberá solventarlo con la mayor brevedad posible.
Precondición	Se ha producido un fallo en el sistema.
Postcondición	El problema ha sido solucionado.
Flujo normal	■ El servicio no está disponible o se ha detectado un fallo. ■ El administrador de sistemas es notificado. ■ Se resuelve la incidencia.

Cuadro 5.37: Caso de uso gestión de fallos

Gestión de bugs

Actor	Administrador de Sistema.
Descripción	El administrador de sistema deberá resolver aquellos errores reportados en el sistema así como fallos de seguridad.
Precondición	Existe un error o fallo de seguridad.
Postcondición	La incidencia ha sido resuelta.

Flujo normal	■ Un error es reportado en el servicio mediante un trigger o email. ■ El administrador detecta el fallo. ■ Se resuelve el fallo.
--------------	--

Cuadro 5.38: Caso de uso gestión de bugs.

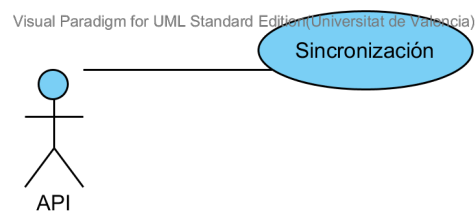


Figura 5.7: Diagrama de caso de uso para la API.

Sincronización

Actor	API.
Descripción	Forma la puerta de enlace de los programadores con el sistema.
Precondición	Se ha realizado una petición por parte del programador.
Postcondición	Se ha resuelto la petición.

Flujo normal	■ El programador solicita una petición de sincronización. ■ Se reciben o envían los datos correspondientes. ■ Se finaliza la petición.
Flujo alternativo	La acción solicitada no existe o no está disponible: se debe contestar con un mensaje de error.

Cuadro 5.39: Caso de uso sincronización

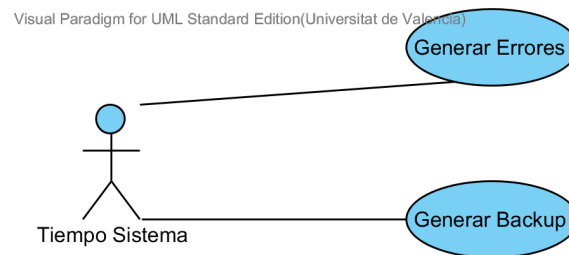


Figura 5.8: Diagrama de caso de uso para el tiempo del sistema.

Generar errores

Actor	Tiempo del sistema.
Descripción	Se deberán detectar fallos de conexión por parte de los programadores.
Precondición	Un programador no ha cumplido con la frecuencia mínima de sincronización.
Postcondición	Se genera un error que será informado al usuario.

Flujo normal	■ La frecuencia mínima de conexión no se ha cumplido. ■ El usuario es reportado mediante el portal web o demás dispositivos de contacto.
--------------	---

Cuadro 5.40: Caso de uso generarErrores

Generar backup

Actor	Tiempo del sistema.
Descripción	Dada una hora determinada en el sistema, se generará una copia de seguridad previamente programada.
Precondición	■ Existe una copia de seguridad programada. ■ Existe espacio suficiente en el disco.
Postcondición	Se ha generado una copia de seguridad.
Flujo normal	■ El administrador de sistemas programa una copia de seguridad (caso de uso 5.36). ■ La hora programada y la actual coinciden. ■ Se genera un backup en el destino programado.
Flujo alternativo	No existe espacio suficiente en el disco: se generará una incidencia que deberá ser reportada al administrador de sistema.

Cuadro 5.41: Caso de uso generar backup

5.1.3. Análisis del modelo de datos

En este apartado trataremos de describir qué datos salvaremos y cuál será su estructura para ser almacenados. La detección de qué información puede resultar útil ha sido analizada junto a un equipo de expertos agrícolas. Además, la estructura de datos que se expone a continuación ha sido pensada para que pueda ser ampliada en un futuro si fuese necesario.

Cabe mencionar que una vez realizada la sincronización, toda la información existente en el programador también estará en el servidor. Diremos así que las estructuras de datos del programador están casi contenidas en el servidor tal y como se representa en la figura 5.9.

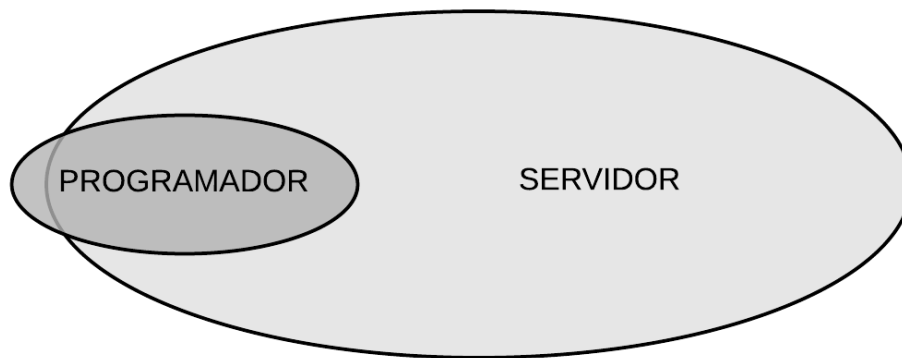


Figura 5.9: Datos en programador-servidor.

Como ha sido común a lo largo de la memoria, dividiremos de nuevo el apartado en dos grandes bloques, Programador y Servidor aunque, por las razones expuestas en el párrafo anterior, en este caso empezaremos por la estructura del servidor.

5.1.3.1. Servidor

La estructura en el servidor resultará la más compleja debido a que guardará el grueso de la información.

En la tabla se muestra el resultado del análisis. En ella aparecen las diferentes entidades que permitirán almacenar dicha información en la base de datos del sistema.

Entidad	Descripción
Cliente	Guarda la información personal del cliente. Esta entidad debe de guardar especial seguridad son datos que requieren un nivel de seguridad básico según la LOPD que estudiamos en el apartado de Viabilidad legal (Página 71).
Usuario	Esta es el enlace web del cliente. En este ítem únicamente guardaremos la información relativa para realizar el acceso a la página web.
Programador	Almacenaremos la información relativa a los programadores existentes.
Error	Esta entidad guardará todos los mensajes de notificación que se pueden mostrar en el servidor. Cabe la posibilidad de introducir los mismos mensajes en distintos idiomas
ProgError	Guarda el histórico de errores aparecidos en un programador.
Programa	Esta entidad almacena los datos relativos a un programa tales como su hora de inicio, fin, etc.
HistoricoProgramas	Almacenaremos cronológicamente los programas ejecutados en el programador.
Sync	Guarda el histórico de conexiones de cada programador.
HistorialSensor	En dicha entidad almacenaremos el histórico de los valores de los sensores a lo largo del tiempo.
Sensor	Guarda la información de configuración de los sensores tales como la fórmula de cálculo o los valores máximos y mínimos.
HistorialElectrovalvula	Guarda el historial de acciones de las electrovalvulas.
Electrovalvula	Esta entidad guardará la información sobre las electro-válvulas presentes en el sistema.
Conexión	Almacena los datos de configuración relativos a la conexión, tales como el host, el puerto de conexión, el PIN, etc.

APN	Guarda los datos relativos a la configuración de cada operadora móvil para realizar una conexión GPRS.
-----	--

Cuadro 5.42: Entidades presentes en la estructura de datos del servidor.

En la figura 5.10 se muestra el diagrama de clases que formará la estructura en la base de datos. Cabe decir que, a la hora de implementar dicha estructura, algunas de las entidades podrán ser simplificadas al igual que pueden aparecer otras nuevas que permitan resolver relaciones del tipo muchos a muchos.

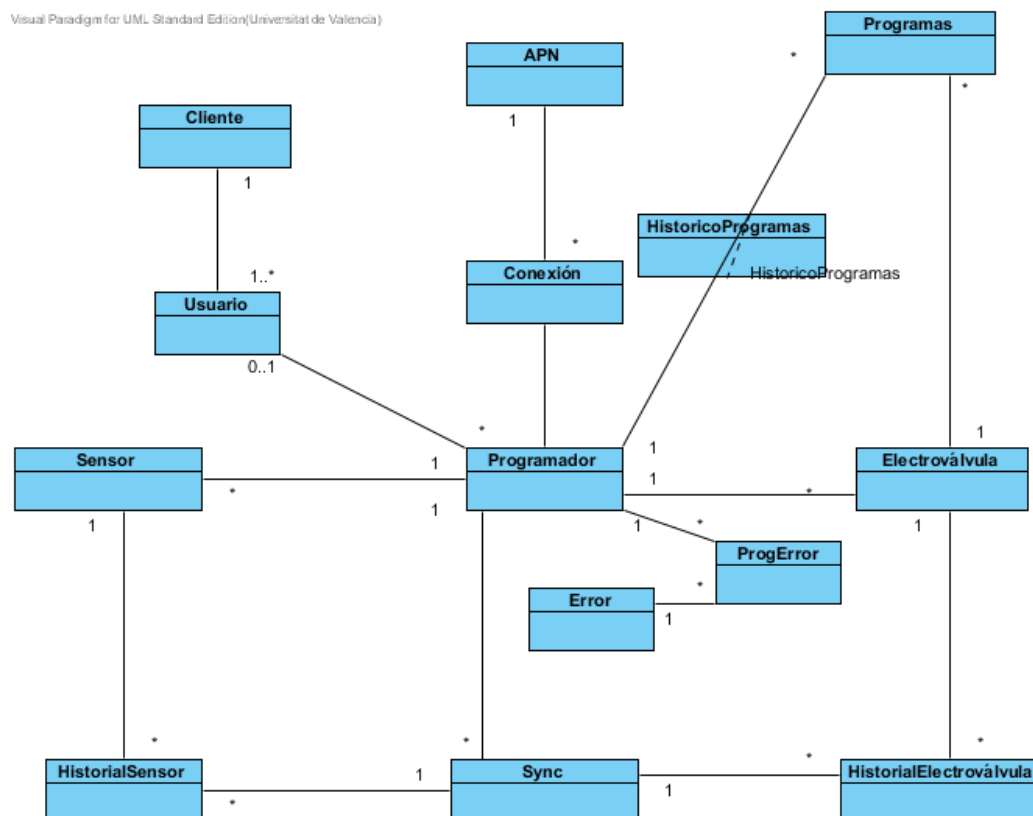


Figura 5.10: Diagrama de clases del servidor.

5.1.3.2. Programador

En el programador únicamente se debe almacenar la información relativa al propio programador y programación. Esta información debe ser almace-

nada de forma permanente ¹. En la tabla 5.43 mostramos los diferentes tipos de memorias existentes en el microcontrolador y sus características [62]. Debemos escoger entre ellas para salvar los datos según convenga.

Tipo	Tamaño	Descripción
Flash	32KB	Es donde se almacena el código del programa. Aunque se puede reescribir en tiempo de ejecución, su velocidad es lenta y el número máximo de ciclos está estimado en 10.000. Su gran desventaja es que si realizamos escrituras en la misma deberemos prestar atención y no escribir allá donde se encuentra el programa.
SRAM	2KB	Se trata de una memoria de tipo volátil en la que se perderán los datos una vez perdida la alimentación. Su velocidad es mucho mayor.
EEPROM	1KB	Es una memoria de tipo no-volátil. Su velocidad de escritura es más rápida que la memoria Flash y su número máximo de ciclos está estimado en 100.000 escrituras.

Cuadro 5.43: Diferentes tipos de memoria existentes en el microchip ATmega328.

Dado que la capacidad de almacenamiento de datos en el programador es muy limitada, solo deberemos guardar aquella información imprescindible para un correcto funcionamiento, el resto será almacenada únicamente en la parte servidora. Diremos así que la información guardada en el programador es una copia local de la información básica sobre los programas existentes en el servidor.

En la tabla 5.44 se muestran las entidades que utilizaremos para su ejecución. La mayoría de estas son comunes a la parte servidor pero solo con los

¹En los microcontroladores es común utilizar únicamente el almacenamiento en memoria RAM que es volátil.

datos referentes al programador y con un número menor de columnas dado que mucha de la información en el servidor no será necesaria.

Entidad	Memoria	Descripción
Sistema	SRAM	Contiene todas las variables del sistema que no son de crucial importancia.
Mensajes	FLASH	Descrito en tabla 5.42.
Programador	EEPROM	Contiene una única instancia referente a la entidad descrita en 5.42.
Información programador	FLASH	Descrito en tabla 5.42.
Programas	EEPROM	Descrito en tabla 5.42.
Electro-válvulas	EEPROM	Descrito en tabla 5.42.
Configuraciones	FLASH	Descrito en tabla 5.42.
Sensores	FLASH	Descrito en tabla 5.42.
Conexión	FLASH	Descrito en tabla 5.42.

Cuadro 5.44: Información no volátil en el programador.

Cabe decir que toda la información será almacenada en la memoria SRAM. La memoria FLASH y EEPROM guardarán una copia con los mismos datos que permitirán recuperar los datos después de un reinicio del sistema.

En la figura 5.11 se representa el diagrama en el que se estructurarán los datos.

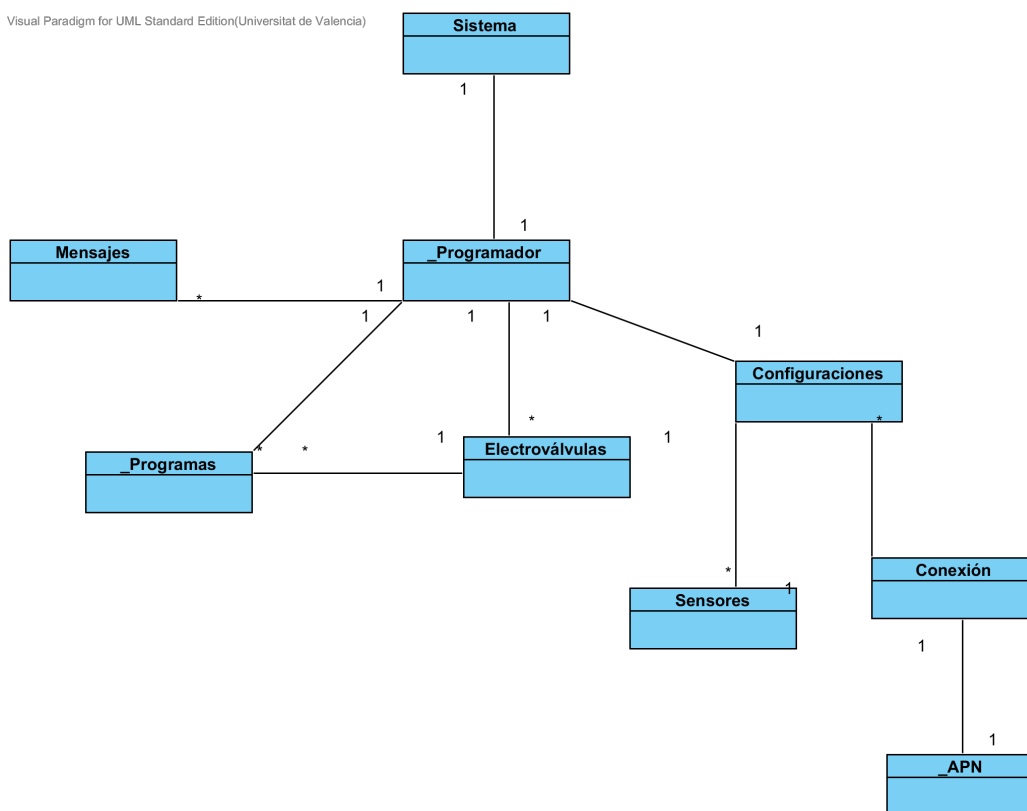


Figura 5.11: Diagrama de clases del programador.

Algunas de estas entidades tan solo contendrán un campo aunque en las tablas del servidor se contengan muchos más. Diremos que la información que guardamos en el programador es tan solo una copia local de la información necesaria para dicho programador.

5.1.4. Análisis de las interfaces de usuario

En este apartado trataremos de analizar la forma con que el cliente interactuará con el producto. El objetivo de este análisis es el de crear una interfaz gráfica sencilla a la vez que amigable.

Dado que existen dos² interfaces de comunicación totalmente distintas (de nuevo hablamos de la interfaz del programador y la del servidor), deberemos dividir el apartado en dos, para crear la mejor interfaz de usuario posible. Cabe recalcar que, aun siendo tan distintas las formas de interacción con el sistema, se intentarán crear patrones de acceso comunes que permitan al

²No consideraremos la interfaz de comunicación del administrador de sistemas puesto que la gestión de sus funciones se realizará a través de programas externos al nuestro.

usuario final encontrar similitudes en el mismo, puesto que su conocimiento técnico, recordamos, es bajo.

Como se ha definido en el apartado de Especificación, la prioridad de la interfaz es que resulte lo más intuitiva posible para el usuario.

Dado que no se cuenta en el equipo con ningún experto en el diseño de interfaces, todas las interfaces diseñadas han sido inspiradas en productos similares o que al menos ofrecen una interfaz que permitiría, con algunas modificaciones, adaptarse a nuestro producto.

5.1.4.1. Análisis de interfaz del programador

En el caso del programador deberemos diseñar tanto la interfaz de menús como la interfaz física de usuario.

Es importante subrayar que las operaciones que realizaremos en el programador no son tan extensas como las que se pueden realizar mediante el servicio web. Además, aunque se podrán realizar todas las operaciones de creación y modificación de programas, se espera que el usuario utilice la interfaz web para ello.

Interfaz física

Para el diseño de esta interfaz hemos recurrido a programadores de riego ya existentes en el mercado. El tipo de programador en el que basaremos nuestro diseño será de aquellos orientados a pequeños sistemas. El escogido es el programador WP de la compañía Rainbird, anteriormente estudiado en el capítulo Estado del Arte(Página 9).

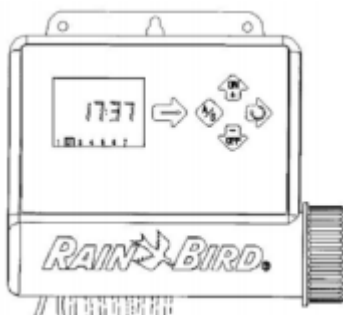


Figura 5.12: Interfaz programador Rainbird WP.

Fuente: http://www.rainbird.eu/files/rainbird/manuel_utilisation/201_manuel.pdf

Como podemos observar en la figura 5.12, este programador consta de tan solo 5 botones y una pantalla LCD que mostrará los datos necesarios.

Mediante los 5 botones, con formas y dibujos que describen su función, el usuario podrá introducir todos los datos necesarios para su funcionamiento.

Interfaz gráfica

Dado que el tamaño del LCD tendrá un tamaño reducido, la interfaz deberá estar preparada para ello. Se han identificado diferentes tipos de menú:

- Menú de tipo lista
Será en el que mostraremos los diferentes elementos en una lista. Estos pueden ser programas, menús de configuración, etc.
- Introducción de datos
Es aquel menú diseñado para introducir los datos, se deberá mostrar el dato a introducir y el valor del mismo.

5.1.4.2. Análisis de la interfaz de la aplicación web

La interfaz web será el medio de interacción usual del usuario con el servicio. Dicha interfaz debe permitir al usuario acceder a todas las características del servicio de una forma sencilla e intuitiva.

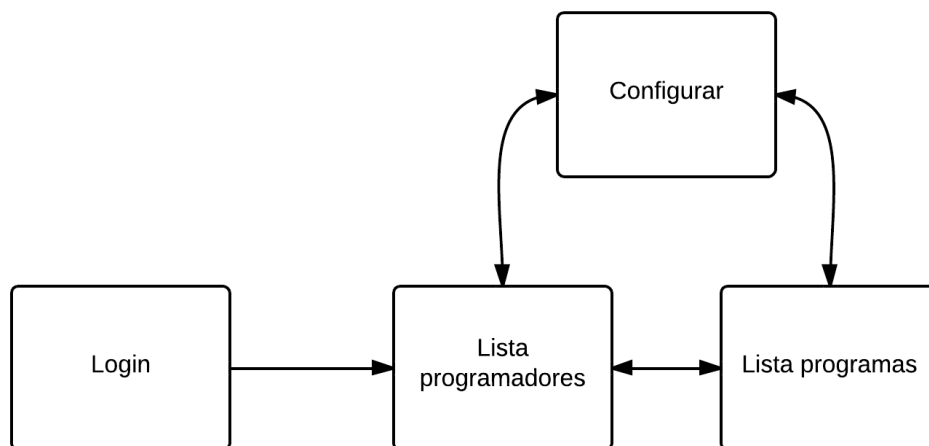


Figura 5.13: Diagrama de flujo entre los diferentes menú web del servicio.

En la figura 5.13 se muestra el flujo de menús del servicio, de los cuales extraeremos 3 tipos de esquemas diferentes: login, lista y configuración. Además, las dos últimas estarán contenidas dentro de una interfaz 5.14 que permitirá acceder a las acciones globales del sistema.

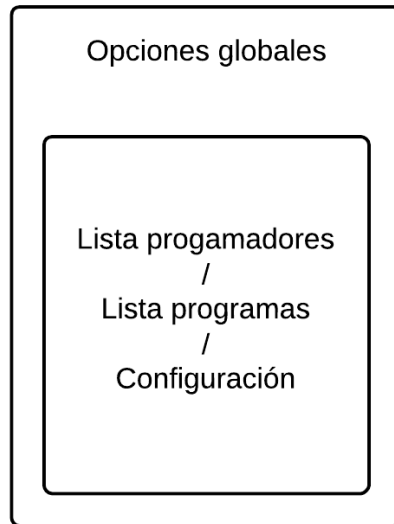


Figura 5.14: Esquema de la web una vez realizado el proceso de identificación.

Una vez identificados los bloques y la estructura de nuestro sistema, podremos resumir que, a grandes rasgos, se trata de un sistema de logueo, listas y configuraciones. Esta estructura nos ha recordado claramente a la seguida por el servicio de espacio en la nube dropbox³ en su interfaz web. Esta destaca por su sencillez de uso.

³<https://www.dropbox.com>

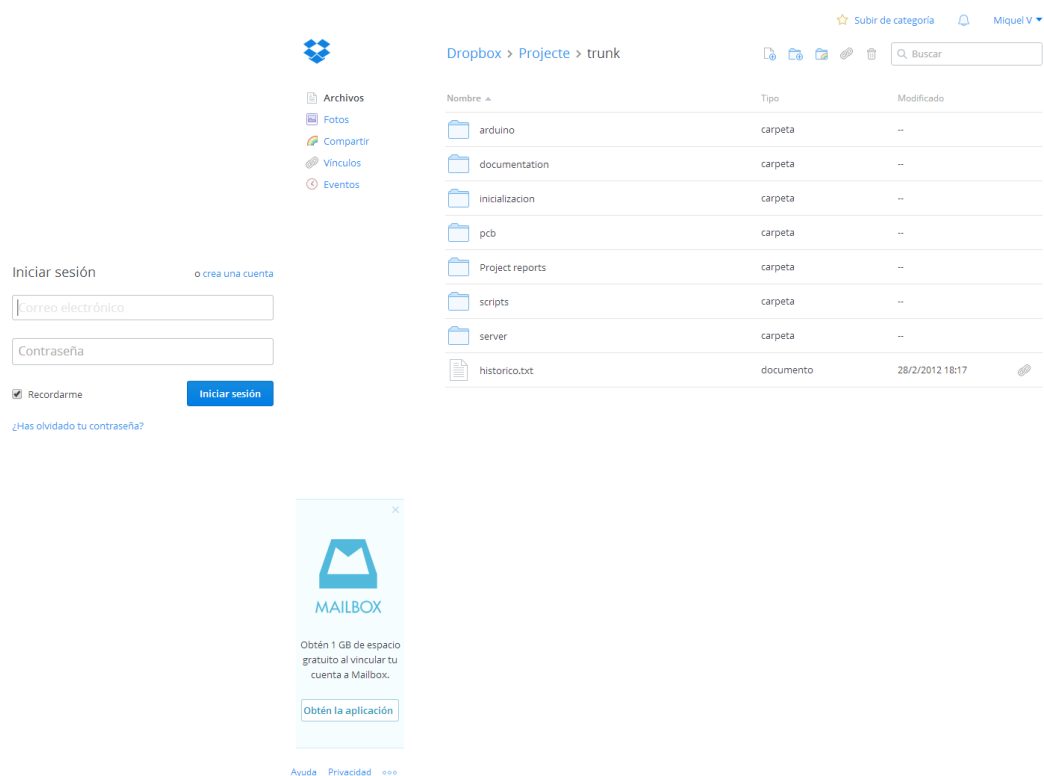


Figura 5.15: Interfaz web de login y carpetas en dropbox.

5.1.5. Análisis de la fuente de alimentación

Una de los componentes claves en el programador es la fuente de alimentación. Esta debe ser capaz de transformar las tensiones de entrada provistas por la batería y el generador en tensiones válidas para el sistema.

En la figura 5.16 obsevamos un esquema energético necesario, con sus entradas y salidas.

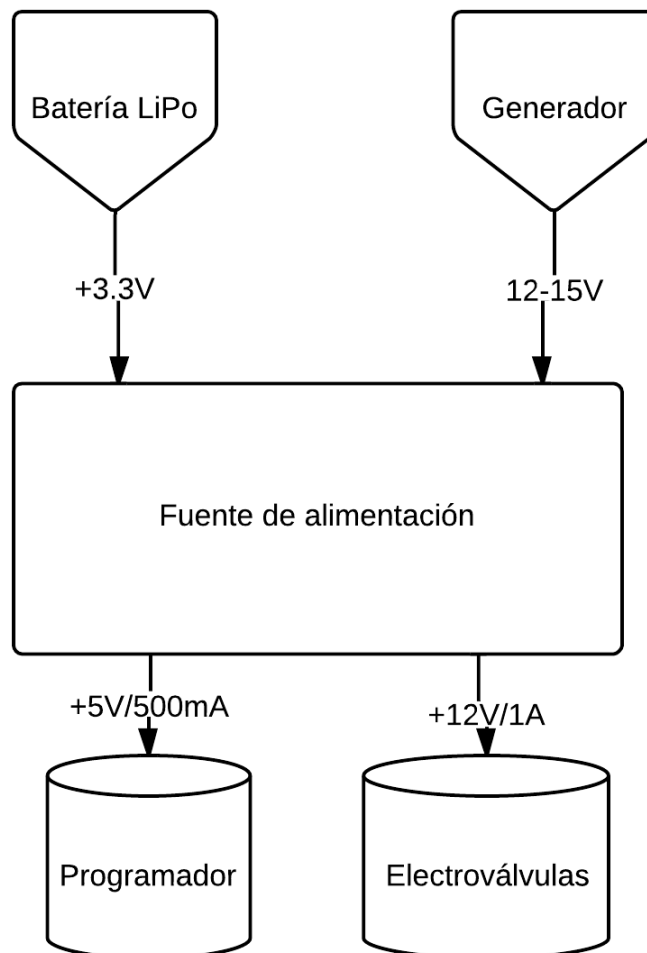


Figura 5.16: Esquema de energía del sistema.

El posterior diseño de la fuente de alimentación deberá adaptarse a este esquema. A continuación describiremos y analizaremos cada uno de los elementos.

- **Batería LiPo:**

Con una capacidad de 2000mAh, esta batería ofrece un voltaje que oscila de 3.7 a 3.9V. Ofrece un circuito de protección que permitirá cortar la corriente cuando se alcance la mínima carga⁴. La carga se deberá rea-

⁴Recordemos que la descarga total de la celda en este tipo de baterías significa la muerte de la misma.

lizar a 4.2V mediante un elemento de control de carga.
En la figura 5.17 observamos la tensión en la batería dada la carga máxima y conectada a una impedancia de 10 MOhm.

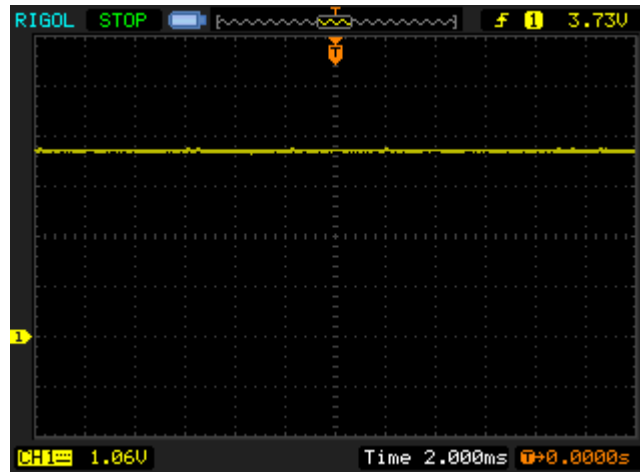


Figura 5.17: Tensión de salida de la batería.

- **Generador de energía con batería integrada:**

Este generador, que parecía inicialmente el más apto para el proyecto, ha resultado fallido pues su funcionamiento no era el esperado.

Según las especificaciones del fabricante, este producto era capaz de generar energía y almacenarla en una batería interna, proporcionando en la salida del mismo una tensión estable de 3.3V. Al probar el producto se observó que solo se suministraba energía a su salida cuando el generador estaba actuando, siendo además dicha tensión variable según el caudal. En la figura 5.18 se muestra la salida obtenida en el generador encendido

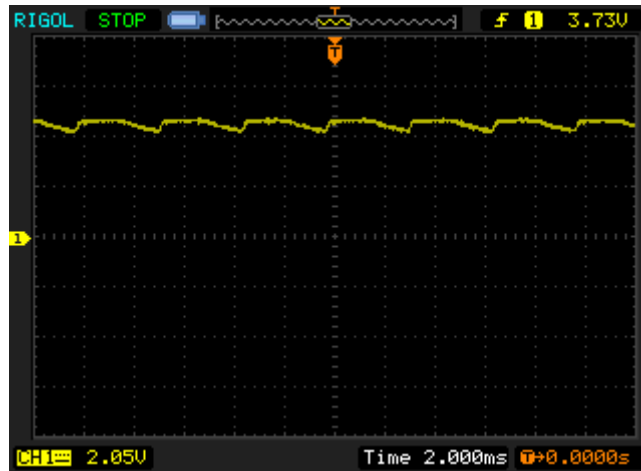


Figura 5.18: Tensión de salida del generador cuando está funcionando.

Inicialmente se pensó que era un problema de la batería pero, tras desmontar el generador y comprobar que su funcionamiento era correcto, se extrajo el circuito del mismo (figura 5.19). Aunque el circuito es correcto casi en su totalidad, el transistor situado a la salida es erróneo debido a que es de tipo NPN cuando debería ser PNP, lo que hace que la tensión de la batería fluya hacia el emisor cuando exista tensión en la base, es decir, cuando se está generando energía.

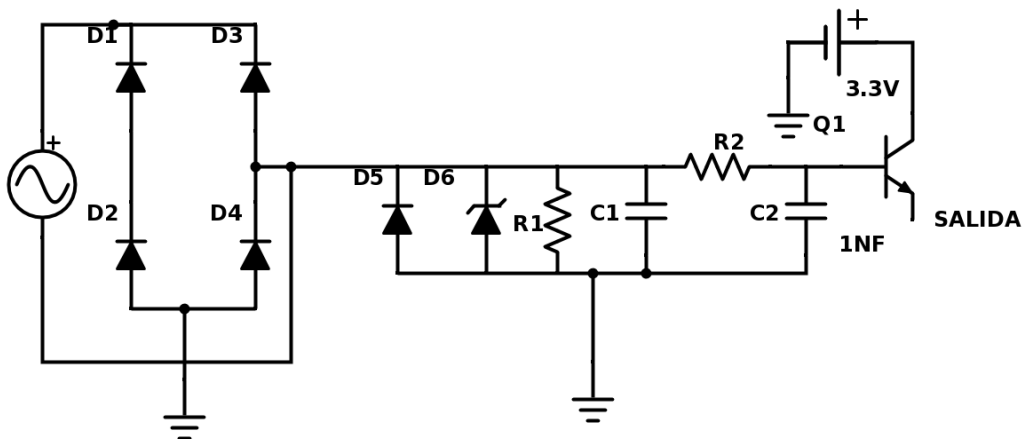


Figura 5.19: Esquema del circuito extraído del generador.

Aunque el problema podría haber sido solucionado, se observó que la corriente generada era mucho menor de la esperada ya que el sistema

contaba con dos pasos de energía por inducción. Por ello se optó por comprobar las cualidades del siguiente generador.

- **Generador de energía sin batería:**

Se trata de un minigenerador hidráulico de mecánica sencilla puesta con un motor DC conectado a una turbina sin ningún otro paso de inducción más que el propio motor. Además, no incluye ningún tipo de circuito electrónico ni batería, por lo que el procesamiento de la señal deberá hacerse en la fuente de alimentación. En la figura 5.20 se muestra la salida de los dos generadores (CH1-Actual, CH2-Anterior) sin ninguna carga en la salida. Como se puede observar, la salida del generador cuenta con un voltaje mucho mayor que el anterior. Aún así, la tensión en vacío es fácil de conseguir, por lo que en la gráfica de la figura 5.21 se representa la salida del programador con el Arduino conectado. Esta tensión ha sido lograda a mitad caudal pues el regulador incluido en el Arduino no permite tensiones mayores de 12V.

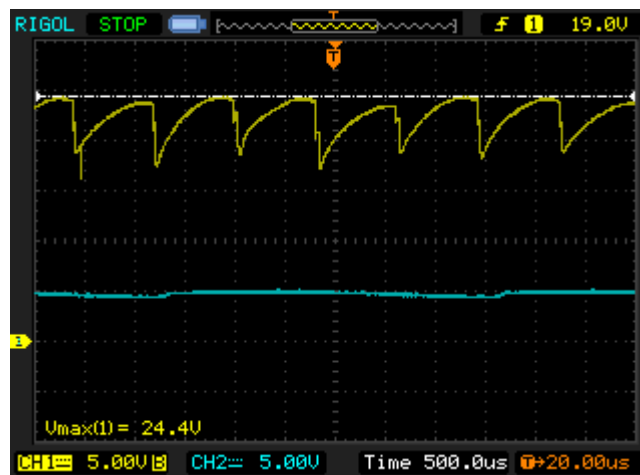


Figura 5.20: Salida de los generadores en vacío

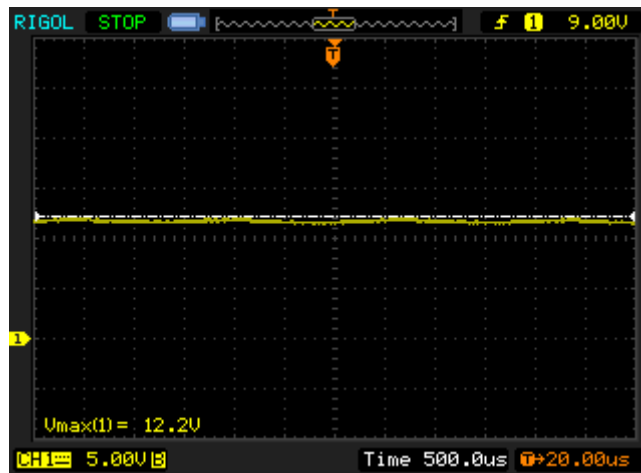


Figura 5.21: Salida del generador conectado al Arduino

- **Electroválvulas** - Dorot S92-2 :

Son electroválvulas de tipo *latch* las cuales no consumirán ninguna corriente una vez finalizado el cambio de estado. Para activar las mismas será necesario un pulso de 7.5 a 30VDC durante un tiempo de 15 a 100ms. La potencia máxima de la electroválvula es de 2W y su impedancia interna es de 5.1 Ohms.

En la figura 5.22, se observa cómo afecta a la salida del generador (a mitad caudal y alimentando al Arduino) cuando se acciona el cierre de la electroválvula. Al realizarse el accionamiento de la electroválvula, la tensión en la línea caerá durante más de 18ms, introduciendo además picos de tensión negativa. Esto se debe a que la electroválvula es una bobina y demandará una corriente muy grande durante este proceso. Este efecto adverso se puede paliar, tal y como se refleja en la figura 5.23 añadiendo un condensador en paralelo al generador, lo que nos permitirá seguir suministrando la tensión mínima de funcionamiento.

