



Universidad
de Cádiz

ESCUELA SUPERIOR DE INGENIERÍA

INGENIERÍA INFORMÁTICA

**Asistente para el diseño de la interfaz para control,
seguimiento y sharing de robots en tiempo real**

Manuel López Urbina
Director: Arturo Morgado Estévez

Cádiz, 27 de mayo de 2017



ESCUELA SUPERIOR DE INGENIERÍA

INGENIERÍA INFORMÁTICA

RobotUI:

Asistente para el diseño de la interfaz para control, seguimiento y sharing de robots en tiempo real

- Departamento: Ingeniería en Automática, Electrónica, Arquitectura y Redes de Computadores
- Director del proyecto: Arturo Morgado Estévez
- Autor del proyecto: Manuel López Urbina

Cádiz, 27 de mayo de 2017

Fdo: Manuel López Urbina

0.1. Agradecimientos

Este proyecto significa la culminación de mi carrera, por lo que me gustaría dedicárselo a todas las personas que me han ayudado a conseguir acabarla.

En primer lugar me gustaría agradecerle, de igual modo que en el PFC de la Ingeniería Técnica, a mi familia el apoyarme y ayudarme durante estos años, y el esfuerzo que han hecho para que yo haya podido culminar mi ingeniería.

Mención especial para Natalia Luciano, mi pareja, la cual ha estado siempre apoyándome en mis objetivos y tanta paciencia y comprensión me ha mostrado tras tantos días, meses e incluso años de estudio y dedicación.

Agradecimientos a D. Arturo Morgado por su ayuda y dedicación durante la realización y dirección de este proyecto, así como su carácter amable y servicial que han hecho más ameno el trabajo realizado.

También me gustaría agradecérselo a mis compañeros, con los que tantos ratos inolvidables he pasado, y que tanto me han ayudado.

Por último quiero dedicarle este proyecto a todos los estudiantes de informática, en especial a todos los amantes del fascinante mundo de la robótica, a los que espero que mi trabajo les sea de utilidad.

0.2. Licencia

Este documento ha sido liberado bajo Licencia GFDL 1.3 (GNU Free Documentation License). Se incluyen los términos de la licencia en inglés al final del mismo.

Copyright © 2017 Manuel López Urbina.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Resumen

RobotUI es un proyecto web para el control y seguimiento de dispositivos robóticos en la red por multitud de usuarios.

Palabras clave: Internet, aplicación web, robótica, robots, interfaz de usuario, streaming de vídeo, control remoto, robot sharing, Sails.js, Node.js, Socket.io, Websockets, compartir, asistente, tiempo real, Raspberry Pi.

Índice general

0.1. Agradecimientos	I
0.2. Licencia	III
Índice general	I
Índice de figuras	v
1. Introducción	1
1.1. Introducción y antecedentes	1
1.2. Objetivos	4
1.3. Acerca de este documento	5
2. Conceptos básicos	7
2.1. Transmisión y comunicación	7
2.2. Socket	7
2.3. WebSocket	8
2.4. La arquitectura TCP/IP y el modelo OSI	9
2.5. Streaming	10
2.6. Framework	11
3. Estado del arte y herramientas utilizadas	13
3.1. Estado del arte	13
3.2. Tecnologías software utilizadas	13
3.2.1. L ^A T _E X	14
3.2.2. WebStorm	14
3.2.3. Github	14
3.2.4. Git	14
3.2.5. DigitalOcean	15
3.2.6. Node.js	16
3.2.7. Sails.js	16
3.2.8. Npm	17
3.2.9. SocketIO	17
3.2.10. FFmpeg	17
3.2.11. Bootstrap	18
3.2.12. JQuery	18
3.2.13. Mongo DB	19
3.2.14. Pm2	19
3.2.15. Robomongo	20

3.3. Tecnologías hardware y materiales utilizados	20
3.3.1. Raspberry Pi Model B	20
3.3.2. Controladora de motores L298N	22
3.3.3. Batería LiPo	22
3.3.4. Tarjeta de expansión con batería de Litio para Raspberry Pi	23
3.3.5. Cámara USB de alta definición	24
4. Desarrollo software	27
4.1. Metodología de desarrollo	27
4.2. Recolección de requisitos	28
4.2.1. Requisitos funcionales	28
4.2.2. Requisitos no funcionales	30
4.3. Diagramas de casos de uso	30
4.4. Descripción de los casos de uso	32
4.5. Diagrama de clases	41
5. Desarrollo de la interfaz	43
5.1. Elementos de la interfaz	43
6. Comunicaciones	45
6.1. Introducción	45
6.2. Conexión y suscripción	47
6.3. Desconexión	52
6.4. Envío de comandos al robot	55
6.5. Captura de datos del robot	56
7. Robot de pruebas	59
7.1. Análisis	60
7.1.1. Análisis de requerimientos hardware	60
7.1.2. Análisis de requerimientos software	60
7.2. Montaje	62
7.2.1. Interconexión de elementos	62
7.3. Software de control	66
7.3.1. Entrada/Salida	66
7.3.2. Comunicaciones	67
8. Organización temporal	77
8.1. Planificación temporal de tareas	80
8.1.1. Hito 1: Planificación y análisis	80
8.1.2. Hito 2: Definición de requisitos	81
8.1.3. Hito 3: Comienzo de desarrollo de la aplicación	81
8.1.4. Hito 4: Desarrollo de la aplicación, módulo componentes	81
8.1.5. Hito 5: Desarrollo de la aplicación, módulo interfaz	81
8.1.6. Hito 6: Desarrollo del módulo de comunicaciones	82
8.1.7. Hito 7: Construcción del vehículo de pruebas	82
8.1.8. Hito 8: Programación del vehículo de pruebas	82
8.1.9. Hito 9: Documentación	82