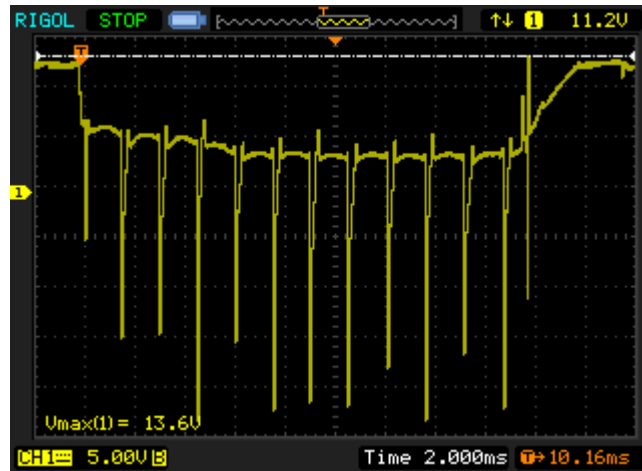


Figura 5.22: Salida del generador conectado al Arduino

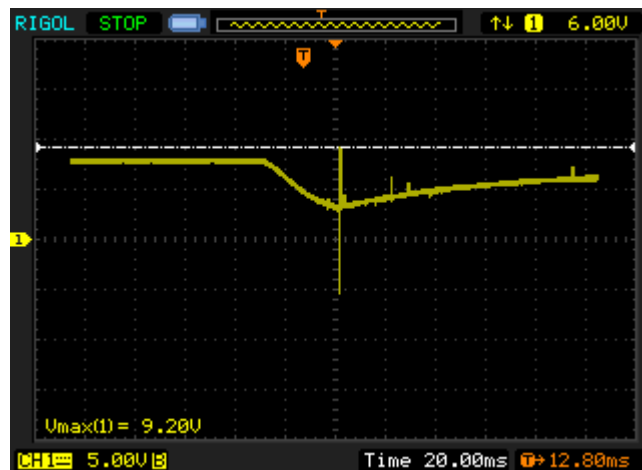


Figura 5.23: Salida del generador conectado al Arduino y un condensador de 4200uF

■ Programador:

El programador es el sistema basado en Arduino. Una de las principales razones del alto consumo en la plataforma Arduino es el regulador DC que incorpora por ser de tipo lineal, lo que transforma en calor el exceso de energía. Es por ello que será necesaria la incorporación de un regulador DC de altas prestaciones que permita disminuir lo máximo posible dicho consumo.

Es necesario destacar que, cuando se realicen conexiones al servidor, el módulo GPRS puede alcanzar picos de hasta 1A, por lo que será necesaria la presencia de condensadores que garanticen estos picos de corriente.

Algunos de los aspectos importantes a la hora del diseño de la fuente de alimentación serán los siguientes:

- **Tipo de conversores DC-DC:**

Deberán ser de alto rendimiento, evitando así conversores de tipo lineal que disipan en forma de calor el excedente energético, por lo que son de una eficiencia energética muy baja.

- **Tipo de encapsulado:**

Otra de las características a tener en cuenta es el tipo de encapsulado en el que se produce. A poder ser, durante la fase de prototipado, se deberán utilizar encapsulados tipo PDIP pues su mayor tamaño de patillas facilita la soldadura manual. Se deben evitar en todo momento encapsulados que no permitan un montaje con soldador.

- **Evitar elementos inductivos:**

Se deberá evitar, en la medida de lo posible, la presencia de elementos inductivos en los circuitos pues su rendimiento suele ser menor y la calidad de la bobina afecta en gran medida a la calidad de la señal resultante.

5.2. Diseño

Completada ya la fase de análisis del sistema, se procederá al diseño del mismo. En este apartado se diseñará el sistema a bajo nivel. Es el paso previo a la realización del mismo. Así pues, definiremos su arquitectura, definiremos los diagramas de secuencia para los casos de uso del apartado anterior, definiremos el diagrama de datos tal y como deberá implementarse en la siguiente fase del proyecto y, finalmente, definiremos cómo deberán ser las interfaces de usuario para cada uno de los casos detectados en el apartado de análisis.

5.2.1. Arquitectura del sistema

La arquitectura del sistema trata de definir el tipo de estructura que se utiliza en el sistema. Según la profundidad con la que se analice el sistema,

podremos definir diferentes tipos de arquitecturas. Procederemos pues a realizar su definición desde un punto de vista más general e iremos avanzando hasta concretar en cada uno de los subsistemas.

Desde la definición de los objetivos, sección 2.2, se ha planteado el sistema como un servicio en la nube que permite a un usuario interactuar con sus programadores de riego. Para poder ofrecer dicho servicio, se recurrirá a una estructura clásica cliente-servidor, tal como se indica en la figura 5.24, en la que las peticiones se realizarán al servidor a través de Internet, el cliente las realizará mediante la interfaz web y el programador mediante la API, encontrándose ambos servicios alojados en el servidor.

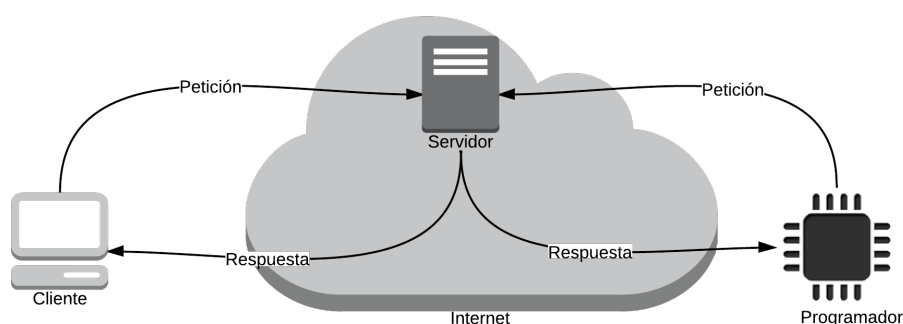


Figura 5.24: Arquitectura cliente-servidor en una visión global.

Por lo que respecta a la arquitectura en el servidor (figura 5.25), de nuevo recurriremos a una estructura cliente-servidor. El servidor tratará de responder a las peticiones realizadas por los clientes a través del servicio web y de los programadores a través de la API. Para resolver estas peticiones, será necesario consultar la base de datos, por lo que también seguiremos dicha estructura con la misma. Cabe resaltar que, esta arquitectura es válida ya que el programador es el que realiza las peticiones de conexión, datos, etc., como se ha definido en el apartado de Especificación del sistema (Página 46).

Esta estructura además sigue el esquema de MVC:

- **Modelo:**
Es la capa en el que se guarda la lógica de negocio. Las peticiones realizadas a través de la vista se comunican con el modelo a través del controlador.
- **Vista:**
Está formada por la interfaz web y la API. Permite la entrada de datos

al sistema por parte del cliente y el programador. Diremos que es la capa que permite comunicarse con el sistema.

- **Controlador:**

Es la capa encargada de responder mediante peticiones al modelo a eventos generados por la vista.

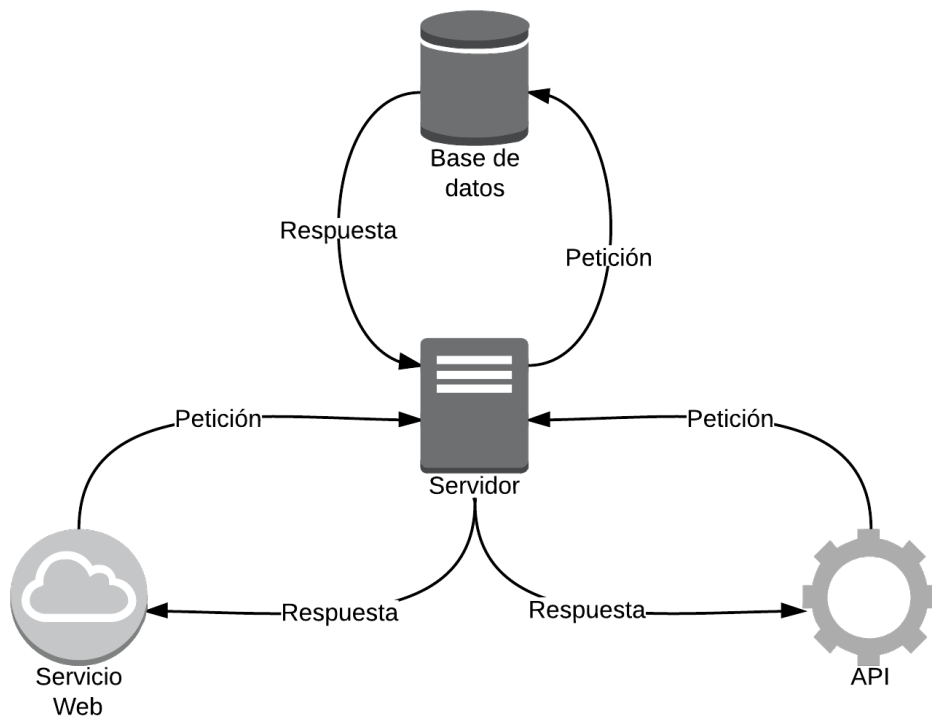


Figura 5.25: Arquitectura cliente-servidor en el servidor.

Por lo que respecta a la estructura interna del programador, se ha definido una estructura por capas⁵. Tomando como objeto principal el núcleo del sistema, como se muestra en la figura 5.26, este será el encargado de realizar peticiones a las distintas capas existentes.

⁵Que podría ser encapsulada también dentro de una estructura cliente-servidor.

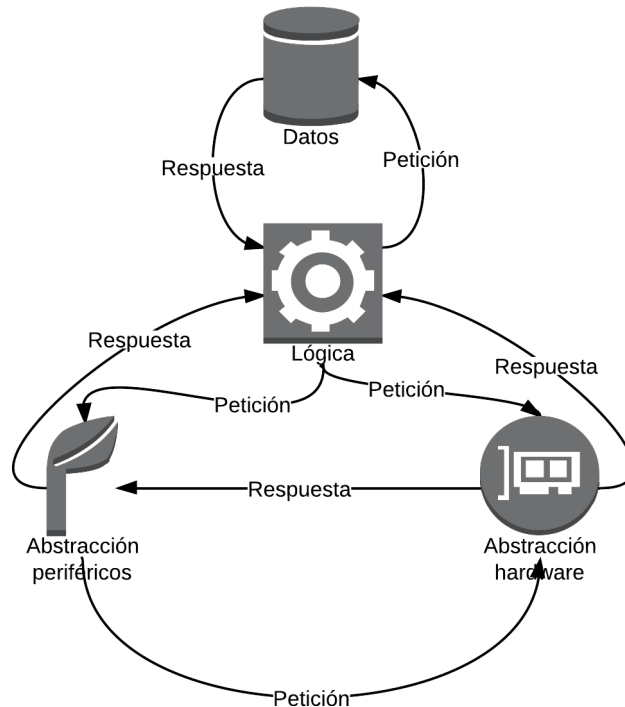


Figura 5.26: Arquitectura por capas en el programador.

5.2.2. Diseño de los datos

Durante el apartado de análisis concluimos que el grueso de los datos almacenados en el programador resultaban una copia local de aquellos almacenados en el servidor⁶. También se debatió sobre el tipo de memoria existente en el programador y cuál se debería utilizar en cada caso.

Durante el diseño de los datos, trataremos de concretar cuáles serán los datos almacenados en cada una de las clases. De nuevo dividiremos el apartado en servidor y programador en orden opuesto al resto de la memoria.

5.2.2.1. Diseño de datos en el servidor

Para entender el diseño de datos en el servidor, será necesario entender, aunque sea de una forma muy primaria, el funcionamiento de Symfony. Para

⁶Siempre y cuando los datos estuviesen sincronizados.

ello se recomienda la lectura del apéndice Symfony: la arquitectura (página 251).

Inicialmente se diseñará el diagrama de clases extendido a implementar en el servidor pero, debido a que el sistema va a ser implementado sobre Symfony, se deberá distribuir esta clase en distintos *bundles*.

Estructura de datos: Diagrama de clases extendido

En la figura 5.27 se muestra el diagrama de clases extendido en el servidor. A diferencia del diagrama de clases mostrado en el apartado anterior, en este se muestran los atributos que serán necesarios en cada una de estas clases.

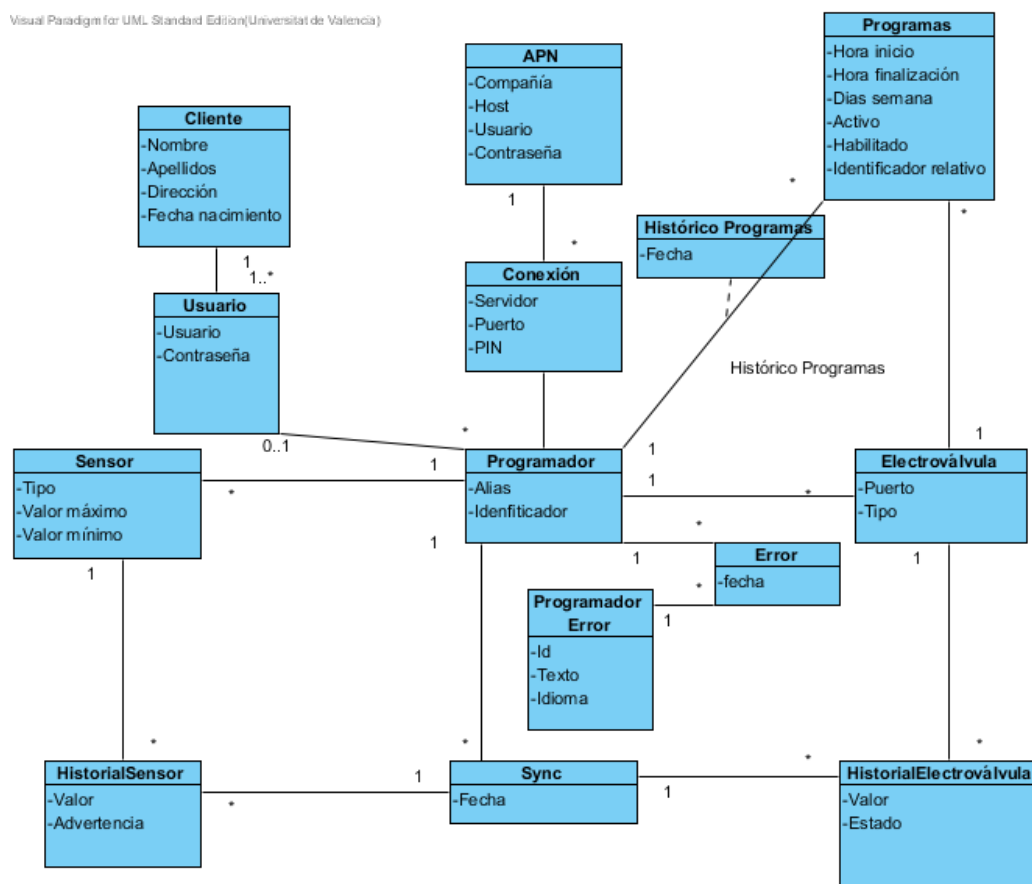


Figura 5.27: Diagrama de clases del servidor junto a sus atributos.

Estructura de *bundles* en el proyecto

Como se explica en el apéndice, un *bundle* trata de actuar como contenedor de una o más clases de el sistema. Así pues, aunque no es necesario crear

distintos *bundle* para desarrollar nuestro sistema, sí que es muy recomendable, pues ayudará a mejorar su estructura y organización.

En la figura 5.28 se muestran todos los *bundles* que se crearán en el proyecto. En ellos aparece además todas las clases, definidas en el diagrama anterior 5.27, contenidas.

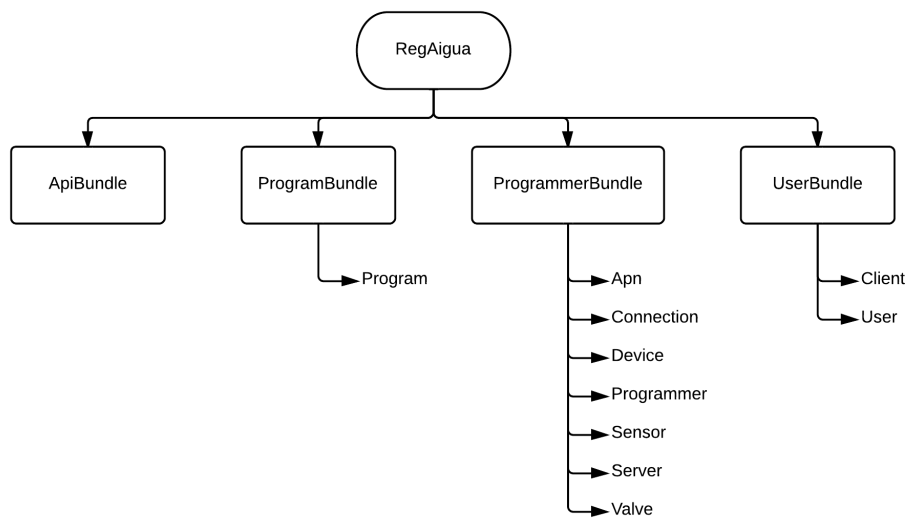


Figura 5.28: Bundles del proyecto

5.2.2.2. Diseño de datos en el programador

Como se ha comentado ya en distintas ocasiones, los datos guardados en el programador son una versión reducida de la base de datos para una instancia de la clase programadores y únicamente con la información útil para este programador.

Dado que nos encontramos en un entorno de recursos limitados, no se recurrirá a ningún sistema de bases de datos para almacenar la información. Esta será almacenada consecutivamente a partir de una dirección de memoria de acuerdo a estructuras de datos de C. Esta información será guardada y recuperada en bloque y se guardará el valor de las variables independientemente de si han sido utilizadas o no.

En la figura 5.29 se muestra el diagrama de clases. Si lo observamos, podemos ver que a algunas de las clases se les ha añadido una barra baja delante. Esto es porque no contienen los mismos datos que en el servidor pues no son relevantes para un correcto funcionamiento.

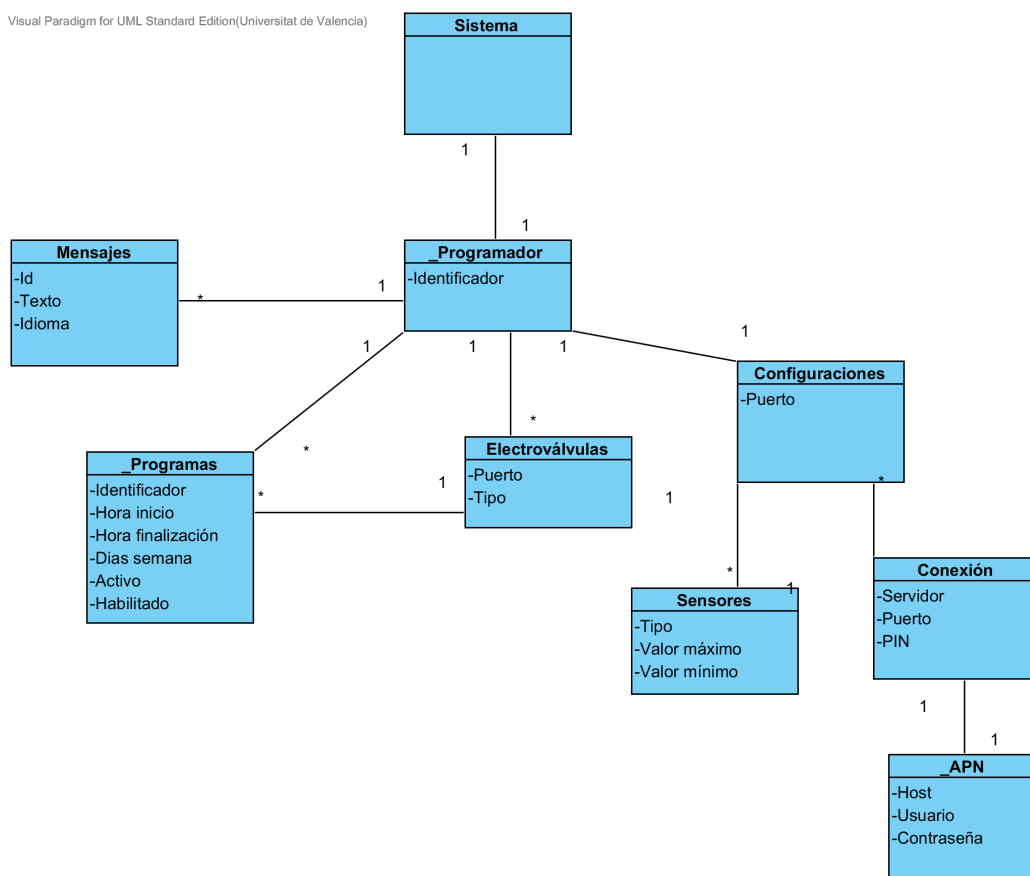


Figura 5.29: Diagrama de clases del programador junto a sus atributos.

En el caso del programador, es muy importante el estudio del tipo de datos utilizado para almacenar cada atributo de las clases así como el número de instancias máximo de cada clase, para así poder establecer el tamaño máximo de memoria necesario en el microcontrolador.

Programador

Consta de una única instancia en todo el programa. Es la clase principal y la que guarda relación con el resto.

Campo	Tipo	Descripción
Identificador	Int	Representa al identificador único del programador. Esta identificación se utiliza en las comunicaciones con el servidor.

Cuadro 5.45: Atributos de la clase Programador.

Mensajes

Guarda todos los mensajes en diferentes idiomas. Cada instancia de los Mensajes estará almacenada en un vector dentro de `_Programador`

Campo	Tipo	Descripción
Id	int	Guarda el identificador del tipo de mensaje.
Texto	char *	Guarda el texto a mostrar como mensajes de error y advertencias.
Idioma	int	Identifica al idioma utilizado.

Cuadro 5.46: Atributos de la clase `Mensajes`.

`_Programas`

Guarda los datos referentes a los programas. Las instancias estarán guardadas en un vector estático almacenado en la clase `_Programador`.

Campo	Tipo	Descripción
Identificador	int	Representa el identificador único de programa a nivel de programador.
Hora inicio	int	Representa el tiempo en segundos de un día en el que se inicia el programa.
Hora fin	int	Representa el tiempo en segundos de un día en el que se finaliza el programa.
Dias semana	char	Representa de forma binaria qué días de la semana se deberá activar el programa.
Activo	bool	Indica si el programa se encuentra activo en el momento.
Habilitado	bool	Indica si el programa se encuentra habilitado en el momento.

Cuadro 5.47: Atributos de la clase `_Programas`.

Electro-válvulas

Almacena la información relativa a las electro-válvulas. Se creará un vector de tipo estático de tamaño igual al número máximo de electro-válvulas conectadas.

Campo	Tipo	Descripción
Puerto	int	Número de puerto físico en el que se conecta la electro-válvula.
Tipo	ENUM	Tipo de electro-válvula de la que se trata: Riego o Abonado.

Cuadro 5.48: Atributos de la clase electro-válvulas.

Conexión

Guarda la información relativa a la conexión. Solo guardamos la tupla configurada en el programador.

Campo	Tipo	Descripción
Servidor	char *	Guarda la información relativa a la dirección de conexión al servidor.
Puerto	int	Puerto por el que se realiza la conexión al servidor.
PIN	int	código PIN necesario para realizar la conexión.

Cuadro 5.49: Atributos de la clase tipo conexión.

_APN

Guarda la información de configuración de la compañía de telecomunicaciones para realizar la conexión.

Campo	Tipo	Descripción
Host	char *	Host asociado a la compañía.
Usuario	char *	Nombre de usuario para realizar la conexión.
Contraseña	char *	Contraseña para realizar la conexión.

Cuadro 5.50: Atributos de la clase tipo _APN.

Sensores

Guarda la información de configuración de los sensores configurados. La información se guarda en la clase configuraciones en un vector de tipo estático.

Campo	Tipo	Descripción
Tipo	ENUM	Indica el tipo de sensor del que se trata.
Valor máximo	int	Valor máximo del sensor.
Valor mínimo	int	Valor mínimo que puede alcanzar el sensor.

Cuadro 5.51: Atributos de la clase tipo Sensores.

5.2.3. Diseño de los casos de uso: Diagramas de secuencia

Ya definida la estructura de los datos, deberemos pasar al diseño de los diagramas de secuencia. Estos tratan de reflejar cómo se realiza la interacción entre los mismos con las funciones que aparecen en el diagrama de clases.

Como se ha realizado a lo largo de muchos apartados del proyecto, estableceremos tres subapartados: programador, servidor y comunicaciones. En ellos se mostrarán los diagramas de secuencia más importantes del sistema.

5.2.3.1. Diagramas de secuencia en el programador

Pulsación de botones

El accionamiento de cualquiera de los botones del programador influye en dos de los actores del programador, por lo que puede resultar compleja su comprensión.

El primero de ellos, el Usuario Programador, es la persona que acciona el botón. Esta acción guardará en el autómata qué botón se ha pulsado pero no realizará ninguna acción más. Para que se lleven a cabo dichas acciones, entrará en juego el actor Tiempo Programador, que comprobará si se ha pulsado algún botón y llamará a la función CheckActions del estado en que se encuentre⁷.

⁷Es necesario recordar que todos los estados heredan de la clase virtual StateVir que obliga a implementar dicho método.

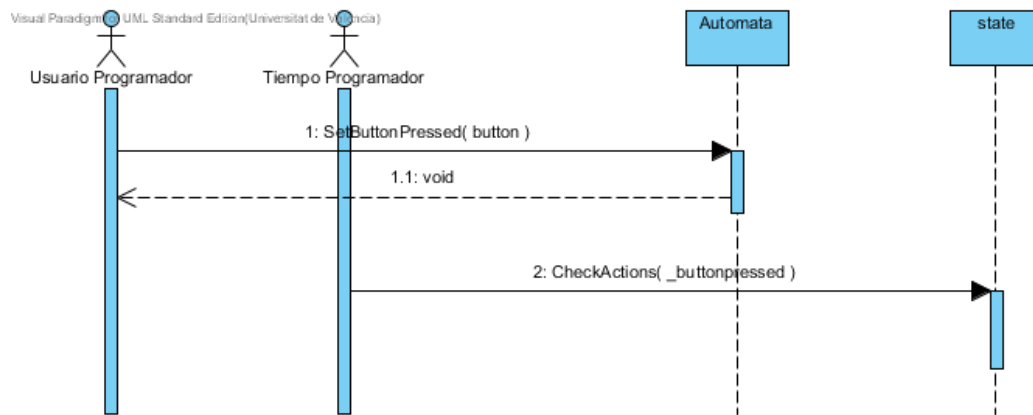


Figura 5.30: Diagrama de secuencia para el la interacción física

Comprobar acciones

Comprueba en el programador si hay acciones a realizar. Esta acción será llamada generalmente por el actor Tiempo Programador, aunque puede ser llamada de manera forzada desde otros ámbitos.

Las comprobaciones a realizar son sobre los programas (si deben o no ser activados) y sobre los sensores. Finalmente, si se ha producido algún cambio en los programas o se ha generado alguna alerta, se realizará la sincronización con el servidor.

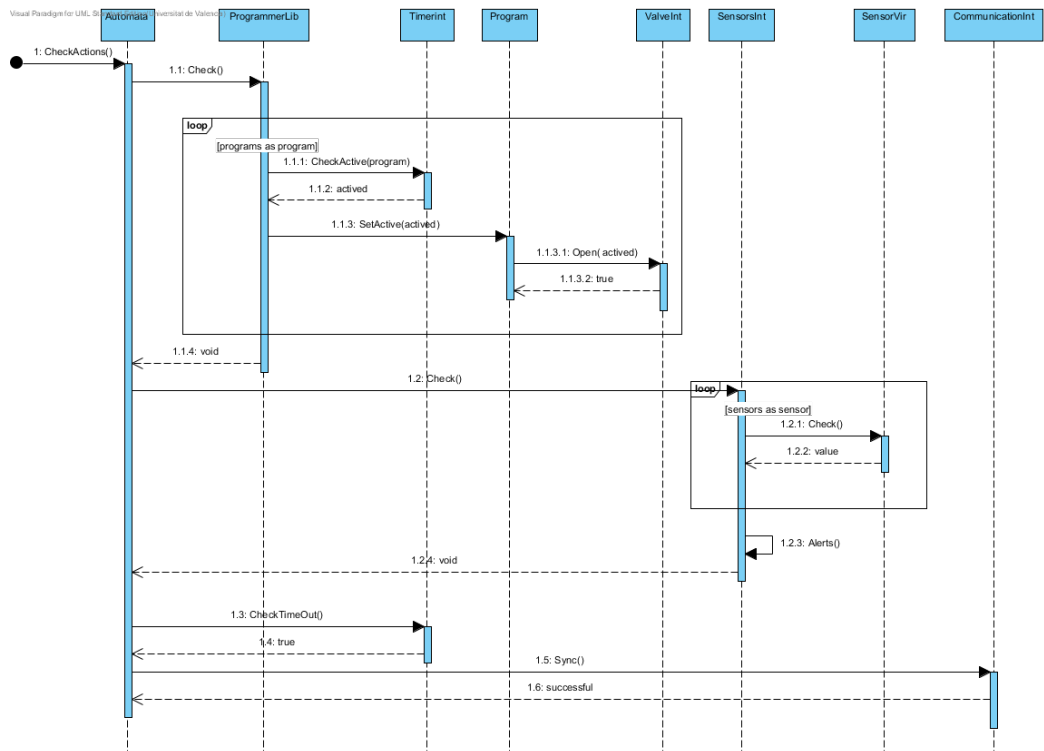


Figura 5.31: Diagrama de secuencia para la comprobación de acciones

5.2.3.2. Diagramas de secuencia en el servidor

Antes de pasar a definir los diagramas de secuencia en el servidor, será necesario definir el diagrama de secuencia interno de Symfony, el *framework* utilizado.

Symfony

En la figura 5.32 se muestra el diagrama de secuencia del funcionamiento interno de Symfony para una petición de ejemplo de www.ejemplo.com/pagina. Se trata de una aproximación para entender el funcionamiento del mismo. Para más información, consultar la bibliografía [24, p 156].

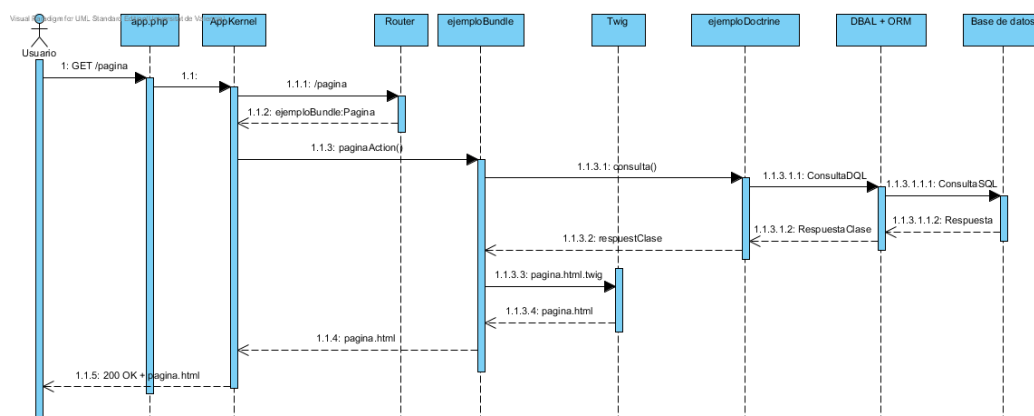


Figura 5.32: Diagrama de secuencia interno de symfony para una petición de ejemplo

Las clases `app.php`, `AppKernel`, `DBAL + ORM` son internas de Symfony y no se modificarán durante el desarrollo de nuestro proyecto.

Los siguientes diagramas de secuencia están simplificados omitiendo las clases internas de Symfony, permitiendo así mayor sencillez en el diagrama.

Symfony, además, nos permite añadir configuraciones de seguridad, lo cual nos posibilitará identificar si el usuario se encuentra logueado en la web, redirigiendo en caso de ser necesario al menú de login. Por esta razón, dicha acción no se mostrará en el resto de diagramas de flujo.

Login

En cuanto al diagrama de secuencia para la acción de autenticación o login, figura 5.33, el módulo `security.context` se encarga de realizar la comprobación en el servidor de que el usuario existe en el sistema. Posteriormente se identifica el tipo de usuario logueado para generar una u otra página.

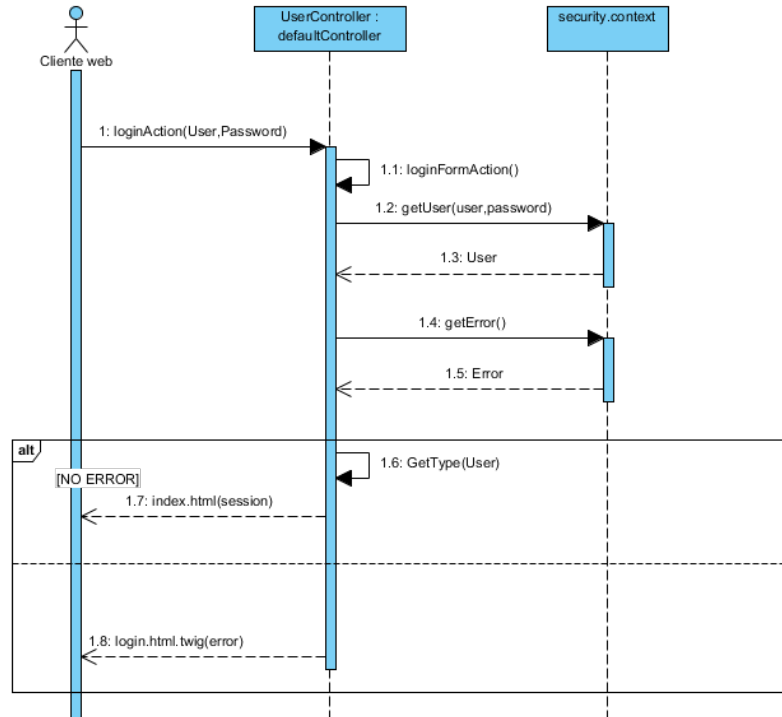


Figura 5.33: Diagrama de secuencia para el login del usuario y administrador

Lista de programadores

Obtiene la lista de programadores. Recibiremos la lista de todos los programadores asociados a un usuario. Se obtiene el usuario mediante la sesión para verificar que el programador que se muestra pertenece al mismo.

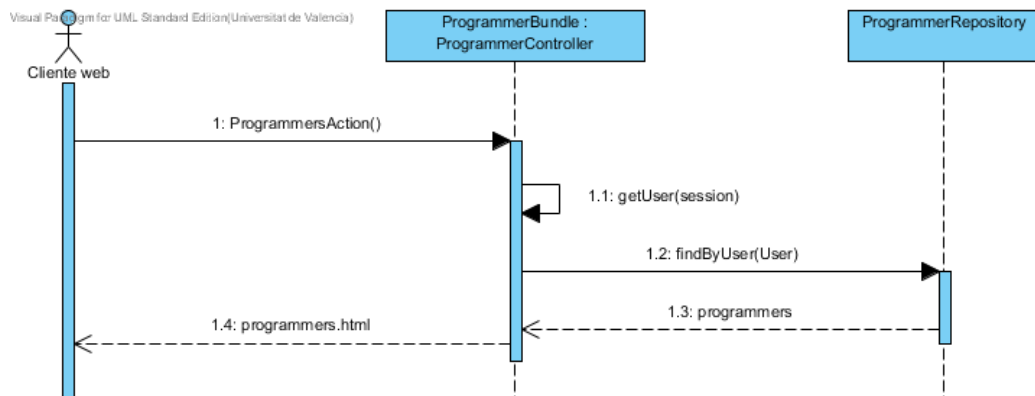


Figura 5.34: Diagrama de secuencia para obtener la lista de programadores de un usuario.

Lista de programas

Obtiene la lista de programas. Esta acción admite escoger entre la lista de todos los programas o únicamente los activos. En ambos casos se mostrarán aquellos que no han sido eliminados. Obtendremos mediante la sesión el usuario identificado, que nos permitirá mostrar solo los programas asociados a un programador asociado al mismo.

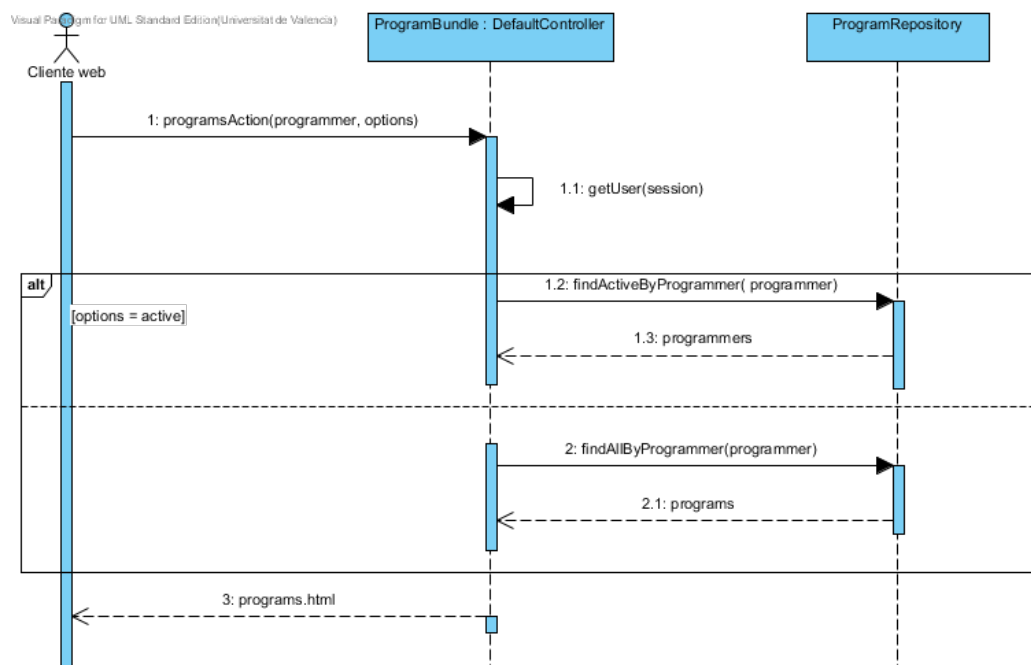


Figura 5.35: Diagrama de secuencia para obtener la lista de programas de un programador

Añadir nuevo programa

En el siguiente diagrama de flujo se indican las interacciones entre clases realizadas para añadir un nuevo programa.

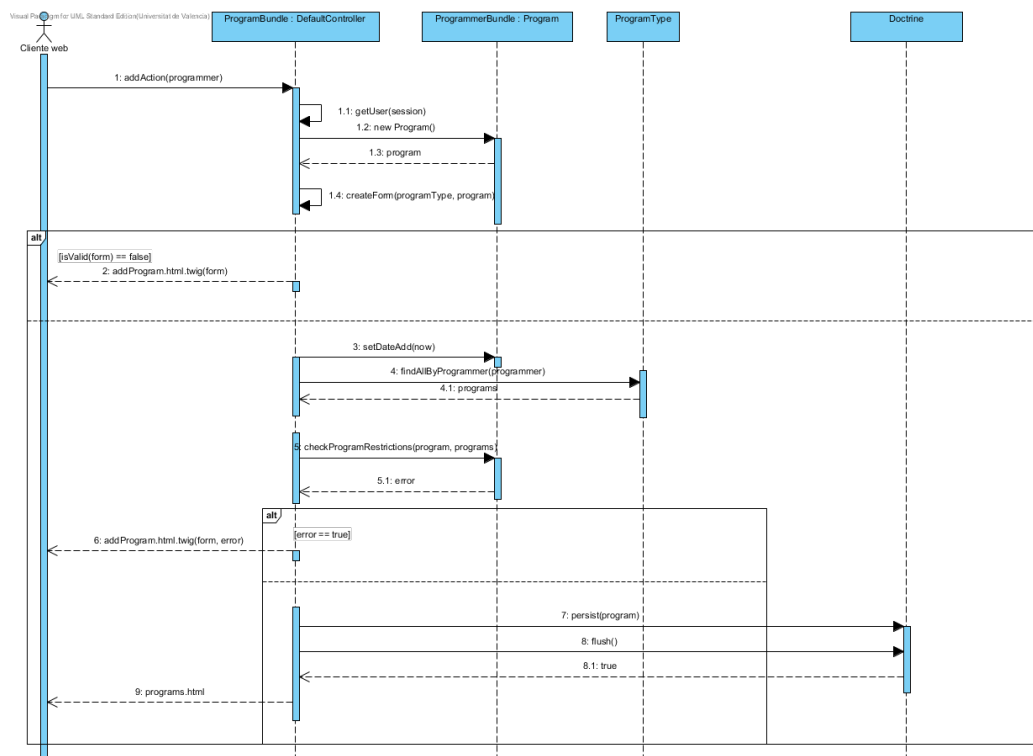


Figura 5.36: Diagrama de secuencia para añadir un nuevo programa a un programador

Como se ha establecido en las especificaciones, la sincronización de los programas viene dictada por el programador. Esto implicará que, si añadimos un nuevo programa en el servidor, el sistema no será consistente hasta que el programador realice una sincronización.

Como el programador es quien genera los nuevos identificadores locales para los programas (*idrel* en la entidad *Program* del servidor), pero necesitamos un identificador que lo diferencie de otros programas creados, se ha optado por inicializarlo con un número mayor que 10.000 (número virtualmente inalcanzable para nuestro programador y futuros programadores debido a que un único programador debería contener 10000 programas).

5.2.3.3. Diagramas de secuencia en las comunicaciones

En este apartado se mostrarán los diagramas de secuencia referentes a las comunicaciones. Estos estarán presentes desde un punto de vista tanto del programador (al realizar la petición de conexión) como de la API REST que reside en el servidor.

Envío y recepción de datos

Se encarga de gestionar el envío y la recepción de datos con el servidor mediante comunicación TCP.

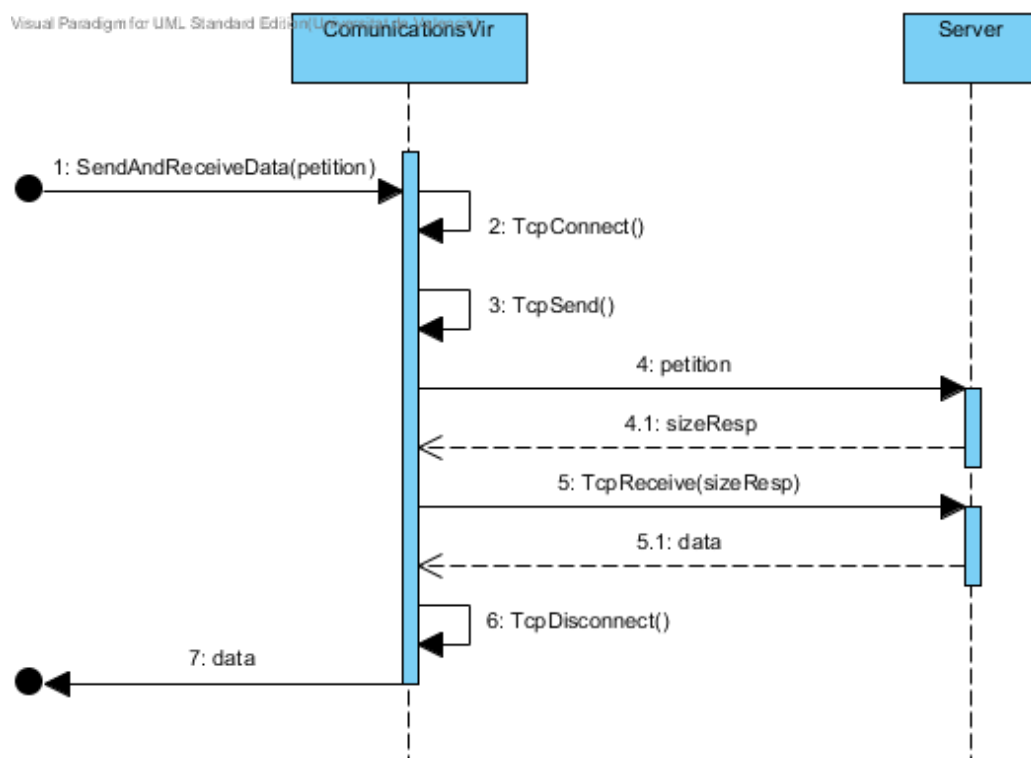


Figura 5.37: Diagrama de secuencia para el envío de datos mediante el módulo de comunicaciones

Carga de los programas

Se encarga de copiar todos los programas existentes en el servidor al programador. Para ello la memoria del programador deberá estar vacía.

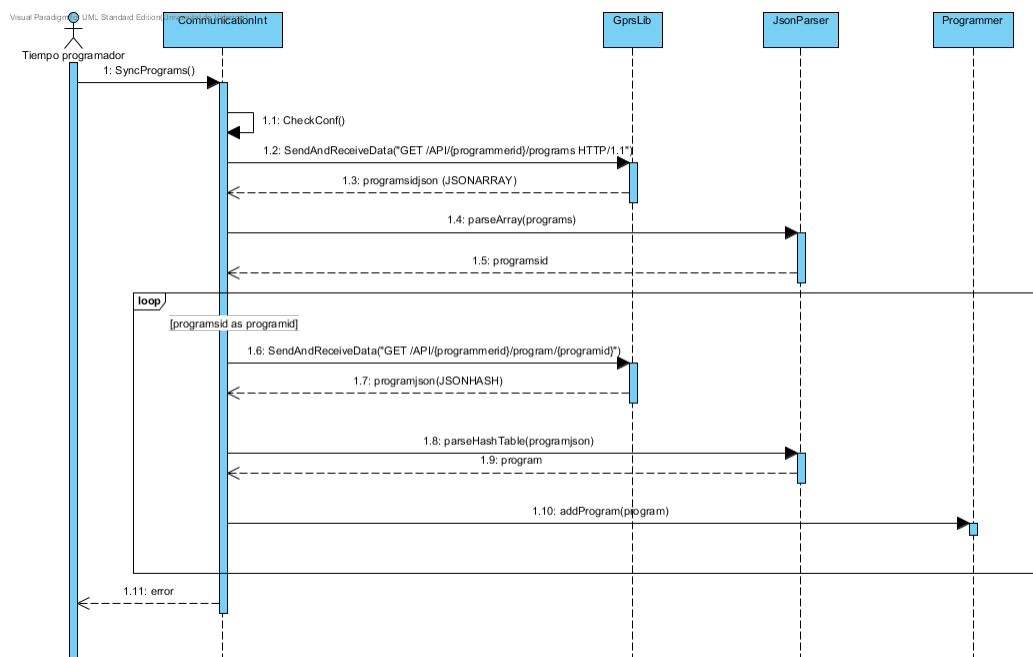


Figura 5.38: Diagrama de secuencia para la carga de todos los programas

API actualizar programa

En el siguiente diagrama de flujo se muestran las acciones realizadas cuando se recibe una petición de modificación de programa en el servidor.

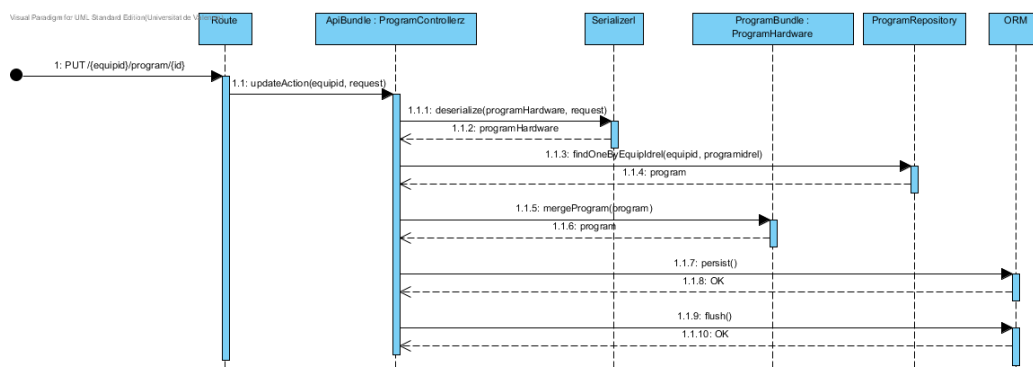


Figura 5.39: Diagrama de secuencia para la actualización de un programa mediante la API REST

5.2.4. Diseño de las interfaces de usuario

Una vez sentados los patrones de diseño en el apartado de análisis 5.1.4, procederemos a concretar su diseño.

Recordemos que durante el proceso de análisis dividimos la interfaz del programador en física y gráfica. En este caso seguiremos el mismo orden y esquema.

5.2.4.1. Diseño de interfaz del programador

Interfaz física

En nuestra interfaz se ha decidido incluir una pantalla LCD 16x2⁸ en la que se mostrarán los menús. Para interactuar con el sistema incluiremos un total de cinco botones que permitirán navegar por los menús mostrados en el LCD. El esquema de diseño del dispositivo es el mostrado en la figura 5.40.

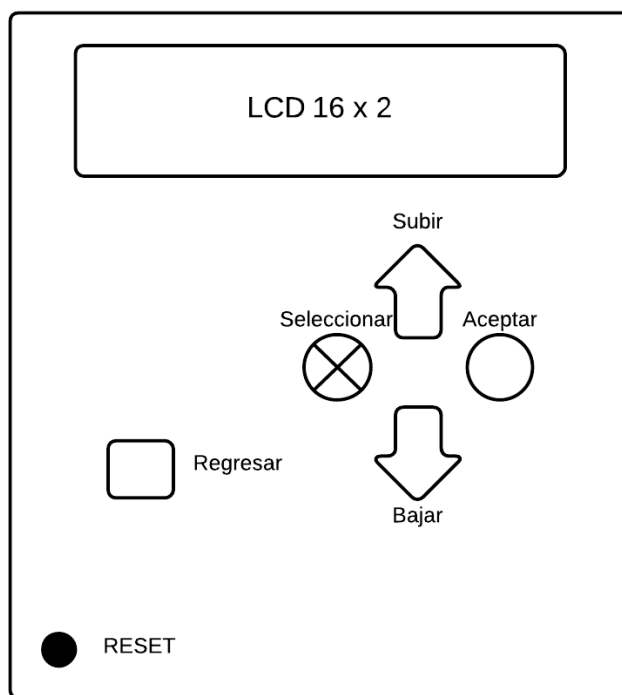


Figura 5.40: Esquema de la interfaz física.

Interfaz gráfica

Para la interfaz gráfica en el programador deberemos diseñar tres tipo de representación de datos. Ahora que conocemos el tipo de pantalla del que

⁸Se ha escogido este tipo de pantalla porque resulta mucho más económico para el desarrollo de un prototipo que crear una pantalla LCD exclusiva para nuestro programador.

dispondremos, procederemos al diseño de cada una de las interfaces.

- Menú de tipo lista

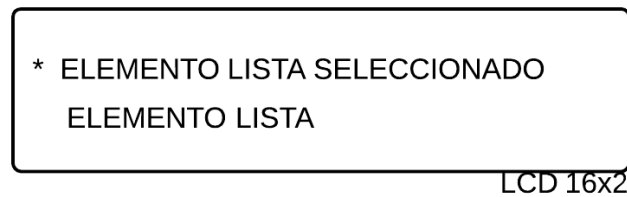


Figura 5.41: Esquema de la interfaz del programador tipo lista.

Será el tipo de menú que ejecutaremos en el caso en que necesitemos introducir datos.

A modo de cursor, tendremos un asterisco que nos indicará cuál es el elemento de la lista seleccionado.

El último elemento de la lista se mostrará en la línea de arriba, quedando la de abajo en blanco. Esta representación permitirá al usuario detectar que ha llegado al final de un menú o programa. Un caso homólogo pasará si se selecciona el primer elemento de la lista.

- Introducción de datos

En este tipo de menú mostraremos la información requerida en la primera línea guardando la segunda para realizar la introducción de los datos.

El usuario podrá incrementar y disminuir el valor a introducir mediante los pulsadores físicos incluidos en el programador.

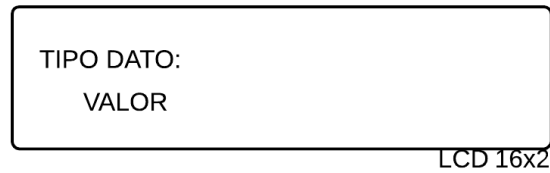


Figura 5.42: Esquema de la interfaz del programador tipo introducción de datos.

5.2.4.2. Diseño de la interfaz de la aplicación web

En el proceso de análisis concluimos que debíamos crear un total de 3 tipos de interfaz además de la vista general que contendría dos de estas.

- **Identificación**

En la figura 5.43 se muestra el esquema de la interfaz de identificación. Este panel se mostrará centrado en la pantalla sin mayores datos.

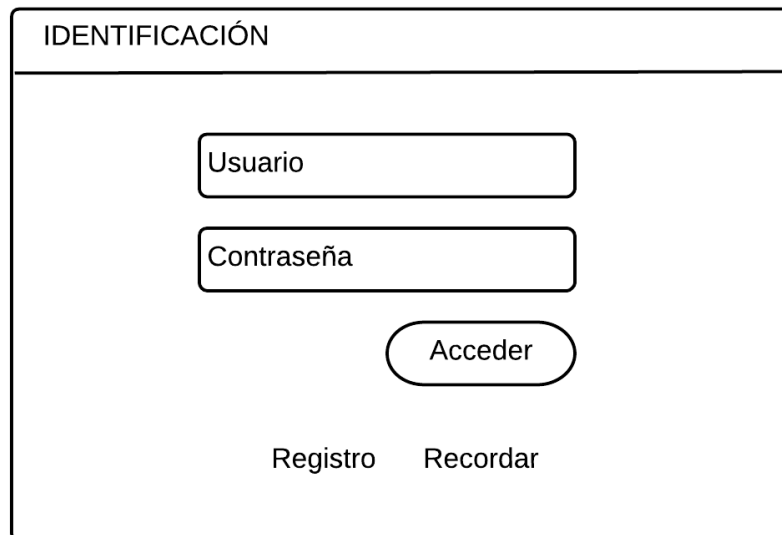


Figura 5.43: Esquema de la interfaz de identificación.

Es el panel inicial con el que se encuentra un usuario anónimo. Además de la operación de identificación, se podrán realizar las acciones de registro y recordar contraseña.

■ Lista

En la figura 5.44 se muestra el esquema para mostrar una lista vía web. A modo de ejemplo se realizado la lista de los programas en un programador, pero el diseño será similar en todos aquellos casos de uso de tipo lista, tan solo se deberán cambiar los valores introducidos en las columnas. La última columna siempre se incluirán mediante iconos identificativos las posibles acciones a realizar sobre la fila en concreto.

Inicio	Fin	Electroválvula	Tipo	Opciones
10:00	12:00	1	Riego	
10:00	12:00	1	Riego	
10:00	12:00	1	Riego	
10:00	12:00	1	Riego	

Figura 5.44: Lista de programas vía web.

■ Introducción de datos

En la figura 5.45 se muestra el esquema para la introducción de nuevos datos. De nuevo se trata de un ejemplo para el caso de uso de introducción de un nuevo programa.

El formulario 'Nuevo programa' contiene los siguientes elementos:

- Hora inicio:** Campo de texto con el placeholder 'Valor'.
- Hora fin:** Campo de texto con el placeholder 'Valor'.
- Dias:** Campo de texto con el placeholder 'Valor'.
- Electroválvula:** Campo de lista desplegable con el placeholder 'Lista'.
- Botones:** 'Cancelar' y 'Aceptar'.

Figura 5.45: Introducción de nuevo programa vía web.

Para otros casos de uso que estén englobados dentro de la estructura de introducción de datos, tan solo tendremos que cambiar el tipo de datos y el título del mismo.

■ Panel global

El panel global 5.46 contiene al resto de submenús. Estos se mostrarán en el centro de la página.

El bloque de mensajes mostrará cualquier tipo de mensaje generado por la aplicación.

En el encabezado mostraremos información al respecto de la aplicación así como el acceso a acciones comunes.

Por lo que respecta al pie de página, se facilitará el acceso a información de tipo legal, contacto y datos de interés.

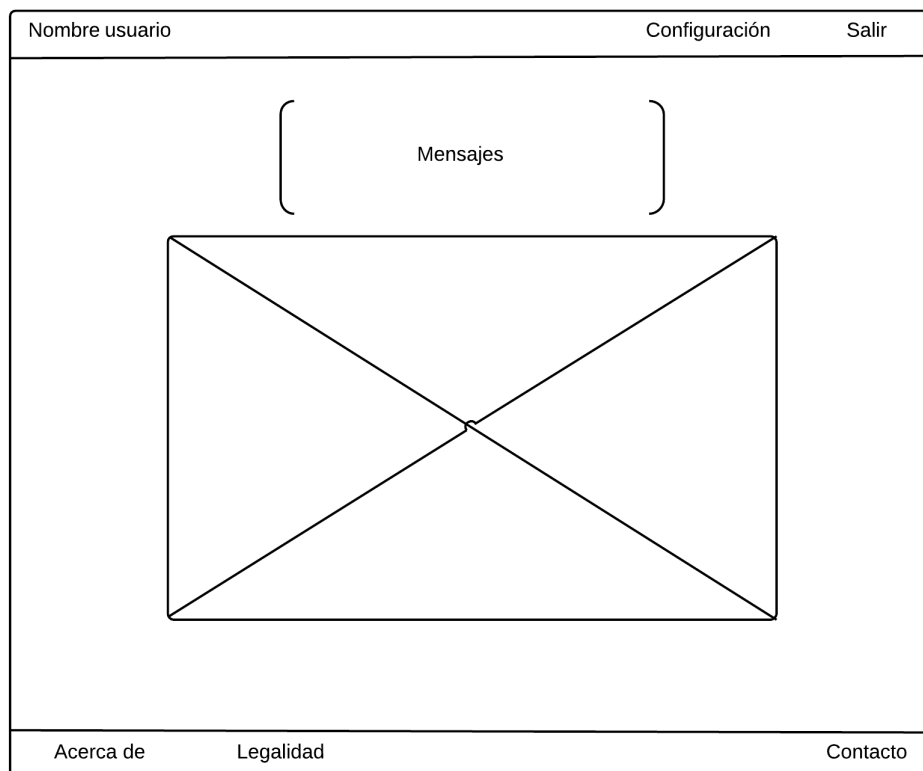


Figura 5.46: Contenedor global.

5.2.5. Diseño de la fuente de alimentación

En este apartado trataremos de definir los componentes utilizados para diseñar la fuente de alimentación que abastecerá de energía al sistema de forma ininterrumpida.

En la figura 5.47 reflejamos la estructura que tendrá el mismo y los componentes que lo forman.

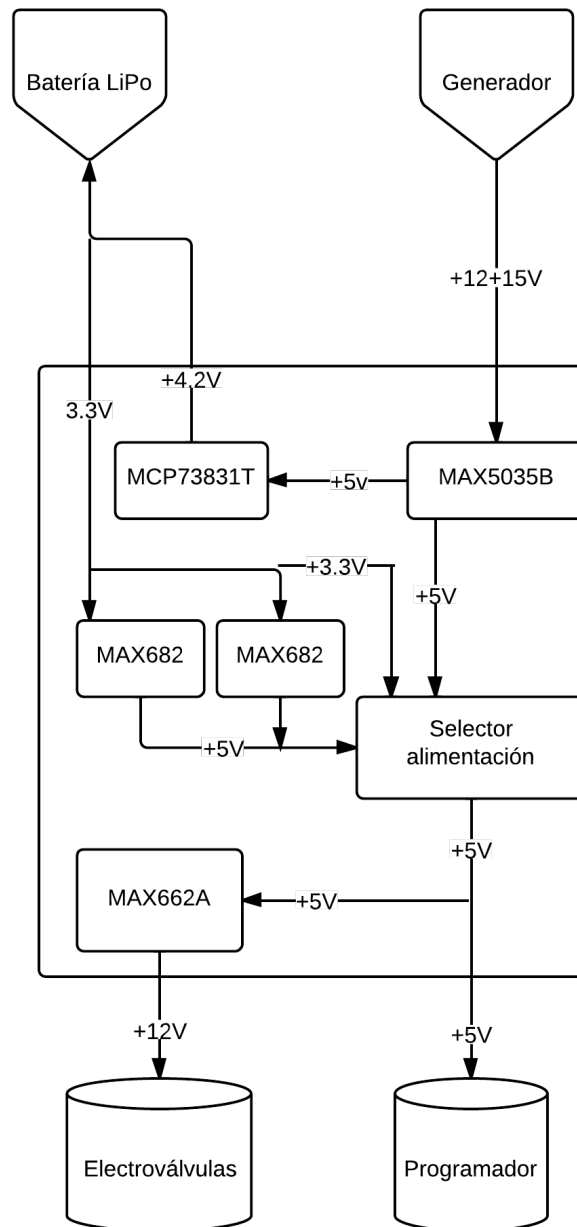


Figura 5.47: Diseño de los elementos de la fuente de alimentación

A continuación realizaremos una breve explicación de cada uno de los módulos y su función en el sistema:

- **MAX5035B:**

Convertidor DC-DC de alta eficiencia tipo Step-Down. Permite convertir corriente continua, en un rango de entrada 7.5V a 76V, a 5V de salida con un máximo de 1A.

- **MCP73831T:**

Gestor de carga para baterías de Li-Ion y Li-Polymer de una celda. Permite, mediante muy pocos elementos externos, controlar la carga de la batería. Para ello será necesaria una entrada de entre 3.75V y 6V. La carga se realizará a un máximo de 500mA.

- **MAX682:**

Se trata de una bomba de carga (*charge-pump*) que permite elevar la tensión de 3.3V (tensión nominal de la batería de li-po) a 5V. Para su correcto funcionamiento será necesaria una entrada que oscile de 2.7 a 5.5V (importante pues la tensión en la batería varía según su carga). Cada uno de los módulos es capaz de generar una corriente máxima de 250mA, por lo que será necesario el uso de dos de los módulos en paralelo para poder abastecer la energía requerida por las conexiones GPRS.

- **Selector de alimentación:**

Formado por un amplificador operacional sin realimentación que actúa como comparador y un transistor MOSFET que permite el paso de tensión desde los conversores MAX682 cuando falla la tensión provista por el generador.

- **MAX662A:**

Convertidor DC-DC que permite elevar la tensión desde a 12V/30mA dado un rango de tensiones de 4.4V a 5.5V. Esta corriente será derivada a la carga de un condensador de alta capacidad que permitirá la ejecución de las electroválvulas. Dado el bajo amperaje del mismo, será necesario un tiempo mínimo nada despreciable de carga de los mismos. En la figura 5.48 se muestra una gráfica del tiempo de carga del condensador, donde podemos observar que el siguiente disparo no podrá realizarse hasta un mínimo de 2 segundos más tarde, supuesto un condensador de 3200uF.

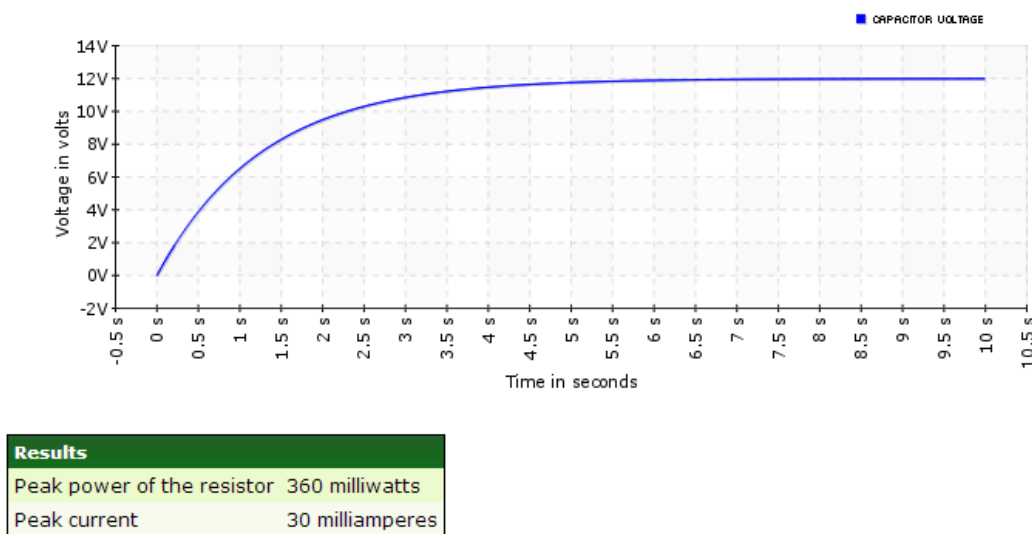


Figura 5.48: Tiempo de carga del condensador de 4200uF dado un amperaje de 30mA

5.3. Implementación

una vez terminados los apartados de análisis y diseño, ya contamos con toda la información para proceder con el desarrollo del sistema. Es por ello que, a lo largo de esta sección, explicaremos cómo se ha ido llevando a cabo el sistema hasta conseguir que sea totalmente funcional siguiendo los esquemas y diagramas anteriores.

El esquema que seguiremos en este apartado será el siguiente: en primer lugar estableceremos cuál será nuestro entorno de desarrollo. En él se explicará tanto el hardware como el software utilizado en toda la fase de desarrollo. En segundo lugar, se expondrán partes del desarrollo más relevantes, tanto programación⁹ como hardware.

5.3.1. Entorno de desarrollo

Como ya se ha dicho anteriormente, en este apartado expondremos todo el software y hardware utilizado para el desarrollo que pueden presentar problemas a la hora de la ejecución del proyecto. Además, se especificarán las versiones del mismo para evitar posibles errores a la hora de reproducir el código.

⁹Recordamos que el código completo de la aplicación, se puede consultar en el CD-ROM adjunto.

- **Entornos de desarrollo y librerías**

Atmel Studio *6.2.1153 32-bits*

Arduino Ide for Atmel Studio 6.2 *1404.1* ArduinoJsonParser *v1.2 commit 80e0a51c157e16c52d76d89c93f74fb9c62c4038*

- **Servidor**

Ubuntu *12.04.4 LTS 64-bits*

Symfony *2.5.0*

PHP *PHP 5.5.3-1ubuntu2.3*

MySQL *5.5.37-0ubuntu0.12.04.1*

- **Hardware**

Arduino MEGA ADK *R3*

Arduino UNO *R3*

GPRS Quadband Module *Hilo 1.009*

La instalación y configuración no específica de los productos no han sido tratadas en esta memoria ya que no forman parte del objetivo del proyecto y pueden encontrarse fácilmente en la red.

Configuración de dependencias en Symfony

Para utilizar las dependencias necesarias en el *framework*, se hará uso de Composer¹⁰. Esta herramienta nos permitirá definir las dependencias necesarias en el archivo *composer.json* definido en la raíz del proyecto Symfony. En la figura 5.49 se muestran las dependencias de nuestro proyecto, incluyendo la versión necesaria. Finalmente, deberemos ejecutar el comando *composer install*, para que se lleve a cabo la instalación de las mismas.

¹⁰<https://getcomposer.org/>

```

9      "require": {
10         "php": ">=5.3.3",
11         "symfony/symfony": "2.5.*",
12         "doctrine/orm": "~2.2,>=2.2.3",
13         "doctrine/doctrine-bundle": "~1.2",
14         "twig/extensions": "~1.0",
15         "symfony/assetic-bundle": "~2.3",
16         "symfony/swiftmailer-bundle": "~2.3",
17         "symfony/monolog-bundle": "~2.4",
18         "sensio/distribution-bundle": "~3.0",
19         "sensio/framework-extra-bundle": "~3.0",
20         "incenteev/composer-parameter-handler": "~2.0",
21         "symfony/intl": "2.6.*@dev",
22         "doctrine/doctrine-fixtures-bundle": "dev-master",
23         "symfony/serializer": "dev-master"
24     },
25 }

```

Figura 5.49: Dependencias en archivo composer.json

Permisos en los directorios

Para un correcto funcionamiento del servidor, Symfony necesita tener permisos de escritura en el directorio *log*. En la figura 5.50 se muestran las acciones necesarias para permitir dicho acceso a Symfony.

```

7      #####
8      # Folder permissions
9      HTTPDUSER=$(ps aux | grep -E '[a]pache|[h]ttpd|[l]www|[w]ww-data|[n]ginx' | grep -v root | head -1 | cut -d\ -f1)
10     setfacl -R -m u:$HTTPDUSER:rwx -m u:whoami:rwx app/cache app/logs
11     setfacl -dR -m u:$HTTPDUSER:rwx -m u:whoami:rwx app/cache app/logs
12

```

Figura 5.50: Estableciendo permisos en el directorio de log

Estas y otras configuraciones han sido automatizadas mediante el archivo *initialize.sh* incluido en el CD que facilitará la configuración del servicio en una nueva máquina.

5.3.2. Detalles de programación

Una vez establecido el entorno de desarrollo, entraremos en detalle en aquellos aspectos más relevantes de la programación tanto del servidor como del programador. Como es de esperar, este apartado estará dividido en el apartado del programador y el del servidor, ya que resultará totalmente distinto su enfoque.

5.3.2.1. Detalles de programación en el programador

Recordamos que el desarrollo del programador se ha realizado con C++. Es necesario prescindir de librerías estándar como la std debido a su gran tamaño. La versión del lenguaje es completa¹¹ por lo que no hay problemas al realizar herencia, definir clases virtuales, etc.

Recepción de datos del servidor

El método *ATMessageReceive* ha sido alguno de los más problemáticos. El problema reside en las limitaciones impuestas tanto en el módulo GPRS como por la placa Arduino utilizada. Para poder entenderlas mejor, en la figura 5.51 se muestra un ejemplo de una petición de los programas recibida sin alterar.

```
1 HTTP/1.0 200 OK
  Date: Wed, 02 Jul 2014 11:30:32 GMT
3 Server: Apache/2.4.6 (Ubuntu)
  X-Powered-By: PHP/5.5.3-1ubuntu2.3
5 Cache-Control: private, must-revalidate
  pragma: no-cache
7 expires: -1
  Connection: close
9 Content-Type: application/json
11 [5,1,3,5,10] --EOF--Pattern--
```

Figura 5.51: Información RAW recibida de una petición al servidor

Como se puede observar, los datos recibidos incluyen las cabeceras HTTP, la mayoría de ellas prescindibles para nuestro propósito. Este texto de más incrementa el tamaño de la cadena recibida, por lo que será importante seleccionar los datos antes de ser guardados en la escasa memoria del microcontrolador.

Dado que el módulo GPRS tan solo nos permite realizar control de flujo por hardware pero Arduino no lo incorpora, será necesario procesar la información antes de ser recibida. Esto es posible gracias a que el baudrate de la comunicación serie ha sido configurado a 9600 bps, lo que significa que la velocidad de transmisión es de 1.04 ms, velocidad muy superior a la de ejecución de instrucciones del microcontrolador.

¹¹En otros microcontroladores su compilador oficial tan solo soportaba versiones de Embedded C++, una versión del mismo pero con funcionalidades podadas.

En la figura 5.52 se muestra una parte de la función `ATMMMessageReceive`, encargada de la recepción de los datos.

```

1  const String responseHeaderPattern = "HTTP/";
   const String beginContentPattern = "\r\n\r\n";
3  const String endPattern = "—EOF—Pattern—";

5  Serial1.println( "AT+KTCPRCV=" + indexTcp + "," + _messageSize );

7  for ( int i = 0; !Serial.available() && i < timeOut/waitTime; i ++ )
   delay( waitTime );

9  //////////////////////////////////////
11 // Read HTTP RESPONSE
   //////////////////////////////////////

13 // DETECT HTTP/ patern
   while ( Serial.available() > 0 )
15 {
   incomingByte = Serial.read();

17   if ( incomingByte == responseHeaderPattern[cont] )
19     cont ++;
   else
21     cont = 0;

23   if ( cont >= responseHeaderPattern.length() )
       break;

25   for ( int i = 0; !Serial.available() && (i < timeOut/waitTime); i ++ )
27     delay( waitNext );
   }

```

Figura 5.52: Código para la recepción de datos

Entre la línea 1 y 3, se muestran los patrones que se desean identificar. En el código de ejemplo se procederá a la identificación de la cadena `HTTP/` que dicta el tipo de respuesta.

. Como se puede observar en el bucle de las líneas 7 y 8, deberemos permanecer a la espera de la llegada de datos para poder continuar. Dicha espera debe estar acotada en el tiempo ya que se debe contemplar la posibilidad de un fallo. El tiempo a esperar viene definido por la variable `waitTime` y esta a su vez viene establecida por las especificaciones del módulo GPRS.

Una vez se han empezado a recibir datos, pasaremos a la identificación de la cadena, saliendo del bucle una vez se haya identificado todo el patrón.

Finalmente, dado que el procesamiento del código es más rápido que la recepción en serie, deberemos permanecer a la espera (también por un tiempo limitado, por si ocurriese un fallo durante la recepción) del siguiente byte.

El código para identificar el resto de patrones será similar, así como el código para extraer los valores deseados.

Máquina de estados: el autómeta

Las acciones a realizar por el usuario en el programador vienen dictadas por una máquina de estados. Esta está implementada en la clase *Autómata*, la cual se encarga de llamar a los diferentes estados.

Dentro de esta clase existen distintos métodos. El primero de ellos, *CheckActions()* figura 5.53, es el que es llamado por el temporizador cada *step* de tiempo. Cada vez que esta función se ejecuta, se comprobará (lineas 6-10) si se debe accionar alguna válvula, el estado de los sensores y si se ha superado el timeout máximo de comunicación con el servidor. En caso de que alguna de las comprobaciones resulte positiva (en el caso de los sensores, que se genere alguna alerta), se procederá al envío de la información al servidor.

```
void Automata::CheckActions()
{
    if ( _state != NULL && _state->nextState != -1 )
        _nextState = _state->nextState;

    if ( _actualState != Resources::STATE_INIT )
    {
        if ( SensorsInt::Check() || ProgrammerLib::Check() || TimeInt::Check() )
            SensorsInt::Send();
    }

    if ( _nextState != _actualState )
    {
        delete _state;
        _state = NULL;

        switch ( _nextState )
        {
            case Resources::STATE_INIT:
                _state = new InitState();
                break;

            case Resources::STATE_ADD:
                _state = new AddState();
                break;

            ....

        }

        _actualState = _nextState;
    }
    else if ( _buttonPressed != Resources::BUTTON_NONE )
    {
        if ( _state != NULL )
            _state->CheckActions( _buttonPressed );
    }
}
```

Figura 5.53: Código del método *CheckActions* definido en la clase *Automata*

Seguidamente se comprueba si se ha realizado una petición de cambio de estado. Las peticiones para realizar un cambio de estado se efectúan mediante el método `SetNextState`. En caso de efectuarse un cambio, se destruirá el estado actual y se creará el nuevo estado, además de que se reseteará la variable que almacena la petición de cambio.