8.2. Diagrama de Gantt	82
9. Guía de Usuario	85
9.0.1. Objetivo de esta guía	86
9.0.2. Dirigido a	86
9.0.3. Obtener RobotUI	86
9.1. Uso de RobotUI	86
9.1.1. Acceso a la aplicación	86
9.1.2. Registro de usuario	87
9.1.3. Ingreso al sistema	89
9.1.4. Registro de un Robot	89
9.1.5. Gestión de permisos	91
9.1.6. Configuración de la interfaz	92
9.1.7. Ejemplo	95
9.1.8. Programa tu robot	96
9.1.9. Control de robots	101
9.1.10. Seguimiento de robots	102
9.1.11. Mensajes	104
9.1.12. Panel de administración	105
10. Comentarios finales	107
10.1. Presupuesto	107
10.2. Conclusiones	108
10.3. Mejoras futuras	109
Anexos	111
.1. Instalación de Node.js	111
.2. Instalación Sails.js	111
.2.1. Prerrequisitos	111
.2.2. Instalación	112
.3. Instalación de MongoDB	113
.3.1. Prerrequisitos	113
.3.2. Instalación	113
.4. Instalación del control de versiones Git	115
.4.1. Configuración	115
.5. Despliegue de una aplicación Sails	116
Bibliografía	119
GNU Documentation Free License	121

Índice de figuras

1.1. Logo RobotUI ¹	3
1.2. Página principal de RobotUI.	3
1.3. Imagen del vehículo de pruebas desarrollado ²	4
2.1. Representación de capas o niveles OSI y TCP/IP.	9
2.2. Representación de los sockets como una interfaz de la capa de transporte del protocolo TCP/IP.	10
3.1. Droplet desplegado en DigitalOcean	15
3.2. Imagen de una Raspberry Pi 3 Model B	21
3.3. Imagen de la controladora de motores L298n utilizada.	22
3.4. Imagen de la batería LiPo utilizada.	23
3.5. Imagen de la tarjeta de expansión con batería de Litio utilizado.	24
3.6. Imagen de la cámara USB utilizada.	24
4.1. Subsistemas existentes en el proyecto junto con el modelo de ciclo de vida utilizado para su desarrollo.	28
4.2. Diagrama de casos de uso para la interacción con la aplicación.	31
4.3. Diagrama de casos de uso para la interacción con el robot de pruebas.	32
4.4. Diagrama de clases para la BD.	42
6.1. Peticiones entre un navegador y un servidor http con y sin el empleo de websockets.	45
6.2. Representación de una sala compuesta por dos clientes.	46
6.3. Página de gestión de usuarios actualizable en tiempo real.	47
6.4. Esquema representativo del flujo de datos originado por un usuario controlador de un robot.	56
6.5. Esquema representativo del flujo de datos originado por un usuario tras la captura de datos procedentes del robot.	57
7.1. Imagen del robot de pruebas desarrollado.	59
7.2. Autómata representativo de los diferentes estados del robot.	61
7.3. Esquema GPIO de una Raspberry Pi Model B+.	62
7.4. Pines de entrada/salida del módulo L298N empleado.	64
7.5. Conjunto Raspberry Pi y módulo de expansión de alimentación.	64
7.6. Batería LiPo que alimenta los motores.	65
7.7. Esquema de conexiones del robot de pruebas.	65

7.8.	Vista superior del vehículo.	66
7.9.	Canales de comunicación abiertos por el robot.	68
8.1.	Panel de actividades - Trello	77
8.2.	Descomposición de las tareas implicadas en el desarrollo del proyecto (Primera Parte).	79
8.3.	Descomposición de las tareas implicadas en el desarrollo del proyecto (Segunda parte).	80
8.4.	Diagrama de Gantt 1. Desarrollo del proyecto.	83
8.5.	Diagrama de Gantt 2. Desarrollo del proyecto.	84
9.1.	Página principal RobotUI.	87
9.2.	Vista de la barra superior en modalidad usuario no identificado en el sistema.	87
9.3.	Vista de la barra superior en modalidad usuario identificado en el sistema.	87
9.4.	Formulario de registro de usuario.	88
9.5.	Información de un usuario.	88
9.6.	Formulario de creación de un robot.	90
9.7.	Vista informativa de un robot.	90
9.8.	Panel de configuración de los permisos de un robot.	92
9.9.	Panel informativo de un robot privado frente a uno público.	92
9.10.	Panel de configuración de la interfaz sin elementos añadidos.	93
9.11.	Panel de elemento (izquierda) y panel de acciones (derecha) para la configuración de la interfaz.	94
9.12.	Formulario para la creación de un botón.	95
9.13.	Vista tipo de una interfaz configurada.	96
9.14.	Panel informativo de un robot donde se aprecia su identificador.	97
9.15.	Robot a la espera de conexión entrante.	100
9.16.	Índice robots públicos.	101
9.17.	Panel informativo de las características del robot de la interfaz.	101
9.18.	Panel informativo de las órdenes enviadas al robot junto con una entrada de comandos para enviar nuevas órdenes directamente.	102
9.19.	Panel informativo de un robot en su modalidad ocupado.	102
9.20.	Vista del panel de seguimiento de un robot en tiempo real.	103
9.21.	Vista del panel de control de un robot junto con el listado de usuarios realizando el seguimiento.	104
9.22.	Formulario de redacción de un mensaje.	105
9.23.	Paneles de administración para robots y usuarios.	106
1.	Iniciando Sails.	112
2.	Sails en funcionamiento ³	113
3.	Utilización del gestor de procesos Pm2 en el entorno de producción.	118

Capítulo 1

Introducción

*Lo mejor que podemos hacer por otro
no es sólo compartir con él nuestras riquezas,
sino mostrarle las suyas*
Benjamin Disraeli

1.1. Introducción y antecedentes

La robótica es una rama de la ingeniería, la cual se ocupa del diseño, construcción, operación y uso de robots¹, así como sistemas informáticos para su control, retroalimentación sensorial y procesamiento de información. Entre las diversas disciplinas aplicadas a la robótica podemos encontrar: la mecánica, la electrónica, la informática, la inteligencia artificial, la ingeniería de control y la física, entre otras muchas, de lo cual podemos considerar la robótica como una ciencia multidisciplinaria.

En la actualidad, los robots comerciales e industriales son ampliamente utilizados y cada día realizan tareas de forma más exacta o más barata que los humanos. También se les utiliza en trabajos demasiado sucios, peligrosos o tediosos. Los robots son muy utilizados en plantas de fabricación, montaje y embalaje, en transporte, en exploraciones en la Tierra y en el espacio, cirugía, armamento, investigación en laboratorios y en la producción en masa de bienes industriales o de consumo. Otras aplicaciones incluyen la limpieza de residuos tóxicos, minería, búsqueda y rescate de personas y localización de minas terrestres. En definitiva, la robótica está presente en prácticamente cualquier ámbito que podamos imaginar.