Finalmente, se comprueba si el usuario ha interactuado con la interfaz física, líneas 33-36. En caso de haber accionado algún pulsador, se llamará a la función `CheckActions()` del estado con la id del mismo. Para accionar el *flag* de pulsación de botón, se realizará mediante el método `SetButtonPressed`, en el que se pasará como parámetros el id del mismo.

Máquina de estados: los estados

Como se ha podido observar, en el código anterior (figura 5.53) la llamada al método `CheckActions( _buttonPressed )` se realiza independientemente del estado en el que se encuentre. Esto es posible debido a que todos los estados definidos en el programador heredan de la clase `StateVir`, figura 5.54, en el que aparece definido ese mismo método como virtual puro, lo que obligará a reescribirlo en las clases hijas.

```
class StateVir
2 {
  public:
4   StateVir ()
    {
6     nextState = -1;
      _selected = 0;
8   }

10   virtual void CheckActions( int & button ) = 0;

12   public:
      int nextState;
14
16   protected:
      int _selected;
};
```

Figura 5.54: Código referente a la clase `StateVir`

En la figura 5.55 se muestra el esquema de funcionamiento de la herencia. Esta estructura de herencia delega en el sistema la elección del estado al que llamar, por lo que evitará posibles fallos al mantener el código.

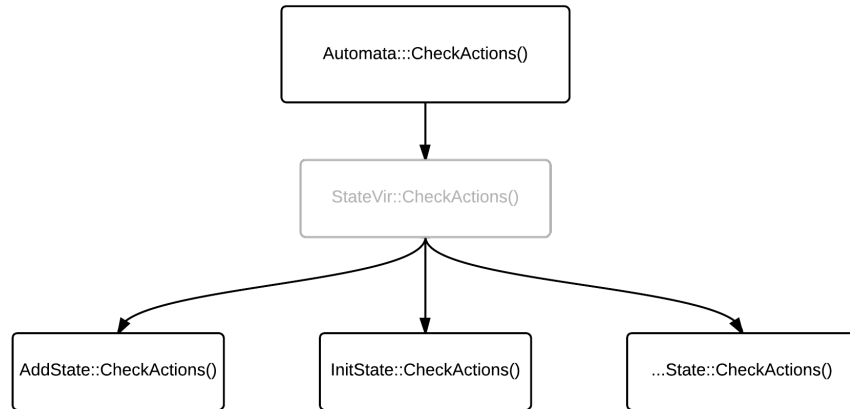


Figura 5.55: Esquema de herencia de los estados

Estructura: código genérico y herencias

Siguiendo el ejemplo del apartado anterior, se ha procurado que la estructura del sistema sea lo más genérica posible. La razón de ello es que, a la hora de añadir nuevos periféricos, el código a modificar en el núcleo sea mínimo.

En la la figura 5.56 se muestra un esquema del orden seguido para crear las clases. Podemos observar que resulta importante que estén en mayúsculas y minúsculas, pues contribuirá a identificar de qué tipo son.

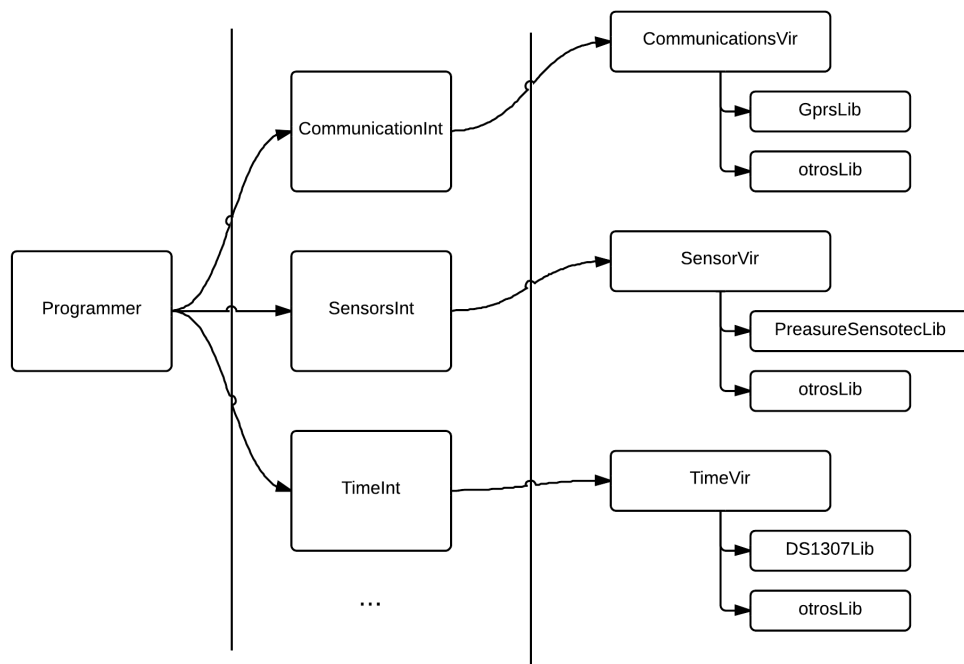


Figura 5.56: Estructura del sistema

- *claseInt*

Las clases con este sufijo contendrán métodos *static*. Será necesario que se inicialicen mediante el método `Init()` durante la configuración inicial del programador. Al ser de tipo *static* no deberá ser instanciada cuando se desee acceder a sus métodos.

Definiremos clases de este tipo como contenedores de instancias del mismo tipo. Así pues, agrupará tipos de sensores, tipos de obtención horaria, tipos de comunicaciones, etc. que se instanciarán durante el proceso de inicialización. En la figura 5.57 podemos ver un ejemplo de la función de inicialización de la interfaz de comunicaciones. En ella se instancia el módulo GPRS (el único disponible actualmente).


```

1  //////////////////////////////////////////////////
2  // Variables
3
4  CommunicationsVir * CommunicationInt::_communication = NULL;
5
6  ...
7
8  bool CommunicationInt::Init( const int communication )
9  {
10
11     switch ( communication )
12     {
13 #ifdef PER_GPRS
14     case GPRS:
15         _communication = new GprsLib( _pinCode , _apn , _user , _passwd );
16
17         if ( !_communication->Config( _server , _port ) )
18         {
19             PhysicalInterfaceInt::PrintError( "ERROR GPRS CONF" );
20             return false;
21         }
22
23         break;
24 #endif
25     }
26
27     return true;
28 }
29

```

Figura 5.57: Código de el método Init() en la clase communicationInt

■ claseVir

Se trata de las clases de las que heredarán un mismo tipo periféricos. En ella están definidos métodos de tipo virtual puro que obligarán a ser reescritos en las clases hijas.

Este tipo de estructura, permite crear un puntero de este tipo en las clases de tipo interfaz vistas en el punto anterior, el cual se instanciará con una clase de tipo Lib.

En la figura 5.58 podemos observar cómo se realiza la llamada a *SendAndReceiveData(...)* de *_communication* independientemente de la clase en la que esté instanciada.

```

1 bool CommunicationInt::GetPrograms()
2 {
3     char response[50];
4     _communication->SendAndReceiveData( "GET /api/6961
5         f4ac90b5ec8a8cddb33bad2f7f2/programs HTTP/1.0", response );
6     ...

```

Figura 5.58: Código de el método GetPrograms en la clase communicationInt

■ *claseLib*

Las clases de tipo Lib serán aquellas heredadas de las anteriores, las tipo Vir. Estas tratan a un tipo de periférico específico. Tan solo se debe poder acceder a ellas mediante los métodos virtuales del padre.

En la figura 5.60 se muestra parte de la definición de los métodos en *gprsLib.h*. En él se puede observar cómo hasta la línea 10 son públicos y se corresponden con los definidos en la clase padre 5.59. El resto de métodos pasan a ser de tipo privado y serán para la organización interna de la clase.

```

1 class CommunicationsVir
2 {
3     public:
4         virtual bool Config( const char * host, const char * port ) = 0;
5         virtual bool SendData( const char * ) = 0;
6         virtual bool SendAndReceiveData( const char * data, char * receivedMessage
7             ) = 0;
8         virtual bool EndTransmission() = 0;
9     };

```

Figura 5.59: Código de la clase padre communicationsVir

```

2  class GprsLib : public CommunicationsVir {
3
4  public:
5      GprsLib( const char * pinCode, char * apn, char * user, char * passwd );
6      // Metodos virtuales heredados
7      bool Config( const char * host, const char * port );
8      bool SendData( const char * data );
9      bool SendAndReceiveData( const char * data, char * receivedMessage );
10     bool EndConnection(){ return false; }
11
12     // Metodos gestion interna
13 private:
14     bool TcpConnect();
15     bool TcpDisconnect();
16     bool TcpSend( const String & data, String & response );
17     bool TcpReceive( String & response, int & responseSize );
18     ...

```

Figura 5.60: Código de la clase GprsLib

Para cumplir con las especificaciones, se ha procurado aislar al máximo el sistema. Es por ello que tan solo aparecen llamadas a funciones específicas de la plataforma en las clases de tipo Lib. Queda pendiente para un futuro crear un *wrapper* o capa de abstracción que encapsule todas las llamadas necesarias al sistema, reduciendo así la dependencia con el sistema y facilitando la migración a otras plataformas.

Resources.h: Fichero de recursos comunes y configuraciones

En este fichero se definen dos namespace:

- Resources

Este actúa como contenedor de clases de poca relevancia¹² tales como *DateTime*, *Time*, etc. además de incluir definiciones de interés global. Como se puede observar en la figura 5.61, aparecen las definiciones que permiten identificar los distintos estados, idiomas y botones. En ningún caso se hace referencia a cuestiones específicas de la plataforma, para ello se definirá el siguiente namespace.

¹²Que actúan casi como una estructura de datos pero con constructores.

```

namespace Resources
2 {
  /** Estados del automata */
4  enum States
  {
6    STATE.CONFIG,
    STATE.INIT,
8    ...
    STATE.DISABLE,
10   TOTALSTATES
  };
12  /** Idiomas implementados */
  enum {
14    SPANISH,
    TOTALLANGUAGES
16  };
  /** Botones existentes en la placa */
18  enum Buttons
  {
20    BUTTON.NONE,
    BUTTON.UP,
22    ...

```

Figura 5.61: Código perteneciente al namespace Resources en resources.h

- Configuration

En este namespace aparece la configuración específica de la placa. Como se puede observar en el código que aparece en la figura 5.62, en él se establece el *pinout* configurado en el sistema, qué periféricos aparecen, etc. También se reflejará la id de la placa que permitirá su identificación para la API.

```

namespace Configuration
2 {
    // BOARD IDENTIFICATOR
4     #define BOARDID "2320C571CDE42099B0DF4FDE76E7534"

    // RESTRICTIONS
6     #define MAXPROGRAMS 10
8     const int MAXVALVES = 4;

10    // Pinout

12    #ifdef ARDUINO
        // SENSORS
14        const int PIN_PREASURESENSOR_IN = 0;

        // VALVES
16        const int PIN_VALVES[] = { 5, 6, 8, 9 };

        // GPRS MODULE
20        #define PIN_GPRS_SWITCH 13

        // LCD
22        #define PIN_LCD_D4 5
24        #define PIN_LCD_D5 4
        #define PIN_LCD_D6 3

26        ...

28        // Devices

30        #define PER_LCD162
32        #define PER_GPRS
        ...

34    #endif

```

Figura 5.62: Código perteneciente al namespace Configuration en resouces.h

Json: Formato de transferencia

La implementación de Json en Arduino ha sido relativamente sencilla, puesto que la deserialización ha sido implementada por una librería desarrollada para c++ por un tercero¹³. Esto nos permite convertir la cadena de texto recibida por la clase de comunicaciones a una estructura de tipo JsonArray o JsonHashTable. Llegados a este punto, queda el paso de conversión de una clase de c++ a una estructura JSON y de una estructura de tipo JsonHashTable a una clase c++. Estos puntos han sido resueltos forzando al programador a definir dos métodos en las clases que queramos que sean de tipo serializable: *GetJson* y *SetJson*. En la figura 5.63 se muestra el ejemplo de ellos para la clase programa.

¹³<https://github.com/bblanchon/ArduinoJsonParser>

```

1 class Program {
2     public:
3         ...
4
5         char * getJson()
6         {
7             char buffer[100];
8             sprintf ( buffer, "{ \"id\":%i, \"weekdays\":\"%s\", \"startTime\":%i, \"
9             stopTime\":%i, \"valveId\":%i, \"active\":%d}",
10                 _id,
11                 _weekdays,
12                 _startTime.getTimeInSeconds(),
13                 _stopTime.getTimeInSeconds(),
14                 _valveId,
15                 _active
16             );
17
18             return buffer;
19         }
20
21         void setJson( JsonHashTable hashTable )
22         {
23             _valveId = hashTable.getLong( "weekdays" );
24             _startTime.setTimeInSeconds( hashTable.getLong( "startTime" ) );
25             _stopTime.setTimeInSeconds( hashTable.getLong( "stopTime" ) );
26             _active = hashTable.getBool( "active" );
27         }
28
29         ...

```

Figura 5.63: Código perteneciente a la clase Program

Como podemos observar, el método *GetJson* devuelve una cadena de texto siguiendo la sintaxis de JSON con todos sus atributos. Esta función permite ser llamada en el paso previo al envío de información en la clase de comunicaciones, facilitando su serialización.

En el caso de querer rellenar la instancia del objeto con una instancia de tipo *JsonHasTable*, haremos uso del método *SetJson*. En él se extraen los datos de la tabla hash almacenándose en los atributos de la misma clase.

Librería GPRS

La librería GPRS ha sido una de las partes que más tiempo han consumido durante el desarrollo.

En esta librería se han tenido que implementar las comunicaciones a bajo nivel. Para ello se ha tenido que consultar el manual de instrucciones de el módulo gsm utilizado¹⁴. En él aparecen todas las instrucciones AT disponibles para efectuar las operaciones de conexión así como de configuración del

¹⁴Se puede consultar en el CD-ROM incluido [71]

dispositivo.

Una de las principales razones del coste de desarrollo de esta clase viene dado por el error cometido a la hora de interpretar las respuestas, además de la escasa información en la red sobre el mismo más allá del anterior manual citado.

Como hemos dicho previamente, el error se producía a la hora de interpretar los mensajes recibidos vía serie por el módulo. Estos se daban debido a que el módulo ofrecido por Sagem viene preconfigurado de fábrica con la opción *verbose* y *echo* activadas. La primera de ellas obliga al módulo a comunicarse de una forma comprensible para el ser humano, que resulta mucho más pesada y con tendencia a fallo. En el ejemplo incluido en la figura 5.64 vemos cómo aparecen cadenas del tipo OK y CONNECT además de los espacios y retornos de carro incluidos antes y después de las cadenas (doble en el caso previo al CONNECT). Además está activado el modo echo, que repite el mensaje enviado por el servidor.

```
1 AT
3 OK
  AT+CPIN=" 5248"
5 OK
7 AT+CGATT=0
9 OK
  AT+KCNXCFG=0,"GPRS","internet","",""
11 OK
13 AT+KTCPCFG=0,0," 90.175.132.220",80
15 +KTCPCFG: 1
17 OK
  AT+KTCPCNX=1
19 OK
21 AT+KTCPSND=1,55
23
25 CONNECT
27 OK
  +KTCPClose: 1,8
  +KTCPCData: 1,513
29 AT+KTCPClose=1,1
31 OK
```

Figura 5.64: Texto RAW recibido con verbose y echo activados

Si la respuesta la comparamos con la de la figura 5.65 con la recibida después de desactivar dichos modos mediante los comandos *AT+ATV0* y *AT+ATE0*, veremos que, aunque es mucho menos comprensible para nosotros, resultará mucho más sencilla de interpretar para el Arduino.

```
1 0000+KTCPCFG: 1
  0010+KTCP_NOTIF: 1,8
3 +KTCP_DATA: 1,513
  0
```

Figura 5.65: Texto RAW recibido con verbose y echo desactivados

Sobre la librería implementada, cabe indicar que su implementación no ha sido completa y se ha limitado el desarrollo de aquellos comandos AT necesarios para el proyecto junto a las respuestas relevantes del mismo.

5.3.2.2. Detalles de programación en el servidor

En este apartado trataremos de aclarar algunos aspectos relevantes de programación en el servidor. Recordemos que esta ha sido realizada en PHP5 sobre el *framework* Symfony2.

HardwareProgram: facilitando la comunicación

Como se definió durante el apartado de análisis de datos 5.1.3.1 (página 109), los datos almacenados en el programador forman una pequeña parte de la información almacenada en el servidor. Esto implicaba, entre otras cosas, que algunas de las clases en el programador contasen con menos atributos que en el servidor. Esta diferencia en la información provoca cierta dificultad a la hora de realizar la serialización y deserialización de los objetos.

Para resolver este problema, se ha generado la clase auxiliar *HardwareProgram*, que permite convertir un objeto de tipo Program almacenado en el servidor a uno con los mismos atributos y referencias existentes en el programador y a la inversa.

Para realizar dichas acciones se vale de las funciones *setProgram* y *mergeProgram* (figura 5.66). La primera de ellas se encarga de inicializar una nueva instancia de *HardwareProgram* a partir de una instancia de Programa. Este método se llamará durante el proceso de serialización. *mergeProgram*, junta una instancia de tipo Programer con otra de tipo HardwareProgram, devolviendo una instancia de tipo Programa. Resulta evidente que este segundo se utilizará en el proceso de incorporación de un programa enviado por el programador.


```

2 <?php
3
4 public function setFromProgram( Program $program )
5 {
6     $this->id = $program->getIdrel();
7     $this->startTime = $program->getStartTime();
8     $this->stopTime = $program->getStopTime();
9     $this->weekdays = $program->getWeekdays();
10    $this->active = $program->getActive();
11    $this->valveid = $program->getValve()->getIdRel();
12 }
13
14 public function mergeProgram( Program $program, $valve )
15 {
16     $program->setIdrel( $this->id );
17     $program->setStartTime( $this->startTime );
18     $program->setStopTime( $this->stopTime );
19     $program->setWeekdays( $this->weekdays );
20     $program->setActive( $this->active );
21     $program->setValve( $valve );
22     return $program;
23 }

```

Figura 5.66: Método mergeProgram en clase HardwareProgram

En el siguiente apartado observaremos el uso del método *mergeProgram*.

API: recibiendo un nuevo programa desde el programador

En la figura 5.67 observamos parte del código referente al método *newAction*. Este método permite incorporar a la base de datos del servidor un nuevo programa creado desde el programador.

```

<?php
2 public function newAction( $equipid , Request $request )
{
4     ...

6     $data = $request->getContent();
    $hardwareProgram = $serializer->deserialize( $data , 'RegAigua\
    ProgramBundle\HardwareEntity\HardwareProgram' , 'json' );

8     $programmer = $em->getRepository( 'ProgrammerBundle:Programmer' )->
    findOneByEquip( $equipid );
10    if ( !$programmer )
        throw $this->createNotFoundException( "PROGRAM NOT FOUND" );

12    $valve = $em->getRepository( 'ProgrammerBundle:Valve' )->
    findOneByEquipIdrel( $equipid , $hardwareProgram->getValveid() );
14    if ( !$valve )
        throw $this->createNotFoundException( 'NO VALID VALVE' );

16    $program = new Program();
18    $program = $hardwareProgram->mergeProgram( $program , $valve );
    $program->setDateAdd( new \DateTime( 'now' ) );
20    $program->setProgrammer( $programmer );

22    $em->persist( $program );
    $em->flush();

24    $response->setStatusCode( '201' );
26    return $response;
}

```

Figura 5.67: Método NewProgram en la clase ApiBundle:ProgramController

En la línea 7 obtenemos la cadena JSON enviada por el programador. Esta función se realiza mediante el método *getContent()* que ofrece Symfony. Seguidamente, se procederá a la deserialización mediante el método *deserialize*, también ofrecido por el *framework*, obteniendo así un objeto de tipo *HardwareProgram*.

Llegados a este punto, será necesario recuperar la instancia de programador en la base de datos. Este paso se realiza durante las líneas 8-11, que termina reportando un error 404 al hardware si no se ha podido obtener.

Seguidamente se realizará el mismo proceso en la búsqueda de la electroválvula. La búsqueda se realizará mediante el identificador único del programador y el identificador relativo de la válvula recibido. Aunque este segundo no es único en la base de datos, sí lo será junto a la clave del programador.

Habiendo obtenido el programador y la electroválvula, crearemos una nueva instancia de Programa (línea 17) la cual será rellanada con los datos de hardwareProgram gracias al método *mergeProgram* explicado anteriormente.

Finalmente, completaremos los datos de fecha de creación y el programador obtenido anteriormente y procederemos a su guardado en la base de

datos.

Si todo el proceso se ha realizado correctamente, el programador debería recibir una notificación 201 indicando que se ha completado la operación.

Generando formularios

Generar un formulario para que el usuario pueda añadir un nuevo objeto resulta sencillo en Symfony.

En el código de la figura 5.68 se ejemplifica cómo se realiza el formulario para crear un nuevo programa:

```
1 <?php
2 $program = new Program();
3 $form = $this->createFormBuilder( $program )
4     ->add( 'startTime', 'time' )
5     ->add( 'stopTime', 'time' )
6     ->add( 'active', 'checkbox', array( 'required' => false ) )
7     ->add( 'valve' )
8     ->add( 'weekdays', 'choice', array(
9         'mapped' => false,
10        'expanded' => true,
11        'multiple' => true,
12        'choices' => $dayweeks
13    ) )
14     ->getForm();
15
16 $form->handleRequest( $petition );
17
18 if ( $form->isValid() )
19 {
20     ...
21
22     return $this->redirect( $this->generateUrl( 'programs', array( 'programmerid'
23         => $programmerid ) ) );
24 }
25
26 return $this->render(
27     'ProgramBundle:Default:add.html.twig',
28     array( 'form' => $form->createView(),
29         'programmerid' => $programmerid )
30 );
```

Figura 5.68: Código para la creación de un formulario con un nuevo programa

Mediante el método *createFormBuilder()* que nos proporciona Symfony podremos crear un formulario a partir de un objeto. Seguidamente (4-13) añadiremos los atributos que deseamos que se muestren, pudiendo modificar el tipo de representación. Mediante el método *handleRequest()* obtendremos de nuevo el objeto con los datos devueltos por el formulario. El siguiente paso será verificar que los campos rellenados son correctos mediante el método *isValid()*. En este paso se verificarán las comprobaciones indicadas en el

método `createFormBuilder()` además de aquellas que se añaden por defecto que provienen de las restricciones de la base de datos. Si el resultado ha sido correcto, realizaremos las operaciones necesarias y devolveremos la vista con la lista de programas (ya con el nuevo programa añadido). En caso de que no se cumplan todas las restricciones, en la línea 23, veremos que se devuelve la vista que contiene el formulario.

Cabe indicar que la primera vez que se llama al método que contiene estas líneas (*addProgram*) aún no se ha creado el formulario, por lo que no se cumple la comprobación *isValid* de modo que, mediante el return de la línea 23, generaremos el formulario.

Nuevo programa: generando idrel

Como se ha indicado en el apartado de diseño, cuando creamos un nuevo programa en el servidor, será necesario crear un número de identificador relativo *idrel* único y mayor que 1.000.

Como doctrine no permite utilizar funciones como *nextVal()* que nos permitirían obtener el siguiente identificador de una secuencia, se ha recurrido a unos disparadores de doctrine (figura 5.69) que se ejecutarán durante el proceso de guardado del método en la base de datos.

```
1 <?php
2 /**
3  * @ORM\PostPersist()
4  * @ORM\PostUpdate()
5  */
6 public function generateIdrel()
7 {
8     if ( !$this->idrel )
9         $this->idrel = $this->id + 1000;
10 }
11
```

Figura 5.69: Código para generar un nuevo identificador único

Se recurre a estos disparadores de doctrine porque, para crear el identificador único mayor de 1000, será necesario obtener la id del mismo programa. Esta operación no puede llevarse a cabo antes de que se cree la entrada en la base de datos, que es cuando se genera la misma.

5.3.3. Detalles de contrucción del hardware

Una vez descritos los detalles de la programación tanto en el servidor como en el programador, procederemos con la construcción del hardware.

Primer prototipo

Inicialmente se construyó un prototipo sobre una placa perforada (figura 5.70). Este prototipo ha sido construido únicamente para que resultase más cómoda la interconexión de los distintos periféricos utilizados y así facilitar la fase de desarrollo del núcleo del programador.

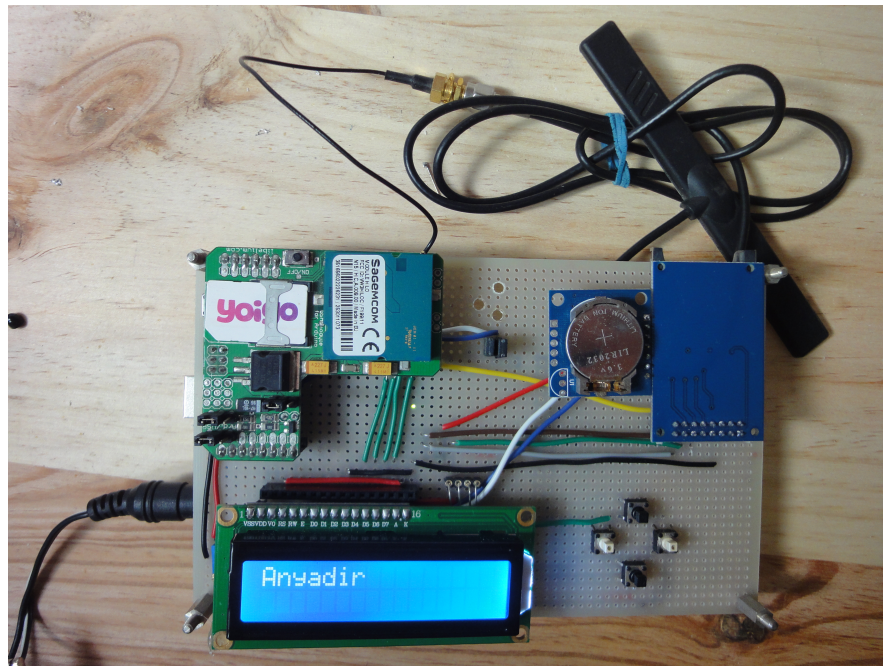


Figura 5.70: Prototipo construido para desarrollo.

Construcción de las pcb

El hardware utilizado en el prototipo final, también ha sido diseñado y construido por el educando. En la figura 5.71 se muestra el diseño de la pcb que contiene el puente H usado para activar las electroválvulas. Este será impreso en papel transparente para su posterior insolación en una pcb positiva.

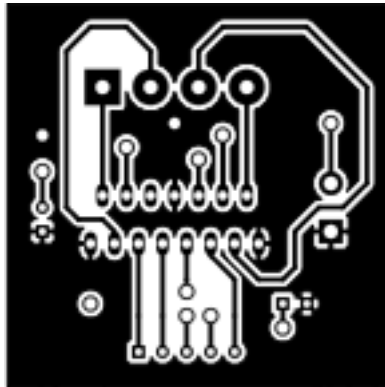


Figura 5.71: Resultado del diseño de la pcb encargada de activar las electroválvulas.

Tras el proceso de insolado (figura 5.72), se deberá revelar la pcb sumergiéndola con una mezcla de agua con sosa cáustica. A continuación, se deberá sumergir en una solución ácida que atacará el cobre que ha quedado al descubierto.

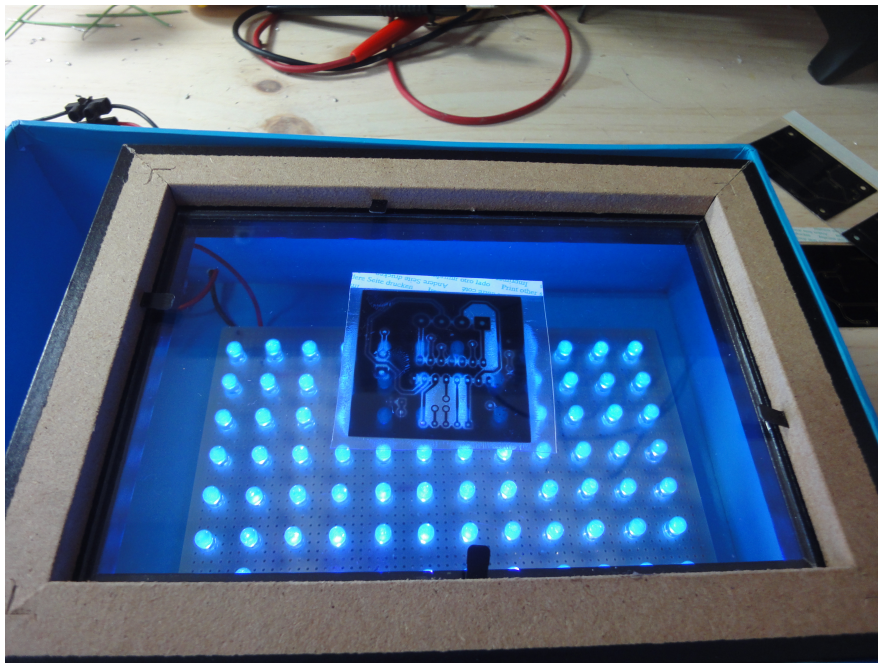


Figura 5.72: Fotolito en la insoladora.

Finalizado ya este proceso, los componentes han sido soldados obteniendo como resultado la pcb mostrada en la figura 5.73. Como se puede observar,

para abaratar costes y aumentar la sencillez en el proceso de fabricación, se ha utilizado una placa de una única cara, razón por la que ha sido necesario el uso de cables para la unión de algunas de las pistas.

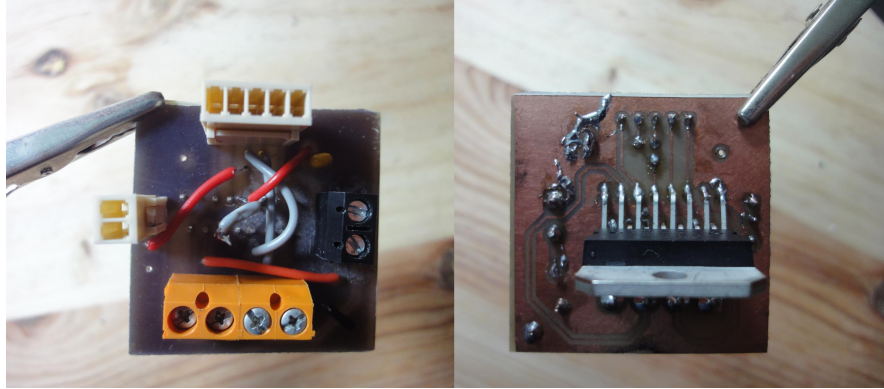


Figura 5.73: Resultado de la pcb una vez ensamblados los componentes.

Capítulo 6

Pruebas y resultados

Ya desarrollado el sistema, se procederá a la fase de pruebas. A lo largo de este capítulo se describirá las pruebas que se han realizado para asegurar el correcto funcionamiento del sistema así como los resultados obtenidos en las mismas.

Las pruebas se dividirán en tres grandes bloques. Cada uno de ellos tendrá un objetivo diferente y por ello se realizarán pruebas de diferente índole.

6.1. Descripción de las fases y pruebas a realizar

6.1.1. Descripción de las pruebas independientes

En esta fase analizaremos el programador y el servidor de forma independiente de manera que, si se produce algún fallo durante el proceso, podamos detectar más fácilmente la causa.

En esta fase se comprobará que se cumple con las especificaciones indicadas en la sección 4.3. Para ello recurriremos a diferentes tipos de pruebas:

- Pruebas funcionales
Permitirán comprobar que se ha concluido con los requisitos definidos en el alcance del proyecto en la sección 4.4.3.
- Pruebas de usabilidad
Tratarán de evaluar cómo de sencillo resulta el uso del sistema y cómo repercute en la curva de aprendizaje del usuario. También se someterá a examen el aspecto final del sistema.

Para evaluar la usabilidad del producto se pedirá a personas con diferente nivel de conocimiento tecnológico¹ que efectúen diferentes operaciones. Una vez realizadas, deberán contestar a una encuesta para ver el grado de satisfacción con el producto. Además se evaluará el tiempo que se ha requerido para realizar las operaciones.

- **Pruebas de rendimiento**

Durante las pruebas de rendimiento se someterá el producto a pruebas de estrés que permitirán calcular parámetros de rendimiento. Los parámetros a evaluar dependerán del apartado que estemos evaluando.

Procedamos pues a la explicación de las pruebas. Dado que, como se ha dicho anteriormente, las pruebas se realizarán de forma independiente, dividiremos este apartado en pruebas en el programador y pruebas para en el servidor. En ambos casos realizaremos los diferentes tipos de pruebas anteriormente citados.

6.1.1.1. Pruebas del programador

Puesto que las pruebas no se realizan en un entorno aislado, será necesario hacer una breve descripción del entorno creado para las mismas.

- **Descripción de la instalación**

El prototipo será instalado en un laboratorio. La alimentación de este se realizará a través de una fuente de alimentación simulando el generador hidroeléctrico.

- **Descripción de las pruebas de funcionalidad** Las funcionalidades a evaluar serán:

1. El usuario deberá ser capaz de configurar el programador.
 - 1.1. Se debe poder cargar la configuración desde una tarjeta SD.
 - 1.2. Se debe poder cargar la configuración manualmente.
 - 1.2.1. Configurar Hora
 - 1.2.2. Configurar Electro-válvulas
2. El usuario debe ser capaz de modificar los programas
 - 2.1. Añadir programa
 - 2.2. Eliminar programa

¹Supondremos un nivel agrario medio pues no es el objetivo evaluar la curva de aprendizaje en dicho aspecto.

- 2.3. Activar programa
- 2.4. Desactivar programa
- 3. El usuario debe poder visualizar el estado del programador
 - 3.1. Ver mensajes
- 4. El usuario debe poder accionar las electro-válvulas manualmente
 - 4.1. Abrir electro-válvula
 - 4.2. Cerrar electro-válvula
- 5. El programador debe ser capaz de comunicarse con el servidor
 - 5.1. Llegado el tiempo máximo de sincronización
 - 5.2. Efectuado algún cambio en los programas
 - 5.3. Dado algún error
- 6. El sistema debe ser capaz de detectar errores
 - 6.1. Presión de línea

■ Descripción de las pruebas de usabilidad

La evaluación sobre la experiencia del usuario tratará de evaluar los siguientes aspectos:

- En el entorno gráfico
El usuario deberá evaluar cómo de comfortable le ha resultado el entorno gráfico a la hora de realizar las configuraciones. Se evaluará así el aspecto visual del mismo, su simplicidad de uso y la tipografía y estilo utilizada.
- En el entorno físico
El usuario evaluará en este caso la comodidad al utilizar el sistema físico, refiriéndose en este caso a la hora de realizar las conexiones de electro-válvulas, sensación a la hora de interactuar con los pulsadores y grado de intuición de los mismos, visibilidad del LCD, etc.

Las operaciones que deberá realizar el usuario serán:

- Configuración manual de las electro-válvulas
El usuario deberá configurar una electro-válvula de riego y otra de abonado, conectadas en el puerto 1 y 2 respectivamente.
- Configuración manual de programas
El usuario deberá introducir tres programas (dos de riego y uno de abonado). Estos programas deberán estar habilitados.

Nombre	Edad	Perfil tecnológico	Perfil agrícola	Otros
Onofre Vento	72	Bajo	Alto	Sí
Jose Mañó	28	Alto	Medio-Alto	Varios
Carlos Reig	26	Muy Alto	Bajo	No

Cuadro 6.1: Usuarios que realizarán las pruebas de usabilidad en el programador

- Accionamiento manual de la electro-válvula de riego
El usuario deberá activar la electro-válvula de riego y posteriormente desactivarla.

En la tabla 6.1 se describe el perfil de los tres usuarios que realizarán las pruebas. Como podemos observar, el nivel de experiencia con otros programadores del mercado es distinto, lo que nos permitirá evaluar el grado de satisfacción del cliente desde distintos puntos de vista.

■ Descripción de las pruebas de rendimiento

Las pruebas de rendimiento en el hardware estarán ligadas al estado de la batería. Evaluaremos el tiempo estimado de la batería.

Las pruebas a realizar serán:

- Tiempo de duración de la batería
Evaluaremos el tiempo de duración de la batería cargada al máximo y sin realizar ningún accionamiento de electro-válvulas pero sí sincronizaciones.

6.1.1.2. Pruebas del servidor

Aunque las pruebas en el servidor siguen el mismo esquema que en el caso del programador, las pruebas a realizar para la evaluación guardan una gran diferencia.

■ Descripción de la instalación

Para llevar a cabo las pruebas realizadas en el servidor durante esta fase deberemos realizar la instalación descrita la sección de conclusiones 4.2, en la que el servidor estará instalado en una máquina virtual sobre un ordenador personal y con la que realizaremos el análisis completo.