Por otra parte, ninguno de los sistemas robóticos actuales podrían ser funcionales sin un software adecuado para su manejo y control, en ocasiones siendo éste tremadamente complejo y específico para garantizar una correcta sincronización entre los diferentes elementos hardware y software implicados con la finalidad de garantizar una correcta

¹Robot: Máquina automática programable capaz de realizar determinadas operaciones de manera autónoma y sustituir a los seres humanos en algunas tareas, en especial las pesadas, repetitivas o peligrosas; puede estar dotada de sensores, que le permiten adaptarse a nuevas situaciones.

armonía del conjunto.

Por estas razones cada vez son más las escuelas que hacen uso de la robótica para que los estudiantes se interesen en la tecnología, ya que pueden encontrar un entorno divertido donde aprender y que ofrece multitud de ventajas:

1. **Los niños lo encuentran divertido:** hay varios concursos orientados a distintos grupos de edad que pueden canalizar la competencia de una manera positiva. Por ejemplo, se le puede pedir a los niños que construyan un robot y luego hacer competiciones.
2. **Es una manera eficaz de enseñarles programación a los estudiantes:** la programación puede ser muy abstracta. Al tener que controlar un robot físico y ver lo que sale mal, los estudiantes aprenden lo que los robots pueden y no pueden hacer. También aprenden la necesidad de dar instrucciones precisas.
3. **Desarrolla habilidades útiles:** capacidad de resolución de problemas, trabajo en equipo, capacidad de análisis, y un largo etcétera.

Actualmente, además, existe una tendencia a la interconexión de aquellos elementos más cotidianos con internet, fenómeno conocido como Internet de las cosas o Internet of things en inglés. Con ello, se permite el control de multitud de dispositivos, desde gestión de stocks, geolocalización, control remoto, y un largo etcétera de posibilidades. Según la empresa Cisco², en 2020 habrá en el mundo aproximadamente 50 mil millones de dispositivos con un sistema de conexión al internet de las cosas³ ¿Por qué no añadir también nuestros proyectos robóticos a la red?

De todo lo anterior se extrae, por tanto, la necesidad de elaborar un sistema que, además de acercar la robótica a los estudiantes, permita compartir sus proyectos con otros usuarios en internet. Todos hemos visto alguna vez vídeos en las redes sociales donde los usuarios nos muestran sus dispositivos en funcionamiento donde, en ocasiones, nos gustaría poder tomar control sobre ellos o visualizar su manejo en tiempo real.

Por tanto el sistema resultante debe cubrir dos necesidades principales, la primera, dotar al usuario de las herramientas necesarias para permitir la configuración de una interfaz de control para dispositivos sin necesidad de amplios conocimientos de programación, y la segunda, cubrir la necesidad paralela en la que los usuarios, orgullosos de sus creaciones, dispongan de una manera de compartir sus robots con el resto del mundo de una manera más dinámica. Es decir, en la que otros usuarios, a modo de espectadores, puedan visualizar el control de los dispositivos por parte de su creador,

²Cisco Systems es una empresa global con sede en San José, California, Estados Unidos, principalmente dedicada a la fabricación, venta, mantenimiento y consultoría de equipos de telecomunicaciones.

³ Estudio referenciado en el libro *Internet of Things - From Research and Innovation to Market Deployment* [11].

como si de una sesión de vídeo en streaming se tratara. También se dotará de la posibilidad de permitir el control por otros usuarios externos.

Así que dadas las motivaciones existentes y, junto que la programación web y la robótica son temas que causan en mi un especial interés, hicieron que me lanzara a la elaboración de este proyecto que unifica ambos campos anteriormente citados.

Así surgió *RobotUI* y con él un nuevo concepto llamado *RobotSharing*.



Figura 1.1: Logo RobotUI ⁴.

RobotUI (nombre del sistema resultante) será una combinación de un elemento software (aplicación web) y hardware (vehículo de pruebas y demostración) surgido como muestra de la solución obtenida a los citados problemas.

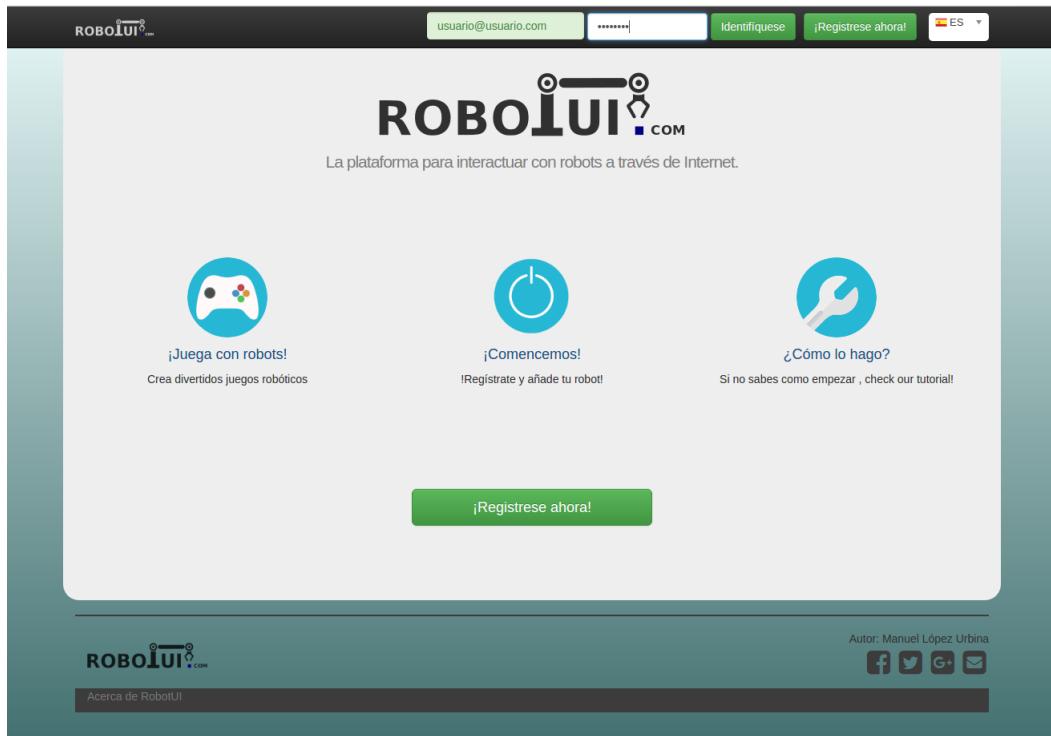


Figura 1.2: Página principal de RobotUI.