- Descripción de las pruebas de funcionalidad Las funcionalidades a comprobar serán:

1. Acceso a las funcionalidades del sistema según el rol que tenga asignado el usuario una vez realizada la validación.
 - 1.1. El cliente debe ser capaz de cerrar la sesión impidiendo que otros usuarios puedan acceder a ella.
2. El cliente deberá poder visualizar la lista de programadores asociados a su cuenta y operar sobre ellos
 - 2.1. Configurar las electroválvulas
 - 2.2. Generar autoconfig
 - 2.3. Visualizar los eventos
3. El cliente deberá poder visualizar la lista de programas asociados a un programador, permitiendo
 - 3.1. Gestionar los programas
 - 3.2. Visualizar su histórico
4. El sistema debe ser capaz de detectar y generar errores:
 - 4.1. Generar un error de conexión en el programador si no se ha realizado el número mínimo de sincronizaciones.
 - 4.2. Detectar errores de coherencia durante la introducción de los programas.
 - 4.3. El cliente debe ser capaz de visualizar las advertencias generadas por el programador.
5. El administrador debe ser capaz de gestionar los programadores
 - 5.1. Asociando programadores a clientes
 - 5.2. Desasociando programadores a clientes
 - 5.3. Asociando tarjetas SIM a programadores
 - 5.4. Desasociando tarjetas SIM a programadores
6. El sistema debe tener un servicio de comunicación (API) que permita
 - 6.1. Sincronizar programas
 - 6.2. Sincronizar sensores
 - 6.3. Sincronizar electroválvulas
 - 6.4. Recibir avisos

■ Descripción de las pruebas de usabilidad

Para determinar la usabilidad del producto, se realizará la prueba del mismo entre los principales navegadores y plataformas del mercado.

Nombre	Edad	Perfil tecnológico	Perfil agrícola	Pruebas
Onofre Vento	72	Bajo	Alto	Cliente
Jose Mañó	28	Alto	Medio-Alto	Cliente y Administración
Carlos Reig	26	Muy Alto	Bajo	Cliente y Administración

Cuadro 6.2: Usuarios que realizarán las pruebas

En cuanto a la interacción de los usuarios con el mismo, tres usuarios de distinto nivel de conocimiento informático evaluarán el servicio a la hora de gestionar los programadores a través de la interfaz web. También se tratará de evaluar la experiencia del administrador al realizar sus funciones, aunque esta segunda guardará un menor interés debido a que es una opción totalmente minoritaria.

Las operaciones que deberá realizar el usuario serán:

1. Configuración completa de un programador
El usuario deberá configurar dos electro-válvulas, una de riego y otra de abonado. Además, deberá crear un total de tres programas: uno de abonado y dos de riego.
2. Eliminar un programa
El usuario deberá eliminar el programa de riego que contenía el programa de abonado.
3. Desactivar programa
El usuario deberá desactivar un programa.
4. Activar programa
El usuario deberá activar el programa desactivado.

Las operaciones que deberá realizar una persona con rol de administrador servicio serán:

1. Cambiar un programador de propietario
El administrador deberá desasociar y asociar un programador.
2. Cambiar una SIM de programador
El administrador deberá desasociar una tarjeta SIM de un programador y asociarla a otro programador.

En la tabla 6.2 se muestran los usuarios que realizarán las pruebas. Con ellos se pretende cubrir los diferentes perfiles de los clientes del servicio. Después de realizar las operaciones anteriormente expuestas deberán evaluar (en un rango de bajo/medio/alto) las siguientes características:

- **Aspecto visual:**
Se evaluarán la disposición de los textos, su tipografía y los colores utilizados del servicio en global.
- **Simplicidad de uso:**
En ella se evaluará la facilidad con la que han realizado las operaciones.
- **Accesibilidad:**
En este punto deberán evaluar la intuitividad con la que han sido situados los distintos menús.

■ **Descripción de las pruebas de rendimiento**

Las formas de evaluar el rendimiento del servidor serán totalmente distintas a las realizadas en el programador.

Antes de realizar dichas pruebas, la base de datos ha sido rellenada con datos aleatorios. Estos asegurarán la existencia de un total de 50 clientes, cada uno con uno o dos programadores asociados, lo que asegurará un número de programadores mayor a la cota. También se han añadido un total de dos administradores del sistema.

Para realizar las pruebas de rendimiento recurriremos a la aplicación JMeter². Este programa, mantenido por Apache y distribuido bajo la licencia Apache 2, permite a los desarrolladores evaluar el rendimiento de sus aplicaciones escritas en distintos lenguajes (PHP, Java, ASP.NET, etc.). El software cuenta con una GUI que permite al administrador configurar la batería de pruebas a realizar sobre el servicio.

A lo largo de estas pruebas trataremos de evaluar el rendimiento en global de nuestra web en distintos escenarios. Se realizarán pruebas con distinto número de usuarios hasta alcanzar la talla definida en el apartado de especificación del servidor(sección 4.3.2). Además se incluirán distintos tipos de usuarios:

- **Usuario normal:**
Se trata de los clientes del servicio que realizarán operaciones de consulta y gestión de sus programadores.
- **Programadores:**
Contempla los programadores de los clientes que desean realizar conexiones al servicio mediante la API.

²<http://jmeter.apache.org/>

- **Administrador:**

Engloba a los administradores del servicio.

- **Robots o spiders:**

Se refiere a los agregadores de buscadores y demás que analizan la red en busca de información. Estos sólo podrán acceder a las páginas públicas del servicio.

Para cada tipo de usuario se ha definido una ruta de acciones diferentes que permitirán simular la navegación por el servicio de la forma más real posible. El número inicial indica el número de veces que se repetirá la operación:

- **Usuario normal:**

- 1 - Identificación en el servicio.
/user/login
- 1 - Lista de programadores
/programmers
- 6 (Bucle):
 - ◇ 3 - Lista de programas
/programmer/8/programs
 - ◇ 1 - Histórico de programas
/programmer/8/history/programs
 - ◇ 1 - Histórico de sensores
/programmer/8/history/senses
 - ◇ 1 - Histórico de electroválvulas
/programmer/8/history/valves

- **Programadores:**

- 1 - Obtener programas
/api/c1ac2cd5d1f56e60d110ffd28a94ef7b/programs
- 3 - Obtener programa
/api/c1ac2cd5d1f56e60d110ffd28a94ef7b/program/1
- 2 - Actualizar programa
/api/c1ac2cd5d1f56e60d110ffd28a94ef7b/program/5
- 1 - Nueva trama de sincronización
/api/c1ac2cd5d1f56e60d110ffd28a94ef7b/sync/init
- 3 - Sincronización de sensores
/api/c1ac2cd5d1f56e60d110ffd28a94ef7b/sync/3/senses

- 3 - Sincronización de válvulas
`/api/c1ac2cd5d1f56e60d110ffd28a94ef7b/sync/3/valves`
- **Administrador:**
 - 1 - Identificación en el servicio.
`/user/login`
 - 2 - Alta programador
- **Robots o spiders:**
 - 20 - Consulta páginas estáticas

6.1.2. Descripción de las pruebas en un entorno real controlado

Una vez finalizadas las pruebas independientes, procederemos a montar el sistema en un entorno real controlado.

Durante esta fase trataremos de evaluar el correcto funcionamiento del sistema en conjunto. La instalación será en un entorno de riego real aunque no resultará grave en caso de fallo. A lo largo de este proceso, se forzará el sistema para poder evaluar distintos puntos críticos del mismo.

■ Descripción de la instalación

En este caso la instalación que se realizará será completa. Incluiremos pues el generador hidroeléctrico y sensor de presión no incluidos en el apartado anterior. Las peticiones de conexión en este caso sí serán resueltas por el servidor, que estará instalado en nuestro servidor personal.

El lugar elegido para la instalación será el jardín de una residencia particular situada en Massarrojos. El programador será el encargado de gestionar el riego de algunos árboles del mismo jardín.

Es importante resaltar que la instalación resulta casi idéntica a una agraria ya que, aunque la presión es mucho menor, el efecto sobre las electroválvulas es casi nulo. Esto es debido a la mecánica utilizada por las mismas, tal y como se refleja en la figura 6.1, para realizar la apertura y cierre de la misma tan solo se ha de mover una pequeña pieza que bloquea o permite el paso de agua a un pequeño recipiente. Así pues, es la misma presión la encargada de realizar la apertura y cierre de las mismas.

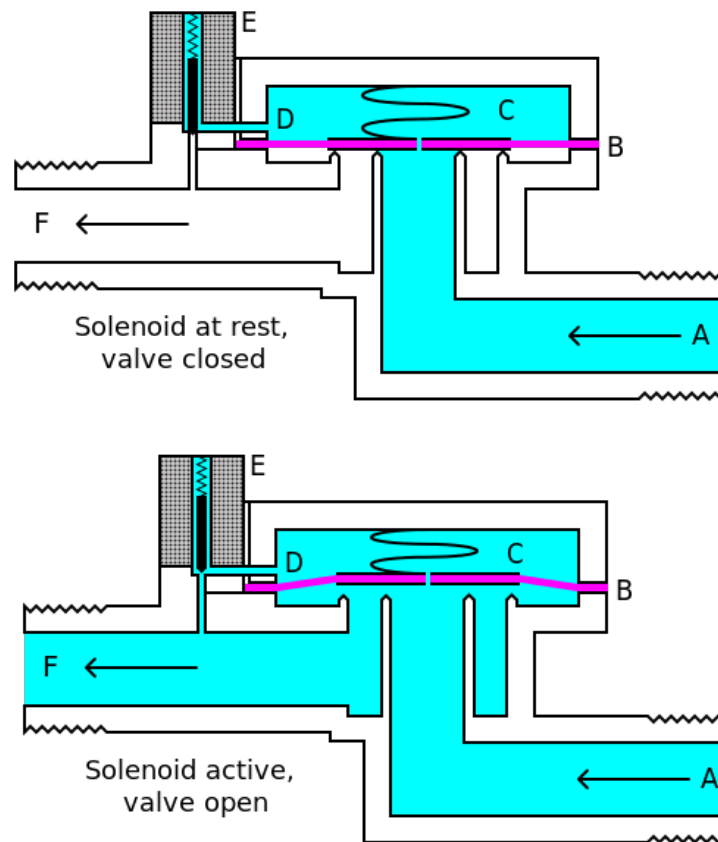


Figura 6.1: Mecánica de una electroválvula.

Fuente: http://en.wikipedia.org/wiki/Solenoid_valve

- **Descripción de la configuración**

El programador se encontrará totalmente reseteado de fábrica mientras que la base de datos se encontrará con datos de otros programadores generados aleatoriamente.

- **Duración de las pruebas**

La duración de las pruebas se estima en un total de 7 días sin fallo del sistema. De suceder un fallo, se resolverá y volverá a empezar la cuenta. Será considerado un fallo del sistema cuando ocurra una acción inesperada. No se considerará un fallo del sistema notificaciones de fallo de agua, presión, etc. si realmente se están dando.

- **Descripción de las pruebas a realizar**

Durante el transcurso de estas pruebas se realizarán diferentes pruebas para probar que se activan los mecanismos de protección y notificación descritos en el apartado de Especificación del sistema (sección 4.3).

Desde el panel de control web, el administrador deberá asociar una SIM a un programador y añadir un nuevo programador.

Los programas se ejecutarán diariamente y serán de 1 hora en el sector de riego y 1 hora en el de abonado.

Se forzará el medio para que el sistema deba efectuar las siguientes funcionalidades y poder comprobar su correcto funcionamiento.

1. Notificación de fallo de presión
2. Notificación de apertura de electro-válvula.

Al final de todo el periodo, se evaluará el estado de la carga de la batería.

6.2. Análisis de los resultados de las pruebas

Una vez descritas todas las pruebas que se realizarán sobre el sistema para certificar su correcto funcionamiento, se describirán los resultados obtenidos una vez realizadas, especificando en cada una de ellas los datos introducidos, la salida esperada y la salida real obtenida después de su ejecución.

6.2.1. Resultados en pruebas independientes del programador

En este apartado trataremos de exponer los resultados referentes a las pruebas realizadas sobre el programador en un entorno que no dependa del servidor o agentes externos como el hidrogenerador.

6.2.1.1. Resultados de funcionalidad en las pruebas independientes

A continuación se procederá a la ejecución de la batería de Pruebas Unitarias del Programador (PUP). En ellas se evaluarán todos los aspectos definidos en la descripción de las pruebas (página 6.1.1.1).

Configuración - Configuración SD

Identificador	PUP 1.1
Entrada	Tarjeta SD con la información
Precondición	Menú principal.
Ejecución	Pulsamos en el menú de configuración. Seleccionamos cargar información desde la SD.
Salida esperada	Se reinicia el programador con los programas configurados en el programador.
Salida obtenida	Se obtiene la salida esperada.

Configuración - Configurar fecha/hora

Identificador	PUP 1.2
Entrada	23:27 02/08/2014
Precondición	Menú configuración
Ejecución	Seleccionamos configurar fecha/hora. Introducimos los datos.
Salida esperada	OK. La hora está actualizada.
Salida obtenida	Se obtiene la salida esperada.



Figura 6.2: Visualización de la hora del sistema.

Configuración - Configurar electroválvulas

Identificador	PUP 1.3
Entrada	Puerto: 0, Tipo: Riego
Precondición	Menú de configuración
Ejecución	Pulsamos en configurar electroválvula. Introducimos los datos.

Salida esperada	OK. La la electroválvula de riego ha quedado asignada al puerto 0.
Salida obtenida	Se obtiene la salida esperada.



Figura 6.3: Lista de electroválvulas del programador.

Gestión de programas - Añadir

Identificador	PUP 2.1
Entrada	Hora Inicio: 08:00 Hora fin: 09:00, Días: Lunes, Electroválvula: 0, Día: Martes
Precondición	Menú principal
Ejecución	Seleccionamos añadir programa. Introducimos los datos.
Salida esperada	OK. Se ha añadido un programa nuevo.
Salida obtenida	Se obtiene la salida esperada.



Figura 6.4: Programa activo del programador.

Gestión de programas - Desactivar

Identificador	PUP 2.2
Entrada	Ninguna.
Precondición	Menú principal.
Ejecución	Seleccionamos la opción Programas Activos. Seleccionamos el programa introducido en PUP 2.1
Salida esperada	OK. El programa desaparece de la lista de programas activos.
Salida obtenida	Se obtiene la salida esperada.

Gestión de programas - Activar

Identificador	PUP 2.3
Entrada	Ninguna
Precondición	Menú principal.
Ejecución	Seleccionamos la opción Programas Inactivos. Seleccionamos el programa introducido en PUP 2.1
Salida esperada	OK. El programa aparece de nuevo en la lista de programas activos.
Salida obtenida	Se obtiene la salida esperada.

Gestión de programas - Eliminar

Identificador	PUP 2.4
Entrada	ninguna
Precondición	Menú principal
Ejecución	Seleccionamos eliminar programa. Seleccionamos el programa introducido anteriormente.
Salida esperada	OK. Se ha eliminado el programa.
Salida obtenida	Se obtiene la salida esperada.

Gestión de advertencias - Ver mensajes

Identificador	PUP 3.1
Entrada	Sensor de presión a 0.
Precondición	Menú principal
Ejecución	Seleccionamos ver mensajes en el menú principal.
Salida esperada	“1-ERROR PRESION” hora
Salida obtenida	Se obtiene la salida esperada.



Figura 6.5: Lista de mensajes del programador.

Gestión manual - Apertura electroválvula

Identificador	PUP 4.1
Entrada	Ninguna
Precondición	Menú principal
Ejecución	Selecciona Apertura manual. Seleccionamos la electroválvula 0.
Salida esperada	OK. Se abre la electroválvula 0.
Salida obtenida	Se obtiene la salida esperada.



Figura 6.6: Mensaje general de confirmación.

Gestión manual - Cierre electroválvula

Identificador	PUP 4.2
Entrada	Ninguna
Precondición	Menú principal.
Ejecución	Seleccionamos Cierre manual. Seleccionamos la electroválvula 0.
Salida esperada	OK. Se ha cerrado la electroválvula 0.
Salida obtenida	Se obtiene la salida esperada.

Detección errores - Presión

Identificador	PUP 5.1
Entrada	Presión de la línea a 0
Precondición	Ninguna.
Ejecución	Ninguna
Salida esperada	Se ha generado un mensaje de error y una conexión al servidor.
Salida obtenida	Se obtiene la salida esperada.

Para hacer las pruebas de comunicaciones en modo independiente, se ha recurrido al servicio apiary³. Este nos brinda la posibilidad de diseñar una APIREST para el test y diseño de un servicio. Además se generará la documentación sobre la misma. En el diseño deberemos incluir el tipo de entradas así como los ejemplos de salida que serán los devueltos al progra-

³<https://app.apiary.io/>

mador. Así pues se realizarán las peticiones a la dirección `http://miquel.apiary-mock.com`

Comunicaciones - Obtener ID programas

Identificador	PUP 6.1
Entrada	Id: c1ac2cd5d1f56e60d1180ffd2a94ef7b
Ejecución	GET /c1ac2cd5d1f56e60d110ffd28a94ef7b/programs
Salida esperada	[1000,1001]
Salida obtenida	Se obtiene la salida esperada.

Comunicaciones - Obtener programas modificados

Identificador	PUP 6.2
Entrada	Id: c1ac2cd5d1f56e60d1180ffd2a94ef7b
Ejecución	GET /c1ac2cd5d1f56e60d110ffd28a94ef7b/programs/modified
Salida esperada	[1000,1001]
Salida obtenida	Se obtiene la salida esperada.

Comunicaciones - Obtener programa

Identificador	PUP 6.3
Entrada	Id: c1ac2cd5d1f56e60d1180ffd2a94ef7b, programid: 1
Ejecución	GET /c1ac2cd5d1f56e60d110ffd28a94ef7b/program/1000
Salida esperada	{“active”:true,“id”:1000,“startTime”:36000, “stopTime”:39600,“valveid”:0,“weekdays”:1010000}
Salida obtenida	Se obtiene la salida esperada.

Comunicaciones - Añadir programa

Identificador	PUP 6.4
Entrada	Id: c1ac2cd5d1f56e60d1180ffd2a94ef7b
Ejecución	POST /c1ac2cd5d1f56e60d110ffd28a94ef7b/program, {“active”:true,“id”:4,“startTime”:36000, “stopTime”:39600,“valveid”:0,“weekdays”:“1010000”}
Salida esperada	HTTP Status Code: 201
Salida obtenida	Se obtiene la salida esperada.

Comunicaciones - Actualizar programa

Identificador	PUP 6.5
Entrada	Id: c1ac2cd5d1f56e60d1180ffd2a94ef7b
Ejecución	PUT /c1ac2cd5d1f56e60d110ffd28a94ef7b/program/1
Salida esperada	HTTP Status Code: 204
Salida obtenida	Se obtiene la salida esperada.

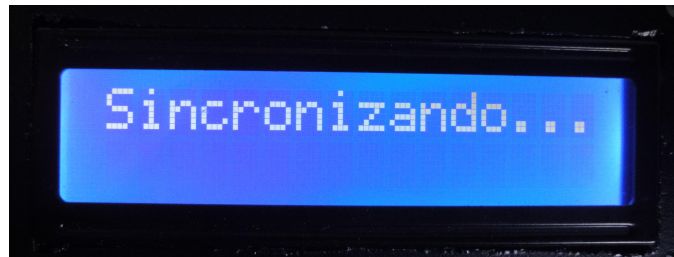


Figura 6.7: Mensaje en pantalla durante la sincronización.

Comunicaciones - Borrar programa

Identificador	PUP 6.6
Entrada	Id: c1ac2cd5d1f56e60d1180ffd2a94ef7b
Ejecución	DELETE /c1ac2cd5d1f56e60d110ffd28a94ef7b/program/1
Salida esperada	HTTP Status Code: 204
Salida obtenida	Se obtiene la salida esperada.

Comunicaciones - Obtener electroválvulas

Identificador	PUP 6.7
Entrada	Id: c1ac2cd5d1f56e60d1180ffd2a94ef7b
Ejecución	GET /c1ac2cd5d1f56e60d110ffd28a94ef7b/valves
Salida esperada	[0,1]
Salida obtenida	Se obtiene la salida esperada.

Comunicaciones - Obtener electroválvulas modificadas

Identificador	PUP 6.8
Entrada	Id: c1ac2cd5d1f56e60d1180ffd2a94ef7b
Ejecución	GET /c1ac2cd5d1f56e60d110ffd28a94ef7b/valves/modified
Salida esperada	[0,1]
Salida obtenida	Se obtiene la salida esperada.

Comunicaciones - Obtener electroválvula

Identificador	PUP 6.9
Entrada	Id: c1ac2cd5d1f56e60d1180ffd2a94ef7b
Ejecución	GET /c1ac2cd5d1f56e60d110ffd28a94ef7b/valve/0
Salida esperada	{id:0,port:0,type:0}
Salida obtenida	Se obtiene la salida esperada.

Comunicaciones - Añadir electroválvula

Identificador	PUP 6.10
Entrada	Id: c1ac2cd5d1f56e60d1180ffd2a94ef7b, {“id”:0,“port”:0,“type”:0}
Ejecución	POST /c1ac2cd5d1f56e60d110ffd28a94ef7b/valve/0
Salida esperada	HTTP Status Code: 201
Salida obtenida	Se obtiene la salida esperada.

Comunicaciones - Actualizar electroválvula

Identificador	PUP 6.11
Entrada	Id: c1ac2cd5d1f56e60d1180ffd2a94ef7b, {“id”:0,“port”:0,“type”:1}
Ejecución	PUT /c1ac2cd5d1f56e60d110ffd28a94ef7b/valve/0
Salida esperada	HTTP Status Code: 204
Salida obtenida	Se obtiene la salida esperada.

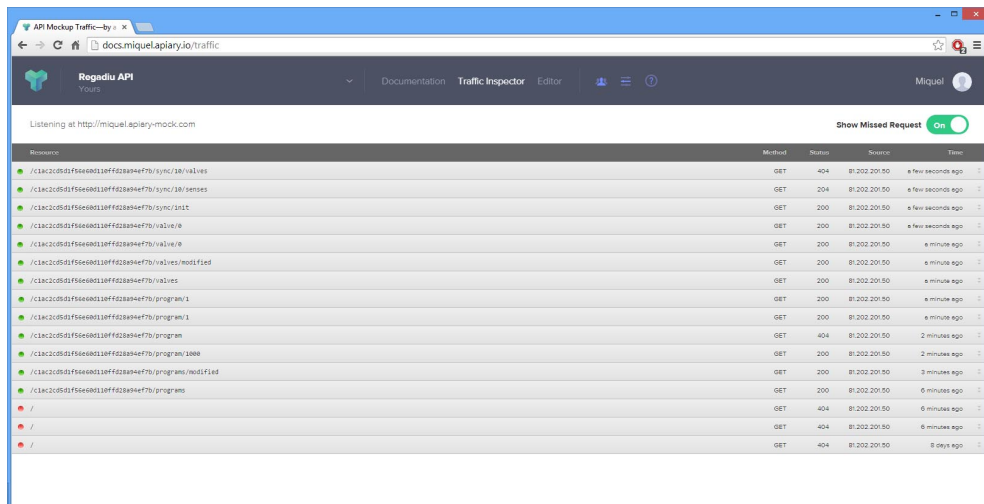


Figura 6.8: Resultado de log de Apiary después de algunas consultas.

Comunicaciones - Eliminar electroválvula

Identificador	PUP 6.12
Entrada	Id: c1ac2cd5d1f56e60d1180ffd2a94ef7b
Ejecución	DELETE /c1ac2cd5d1f56e60d1180ffd2a94ef7b/programs/1
Salida esperada	HTTP Status Code: 204
Salida obtenida	Se obtiene la salida esperada.

Comunicaciones - Iniciar sincronización

Identificador	PUP 6.13
Entrada	Id: c1ac2cd5d1f56e60d1180ffd2a94ef7b
Ejecución	GET /c1ac2cd5d1f56e60d1180ffd2a94ef7b/sync/init
Salida esperada	id:10
Salida obtenida	Se obtiene la salida esperada.

Comunicaciones - Enviar sensores

Identificador	PUP 6.14
Entrada	Id: c1ac2cd5d1f56e60d1180ffd2a94ef7b, [{"id":0,"value":23},{ "id":1,"value":354}]
Ejecución	/c1ac2cd5d1f56e60d1180ffd2a94ef7b/sync/10/senses

Salida esperada	HTTP Status Code: 201
Salida obtenida	Se obtiene la salida esperada.

Comunicaciones - Enviar electroválvulas

Identificador	PUP 6.15
Entrada	Id: c1ac2cd5d1f56e60d1180ffd2a94ef7b, [{"id":0,"open":false}, {"id":1,"open":true}]
Ejecución	/c1ac2cd5d1f56e60d1180ffd2a94ef7b/sync/10/valves
Salida esperada	HTTP Status Code: 201
Salida obtenida	Se obtiene la salida esperada.

Comunicaciones - Enviar errores

Identificador	PUP 6.16
Entrada	Id: c1ac2cd5d1f56e60d1180ffd2a94ef7b, [{"error":"FALLO_PRESION","active":true}, {"error":"FALLO_VALVULA","active":true}]
Ejecución	/c1ac2cd5d1f56e60d1180ffd2a94ef7b/sync/10/errors
Salida esperada	HTTP Status Code: 201
Salida obtenida	Se obtiene la salida esperada.

6.2.1.2. Resultados de usabilidad en las pruebas independientes

La tabla 6.30 refleja la opinión de los usuarios sobre el producto después de realizar las acciones descritas en el apartado de descripción de las pruebas. Podemos observar que el uso de una pantalla no específica del producto penaliza la interacción del usuario con el producto. El resto de valoraciones ha sido realmente positiva.

6.2.1.3. Resultados de rendimiento en las pruebas independientes

A continuación se exponen los resultados extraídos de las distintas pruebas de rendimiento realizadas sobre el producto.

- **Tiempo de duración de la batería:**

Operación	Onofre	Jose	Carlos
Aspecto visual físico	Muy bueno	Muy bueno	Muy bueno
Simplicidad de uso	Media-alta	Alta	Media-alta
Accesibilidad menús	Media	Media	Media
Disposición controles	Muy buena	Muy buena	Muy buena
Visibilidad en interiores	Muy buena	Muy buena	Muy buena
Visibilidad en exteriores	Muy buena	Muy buena	Muy buena

Cuadro 6.30: Valoración de la usabilidad del producto según los administradores

El tiempo de duración de la batería ha sido algo mayor a cinco días. Durante este tiempo se han realizado un total de 63 sincronizaciones con el servidor.

Este tiempo resultará más que satisfactorio pues en un entorno de riego por goteo el tiempo máximo entre tiempos de riego no suele exceder las 24 horas.

6.2.2. Resultados en pruebas independientes del servidor

A continuación se expondrán todas las pruebas realizadas en un entorno independiente sobre el servidor.

6.2.2.1. Resultados de funcionalidad en las pruebas independientes

En este subapartado se exponen todos los datos referentes a los resultados de las pruebas de funcionalidad así como de los datos introducidos o existentes en el servidor para realizar las pruebas.

hay que recordar que para el correcto funcionamiento de todas las pruebas será necesario realizarlas en el orden expuesto, pues los datos generados en cada Prueba Unitaria (PU) permanecerán almacenados.

■ Módulo de login

Se trata del primer módulo a evaluar. En él se deberá comprobar que se puede realizar correctamente el acceso al servicio, tanto para usuarios normales como administradores, siendo cada uno de ellos redirigido al panel correspondiente.

Datos requeridos

Para realizar dichas pruebas, se ha creado en el servidor los siguientes usuarios:

- Usuario: *Miquel*, Contraseña: *miquelPass*, Rol: *usuario*
- Usuario: *Aina*, Contraseña: *ainaPass*, Rol: *administrador*

Módulo login - Identificación usuario

Identificador	PU 1.1
Entrada	Usuario: Miquel, Contraseña: miquelPass
Precondición	No se ha iniciado sesión.
Ejecución	Se accederá a la aplicación que nos redirigirá al formulario de login. Seguidamente, introduciremos los datos.
Salida esperada	Una vez efectuada la validación, el sistema nos debe redirigir a la lista de programadores.
Salida obtenida	Se obtiene la salida esperada.

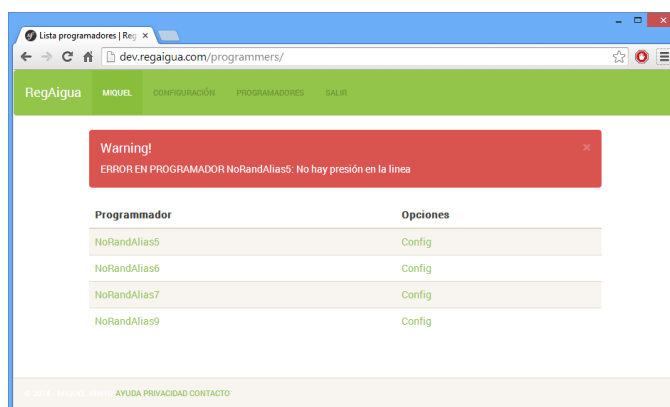


Figura 6.9: Lista de programadores una vez autenticado como usuario

Módulo login - Identificación administrador

Identificador	PU 1.2
Entrada	Usuario: Aina, Contraseña: ainaPass
Precondición	No se ha iniciado sesión.

Ejecución	Se accederá a la aplicación que nos redirigirá al formulario de login. Seguidamente introduciremos los datos del usuario de administración.
Salida esperada	Una vez efectuada la validación, el sistema nos debe redirigir al menú de administración.
Salida obtenida	Se obtiene la salida esperada.

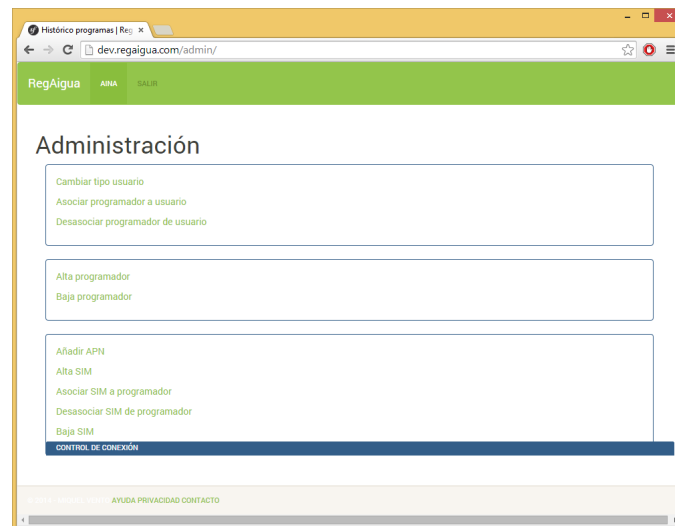


Figura 6.10: Panel de administración una vez identificado como Aina - administradora.

Módulo login - Identificación errónea

Identificador	PU 1.3
Entrada	Usuario: Miquel, Contraseña: incorrecta
Precondición	No se ha iniciado sesión.
Ejecución	Se accederá a la aplicación la cual nos redirigirá al formulario de login. Seguidamente, introduciremos los datos requeridos.
Salida esperada	Una vez efectuada la comprobación, se debe mostrar de nuevo el formulario de identificación junto al error correspondiente.
Salida obtenida	Se obtiene la salida esperada.

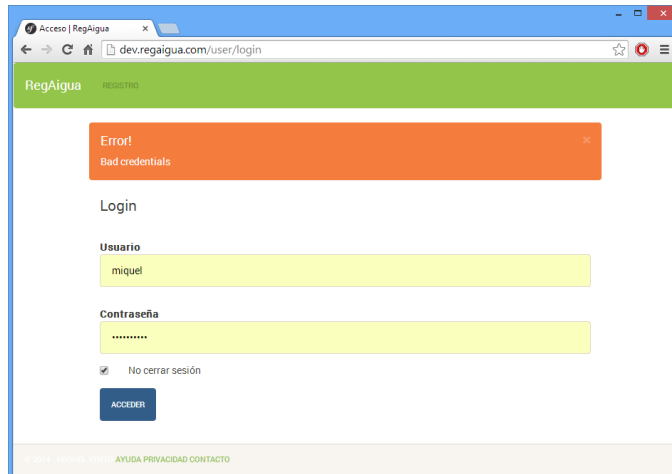


Figura 6.11: Error durante el proceso de identificación

Módulo login - Cierre de sesión

Identificador	PU 1.4
Entrada	Ninguna.
Precondición	Sesión iniciada.
Ejecución	Desde cualquier punto del servicio, pulsaremos sobre la opción Salir.
Salida esperada	Se cerrará la sesión y se mostrará el menú de login. Si se intenta realizar el acceso a cualquier url se redirige a login.
Salida obtenida	Se obtiene la salida esperada.

■ Gestión de programadores

Habiendo realizado las pruebas anteriores, se procederá a la comprobación de los requisitos referentes a la gestión de programadores

Datos en sistema

Para poder realizar dichas comprobaciones, será necesario que el usuario disponga de programadores asociados. Así pues se ha asociado a la cuenta de usuario *Miquel* un total de dos programadores:

- alias: Programador1
- alias: Programador2

A su vez, será necesario que el Programador1 disponga de dos electroválvulas configuradas:

- Puerto: 0 Tipo: *Riego*
- Puerto: 1 Tipo: *Abonado*

Finalmente será necesaria la presencia de algunos datos de sincronización para cada electroválvula y sensado. Estos serán generados aleatoriamente.

Gestión de programadores - Lista programadores

Identificador	PU 2.1
Entrada	Ninguna
Precondición	El usuario accede mediante unos credenciales válidos desde el menú de identificación.
Ejecución	El usuario efectúa una identificación válida.
Salida esperada	El usuario obtiene una lista de programadores pudiendo acceder al menú de configuración del mismo.
Salida obtenida	Se obtiene la salida esperada.

A partir de esta prueba, salvo que se indique lo contrario, se supondrá que el usuario ha efectuado un inicio de sesión válido como cliente y se encuentra en una sesión activa.

Gestión de programadores - Configuración del programador

Identificador	PU 2.2
Entrada	Ninguna
Precondición	El usuario se encuentra en la lista de programadores.
Ejecución	El usuario accede a la configuración del programador1 pulsando sobre el vínculo de configuración.

Salida esperada	Se muestra el panel de configuración referente al programador1 donde podrá cambiar el alias del mismo, acceder a la configuración de las electroválvulas o generar el archivo de autoconfiguración.
Salida obtenida	Se obtiene la salida esperada.

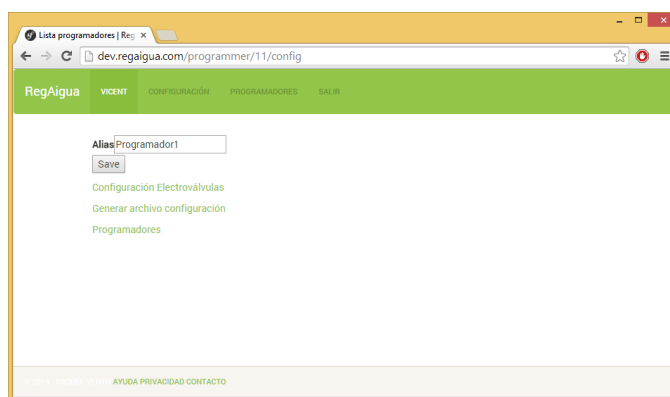


Figura 6.12: Panel de configuración del programador

Gestión de programadores - Desasignar electroválvula

Identificador	PU 2.3
Entrada	Ninguna
Precondición	El usuario se encuentra en el menú de configuración del programador.
Ejecución	El usuario pulsa sobre el enlace de configurar electroválvulas. Se visualiza un nuevo menú donde se lista la asignación actual de las electroválvulas. El usuario pulsa sobre el link de desasignar de la electroválvula del puerto 1.
Salida esperada	Lista de las electroválvulas actualizada con el puerto 1 sin asignar.
Salida obtenida	Se obtiene la salida esperada.

Gestión de programadores - Asignar electroválvula

Identificador	PU 2.4
Entrada	Tipo: Abonado, Puerto: 1
Precondición	El usuario se encuentra en la lista de asignaciones de electroválvulas.
Ejecución	Pulsamos sobre la opción asignar. Introducimos los datos y pulsamos asignar.
Salida esperada	Se muestra la lista de electroválvulas con la nueva asignación.
Salida obtenida	Se obtiene a salida esperada.

Gestión de programadores - Historial de sensores

Identificador	PU 2.5
Entrada	Tipo: Abonado, Puerto: 1
Precondición	El usuario se encuentra en la lista de electroválvulas o cualquier menú que derive de un programador.
Ejecución	El usuario pulsa en la barra el enlace de Programas. Seguidamente se accederá a la pestaña de Historial de sensores.
Salida esperada	La lista de eventos de los sensores ordenados cronológicamente y clasificados por colores.
Salida obtenida	Se obtiene a salida esperada.

Tipo	Valor	Fecha	Alarma
presion	15	10/07/2014	
presion	15	24/04/2014	
presion	19	24/04/2014	
presion	6	04/03/2014	
presion	11	02/02/2014	
presion	23	02/02/2014	
presion	16	22/01/2014	
presion	18	16/11/2013	
presion	10	10/09/2013	
presion	9	07/09/2013	
presion	2	26/06/2013	Alarma!

Figura 6.13: Histórico de sensores del programador

Gestión de programadores - Historial de electroválvulas

Identificador	PU 2.6
Entrada	Tipo: Abonado, Puerto: 1
Precondición	El usuario se encuentra en el historial de sensores o cualquiera de las pestañas del menú.
Ejecución	El usuario pulsa sobre la pestaña Historial de electroválvulas.
Salida esperada	Se obtendrá una lista con los eventos de apertura y cierre de las electroválvulas.
Salida obtenida	Se obtiene la salida esperada.

Tipo	Estado	Fecha
riego	cerrada	02/07/2014
abonado	cerrada	02/07/2014
riego	cerrada	23/05/2014
abonado	abierta	23/05/2014
riego	cerrada	22/04/2014
abonado	abierta	22/04/2014
riego	abierta	22/04/2014
abonado	abierta	22/04/2014
riego	abierta	12/04/2014
abonado	abierta	12/04/2014
riego	abierta	29/03/2014
abonado	cerrada	29/03/2014

Figura 6.14: Histórico de acciones de las electroválvulas del programador

■ Gestión de programas

Una vez configurados los programadores, se realizarán las pruebas referentes a la gestión de los programas ***Datos en sistema***

Para poder realizar los test, se deberá contar con al menos un programa asociado al programador. Por ello, el programador cuenta con un total de dos programas:

- Programa 1: hora inicio: *10:00*, hora de fin: *11:00*, electroválvula: *0*, tipo: *riego*, Días: Lunes y Miércoles
- Programa 2: hora inicio: *10:05*, hora de fin: *10:35*, electroválvula: *1*, tipo: *riego*, Días: Lunes y Miércoles

Gestión de programas - Lista de programas activos

Identificador	PU 3.1
Entrada	Ninguna
Precondición	Nos encontramos en la lista de electroválvulas o cualquier menú que derive de un programador.
Ejecución	Pulsamos en la barra el enlace de Programas.
Salida esperada	Una lista con todos los programas activos del programador.
Salida obtenida	Se obtiene la salida esperada.

Gestión de programas - Añadir programa

Identificador	PU 3.2
Entrada	Hora inicio: <i>08:00</i> , hora de fin: <i>09:00</i> , electroválvula: <i>0</i> , tipo: <i>riego</i> .
Precondición	Nos encontramos en la lista de programas activos o cualquiera de las pestañas del menú.
Ejecución	Pulsamos sobre la pestaña Añadir. Introducimos los datos en el formulario. Pulsamos sobre el botón añadir.
Salida esperada	La lista de programas actualizada con el introducido.
Salida obtenida	Se obtiene la salida esperada.

Gestión de programas - Eliminar programa

Identificador	PU 3.3
Entrada	Ninguna
Precondición	Nos encontramos en la lista programas activos.
Ejecución	Pulsamos sobre el enlace Eliminar del programa añadido en la PU 3.2.
Salida esperada	La nueva lista de programas en la que no aparece el programa eliminado.
Salida obtenida	Se obtiene la salida esperada.

Gestión de programas - Desactivar programa

Identificador	PU 3.4
Entrada	Ninguna
Precondición	Nos encontramos en la lista de programas activos.
Ejecución	Pulsamos sobre la opción Desactivar del programa de riego.
Salida esperada	Obtenemos la lista actualizada de programas activos en la que no aparece el programa desactivado.
Salida obtenida	Se obtiene la salida esperada.

Gestión de programas - Ver todos los programas

Identificador	PU 3.5
Entrada	Ninguna
Precondición	Nos encontramos en la lista de programas activos o cualquiera de las pestañas del menú.
Ejecución	Pulsamos sobre la opción de ver todos
Salida esperada	Lista completa con los programas del programador.
Salida obtenida	Se obtiene la salida esperada.

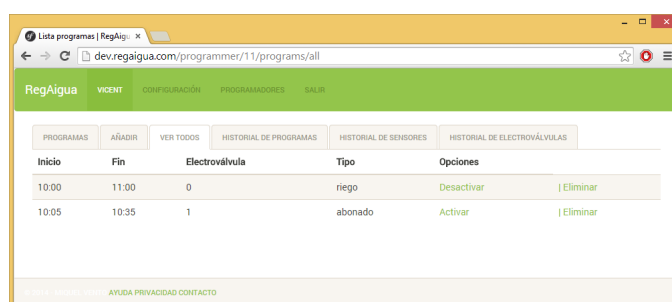


Figura 6.15: Lista de programas activos e inactivos

Gestión de programas - Activar programa

Identificador	PU 3.6
Entrada	Ninguna
Precondición	Nos encontramos en la lista de todos programas.
Ejecución	Pulsamos sobre la opción Activar del programa de riego.
Salida esperada	Obtenemos la lista actualizada de programas activos en la que aparece el programa activado.
Salida obtenida	Se obtiene la salida esperada.

Gestión de programas - Historial de programas

Identificador	PU 3.7
Entrada	Ninguna
Precondición	Nos encontramos en la lista de programas activos o cualquiera de las pestañas del menú.
Ejecución	Pulsamos sobre la pestaña Historial de programas.
Salida esperada	Se muestra una lista con todos los programas del programador ordenados por fecha de creación.
Salida obtenida	Se obtiene la salida esperada.

Fecha de introducción	Fecha borrado	Inicio	Fin	Electroválvula	Tipo	Opciones
02/09/2014	-	10:00	11:00	0	riego	
02/09/2014	-	10:05	10:35	1	abonado	
02/09/2014	02/09/2014	08:00	09:00	0	riego	

Figura 6.16: Lista del histórico de programas

■ Gestión de errores

El sistema debe ser capaz de recibir, generar, mostrar y notificar los errores.

Gestión de errores - Tiempo mínimo

Identificador	PU 4.1
Entrada	Ninguna
Precondición	El tiempo de la última sincronización del programador es mayor a 2 horas.
Ejecución	Ninguna

Salida esperada	Correo electrónico de notificación y mensaje de advertencia en página ERROR EN PROGRAMADOR Programador1: Puede existir un fallo en el servicio o en la conexión principal.
Salida obtenida	Se obtiene la salida esperada.

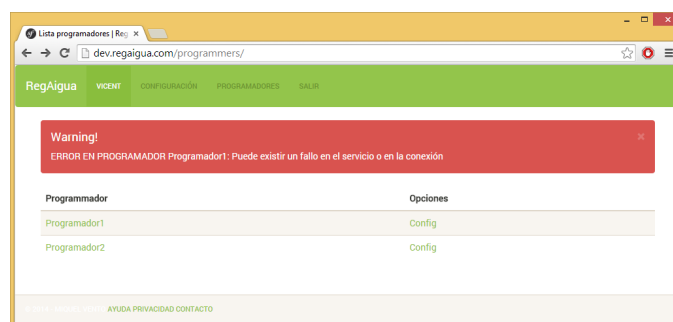


Figura 6.17: Histórico de acciones de las electroválvulas del programador

Gestión de errores - Errores reportados por el programador

Identificador	PU 4.2
Entrada	Ninguna
Precondición	Se han generado errores aleatorios en el programador.
Ejecución	El usuario accede al programa.
Salida esperada	Correo electrónico de notificación y mensaje de advertencia en página ERROR EN PROGRAMADOR Programador1: <i>error</i> principal.
Salida obtenida	Se obtiene la salida esperada.

Gestión de errores - Error en nuevo programa

Identificador	PU 4.3
---------------	--------

Entrada	Hora inicio: <i>07:00</i> , hora de fin: <i>08:00</i> , electroválvula: <i>1</i> , tipo: <i>abonado</i> , Dias: Lunes y Miercoles
Precondición	No existe un programa de riego cuando se está intentando introducir uno de abonado.
Ejecución	El usuario introduce los datos en el menú crear programa.
Salida esperada	Obtenemos el menú añadir programa en el que aparecerá un mensaje indicando que se ha producido un error de coherencia.
Salida obtenida	Se obtiene la salida esperada.

■ Administración

Ya realizadas todas las pruebas desde el punto de vista del cliente, se abarcarán las pruebas referentes a la administración.

Datos requeridos

Para la realización de estas pruebas se presupondrá que se ha realizado una identificación de un usuario tipo administrador (Aina en nuestro caso) y que la sesión sigue activa. Además se deberá registrar un nuevo usuario:

- Usuario: *Carlos*, Contraseña: *carlosPass*, Rol: *usuario*

Administración - Cambio de tipo de usuario

Identificador	PU 5.1
Entrada	Usuario: Carlos, Tipo: administrador
Precondición	El administrador se encuentra en el menú principal de administración.
Ejecución	Pulsamos sobre cambiar tipo de usuario. Introducimos los datos en el formulario y pulsamos Aceptar.
Salida esperada	El usuario Carlos es ahora administrador.
Salida obtenida	Se obtiene la salida esperada.



Figura 6.18: Datos requeridos para el cambio del tipo de usuario

Administración - Alta programador

Identificador	PU 5.2
Entrada	Número único de identificación: 7815696ecbf1c96e6894b779456d330e
Precondición	El administrador se encuentra en el menú principal de administración.
Ejecución	El administrador pulsa sobre alta programador. Introduce el número único de identificación. Pulsa el botón crear.
Salida esperada	Se ha introducido un nuevo programador en el sistema.
Salida obtenida	Se obtiene la salida esperada.

Administración - Asociar programador

Identificador	PU 5.3
Entrada	Programador: changeme - 7815696ecbf1c96e6894b779456d330e, Usuario: Miquel
Precondición	El administrador se encuentra en el menú principal de administración.

Ejecución	El administrador pulsa sobre el vínculo asociar programador a usuario. Selecciona los datos de entrada en el formulario. Se pulsa el botón asociar.
Salida esperada	Se ha asociado el programador al usuario, el cual podrá visualizar desde su menú.
Salida obtenida	Se obtiene la salida esperada.

Administración - Añadir APN

Identificador	PU 5.4
Entrada	Apn: Yoigo, Broadband: internet, Mcc: 214, Mnc: 04
Precondición	El administrador se encuentra en el menú principal de administración.
Ejecución	Pulsamos sobre añadir APN. Se introducen los datos en el formulario. Se pulsa guardar.
Salida esperada	Se ha introducido un nuevo APN en el sistema.
Salida obtenida	Se obtiene la salida esperada.

Administración - Alta SIM

Identificador	PU 5.5
Entrada	Número: 686813025, Pin:5465, Server: www.regaigua.com, Apn: Yoigo
Precondición	El administrador se encuentra en el menú principal de administración.
Ejecución	Se accede al enlace Alta SIM. Se introducen los datos. Se pulsa a Alta.
Salida esperada	Se ha dado de alta una nueva tarjeta SIM.
Salida obtenida	Se obtiene la salida esperada.

Figura 6.19: Formulario para añadir una nueva SIM.

Administración - Asociar SIM

Identificador	PU 5.6
Entrada	changeme - 7815696ecbf1c96e6894b779456d330e, Teléfono: 686813025
Precondición	El administrador se encuentra en el menú principal de administración.
Ejecución	Seleccionamos el vínculo asociar SIM a programador. Seleccionamos los datos de entrada en el formulario. Pulsamos el botón asociar.
Salida esperada	Se ha asociado la SIM al programador.
Salida obtenida	Se obtiene la salida esperada.

Administración - Desasociar SIM

Identificador	PU 5.7
Entrada	changeme - 7815696ecbf1c96e6894b779456d330e
Precondición	El administrador se encuentra en el menú principal de administración.
Ejecución	Pulsamos sobre el vínculo Desasociar SIM de programador. Seleccionamos los datos del formulario. Pulsamos al botón desasociar.
Salida esperada	Se ha desasociado la SIM del programador seleccionado.
Salida obtenida	Se obtiene la salida esperada.

Administración - Baja SIM

Identificador	PU 5.8
Entrada	Número: 686813025
Precondición	El administrador se encuentra en el menú principal de administración.
Ejecución	Se pulsa el vínculo Baja SIM. Seleccionamos los datos en el formulario. Pulsamos el botón Baja.
Salida esperada	La SIM ha sido dada de baja del sistema.
Salida obtenida	Se obtiene la salida esperada.

Administración - Desasociar programador

Identificador	PU 5.9
Entrada	changeme - 7815696ecb1c96e6894b779456d330e
Precondición	El administrador se encuentra en el menú principal de administración.
Ejecución	Se pulsa el vínculo de Desasociar programador a cliente. Seleccionamos los datos del formulario. Pulsamos el botón desasociar.
Salida esperada	Se ha desasociado el programador al cliente.
Salida obtenida	Se obtiene la salida esperada.

Administración - Baja programador

Identificador	PU 5.10
Entrada	changeme - 7815696ecb1c96e6894b779456d330e
Precondición	El administrador se encuentra en el menú principal de administración.
Ejecución	Seleccionamos la opción Baja programador. Introducimos los datos en el formulario. Pulsamos el botón baja.

Salida esperada	Se ha dado de baja el programador del sistema.
Salida obtenida	Se obtiene la salida esperada.

■ API

Ya realizadas todas las pruebas desde un punto de vista de interacción con las personas, realizaremos las pruebas en las que es el programador quien realiza las peticiones.

Dado que nos encontramos en las pruebas independientes, recurriremos a un programa externo para realizar las peticiones y no al programador físico. El programa utilizado es Postman - REST Client. Se trata de un programa que funciona con Google Chrome y se distribuye bajo la licencia Apache2. Con este programa podremos realizar peticiones al servicio de tipo GET, POST, DELETE, PUT, etc.

API - Obtener ID programas

Identificador	PU 6.1
Entrada	Id: c1ac2cd5d1f56e60d1180ffd2a94ef7b
Ejecución	GET /c1ac2cd5d1f56e60d1180ffd2a94ef7b/programs
Salida esperada	[1000,1001]
Salida obtenida	Se obtiene la salida esperada.

API - Obtener programas modificados

Identificador	PU 6.2
Entrada	Id: c1ac2cd5d1f56e60d1180ffd2a94ef7b
Ejecución	GET /c1ac2cd5d1f56e60d1180ffd2a94ef7b/programs/modified
Salida esperada	[1000,1001]
Salida obtenida	Se obtiene la salida esperada.

API - Obtener programa

Identificador	PU 6.3
Entrada	Id: c1ac2cd5d1f56e60d1180ffd2a94ef7b, programid: 1

Ejecución	GET /c1ac2cd5d1f56e60d110ffd28a94ef7b/program/1000
Salida esperada	{“active”:true,“id”:1000,“startTime”:36000, “stopTime”:39600,“valveid”:0,“weekdays”:1010000}
Salida obtenida	Se obtiene la salida esperada.

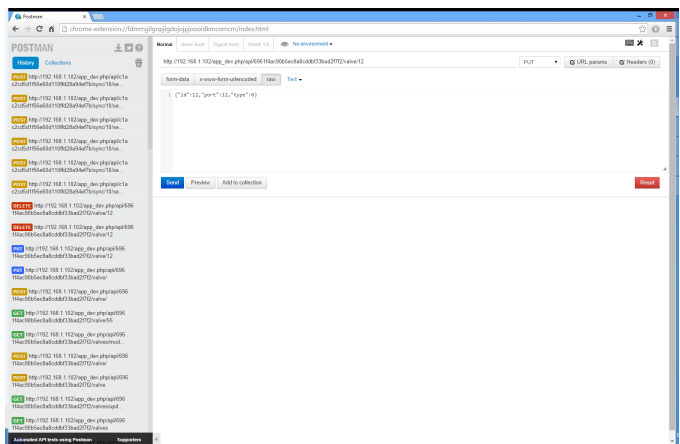


Figura 6.20: Programa utilizado para realizar los test del api rest.

API - Añadir programa

Identificador	PU 6.4
Entrada	Id: c1ac2cd5d1f56e60d1180ffd2a94ef7b
Ejecución	POST /c1ac2cd5d1f56e60d110ffd28a94ef7b/program, {“active”:true,“id”:4,“startTime”:36000, “stopTime”:39600,“valveid”:0,“weekdays”:“1010000” }
Salida esperada	HTTP Status Code: 201
Salida obtenida	Se obtiene la salida esperada.

API - Actualizar programa

Identificador	PU 6.5
Entrada	Id: c1ac2cd5d1f56e60d1180ffd2a94ef7b
Ejecución	PUT /c1ac2cd5d1f56e60d110ffd28a94ef7b/program/1
Salida esperada	HTTP Status Code: 204
Salida obtenida	Se obtiene la salida esperada.

API - Borrar programa

Identificador	PU 6.6
Entrada	Id: c1ac2cd5d1f56e60d1180ffd2a94ef7b
Ejecución	DELETE /c1ac2cd5d1f56e60d110ffd28a94ef7b/program/1
Salida esperada	HTTP Status Code: 204
Salida obtenida	Se obtiene la salida esperada.

API - Obtener electroválvulas

Identificador	PU 6.7
Entrada	Id: c1ac2cd5d1f56e60d1180ffd2a94ef7b
Ejecución	GET /c1ac2cd5d1f56e60d110ffd28a94ef7b/valves
Salida esperada	[0,1]
Salida obtenida	Se obtiene la salida esperada.

API - Obtener electroválvulas modificadas

Identificador	PU 6.8
Entrada	Id: c1ac2cd5d1f56e60d1180ffd2a94ef7b
Ejecución	GET /c1ac2cd5d1f56e60d110ffd28a94ef7b/valves/modified
Salida esperada	[0,1]
Salida obtenida	Se obtiene la salida esperada.

API - Obtener electroválvula

Identificador	PU 6.9
Entrada	Id: c1ac2cd5d1f56e60d1180ffd2a94ef7b
Ejecución	GET /c1ac2cd5d1f56e60d110ffd28a94ef7b/valve/0
Salida esperada	{id:0,port:0,type:0}
Salida obtenida	Se obtiene la salida esperada.

API - Añadir electroválvula

Identificador	PU 6.10
Entrada	Id: c1ac2cd5d1f56e60d1180ffd2a94ef7b, {“id”:0,“port”:0,“type”:0}
Ejecución	POST /c1ac2cd5d1f56e60d110ffd28a94ef7b/valve/0
Salida esperada	HTTP Status Code: 201
Salida obtenida	Se obtiene la salida esperada.

API - Actualizar electroválvula

Identificador	PU 6.11
Entrada	Id: c1ac2cd5d1f56e60d1180ffd2a94ef7b, {“id”:0,“port”:0,“type”:1}
Ejecución	PUT /c1ac2cd5d1f56e60d110ffd28a94ef7b/valve/0
Salida esperada	HTTP Status Code: 204
Salida obtenida	Se obtiene la salida esperada.

API - Eliminar electroválvula

Identificador	PU 6.12
Entrada	Id: c1ac2cd5d1f56e60d1180ffd2a94ef7b
Ejecución	DELETE /c1ac2cd5d1f56e60d110ffd28a94ef7b/programs/1
Salida esperada	HTTP Status Code: 204
Salida obtenida	Se obtiene la salida esperada.

API - Iniciar sincronización

Identificador	PU 6.13
Entrada	Id: c1ac2cd5d1f56e60d1180ffd2a94ef7b
Ejecución	GET /c1ac2cd5d1f56e60d110ffd28a94ef7b/sync/init
Salida esperada	id:10
Salida obtenida	Se obtiene la salida esperada.

API - Enviar sensores

Identificador	PU 6.14
Entrada	Id: c1ac2cd5d1f56e60d1180ffd2a94ef7b, [{"id":0,"value":23},{ "id":1,"value":354}]
Ejecución	/c1ac2cd5d1f56e60d110ffd28a94ef7b/sync/10/senses
Salida esperada	HTTP Status Code: 201
Salida obtenida	Se obtiene la salida esperada.

API - Enviar electroválvulas

Identificador	PU 6.15
Entrada	Id: c1ac2cd5d1f56e60d1180ffd2a94ef7b, [{"id":0,"open":false},{ "id":1,"open":true}]
Ejecución	/c1ac2cd5d1f56e60d110ffd28a94ef7b/sync/10/valves
Salida esperada	HTTP Status Code: 201
Salida obtenida	Se obtiene la salida esperada.

API - Enviar errores

Identificador	PU 6.16
Entrada	Id: c1ac2cd5d1f56e60d1180ffd2a94ef7b, [{"error":“FALLO_PRESION”,“active”:true}, {"error":“FALLO_VALVULA”,“active”:true}]
Ejecución	/c1ac2cd5d1f56e60d110ffd28a94ef7b/sync/10/errors
Salida esperada	HTTP Status Code: 201
Salida obtenida	Se obtiene la salida esperada.

6.2.2.2. Resultados de usabilidad en las pruebas independientes

Compatibilidad con los navegadores

Una vez terminado el análisis de las pruebas funcionales, se procederá a la realización de las pruebas de usabilidad. En ellas aseguraremos la correcta presentación del servicio en los principales navegadores de las principales plataformas:

- Windows

Versión SO: 8.1

Resolución de pantalla: 1680x1050

- Internet Explorer 10
- Google Chrome 36
- Firefox 31
- Linux Versión SO: Ubuntu 14.04 LTS
Resolución de pantalla: 1680x1050
 - Firefox 31
 - Chromium 36
- Mac Versión SO: 10.6.8
Resolución de pantalla: 1280x800
 - Safari 5.1.10
 - Chrome 36
 - Firefox 31
- Plataformas móviles
 - 9 - Android: Chrome Soporte: Motorola Moto G
Versión SO: 4.4.4 KitKat
Resolución de pantalla: 720x1280
 - 10 - iOS: Safari Soporte: iPad2
Versión SO: iOS 7.1.2
Resolución de pantalla: 1024x768
 - 11 - Windows phone: Internet Explorer Soporte: Nokia Lumia 520
Versión SO: Windows Phone 8.1
Resolución de pantalla: 480x800

Para realizar las pruebas se han englobado las páginas en varios tipos, de tal forma que no sea preciso realizar la misma prueba para páginas que solo presentan pequeñas diferencias. Las páginas a testear serán:

- Formulario de inicio:

Siendo la página principal para usuarios no registrados, será muy importante comprobar su correcta visualización. Los resultados se pueden observar en las figuras 6.21, 6.22 y 6.23.

- Lista de programadores:

Ya que presenta una tabla de datos y pestañas. Los resultados se pueden observar en las figuras 6.24, 6.24 y 6.24.

- Añadir programa:

Escogido como formulario de ejemplo además de ser uno de los más utilizados. Los resultados se representan en las figuras 6.27, 6.27 y 6.27.

Si se realiza una comparación exhaustiva, observaremos que existen pequeñas diferencias entre los distintos navegadores y plataformas pero ninguna de ellas impide la correcta visualización de los datos. En el CD adjunto se pueden observar las capturas a mayor tamaño así como las realizadas a otras interfaces.

Opinión de los usuarios

En la tabla 6.77 se muestran los resultados de la encuesta de satisfacción de los usuarios para la parte de cliente:

Operación	Onofre	Jose
Aspecto visual	Bueno	Muy bueno
Simplicidad de uso	Buena	Muy buena
Accesibilidad	Buena	Muy buena

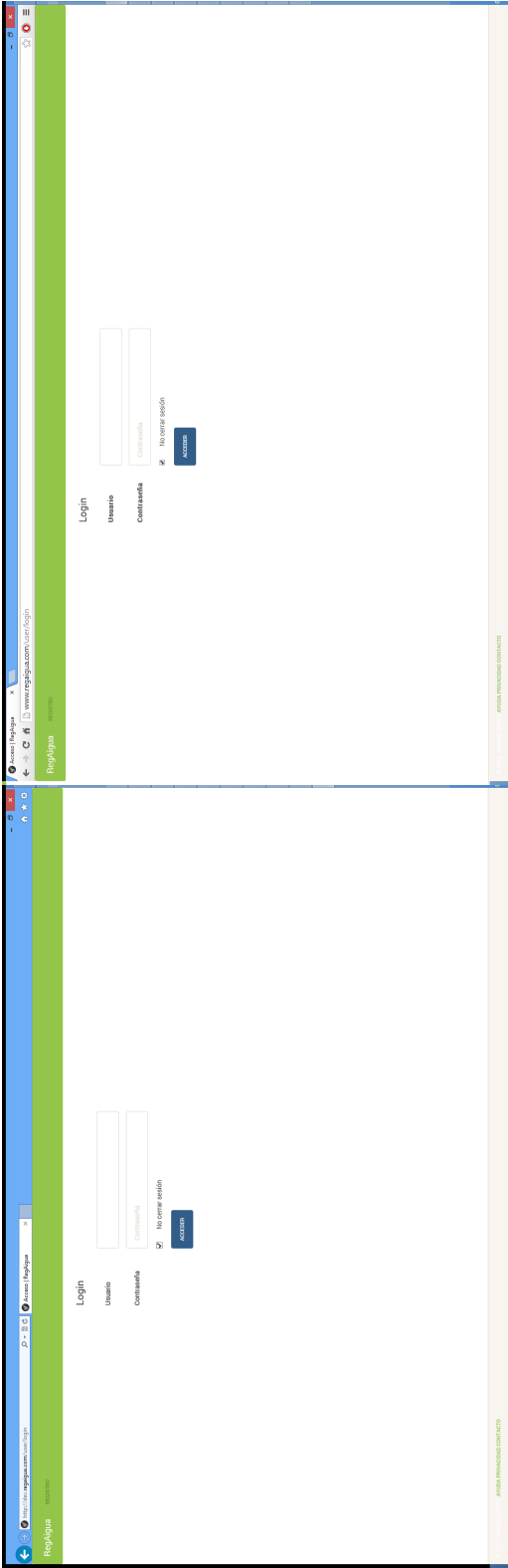
Cuadro 6.77: Valoración de la usabilidad del producto según los usuarios

Por lo que respecta a las operaciones de administración, la valoración de los mismos se representa en la tabla 6.78.

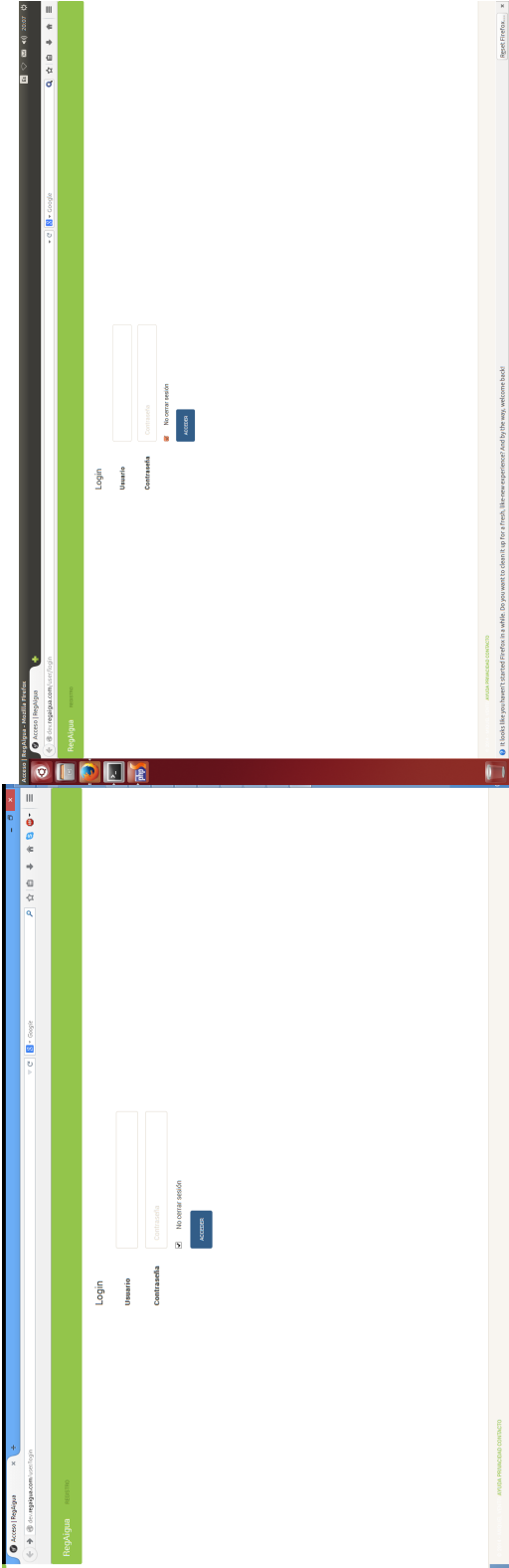
Operación	Jose	Carlos
Aspecto visual	Muy bueno	Muy bueno
Simplicidad de uso	Muy buena	Muy buena
Accesibilidad	Muy buena	Muy buena

Cuadro 6.78: Valoración de la usabilidad del producto según los administradores

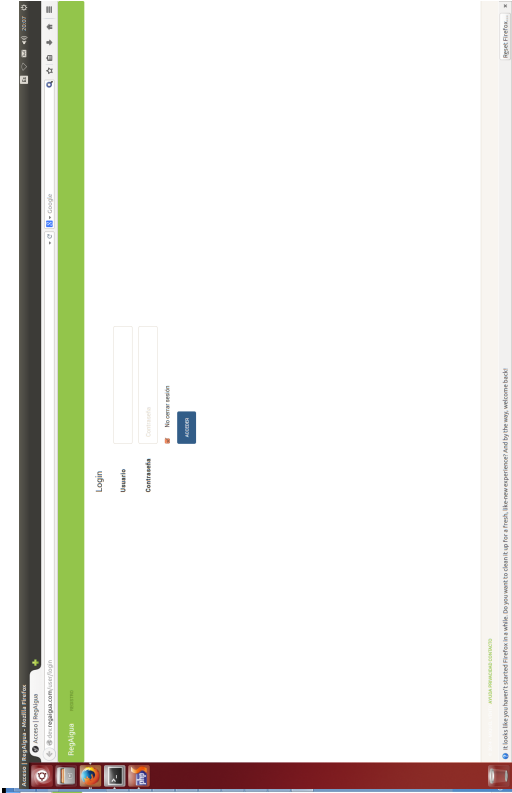
Como se puede observar, el grado de satisfacción de los usuarios ha sido muy bueno por lo que concluiremos que las pruebas de usabilidad del servicio han sido satisfactorias.



(a) Internet Explorer Windows



(c) Firefox Windows

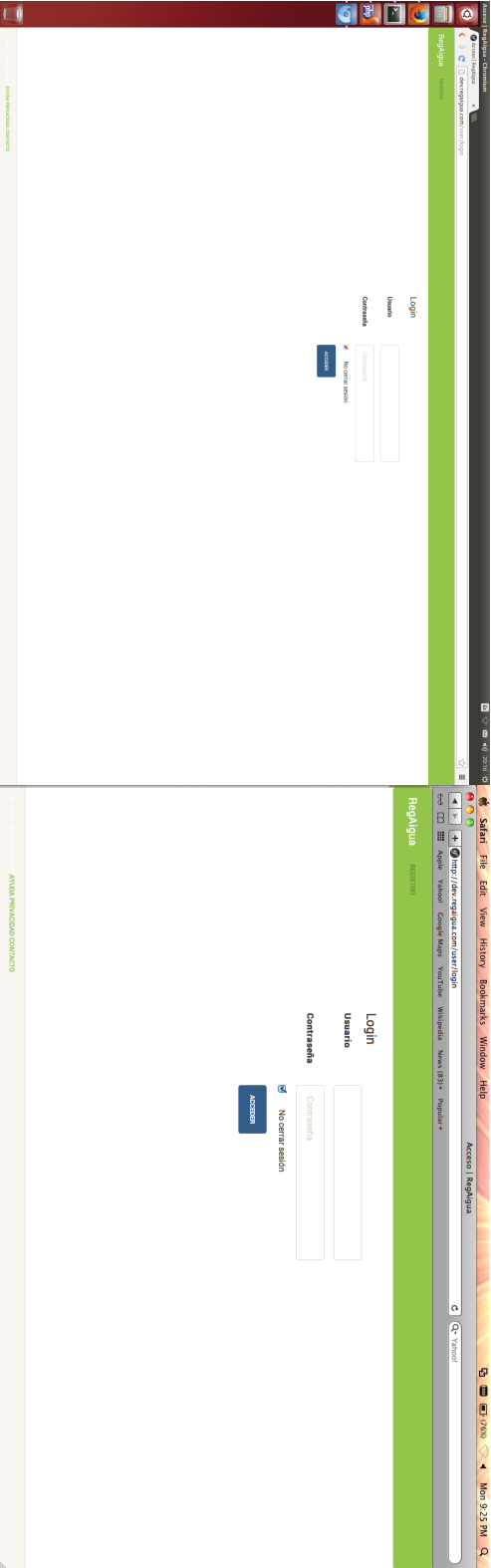


(b) Chrome Windows



(d) Firefox Linux

Figura 6.21: Capturas para el formulario de login(1/3)



(a) Chromium Linux



(b) Safari OSX



(c) Chrome OSX



(d) Firefox OSX

Figura 6.22: Capturas para el formulario de login(2/3)

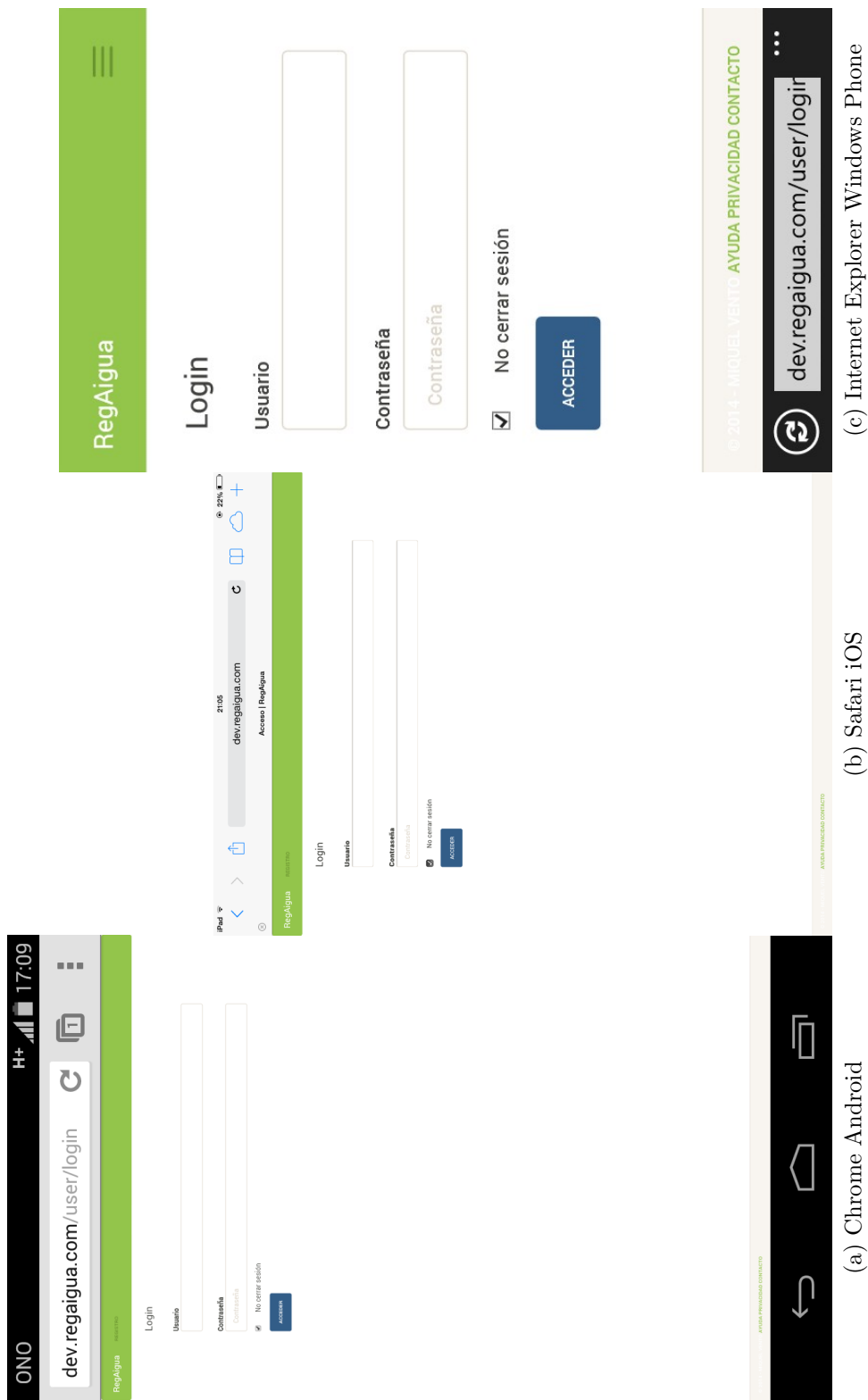
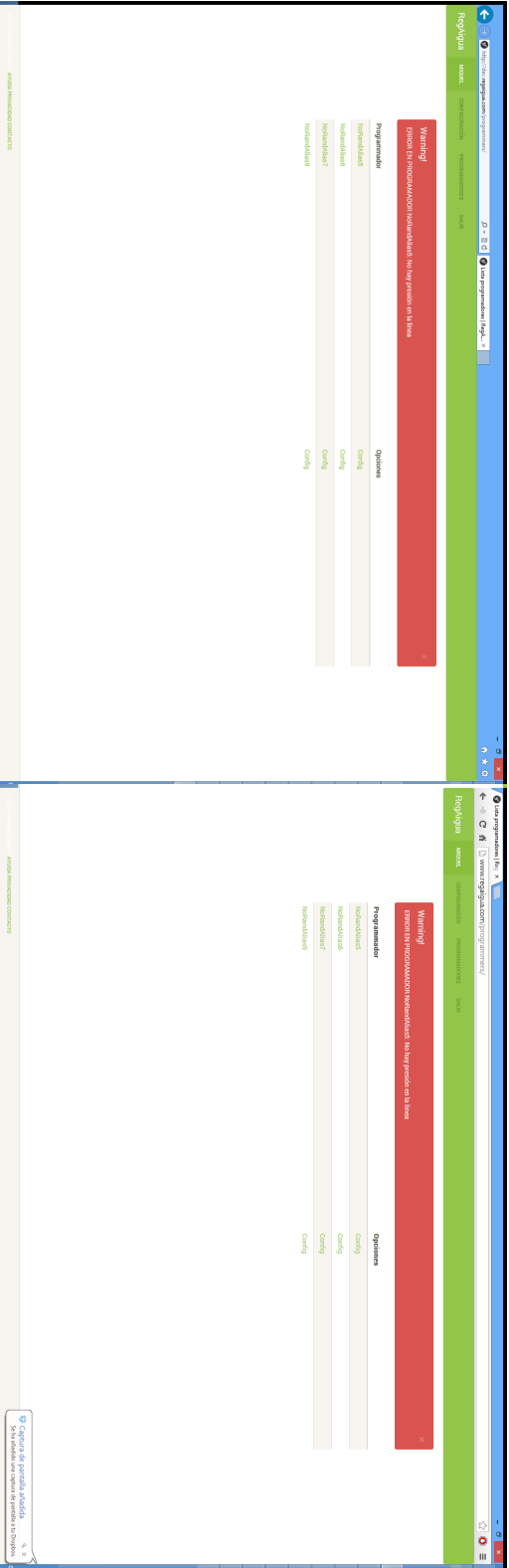