⁴Logotipo RobotUI.

El elemento hardware de este proyecto se compone de un vehículo controlado vía WiFi el cual responde a una serie de señales, *comandos*⁵, a los que responde realizando determinadas acciones. Por otra parte también dispone de una cámara para la captura de imágenes

La interfaz de control generada en la aplicación web se configurará de tal manera que permita el control del susodicho vehículo a modo demostrativo. Este procedimiento servirá de guía para que cualquier usuario pueda proceder a dar de alta sus dispositivos robóticos para su control y difusión con otros usuarios.

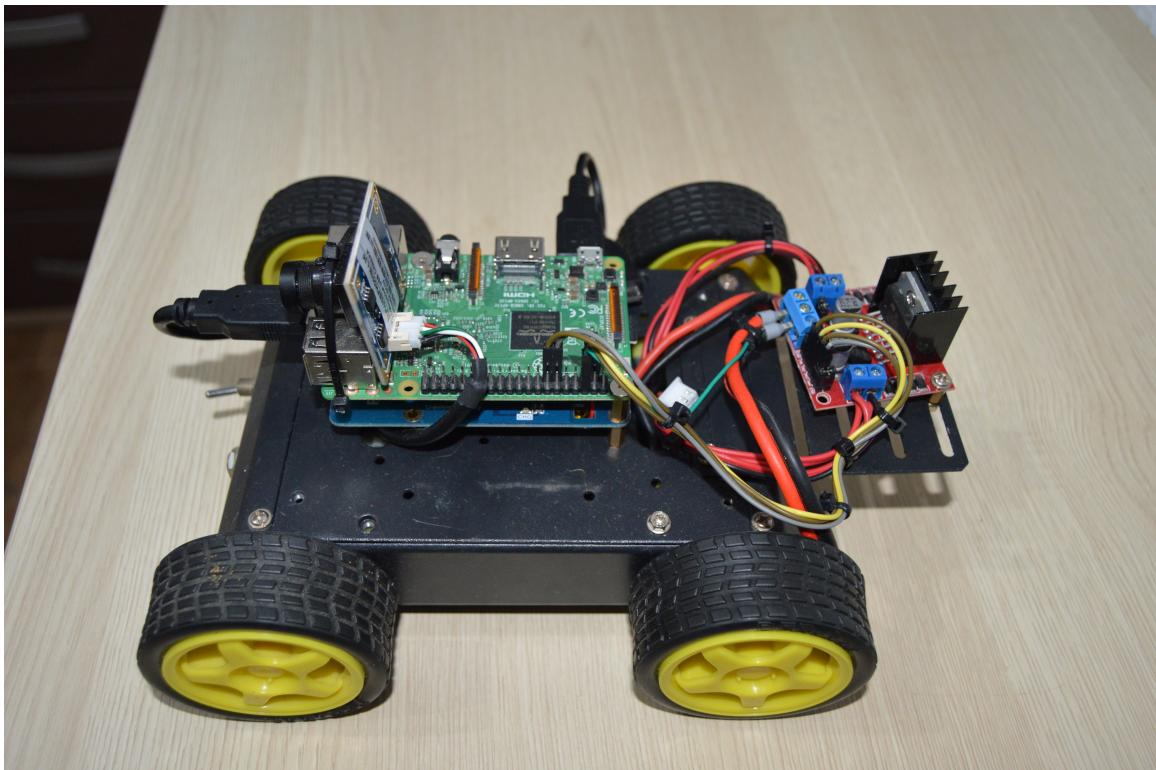


Figura 1.3: Imagen del vehículo de pruebas desarrollado ⁶.

1.2. Objetivos

Como hemos visto, se requiere de multitud de conocimientos a la hora de afrontar un proyecto robótico con ciertas garantías. Este proyecto trata, al menos, de reducir, o facilitar, el área relacionada con la informática, más concretamente con la programación donde multitud de personas ven un impedimento a la hora de comenzar a desarrollar

⁵ Comando: instrucción u orden que el usuario proporciona a un sistema informático, desde la línea de comandos (como una shell) o desde una llamada de programación.

⁶Vehículo desarrollado para probar el funcionamiento de RobotUI. Su desarrollo y características quedan descritas en el capítulo 7.

sus ideas.

Por otro lado, existe la imperiosa necesidad de que los usuarios quieran mostrar sus creaciones al resto del mundo, compartir experiencias, problemas, opiniones, etc, de una forma directa y no mediante la grabación de vídeos del funcionamiento de los proyectos robóticos en cuestión en la que solo se visualiza su funcionamiento sin la posibilidad de interactuar, ya que no disponen de una herramienta adecuada para ello.

De lo anterior deducimos que existe la necesidad de que otros usuarios puedan participar de manera más activa, ya sea visualizando el control del robot por parte de su creador o permitir que otros usuarios tomen el control de esos proyectos en tiempo real.

El sistema a desarrollar dispondrá de dos modos de funcionamiento, el primero de ellos proporciona las herramientas para la configuración de una interfaz a gusto del usuario, en la cual, una vez configurada, el usuario podrá controlar a su antojo el dispositivo. En el segundo modo de funcionamiento, la aplicación permitirá que otros usuarios puedan visitar la interfaz anteriormente configurada y actuar como espectadores en el control del robot por el usuario propietario del mismo. En definitiva, se proporcionará un sistema de control y difusión en uno solo.

En resumen, este proyecto busca facilitar la árdua labor de programación de los proyectos robóticos junto con la posibilidad de compartir las creaciones realizadas con otros usuarios.

1.3. Acerca de este documento

El documento se ha sido elaborado en un lenguaje claro y sencillo para permitir que un estudiante universitario de Ingeniería Informática pueda comprender los contenidos sin apenas dificultad añadida.