Figura 6.23: Capturas para el formulario de login en dispositivos móviles (3/3)



(a) Internet Explorer Windows

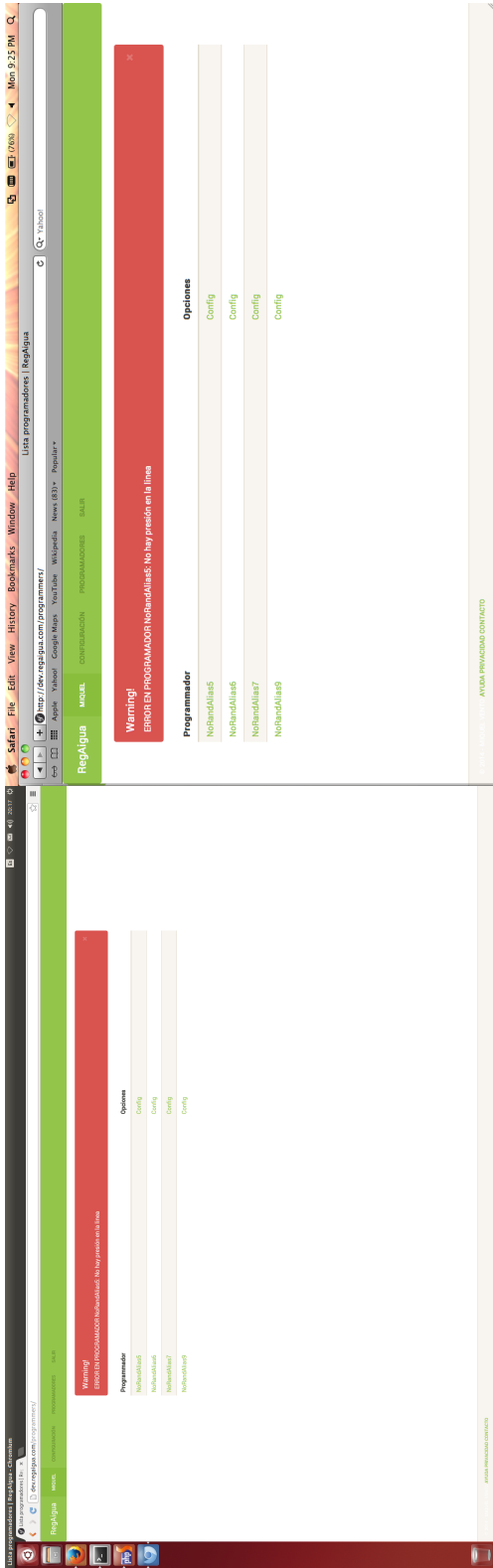


(b) Chrome Windows

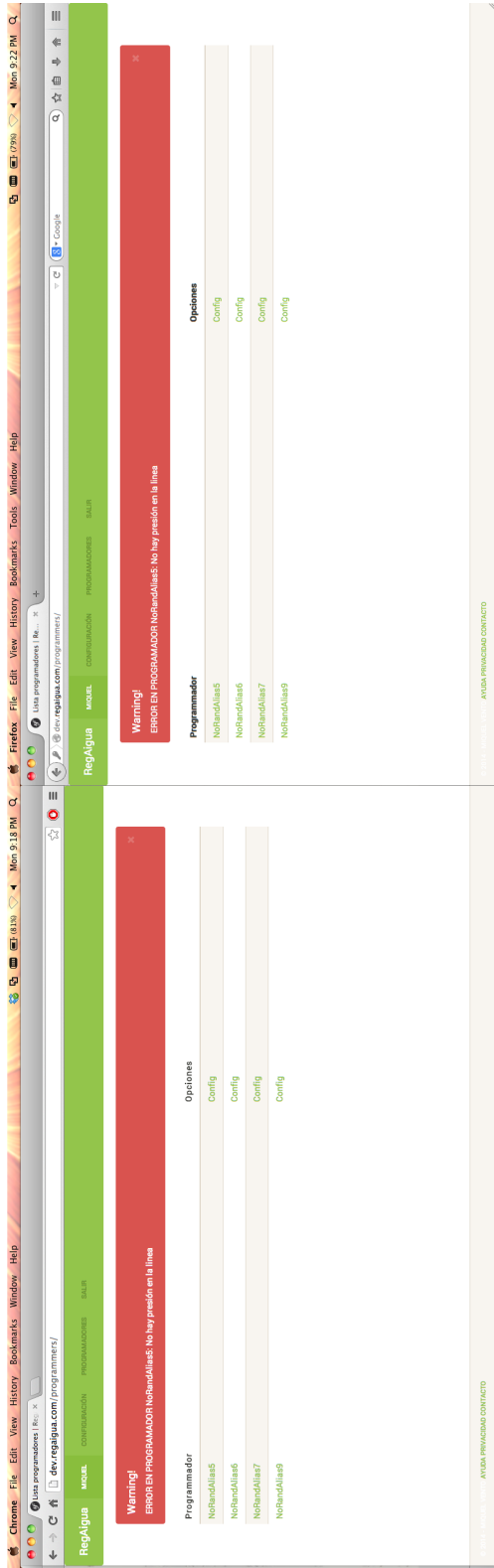


(c) Firefox Windows

Figura 6.24: Capturas para el formulario para añadir programa(1/3)



(a) Chromium Linux

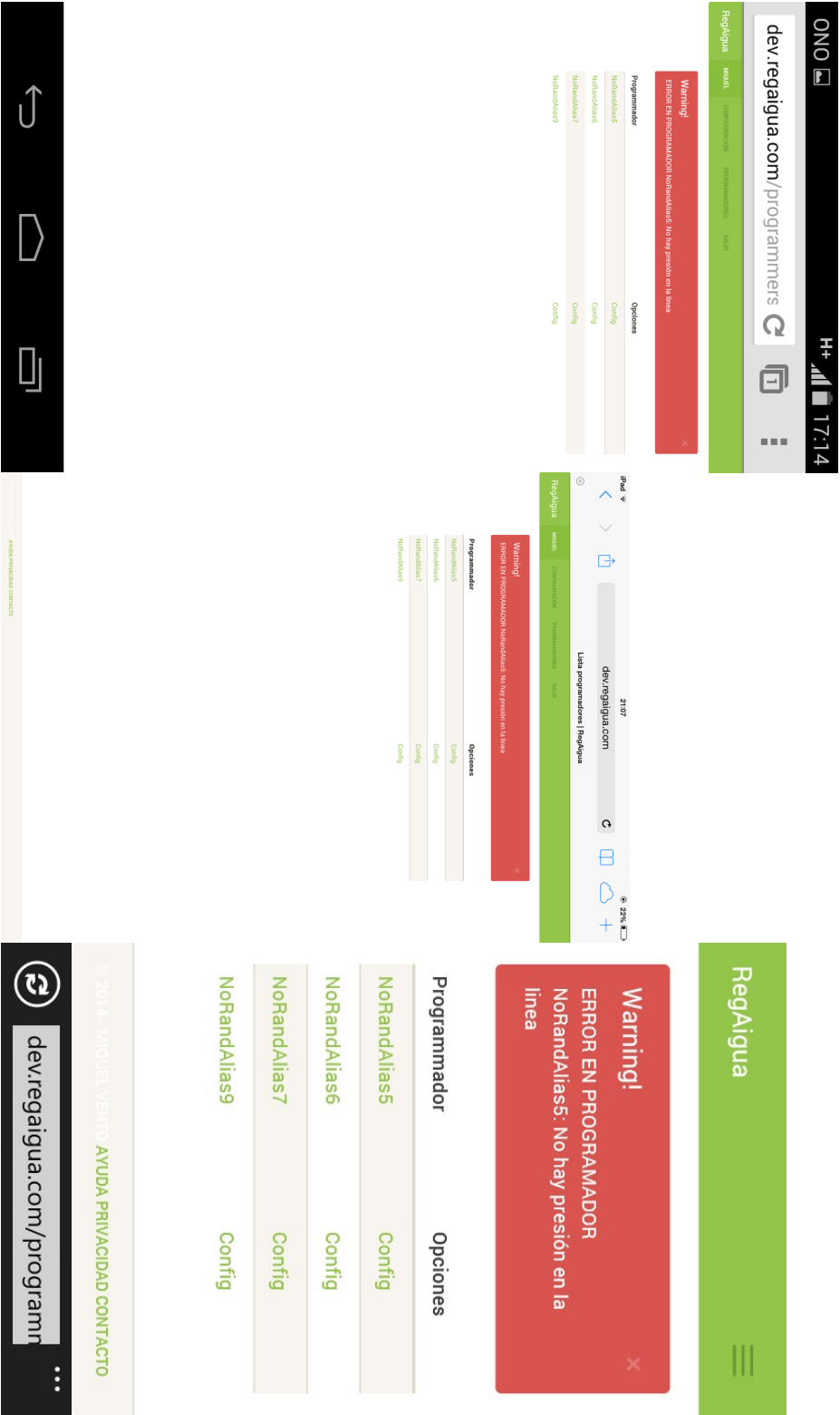


(b) Safari OSX

(c) Chrome OSX

(d) Firefox OSX

Figura 6.25: Capturas para el formulario para añadir programa(2/3)

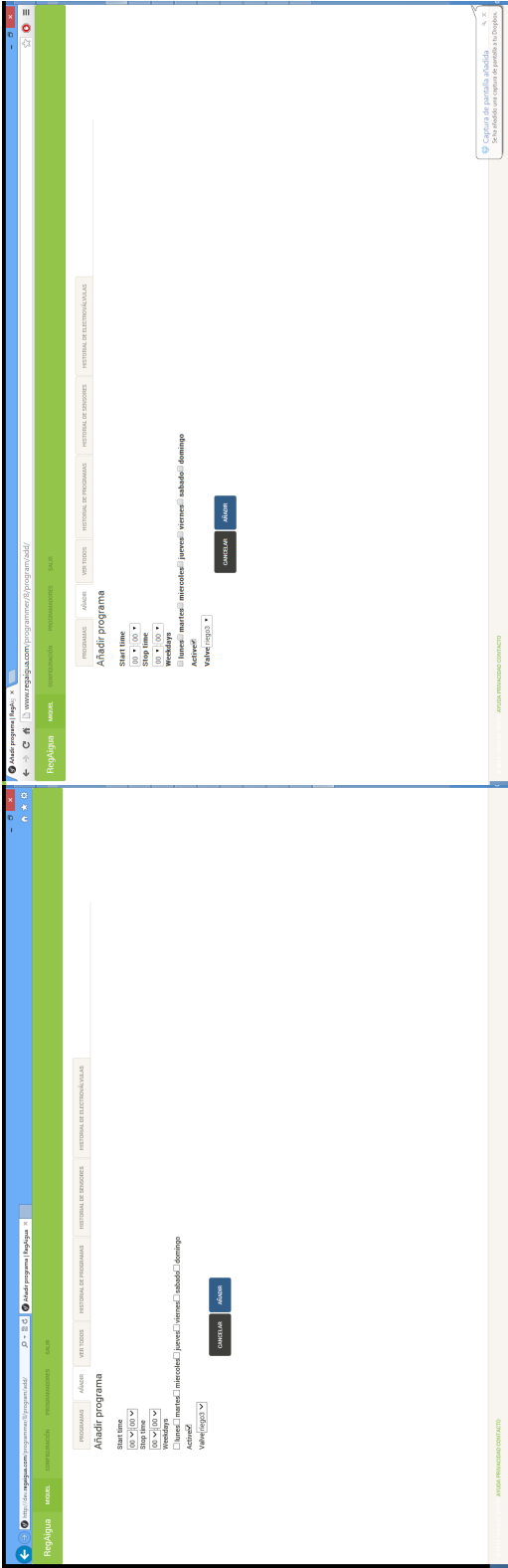


(a) Chrome Android

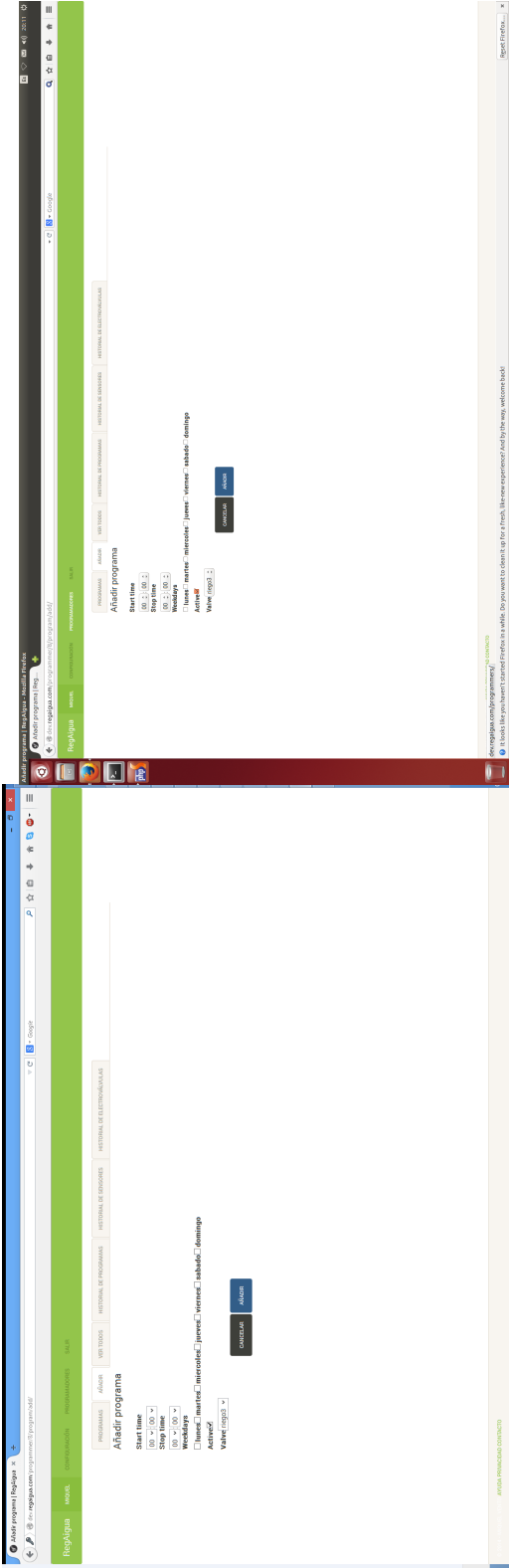
(b) Safari iOS

(c) Internet Explorer Windows Phone

Figura 6.26: Capturas para el formulario para añadir programa en dispositivos móviles (3/3)



(a) Internet Explorer Windows

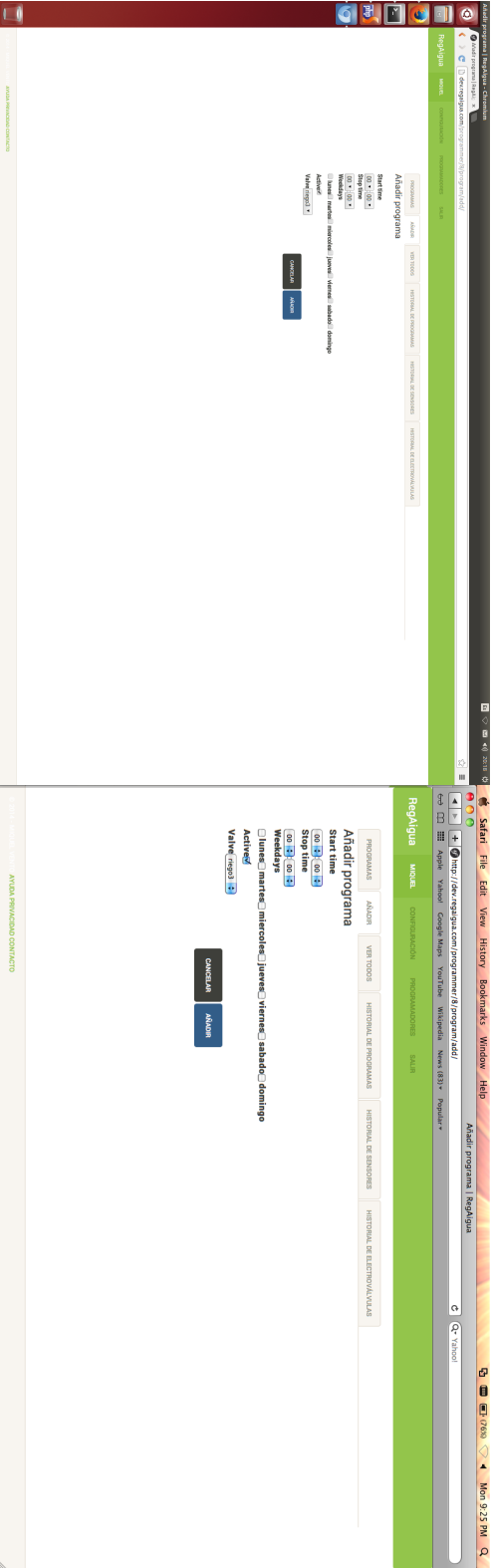


(b) Chrome Windows

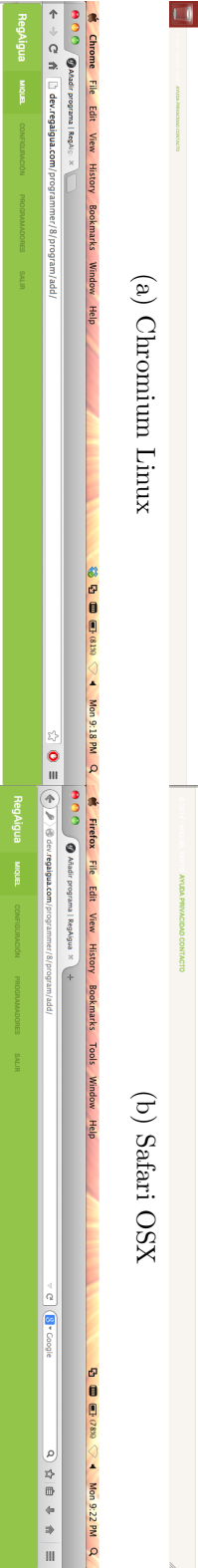
(c) Firefox Linux

(d) Firefox Linux

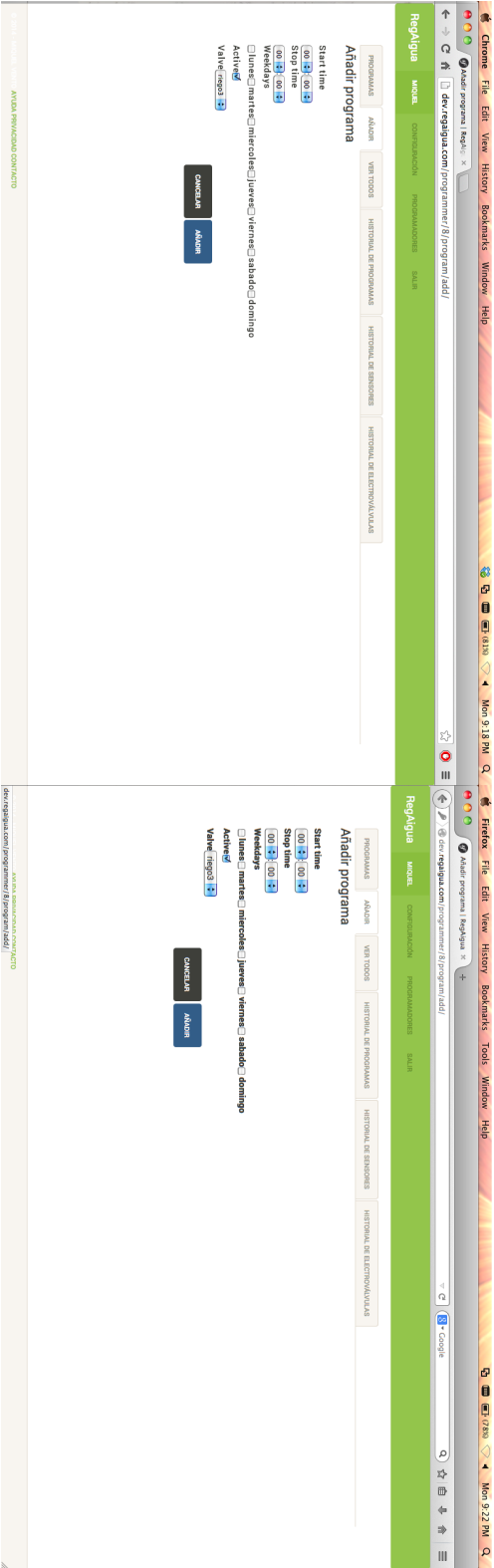
Figura 6.27: Capturas para el formulario para añadir programa(1/3)



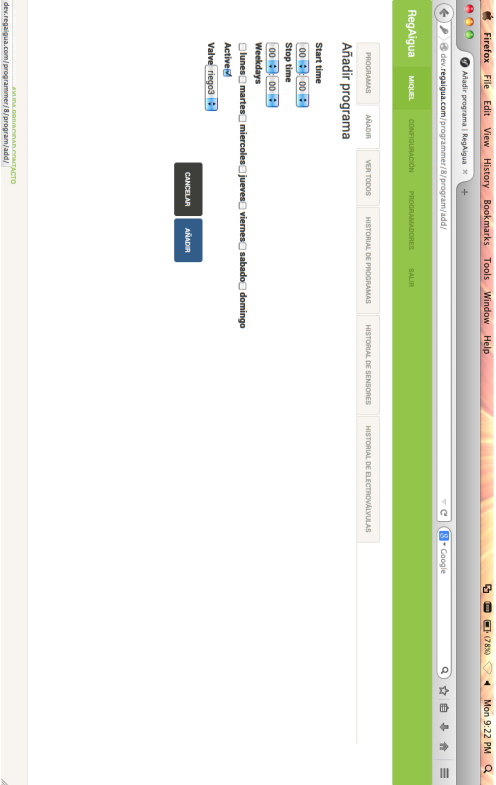
(a) Chromium Linux



(b) Safari OSX

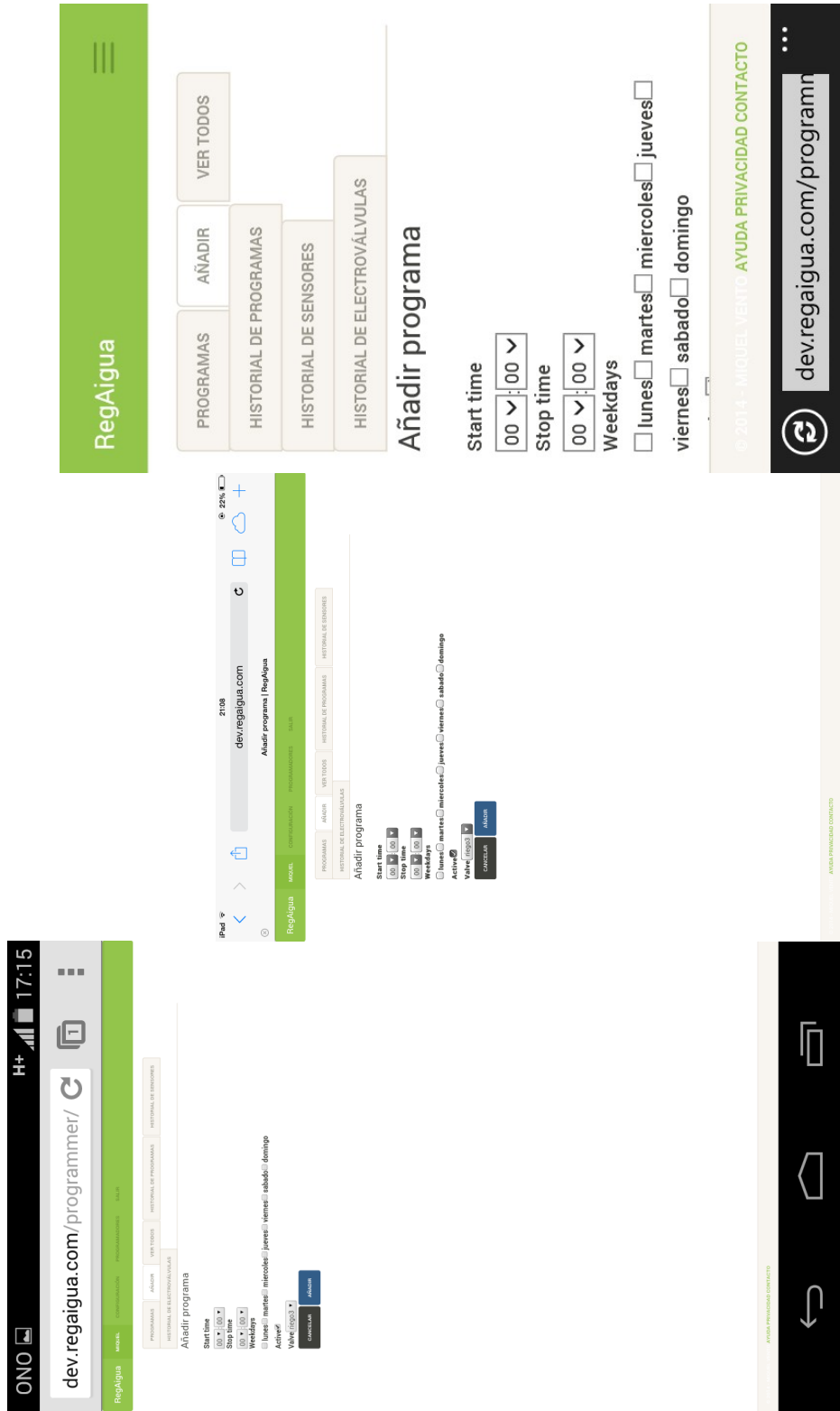


(c) Chrome OSX



(d) Firefox OSX

Figura 6.28: Capturas para el formulario para añadir programa(2/3)



(a) Chrome Android

(b) Safari iOS

(c) Internet Explorer Windows Phone

Figura 6.29: Capturas para el formulario para añadir programa en dispositivos móviles (3/3)

6.2.2.3. Resultados de rendimiento en las pruebas independientes

Como ya se ha definido durante la fase de descripción de las pruebas, estas se realizarán dos veces para distinto número de usuarios concurrentes.

Las pruebas se realizarán para distintos niveles de carga de la web y las tareas a realizar serán las anteriormente descritas.

	Esperada	Medio bajo	Medio alto	Alto	Extremo
Administradores	1	1	2	2	2
Usuario normal	10	25	35	50	75
Robots	5	10	15	20	25
Programadores	10	25	35	50	75

Cuadro 6.79: Usuarios concurrentes en el servicio

Los clientes serán lanzados durante un periodo de 4 segundos. El script de ejecución así como los resultados se pueden consultar en el CD adjunto.

A continuación mostramos las gráficas de tiempos de acceso al servicio para cada uno de los casos.

- **Carga de clientes esperada:**

Como se puede observar en la figura 6.30, el rendimiento para una carga esperada de clientes es más que aceptable. La carga de la mayoría de las páginas ronda los 50 ms, exceptuando la operación de identificación *sendForm* que ronda los 100 ms. Esto es debido a que se realiza una operación de encriptación, lo cual requiere más uso de CPU.

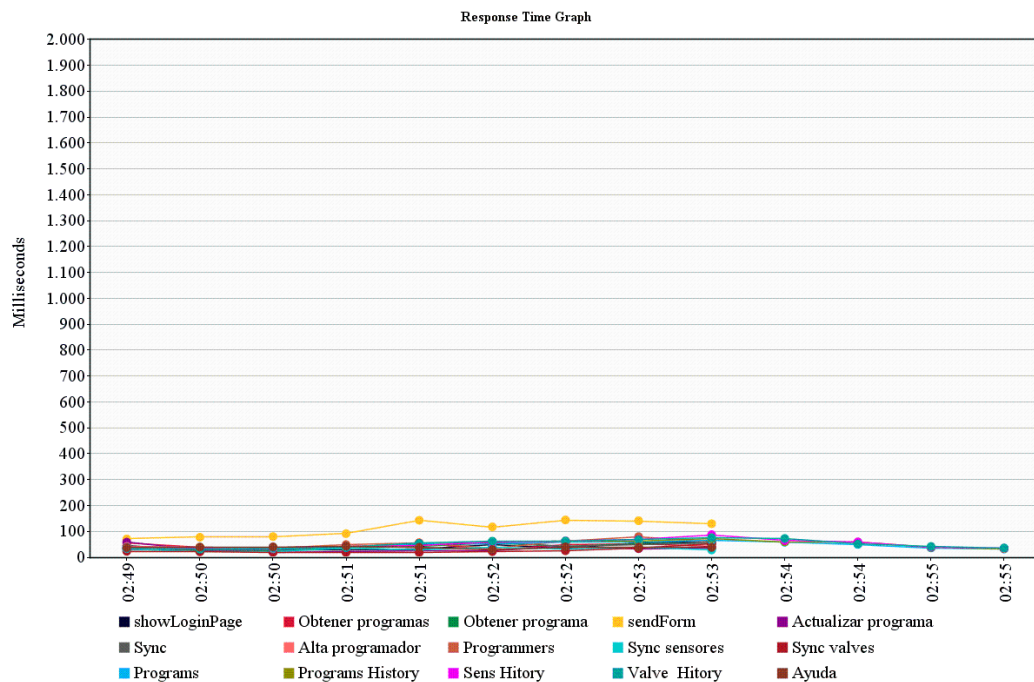


Figura 6.30: Resultados temporales para una carga normal

■ Carga de clientes media baja:

El comportamiento del servicio con una carga de clientes medio-baja, que podemos visualizar en la gráfica 6.31, resulta también más que aceptable. Durante la fase inicial se realizan las operaciones de mayor carga en el servidor tales como la autenticación en el servicio. El tiempo llega a alcanzar un pico de 1200 ms en el mayor punto de carga y concurrencia pero una vez logueado en el sistema, el tiempo de carga oscilará alrededor de 240ms, lo que resulta más que aceptable dadas las operaciones realizadas.

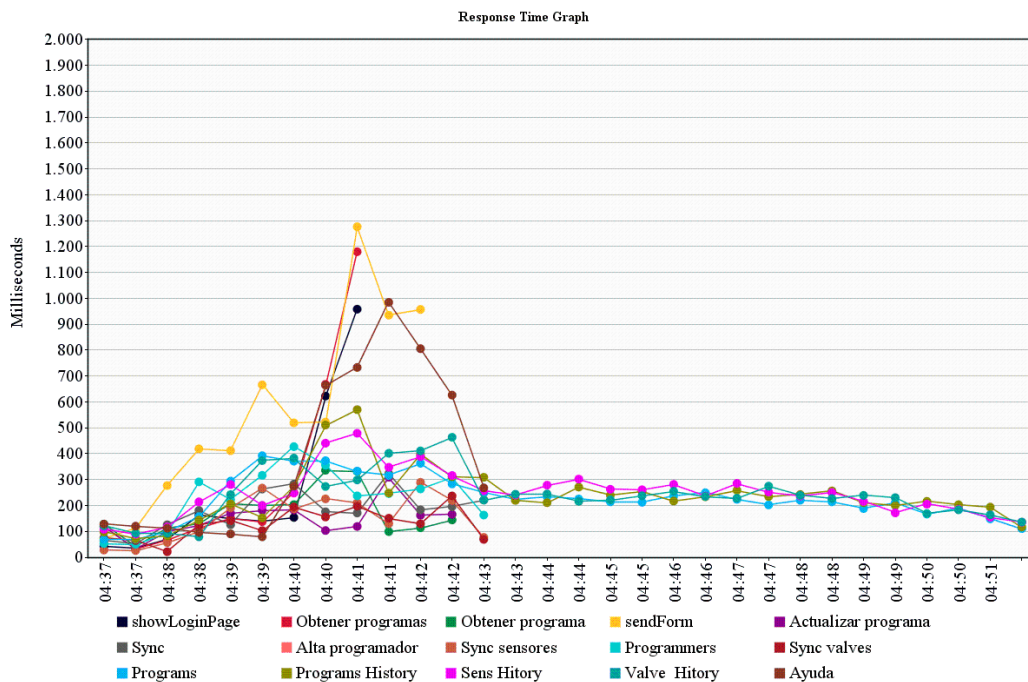


Figura 6.31: Resultados temporales para una carga media baja

■ Carga de clientes media alta:

En la figura 6.32 observamos que durante las operaciones de identificación, el tiempo de comprobación ha aumentado respecto al anterior aunque se respeta el pico máximo de carga. Posteriormente el valor de carga se estabilizará algo por encima de los 350 ms lo cual sigue garantizando una buena experiencia de navegación para el usuario. El valor de carga de la página de login inicialmente no resulta preocupante ya que se asociará a un caso único pues no vuelve a alcanzar dichos valores.

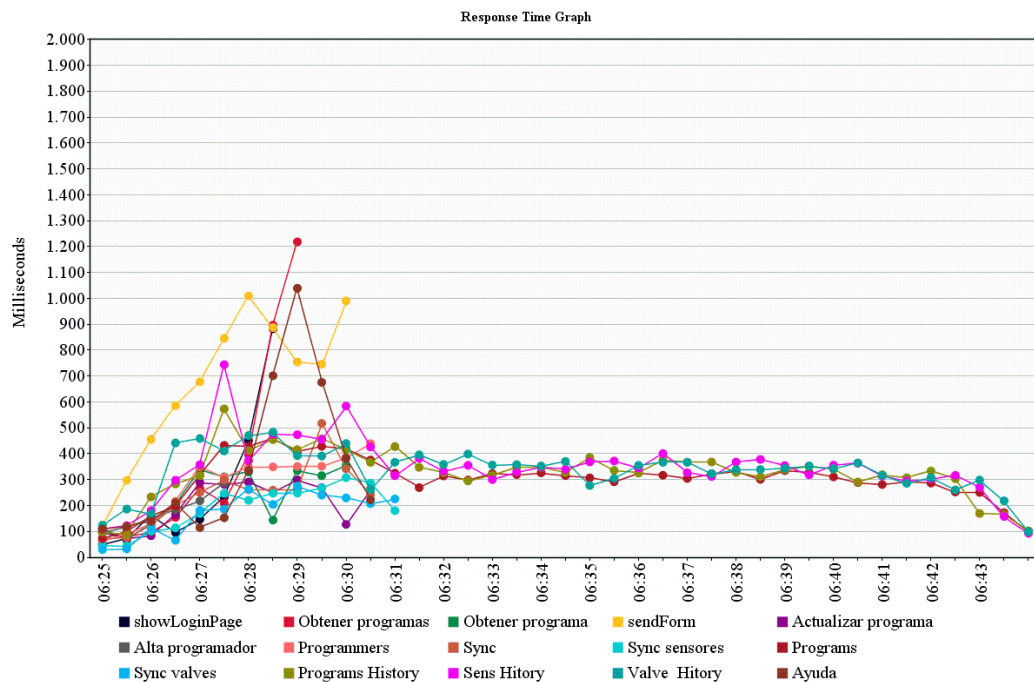


Figura 6.32: Resultados temporales para una carga media alta

■ Carga de clientes alta:

Para una carga de clientes igual a la talla máxima del proyecto, tal y como se refleja en la figura 6.33, la carga de páginas durante el proceso de identificación se eleva a picos por encima de los 1600 ms. De nuevo, no resulta preocupante ya que una vez pasado dicho proceso de identificación los tiempos de navegación rondan los 500 ms, lo que sigue siendo satisfactorio para la experiencia en el usuario dado que se encuentra en el uso máximo del servidor.