Este documento se organiza en los siguientes capítulos:

- En el capítulo 1, Introducción, se comentan las razones que han motivado la creación de este proyecto, así como el propósito del mismo.
- En el capítulo 2, Conceptos básicos, se incluyen definiciones de aquellos conceptos considerados de interés para la correcta comprensión del contenido de la presente memoria.
- En el capítulo 3, Estado del arte y herramientas utilizadas, se realiza una descripción de las diferentes elementos hardware y software empleados durante el desarrollo del proyecto y necesarios para la utilización del mismo. Así como una breve descripción del conocimiento acumulado y tecnologías existentes hasta la fecha.

- En el capítulo 4, Desarrollo software, se realiza un análisis sobre la metodología empleada para el desarrollo software, describiendo los modelos de ciclo de vida utilizados, la descripción de los requisitos funcionales y no funcionales junto con los diagramas de casos de uso, de clases y de secuencia.
- En el capítulo 5, Desarrollo front-end, se recogen aquellos aspectos técnicos de interés referentes a la elaboración de toda la parte visual de la aplicación.
- En el capítulo 6, Comunicaciones, se hace una descripción de las diferentes herramientas proporcionadas por el SDK del framework Sails.js para la gestión de eventos en tiempo real y la descripción de su funcionamiento mediante la explicación de casos prácticos desarrollados en el presente proyecto.
- En el capítulo 7, Robot de pruebas, se recogen aquellos aspectos referentes al montaje, configuración y programación de un robot de pruebas para la demostración y uso de la aplicación desarrollada en el presente proyecto.
- En el capítulo 8, Organización temporal, se recoge todo lo que concierne a la distribución y duración de cada una de las tareas llevadas a cabo durante el desarrollo del proyecto que el presente documento describe.
- En el capítulo 9, Guía de usuario, se describen los diferentes aspectos necesarios para la correcta utilización del conjunto software y hardware de los que se compone el presente proyecto.
- En el capítulo 10, Comentarios finales, se hace mención de las conclusiones obtenidas tras la realización del proyecto además de las posibles mejoras aplicables y presupuesto.
- En el capítulo Anexos 10.3, aparecen los manuales de instalación del software que ha sido necesario para la realización del proyecto.

Capítulo 2

Conceptos básicos

En el presente capítulo se recogen aquellos conceptos, definiciones, protocolos y diferentes aspectos que resultan de especial interés y que ayudarán a la comprensión de los diferentes puntos tratados en el resto de la memoria sin profundizar demasiado en detalles técnicos.

Todos estos conceptos se encuentran estrechamente ligados con tecnologías de la comunicación y transmisión de información, más concretamente en el ámbito de la programación web.

2.1. Transmisión y comunicación

Se denomina *transmisión* como el proceso de transporte de una señal de un lugar a otro y *comunicación* como el intercambio entre dos entes mediante una transmisión, los cuales son capaces de interpretar la información circundante entre ellos y en el cual existen un conjunto de reglas definidas, los protocolos¹, que rigen el proceso.

2.2. Socket

Socket, o también conocido como conector, designa un concepto abstracto mediante el cual dos programas, generalmente situados en computadoras distintas, pueden intercambiar cualquier flujo de datos de manera fiable y ordenada.

La comunicación a través de una red de ordenadores es una tarea compleja en la que para resolverla se ha empleado un enfoque de diseño por capas, pudiéndose hablar por tanto de una arquitectura o pila de protocolos donde cada capa utiliza servicios (funciones) de la capa inferior y ofrece servicios a la capa superior.

El modelo de referencia para la comunicación de ordenadores es el denominado modelo de Interconexión de Sistemas Abiertos OSI (Open Systems Interconnection), el cual queda descrito en la sección [2.4](#) junto con la localización de los sockets en dicho

¹Protocolo: reglamento o serie de instrucciones que se fijan por tradición o por convenio.

modelo.

El término *socket* es también usado como el nombre de una interfaz de programación de aplicaciones (API) para la familia de protocolos de red TCP/IP², provista usualmente por el sistema operativo.

Los sockets constituyen el mecanismo para la entrega de paquetes de datos provenientes de la tarjeta de red a los procesos o hilos apropiados. Un socket queda definido por un par de direcciones IP local y remota, un protocolo de transporte y un par de números de puerto local y remoto.

Cuando se habla de dirección y puerto local/remoto, se sobreentiende que nos referimos a dos procesos (cliente/servidor o nodo/nodo) ya que ambas direcciones IP y puerto pueden coincidir para el intercambio de información entre procesos dentro de una misma máquina y; además, la comunicación puede ser perfectamente bidireccional, asumiendo que el par que la inicia es el cliente y su contrapartida un servidor pero pudiendo ejercer de forma ambivalente ambas partes.

2.3. WebSocket

Comprendido previamente el concepto de *socket* descrito en el punto 2.2, definimos *Websocket* como una tecnología que proporciona un canal de comunicación bidireccional y full-duplex³ utilizada por cualquier aplicación cliente/servidor.

La API de WebSocket está siendo normalizada por el W3C, mientras que el protocolo WebSocket ya fue normalizado por la IETF⁴ como el RFC 6455.

Debido a que las conexiones TCP comunes sobre puertos diferentes al 80 son habitualmente bloqueadas por los administradores de redes, el uso de esta tecnología proporcionaría una solución a este tipo de limitaciones proveyendo una funcionalidad similar a la apertura de varias conexiones en distintos puertos, pero multiplexando diferentes servicios WebSocket sobre un único puerto TCP a costa de una pequeña sobrecarga del protocolo.

² TCP/IP es un conjunto de protocolos que permiten la comunicación entre los ordenadores pertenecientes a una red. La sigla TCP/IP significa Protocolo de control de transmisión/Protocolo de Internet. Proviene de los nombres de dos protocolos importantes incluidos en el conjunto TCP/IP, es decir, del protocolo TCP y del protocolo IP.

³Full Duplex: definido como la capacidad de transmisión y recepción en ambas direcciones al mismo tiempo.

⁴Internet Engineering Task Force (IETF) (en español, Grupo de Trabajo de Ingeniería de Internet) es una organización internacional abierta de normalización, que tiene como objetivos el contribuir a la ingeniería de Internet, actuando en diversas áreas, como transporte, encaminamiento, seguridad. Se creó en los Estados Unidos, en 1986. Es mundialmente conocido porque se trata de la entidad que regula las propuestas y los estándares de Internet, conocidos como RFC.

2.4. La arquitectura TCP/IP y el modelo OSI

En 1977 la Organización Internacional de Estandarización (International Standards Organization , ISO) estableció un subcomité encargado de diseñar una arquitectura de comunicación. El resultado fue el modelo de referencia para la Interconexión de Sistemas Abiertos OSI (Open Systems Interconnection). Dicho modelo define una arquitectura de comunicación estructurada en siete niveles verticales. Dicho modelo es utilizado como base teórica para el desarrollo de la arquitectura TCP/IP, la cual está compuesta por una serie de capas o niveles en los que se encuentran los protocolos que implementan las funciones necesarias para la comunicación entre dos dispositivos en red.

Siendo, por tanto, el modelo OSI el empleado en el estudio de las redes de datos y el modelo o arquitectura TCP/IP como un modelo real empleado es las redes actuales.

En la siguiente figura 2.1 se aprecian los niveles o capas de los modelos OSI y TCP/IP.



Figura 2.1: Representación de capas o niveles OSI y TCP/IP.

Los sockets dentro del modelo TCP/IP se pueden ver como una interfaz con la capa de transporte.

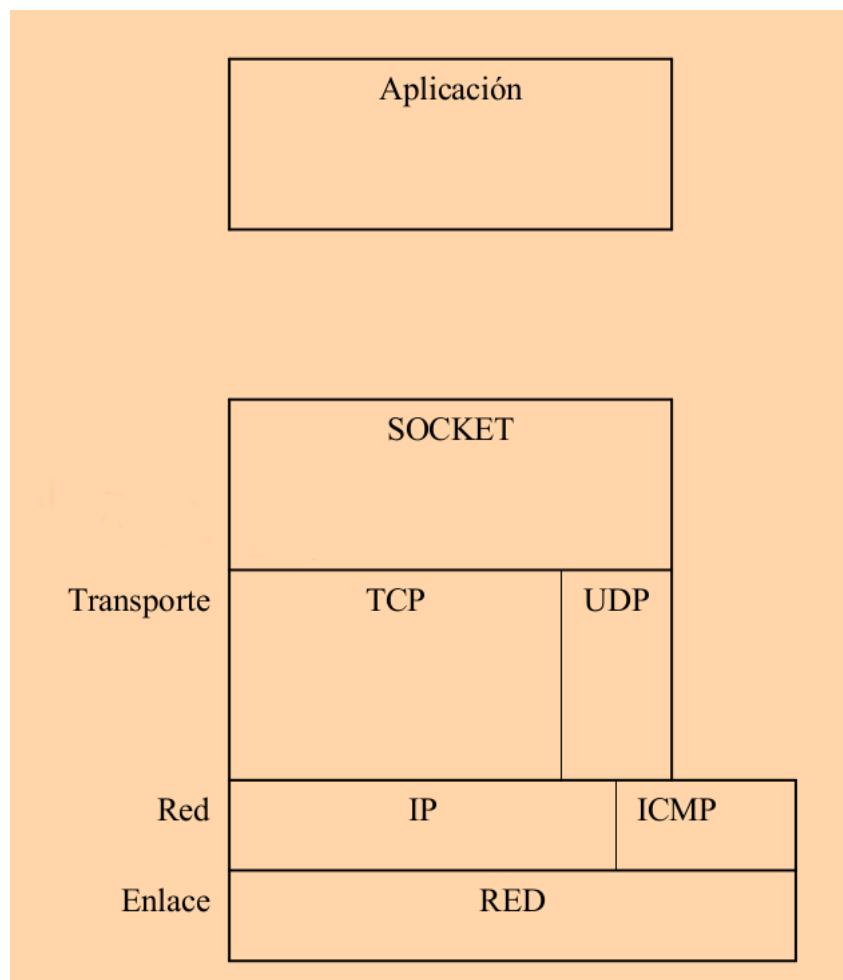


Figura 2.2: Representación de los sockets como una interfaz de la capa de transporte del protocolo TCP/IP.

2.5. Streaming

La retransmisión (en inglés streaming, también denominado transmisión) es la distribución digital de contenido multimedia a través de una red de computadoras, de manera que el usuario utiliza el producto a la vez que se es descargado. La palabra retransmisión se refiere a una corriente continua que fluye sin interrupción, habitualmente audio o vídeo, aplicándose la difusión de vídeo en el presente proyecto.

Este tipo de tecnología funciona mediante un búfer de datos que va almacenando el flujo de descarga en la estación del usuario para inmediatamente mostrarle el material descargado. Esto se contrapone al mecanismo de descarga de archivos, que requiere que el usuario descargue los archivos por completo para poder acceder al contenido.

La retransmisión requiere de una conexión por lo menos de igual ancho de banda que la tasa de transmisión del servicio. La retransmisión de vídeo por Internet se popularizó

a fines de la década de 2000, cuando la contratación del suficiente ancho de banda para utilizar estos servicios en el hogar se hizo lo suficientemente barato.

2.6. Framework

Un framework, también denominado entorno de trabajo o marco de trabajo, es un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar.

Aplicado a informática, concretamente al desarrollo de software, un entorno de trabajo o framework es una estructura conceptual y tecnológica de asistencia definida, normalmente, con artefactos o módulos concretos de software, que puede servir de base para la organización y desarrollo software. Típicamente, puede incluir soporte de programas, bibliotecas, y un lenguaje interpretado, entre otras herramientas, para así ayudar a desarrollar y unir los diferentes componentes de un proyecto.

En general, con el término framework, nos estamos refiriendo a una estructura software compuesta de componentes personalizables e intercambiables para el desarrollo de una aplicación. En otras palabras, un framework se puede considerar como una aplicación genérica incompleta y configurable a la que podemos añadirle las últimas piezas para construir una aplicación concreta.

Los objetivos principales que persigue un framework son:

- Acelerar el proceso de desarrollo
- Reutilización de código.
- Promover buenas prácticas de desarrollo como el uso de patrones.