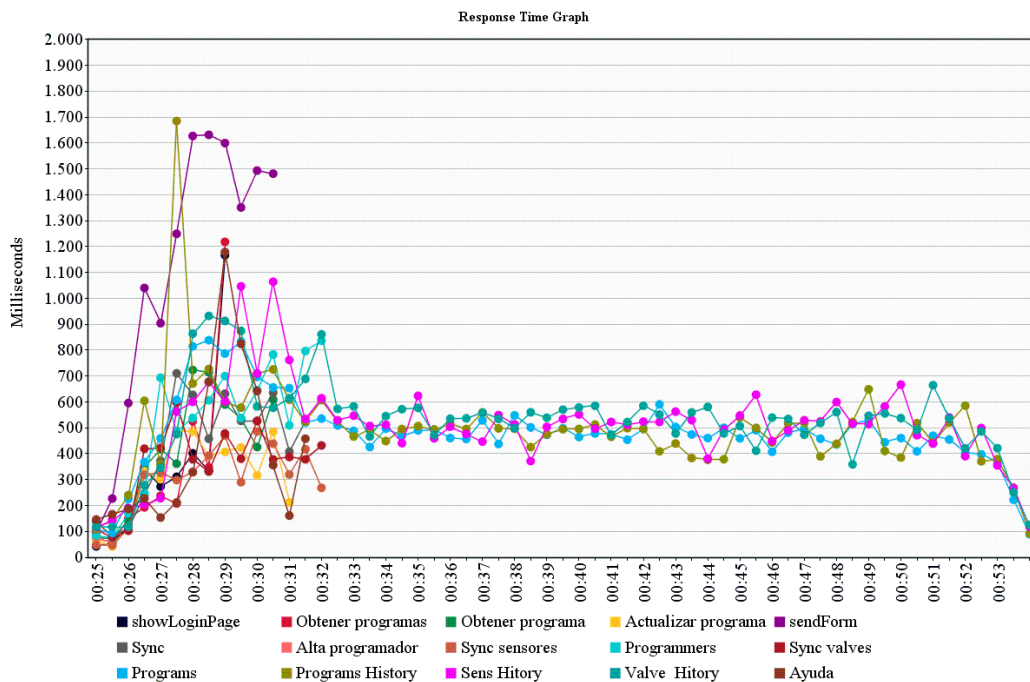


Figura 6.33: Resultados temporales para una carga alta

■ Carga de clientes externa:

Como observamos en la gráfica correspondiente a un acceso al servicio extremo 6.34 (en la cual se utiliza una **escala de tiempos distinta**). Este nivel de carga es un 25 % mayor que la cuota del proyecto sumado además los agentes externos. Como vemos, el tiempo de identificación al servicio se eleva hasta pasados los 1600 ms, lo cual resulta aceptable dada la carga del sistema. Pasado este periodo, el tiempo de carga de las páginas de los usuarios oscilará los 600 ms, lo cual sigue siendo aceptable para el servicio.

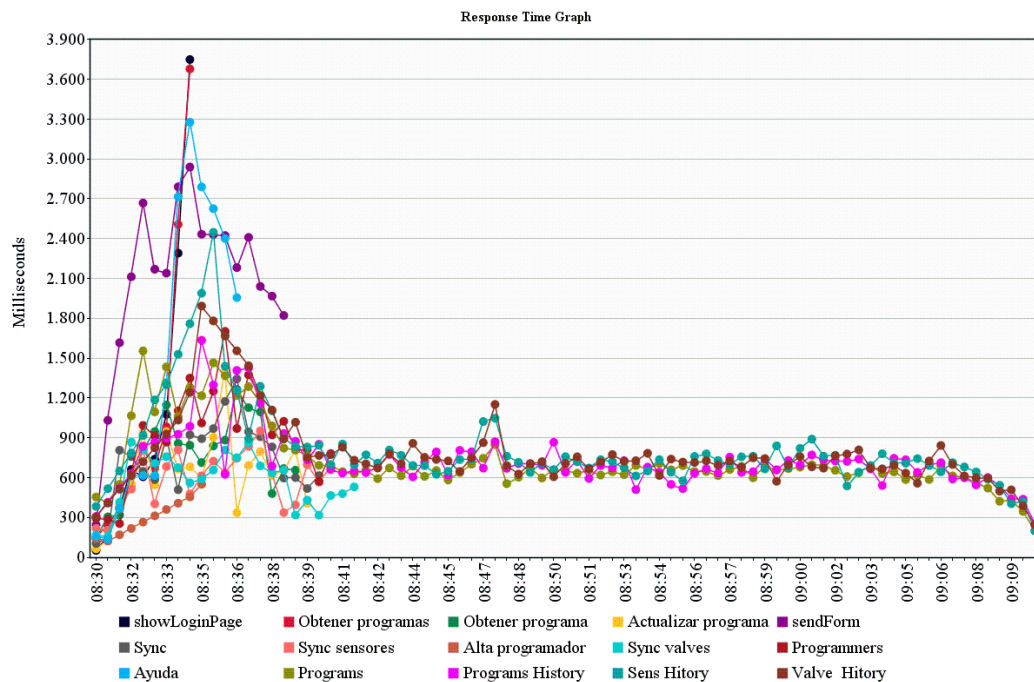


Figura 6.34: Resultados temporales para una carga extrema

Realizadas ya todas las pruebas de rendimiento en el servidor, concluimos que los resultados obtenidos han sido más que aceptables. Los tiempos máximos se alcanzan inicialmente cuando todos los usuarios realizan la operación de identificación concurrentemente, lo cual no será típico en el transcurso normal del servicio.

Podemos concluir, así, que el servicio web ofrece un rendimiento adecuado incluso cuando existe un número de peticiones concurrentes mayor a la talla del proyecto.

6.2.3. Resultados en pruebas sobre entorno real controlado

Las pruebas en un entorno real controlado han resultado exitosas. El sistema instalado se puede ver en la figura 6.35.



Figura 6.35: Instalación realizada para las pruebas de entorno real controlado

A continuación se expone el resumen de las operaciones que se han realizado:

- 186 Sincronizaciones con el sistema.
- 32 Aperturas y cierres de electroválvulas.
- 4 Aperturas/cierres manuales.
- 1 Cambios en la programación mediante la interacción física.
- 1 Cambios en la programación mediante el servicio web.
- 2 Fallos en el caudal de presión.
- 0 Fallos en el sistema.

Estado de la batería

Después de más de 15 días sin estar conectado a un cargador externo, el voltaje de la batería es de 3.91V, el mismo que al iniciar el proceso, por lo que podremos concluir que el proceso de carga de la batería es mayor o igual para ciclos de riego de una hora diaria.

6.3. Evaluación de proyecto al cierre

Una vez finalizadas todas las etapas, se deberá comparar el estudio de costes realizado en el apartado 4.5 con el resultado final.

Para ello realizaremos inicialmente el diagrama de Gantt con los datos temporales reales. Este se refleja en la figura 6.36. Como podemos observar, el proyecto se ha dilatado hasta un total de 263 días respecto a los 259 días estimados sin margen de seguridad. Este retraso ha podido ser absorbido por el margen de seguridad de 9 días que se añadió durante la fase de planificación. Esto implicará que el retraso producido en el proyecto no aumentará el coste del mismo.

Así pues, como el coste del proyecto se había calculado contando con el margen de seguridad y este no ha sido gastado al completo, se producirá un ahorro en el coste real del proyecto. En la ecuación 6.1 se refleja el coste real de los recursos humanos destinados al proyecto. Este nuevo resultado supondrá un ahorro total de 465 € en cuanto a los recursos destinados a los recursos humanos.

$$\text{Costes recursos humanos} = 93 \text{ €} * 263 \text{ días} = 24.459 \text{ €} \quad (6.1)$$

Dado que la desviación temporal es pequeña, no supondrá una variación real respecto a la amortización del software y el hardware, por lo que no será necesario realizar de nuevo el cálculo del mismo.

En la tabla 6.80 se muestra el coste final del proyecto.

Tipo	Coste
Recursos humanos	24.459 €
Software	26 €
Hardware	590,12 €
Total (directo)	25.075,12 €
Total (<i>overhead</i>)	30.090,14 €

Cuadro 6.80: Resumen del coste económico total real del proyecto

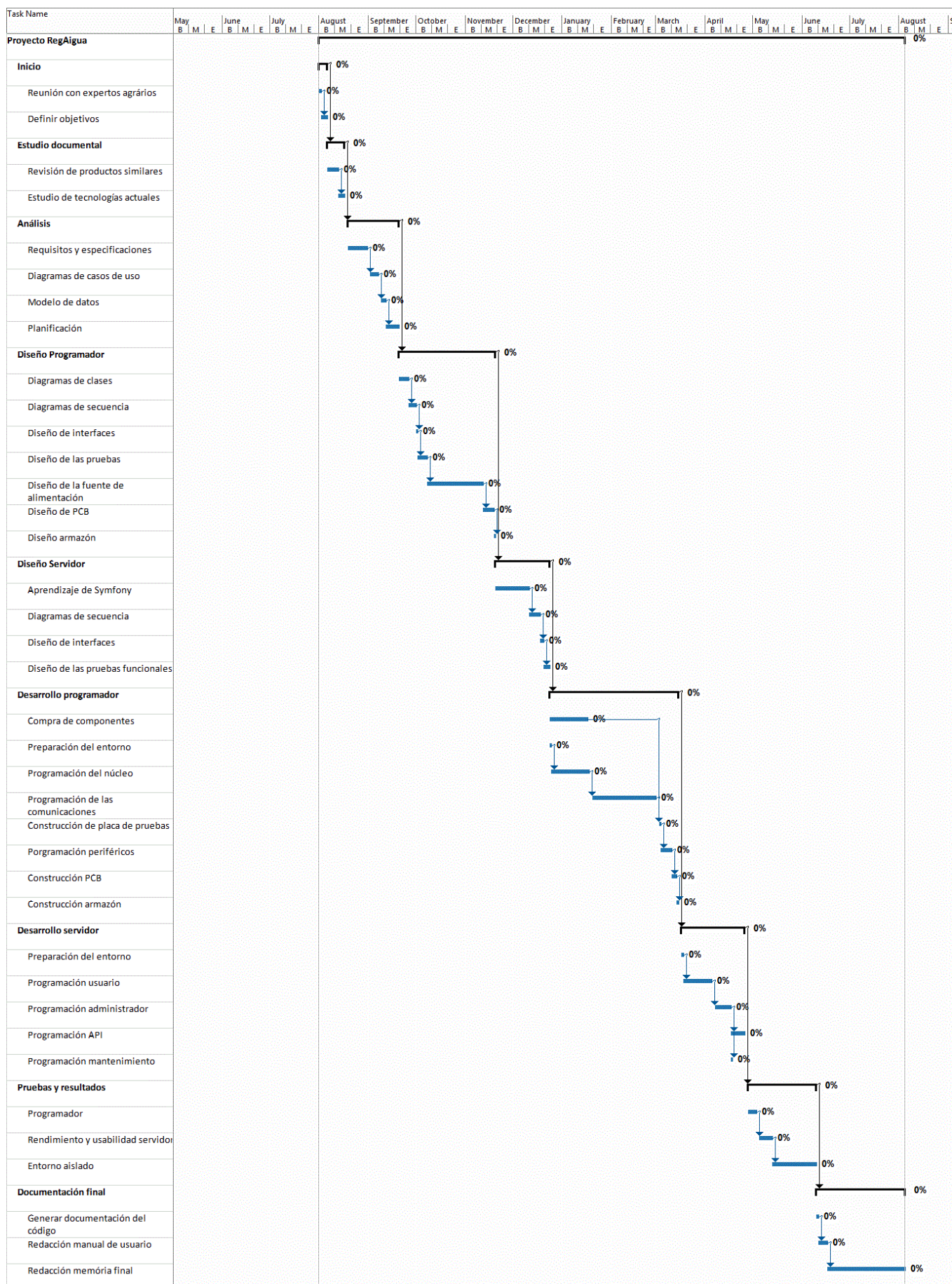


Figura 6.36: Diagrama de Gantt a partir de los datos reales

Capítulo 7

Conclusiones y trabajo futuro

Una vez finalizado el proyecto, será importante reflexionar respecto al mismo.

A lo largo de este capítulo trataremos de hacer una valoración global del trabajo realizado. El proyectando valorará desde un punto objetivo si se ha logrado cumplir con todos los requisitos planteados en el proyecto, las dificultades que se han encontrado a lo largo del desarrollo así como las posibles mejoras o cambios que se podrían realizar en un futuro para la mejora del producto.

7.1. Conclusiones

En primera instancia cabe destacar que la desviación temporal ha sido muy pequeña. Esto ha sido posible gracias a las técnicas de simulación utilizadas que permitieron estimar con gran precisión el tiempo total en el que se realizaría el proyecto pudiendo calcular así el margen de seguridad.

Por lo que respecta al desarrollo de proyectos TIC, se han puesto en práctica los conocimientos adquiridos durante la carrera aplicando un esquema tradicional de análisis, diseño, implementación y pruebas. También se han incorporado técnicas de cálculo estadístico que han ayudado a mejorar la precisión con la que se han estimado los plazos. Es necesario destacar que, para el desarrollo del servidor, se ha seguido un desarrollo guiado por pruebas¹, por lo que, una vez finalizada la fase de desarrollo, la ejecución de las pruebas ha sido correcta.

También será interesante remarcar que durante el proceso de aprendizaje se han adquirido conocimientos técnicos en tecnologías punteras como el *framework* Symfony, que permitirá al proyectando ampliar su perfil laboral

¹Test-driven development (TDD)

facilitando el acceso a nuevos puestos de trabajo. Dada la gran envergadura del proyecto diremos que también se ha reforzado las competencias en tecnologías ya conocidas.

Tras todo el trabajo realizado en el proyecto y expuesto a lo largo de la memoria presente, podremos decir que:

- Se ha conseguido crear un servicio que permite al agricultor gestionar sus programadores.
 - Realizando el diseño e implementación integro del sistema.
 - Diseñando una API de comunicaciones que permitirá la futura conexión con otros dispositivos.
 - Diseñado un sistema de alertas que permite alertar al agricultor de fallos sin la necesidad de desplazarse.
 - Programado el sistema con tecnologías punteras tales como Symphony, Twitter Bootstrap, jQuery, etc.
 - Comprobado su funcionamiento de forma experimental.
- Se ha conseguido crear un programador que gestiona pequeñas extensiones de riego de forma totalmente autónoma, evitando tener que reemplazar las baterías a menudo. Para ello se ha:
 - Realizado el diseño lógico de un nuevo programador.
 - Diseñado el circuito que abastecerá de energía ininterrumpidamente.
 - Realizado el diseño y fabricación de las PCB.
 - Programado el *firmware*.
 - Comprobado su correcto funcionamiento de forma experimental.
- Nuestro proyecto cumple con todos los requisitos definidos en el apartado 4.1 tanto en el programador como en el servidor.

Por las razones expuestas anteriormente, podremos concluir que el desarrollo del proyecto ha sido un éxito. Así pues, pasaremos al siguiente apartado, en el que se contemplarán cuáles serán los siguientes objetivos.

7.2. Trabajo futuro

En este apartado trataremos de definir las tareas antes del paso a producción así como las posibles mejoras a realizar en el producto final.

Previamente al paso a producción, se deberán realizar las siguientes tareas:

- **Rediseño de la pcb**

Durante la fase de prototipado, se ha optado por la elección de componentes *Through-Hole*, pues el tamaño es mayor que los de montaje superficial, facilitando así la fabricación de la pcb. Antes de realizar el paso a producción, será necesario rediseñar el sistema para el uso de componentes de montaje superficial (SMD), mucho más pequeños y que permitirán optimizar el tamaño de las placas, disminuyendo así en el precio final de producción.

- **Distribución de 10 prototipos**

Que distribuiremos entre gente de confianza para que realicen las pruebas en sus instalaciones agrarias. Esta fase deberá durar un mínimo de 3 meses. Se deberá realizar un estrecho seguimiento de los mismos, y monitorizar el estado del servidor.

- **Financiación**

Tal y como se expone en el apartado de viabilidad económica 4.6.1 se deberá conseguir una cantidad mayor o igual a 3.200 € para el inicio de esta fase.

A continuación se describirá la lista de posibles mejoras que deberían ser implementadas en futuras versiones del producto. Las propuestas que se citarán seguidamente no se han incluido en esta primera versión del producto, bien porque el tiempo a invertir en su desarrollo incrementaría notablemente el tiempo, bien porque se han detectado en la fase de integración en el medio (sección 6.2) y no resultaban críticas. Aun así, algunas de ellas serán necesarias para realizar el paso a producción del producto.

- **Multilenguaje**

Aunque el sistema ha sido preparado para albergar varios idiomas, durante esta primera versión del producto no se ha decidido invertir tiempo en la traducción del mismo.

- **Centralización de conexiones para comunidades agrícolas**

Dado que el mayor coste en el servicio se encuentra en el precio de alquiler de línea, en un futuro se planteará establecer nodos centrales

con conexiones vía radio que permitan concentrar las conexiones de varios programadores de la zona (figura 7.1). Aunque incrementará el precio de instalación, este sistema mixto será óptimo para comunidades agrícolas o cooperativas que decidan incorporar el servicio a sus clientes.

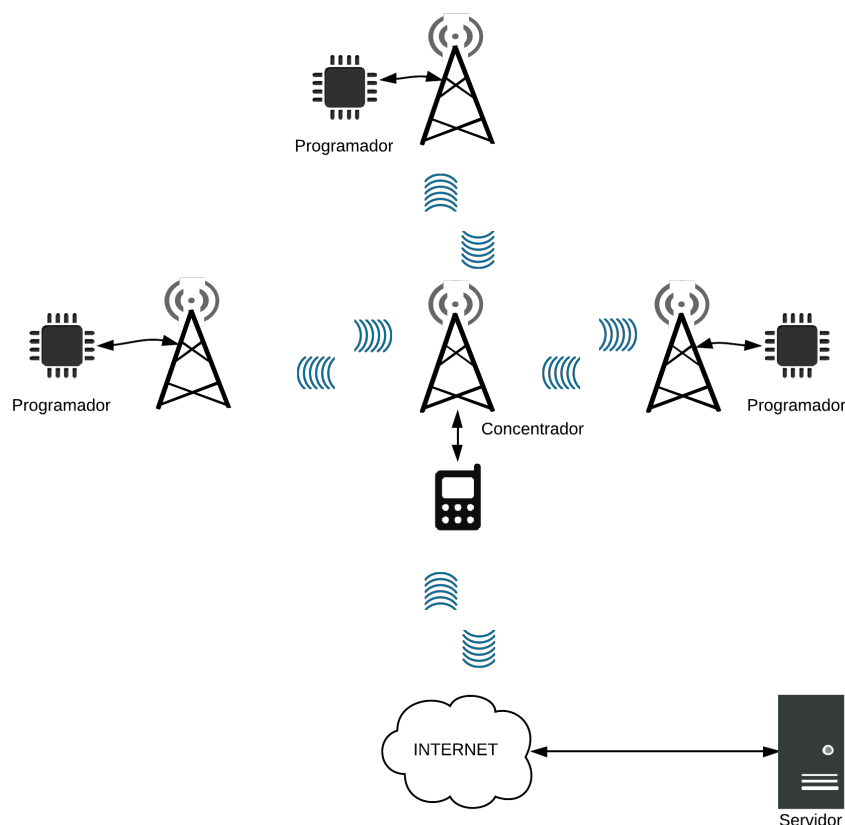


Figura 7.1: Esquema de conexión para un sistema mixto o concentrado.

■ Aplicaciones móviles

Sería de gran utilidad el uso de una aplicación nativa para *smartphone* ya que permitiría el uso de notificaciones vía push, de forma que el usuario podría ser informado de una forma más cómoda del estado de sus programadores.

Gracias a PhoneGap² podríamos portar la aplicación a las principales plataformas móviles de una forma muy sencilla y reutilizando gran parte de nuestro código.

²<http://phonegap.com/>

- **Expansión por distintos tipos de conexión y alimentación**

Una vez finalizado este primer prototipo, sería interesante aprovechar el potencial del mismo para hacerlo más versátil. Gracias a la programación por capas, resultará casi trivial utilizar diferentes tipos de conexiones tales como un módulo Ethernet para redes cableadas que permitan aprovechar las ventajas de cada situación.

- **Implementación de conexiones seguras**

Por restricciones en el hardware utilizado, no ha sido posible utilizar conexiones cifradas entre programador y servidor. Quedará, pues, como trabajo futuro la incorporación de protocolo SSL que permita este tipo de conexiones.

- **Diseño de LCD específico**

La fabricación de un LCD específico para el programador permitiría mejorar la usabilidad del programador así como incrementar su rendimiento, pues asegurarían un menor consumo.

- **Detección de bajante rota**

Una vez abierta una línea de riego, esta se distribuye en distintas bajantes con goteros que se encargan de el riego de todo el campo. A menudo, estas bajantes, son roídas por algún animal o rotas por un accidente, lo que imposibilita que el agua llegue al resto de árboles debido a la pérdida de presión. Sin embargo, la detección de esta pérdida de presión de línea en el origen de la misma (donde están actualmente situados los sensores de presión) es mínima y podría ser detectada como simple ruido en medias extensiones.

En futuras actualizaciones del producto sería interesante la incorporación de sensores a lo largo de la red que permitiesen detectar estas pérdidas de presión generando un aviso al cliente.

Bibliografía

- [1] *Ley orgánica Protección de datos*. BOE, (298), Diciembre 1999. <http://www.boe.es/boe/dias/1999/12/14/pdfs/A43088-43099.pdf>.
- [2] *MINISTERIO DE EMPLEO Y SEGURIDAD SOCIAL*. BOE, (189), Agosto 2004. <http://www.boe.es/boe/dias/2004/08/06/pdfs/A28377-28429.pdf>.
- [3] *Interfaz radioeléctrica reglamentada*. BOE, (276), Noviembre 2005. <http://www.boe.es/boe/dias/2005/11/18/pdfs/A37814-37815.pdf>.
- [4] *MINISTERIO DE EMPLEO Y SEGURIDAD SOCIAL*. BOE, (256), Octubre 2013. <http://www.boe.es/boe/dias/2013/10/25/pdfs/BOE-A-2013-11199.pdf>.
- [5] Arduino: *Arduino Uno*, Abril 2014. <http://arduino.cc/en/Main/arduinoBoardUno>.
- [6] Arduino: *Arduino UNO Rev3 Store*, Abril 2014. http://store.arduino.cc/index.php?main_page=product_info&cPath=11&products_id=195.
- [7] Arduino: *Arduino WiFi Shield*, Abril 2014. <http://arduino.cc/en/Main/ArduinoWiFiShield>.
- [8] Arduino: *What Arduino can do*, Abril 2014. <http://www.arduino.cc/>.
- [9] Atmel: *ATmega328*, Abril 2014. <http://www.atmel.com/devices/atmega328.aspx>.
- [10] Atmel: *Atmel Studio: Release 6.0*, Mayo 2014. <http://www.atmel.com/images/as6installer-6.0.1843-readme.pdf>.
- [11] Bach, Martin: *Beginning SQL Server 2012 for Developers*. Apress, 2012, ISBN 1430237503.

- [12] Bach, Martin: *Expert Consolidation in Oracle Database 12c*. Apress, 2013, ISBN 1430244283.
- [13] Barr, Michel: *Programming Embedded Systems in C and C++*. O'Reilly, Reading, Massachusetts, 1999, ISBN 0471491101.
- [14] Barr, Michel: *Apache: The Definitive Guide*. O'Reilly, 2003, ISBN 9780596002039.
- [15] BricoSolar: *Placa Solar Monocristalina 50Wp/12V*, Abril 2014. http://www.bricosolar.eu/epages/eb0043.sf/es_ES/?ObjectPath=/Shops/eb0043/Products/%22PS50/12%22.
- [16] BricoSolar: *Regulador solar 12V*, Abril 2014. http://www.bricosolar.eu/epages/eb0043.sf/es_ES/?ObjectPath=/Shops/eb0043/Products/%22PS50/12%22.
- [17] Caballer, Vicente: *Valoración agraria: Teoría y práctica*. Ediciones Mundi-Prensa, 2008, ISBN 9788484763451.
- [18] CleanPower: *Aerogenerador 12v*, Abril 2014. <http://www.yecoy.com/producto/0006000000011>.
- [19] CleanPower: *Micro Hydro Generator Pro*, Abril 2014. <http://www.seeedstudio.com/depot/36V-Micro-Hydro-Generator-Pro-p-1280.html>.
- [20] datos, Agencia española de protección de: *Guía de seguridad de datos*, 2008. http://www.agpd.es/portaleswebAGPD/canaldocumentacion/publicaciones/common/pdfs/guia_seguridad_datos_2008.pdf.
- [21] DB-Engines: *DB-Engines Ranking*, Abril 2014. <http://db-engines.com/en/ranking>.
- [22] Debrauwer, Laurent y Fien van der Heydeh: *UML 2: iniciación, ejemplos y ejercicios corregidos*. Ediciones ENI, 2005, ISBN 2746027984.
- [23] Deo, Akshay: *Django: A Web Framework with Immense Potential*, Marzo 2014. <http://www.opensourceforu.com/2013/03/django-a-web-framework-with-immense-potential/>.
- [24] Eguiluz, Javier: *Desarrollo web ágil con Symfony2*. Octubre 2012.
- [25] Energia: *Energia IDE*, Abril 2014. <http://energia.nu/>.

- [26] Farnworth, Demian: *Why use Ruby on Rails?*, Abril 2014. <http://blog.teamtreehouse.com/reasons-why-every-web-designer-needs-to-learn-rails>.
- [27] Freeman, Adam: *Bruce A. Tate and Curt Hibbs*. O'Reilly, 2006, ISBN 0596554788.
- [28] Freeman, Adam: *AngularJS*. O'Reilly, 2013, ISBN 1449355870.
- [29] Freeman, Adam: *Pro ASP.NET MVC 5*. Apress, 2013, ISBN 1430265299.
- [30] Foundation, Apache: *Apache HTTP SERVER PROJECT*, Abril 2014. <http://httpd.apache.org/>.
- [31] HawkEye: *OpenLRS HawkEye*, Abril 2014. http://www.aeroquadstore.com/433MHz_UHF_LRS_long_range_RF_transmitter_receiver_p/lrsng-005.htm.
- [32] Herity, Dominic: *C++ in embedded systems: Myth and reality*. eetindia, Febrero 1998. http://www.eetindia.co.in/ARTICLES/1998FEB/PDF/EEIOL_1998FEB02_EMS_TA.pdf.
- [33] IBM: *Compiled versus interpreted languages*, Abril 2014. http://publib.boulder.ibm.com/infocenter/zos/basics/index.jsp?topic=/com.ibm.zos.zappldev/zappldev_85.htm.
- [34] Industries, Hunter: *PC-BASED CENTRAL CONTROL SOFTWARE FOR WIDE-AREA SYSTEM MANAGEMENT*, Abril 2014. <http://www.hunterindustries.com/irrigation-product/central-control/imms>.
- [35] Industries, Hunter: *WCP/WVC Irrigation Controller*, Abril 2014. <http://www.hunterindustries.com/irrigation-product/controllers/wvpwvc>.
- [36] Instruments, Texas: *MSP-EXP430G2: buy*, Abril 2014. <http://www.ti.com/tool/msp-exp430g2#buy>.
- [37] Instruments, Texas: *MSP430 LaunchPad*, Abril 2014. <http://www.ti.com/tool/msp-exp430g2>.
- [38] ivia: *Datos de riego por estaciones*, Marzo 2014. <http://riegos.ivia.es/>.

- [39] Jeff Forcier, Paul Bissex y Wesley Chun: *Python Web Development with Django*. Addison-Wesley Professional, Octubre 2008, ISBN 0132356139.
- [40] Jones, Monique: *Advantages of using the web framework Django*, Marzo 2014. <https://exploreb2b.com/articles/advantages-of-using-the-web-framework-django>.
- [41] Lau, Dmitri: *Why use Ruby on Rails?*, Abril 2014. <http://www.sitepoint.com/10-reasons-use-angularjs/>.
- [42] libelium: *GPRS/GSM QUADBAND MODULE FOR ARDUINO / RASPBERRY PI (SIM900)*, Abril 2014. <http://www.cooking-hacks.com/gprs-quadband-module-sim900-for-arduino>.
- [43] Lucidchart: *Lucidchart Terms of Service*, Mayo 2014. <https://www.lucidchart.com/pages/tos>.
- [44] MariaDB: *MySQL Editions*, Abril 2014. <https://mariadb.org/en/about/>.
- [45] McRoberts, Michael: *Beginning Arduino*. Apress, Agosto 2013, ISBN 9781430250166.
- [46] Microsoft: *Benefits*, Abril 2014. <http://www.microsoft.com/en-us/server-cloud/products/sql-server/#fbid=5hXq3BP7EJ8>.
- [47] Microsoft: *How to buy SQL Server*, Abril 2014. <http://www.microsoft.com/en-us/server-cloud/products/sql-server/Buy.aspx#fbid=5hXq3BP7EJ8>.
- [48] Microsoft: *IIS Microsoft Oficial*, Abril 2014. <http://www.iis.net/>.
- [49] Microsoft: *MICROSOFT DREAMSPARK STANDARD CONTRATO DE SUSCRIPCIÓN*, Mayo 2014. <https://www.dreamspark.com/Institution/DS-EULA.aspx>.
- [50] Microsoft: *Why use ASP.NET?*, Abril 2014. <http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/ad2a22b9-135c-432a-bc9f-c67f074242b7.mspx?mfr=true>.
- [51] Microsoft: *Why use IIS?*, Abril 2014. <http://www.microsoft.com/web/platform/server.aspx>.
- [52] MikTeX: *Redistributing MiKTeX*, Mayo 2014. <http://miktex.org/copying>.

- [53] Mitsunaga, N.: *An interpreted language with debugging interface for a micro controller*. En *Consumer Electronics (GCCE), 2012 IEEE 1st Global Conference on*, páginas 115–119, Oct 2012.
- [54] msasdyql: *MySQL Editions*, Marzo 2014. <http://www.mysql.com/products/>.
- [55] Nedelcu, Clement: *Dimitri Aivaliotis*. Packt Publishing Ltd, 2013, ISBN 1849517452.
- [56] Nginx: *Why Use Nginx?*, Abril 2014. <http://wiki.nginx.org/WhyUseIt>.
- [57] Oracle: *Why Oracle Insight*, Abril 2014. http://docs.oracle.com/cd/B28359_01/license.111/b28287/editions.htm#DBLIC117.
- [58] Oracle: *Why Oracle Insight*, Abril 2014. <http://www.oracle.com/us/corporate/insight/why/index.html>.
- [59] O’Sullivan, Pdraig: *Comparing PostgreSQL 9.1 vs. MySQL 5.6 using Drupal 7.x*, Abril 2014. <http://posulliv.github.io/2012/06/29/mysql-postgres-bench/>.
- [60] Paradigm, Visual: *Academic Partner Program*, Mayo 2014. <https://www.visual-paradigm.com/partner/academic/academiclicense.jsp>.
- [61] PhpStorm: *License Comparison Matrix*, Mayo 2014. <http://www.jetbrains.com/phpstorm/buy/license-matrix.jsp>.
- [62] playground, Arduino: *Hidden Powers - storing data in Flash and EEPROM*, Mayo 2014. <http://playground.arduino.cc/Code/EEPROM-Flash>.
- [63] PostgreSQL: *Advantages*, Abril 2014. <http://www.postgresql.org/about/advantages/>.
- [64] Pressman, Roger S.: *Ingeniería del software: un enfoque práctico*. Mikel Angoar, 1997, ISBN 2746027984.
- [65] Progres: *Ficha Agrónic 2500*, Marzo 2014. <http://www.progres.es/htmls/productos/es/programadores-riego-y-fertirrigacion/ficha/agronic-2500.html>.

- [66] Progres: *Ficha Avisador GSM*, Marzo 2014. <http://www.progres.es/htmls/productos/es/accesorios/ficha/avisador-gsm.html>.
- [67] Progres: *Ficha Irri 5*, Marzo 2014. <http://www.progres.es/htmls/productos/es/programadores-riego-y-fertirrigacion/ficha/irri-5.html>.
- [68] Progres: *Programadores riego y fertirrigación*, Marzo 2014. <http://www.progres.es/htmls/productos/es/programadores-riego-y-fertirrigacion/ficha/irri-5.html>.
- [69] RainBird: *Ficha Agronic Net II*, Abril 2014. <http://www.progres.es/gestio/adjunts/1589%20FICHA%20A-NET%20II.pdf>.
- [70] RainBird: *RainBird WP Series*, Abril 2014. http://www.rainbird.eu/5-6558-Fiche-produit.php?id_produits=44.
- [71] SAGEM: *AT COMMAND SET HILO/HILONC MODULES*, 2009.
- [72] Schwartz, Marco: *How to Run an Arduino for Years on a Battery*, Abril 2014. <http://www.openhomeautomation.net/arduino-battery/>.
- [73] smarta: *Friends, family and fools*, Abril 2014. <http://www.smarta.com/advice/business-finance/start-up-finance/friends-family-and-fools/>.
- [74] Stones, Richard y Neil Matthew: *Beginning Databases with PostgreSQL: From Novice to Professional*. Apress, 2013, ISBN 1430200189.
- [75] Tilkov, Stefan: *REST vs. Websockets, Oh My*, Abril 2014. <http://www.innoq.com/blog/st/2012/02/rest-vs-websockets/>.
- [76] Trello: *Trello Terms of Service*, Mayo 2014. <https://trello.com/legal>.
- [77] w3techs: *Usage of web servers for websites*, Marzo 2014. http://w3techs.com/technologies/overview/web_server/all.
- [78] Zhao, Michael: *60 miles wifi*. Forbes, Marzo 2007. <http://www.forbes.com/forbes/2007/0409/076.html>.

Apéndice A

Symfony: la arquitectura

Symfony trata de un conglomerado de clases, archivos de configuración, etc. escritos generalmente en PHP y YALM. La *camelización* y el orden de los directorios juegan un papel muy importante a la hora de ejecutar este *framework* pues permitirán identificar automáticamente la presencia de nuevos directorios, archivos de configuración, etc.

```
1 Proyecto/
2 |- app/
3   |- cache/
4   |- config/
5   |- logs/
6   |- ...
7 |- src/
8   |- exampleBundle/
9     |- Controller/
10    |- Data Fixtures/
11    |- Entity/
12    |- Resources/
13    |   |- Config/
14    |   |- Tests/
15    |   |- ...
16    |- ...
17 |- web/
18   |- app.php
19   |- ...
```

Figura A.1: Estructura básica de symfony

En el la figura A.1 se muestra la estructura de directorios básica de Symfony. Seguidamente se realizará una explicación básica de cómo funciona la arquitectura del sistema, aunque se recomienda la lectura de los primeros capítulos del libro desarrollo web ágil con Symfony2 [24] para mayor detalle.

- app

La carpeta donde se encuentran los archivos principales de la arqui-

itectura están situados en el directorio `app`. En él se almacenan las configuraciones globales, recursos para la interacción el framework, etc. En él definiremos, por ejemplo, las configuraciones de routing, encargadas de indicar a la plataforma la ruta de directorios que se pueden utilizar. En este caso, enlazarán directamente a los diferentes bundles que creemos.

- `src`

En esta ruta es donde se almacenan los *bundles*. Un *bundle* es un bloque contenedor en el que se englobará un mismo tipo de datos. El criterio del tipo de datos a almacenar en un mismo *bundle*, es dictado por el desarrollador. Un *bundle* cuenta con una estructura interna propia como la que podemos observar en el caso de *exampleBundle*

- Controller

Contiene los controladores de una estructura MVC. Estos métodos serán llamados a través del archivo de configuración router. Cada clase del controlador debe finalizar siempre en `Action`. En estos métodos escribiremos la lógica del programa y suelen devolver la vista del modelo.

- Data Fixtures

En este directorio se guardarán los ficheros encargados de generar datos falsos para generar las pruebas.

- Entity

Guarda las entidades del sistema. Las clases generadas dentro de este directorio están directamente relacionadas con las entidades en la base de datos. Doctrine, el ORM que soporta oficialmente Symfony, será realmente sencillo crear el esquema con la base de datos.

- Resources

Entre otros directorios, contiene el de configuración. En él se encuentra el archivo *route.yml* que relaciona una dirección con un método del controlador.

- Tests

Contiene las pruebas que se encargan de comprobar el correcto funcionamiento del *bundle*.

- `web`

A este directorio es donde enlaza el servidor web. En él se encuentra la aplicación `app.php` encargada de consultar los archivos de configuración

almacenados en el directorio `app`, para resolver las rutas, seguridad, etc. También será el directorio donde almacenaremos archivos externos que quieran ser consultados con independencia del *framework* como es el caso de las imágenes.

La forma de añadir nuevas funcionalidades al sistema será creando nuevos *bundles*.