Capítulo 3

Estado del arte y herramientas utilizadas

3.1. Estado del arte

En la actualidad podemos observar cómo la robótica cada vez se encuentra más presente en nuestras vidas cotidianas tanto como para facilitarnos ciertas tareas como para entretenimiento. Si realizamos cualquier búsqueda en internet podemos encontrar multitud de proyectos robóticos en la red. Existen excelentes portales como <http://blog.bricogeek.com/> por nombrar alguno de ellos, donde se publican multitud de proyectos novedosos y originales donde sus autores describen su desarrollo y por regla general lo acompañan de un vídeo donde se demuestra su funcionamiento.

Existen también comunidades que organizan competiciones de drones y/o robots, comunidades educativas asociadas a alguna tecnología concreta como pueden ser Arduino o Raspberry Pi, etcétera, donde nuevamente todas las descripciones de los diferentes proyectos se realizan con la documentación oportuna y en la mayoría de los casos acompañadas de un vídeo demostrativo con la inexistencia de interacción directa por parte del usuario.

Dado lo anterior, considero que hay un campo sin cubrir y que resultaría interesante abarcar este espacio con una aplicación con las características de RobotUI, ya que no existe ninguna plataforma que permita esta interacción usuario-máquina o usuarios-máquina en tiempo real y que sea configurable para cualquier dispositivo que se encuentre conectado a la red.

3.2. Tecnologías software utilizadas

A continuación se detallan las diferentes tecnologías/bibliotecas/lenguajes que se han empleado para la elaboración del proyecto y por qué se han escogido por encima de otras posibles soluciones.

3.2.1. LATEX

Web: <https://www.latex-project.org/>

LATEX es un lenguaje de marcado que sirve para la redacción de documentos científicos o técnicos. Con esta herramienta o lenguaje se ha desarrollado la memoria actual del proyecto de final de carrera.

3.2.2. WebStorm



Web: <https://www.jetbrains.com/webstorm/>

WebStorm es un IDE de JavaScript ligero pero potente, perfectamente equipado para el desarrollo del lado del cliente y el desarrollo del servidor con Node.js. Permite la integración con frameworks de desarrollo como Sails.js. Para el desarrollo de la aplicación se optó por este IDE.

3.2.3. Github



Web: <https://about.github.com/>

Repositorio: <https://github.com/lopi87/SAILS-RobotUI>

GitHub es una forja (plataforma de desarrollo colaborativo) para alojar proyectos utilizando el sistema de control de versiones Git. Utiliza el framework Ruby on Rails por GitHub, Inc. (anteriormente conocida como Logical Awesome). Desde enero de 2010, GitHub opera bajo el nombre de GitHub, Inc. El código se almacena de forma pública, aunque también se puede hacer de forma privada, creando una cuenta de pago.

3.2.4. Git



Web: <https://git-scm.com/>

Git es un sistema open-source de control de versiones diseñado para manejar íntegramente las fases de desarrollo de proyectos, simples y complejos, con velocidad y eficiencia.

3.2.5. DigitalOcean



Web: <https://www.digitalocean.com/>

Servidor web para alojar proyectos en la nube. La ventaja de este servicio de VPS ¹ es que te permite desplegar máquinas de cualquier tipo (siempre que sean software libre) de una manera muy fácil y rápida. Además tiene un punto fuerte y es que la información se almacena en discos SSD, con lo que el procesamiento se ve muy mejorado a la hora de computar (en este caso trabajo con Websockets y transmisión de datos).

 A screenshot of the DigitalOcean dashboard. At the top, there's a navigation bar with icons for Droplets, Images, Networking, API, and Support, and a green 'Create Droplet' button. Below the navigation is a search bar labeled 'Search by Droplet name'. The main area is titled 'Droplets' and shows a table of instances. The table has columns for Name, IP Address, Created, and Tags. One instance is listed: 'robotui' (IP 46.101.102.33, created 19 days ago). There are tabs for 'Droplets' and 'Volumes', and a 'More' dropdown menu.

Name	IP Address	Created	Tags
robotui	46.101.102.33	19 days ago	More

Figura 3.1: Droplet desplegado en DigitalOcean

¹ VPS: Servidor Virtual Privado, del inglés Virtual Private Server, es un método de particionar un servidor físico en varios servidores de tal forma que todo funcione como si se estuviese ejecutando en una única máquina. Cada servidor virtual es capaz de funcionar bajo su propio sistema operativo y además cada servidor puede ser reiniciado de forma independiente.

3.2.6. Node js



Web: <https://nodejs.org/es/>

Node.js es un entorno de ejecución para JavaScript construido con el motor de JavaScript V8 de Chrome. Node.js usa un modelo de operaciones E/S sin bloqueo y orientado a eventos, que lo hace liviano y eficiente. Incorpora un sistema de gestión de paquetes llamado, npm, es el ecosistema mas grande de librerías de código abierto en el mundo.

Node.js tiene una arquitectura basada en eventos capaz de E/S asíncronos. Estas opciones de diseño apuntan a optimizar el rendimiento y la escalabilidad en aplicaciones Web con muchas operaciones de entrada/salida, así como para aplicaciones Web en tiempo real (por ejemplo, programas de comunicación en tiempo real y juegos de navegador), lo que lo hacen ideal para este proyecto.

3.2.7. Sails js



Web: <http://sailsjs.com/>

Sails.js es un framework web que facilita la creación de aplicaciones Node.js de nivel empresarial. Está diseñado para asemejarse a la arquitectura MVC de otros frameworks como Ruby on Rails, pero con soporte para el estilo de desarrollo de aplicaciones web más moderno y orientado a datos.

Utiliza Express para funciones como la gestión de peticiones HTTP y Websockets. Su integración con el sistema de Websockets lo hacen especialmente bueno para construir características en tiempo real. Por estas razones se ha considerado como el framework más adecuado hasta la fecha para la elaboración de este proyecto.

3.2.8. Npm



Web: <https://www.npmjs.com/>

npm es el gestor de paquetes por defecto para Node.js, un entorno de ejecución para JavaScript. Utilizado para la descarga de las librerías incorporadas al proyecto.

3.2.9. SocketIO



Web: <https://socket.io/>

Socket.IO es una biblioteca de JavaScript para aplicaciones web en tiempo real. Permite la comunicación bidireccional en tiempo real entre clientes web y servidores. Consta de dos partes: una biblioteca del lado del cliente que se ejecuta en el navegador y una biblioteca del lado del servidor para Node.js. Ambos componentes tienen una API casi idéntica. Al igual que Node.js, es impulsado por eventos.

Socket.IO puede usarse simplemente como un wrapper para WebSocket aunque proporciona muchas más funciones, incluyendo la transmisión a múltiples sockets, almacenamiento de datos asociados a cada cliente y E/S asíncronas.

3.2.10. FFmpeg



Web: <https://ffmpeg.org/>

FFmpeg es una colección bibliotecas software libre que permiten grabar, convertir (transcodificar) y hacer streaming de audio y vídeo. Incluye libavcodec, una biblioteca de codecs. FFmpeg está desarrollado en GNU/Linux, pero puede ser compilado en la mayoría de los sistemas operativos, incluyendo Windows.

FFmpeg es un programa bastante sencillo y de fácil utilización, orientado tanto a personas con conocimientos avanzados como usuarios inexpertos.

El proyecto FFmpeg está compuesto por:

- ffmpeg: es una herramienta de línea de comandos para convertir audio o video de un formato a otro. También puede capturar y codificar en tiempo real desde DirectShow, una tarjeta de televisión u otro dispositivo compatible.
- ffserver: es un servidor de streaming multimedia de emisiones en directo que soporta HTTP (la compatibilidad con RTSP está en desarrollo). Todavía no está en fase estable, y de momento no está disponible para Windows.
- ffplay: es un reproductor multimedia basado en SDL y las bibliotecas FFmpeg.
- libavcodec: es una biblioteca que contiene todos los códecs de FFmpeg. Muchos de ellos fueron desarrollados desde cero para asegurar una mayor eficiencia y un código altamente reutilizable.
- libavformat: es una biblioteca que contiene los multiplexadores/demultiplexadores para los archivos contenedores multimedia.
- libavutil: es una biblioteca de apoyo que contiene todas las rutinas comunes en las diferentes partes de FFmpeg.
- libpostproc: es una biblioteca de funciones de postproceso de vídeo.
- libswscale: es la biblioteca de escalado de vídeo.

Para el desarrollo de RobotUI, concretamente para la transmisión de vídeo desde el robot de pruebas desarrollado hacia el cliente, el módulo utilizado ha sido el de la herramienta de línea de comandos.

3.2.11. Bootstrap



Web: <http://getbootstrap.com/>

Bootstrap es un framework o conjunto de herramientas de código abierto para diseño de sitios y aplicaciones web. Contiene plantillas de diseño con tipografía, formularios, botones, cuadros, menús de navegación y otros elementos de diseño basado en HTML y CSS, así como, extensiones de JavaScript opcionales adicionales. Se ha utilizado en el presente proyecto para la maquetación de la aplicación.

3.2.12. JQuery



Web: <https://jquery.com/>

JQuery es una biblioteca de JavaScript rápida, pequeña y característica. Hace que las cosas como manipulación del código HTML, manejo de eventos, animación, y permite la realización de peticiones Ajax de manera mucho más simple gracias a API de fácil manejo, la cual funciona a través de una multitud de navegadores. Gracias a su combinación de versatilidad y extensibilidad jQuery ha cambiado la forma en que millones de personas desarrollan con JavaScript.

3.2.13. Mongo DB



Web: <https://www.mongodb.com/es>

MongoDB (cuyo nombre proviene de la palabra en inglés “humongous” que significa enorme) es un sistema de base de datos NoSQL orientado a documentos, desarrollado bajo el concepto de código abierto.

La principal particularidad de las bases de datos NoSQL es que en lugar de guardar los datos en tablas como se hace en las bases de datos relacionales, guarda estructuras de datos en documentos similares a JSON con un esquema dinámico (MongoDB utiliza una especificación llamada BSON), haciendo que la integración de los datos en ciertas aplicaciones sea más fácil y rápida.

3.2.14. Pm2



Web: <http://pm2.keymetrics.io/>

PM2 es un gestor de procesos de producción para aplicaciones Node.js con un balanceador de carga incorporado. Permite mantener las aplicaciones en ejecución, recargarlas sin tiempo de inactividad y facilitar las tareas comunes del administrador de sistemas.

3.2.15. Robomongo



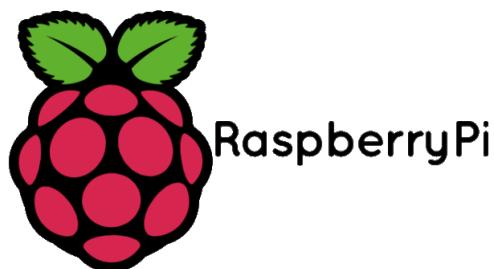
Web: <https://robomongo.org/>

Robomongo es una herramienta de gestión MongoDB de código abierto multi-plataforma basada en shell que incorpora el mismo motor de JavaScript que potencia el shell mongo de MongoDB. Una de sus particularidades es que Robomongo no sólo analiza la semántica del código, sino que también lo ejecuta en una máquina virtual JavaScript interna, lo que permite dotar a la herramienta de un autocompletado en tiempo de ejecución imposible de obtener de forma estática. Esta herramienta ha sido de especial ayuda para la gestión de la base de datos del proyecto.

3.3. Tecnologías hardware y materiales utilizados

A continuación se detallan las diferentes tecnologías hardware que se han empleado para la elaboración del vehículo robótico con la finalidad de servir de prototipo durante el desarrollo de la aplicación y a modo demostrativo de la misma. En los sucesivos puntos se describen las características de cada una de ellas junto con el motivo de su elección.

3.3.1. Raspberry Pi Model B



Web: <https://www.raspberrypi.org>

Raspberry Pi es un ordenador de tamaño reducido y de bajo coste desarrollado en Reino Unido por la Fundación Raspberry Pi, con el objetivo de estimular la enseñanza de ciencias de la computación en las escuelas. Es ampliamente utilizado y de uso muy extendido por lo que ha sido el principal motivo de su elección, además de su bajo cote

y versatilidad.

La Raspberry Pi 3 es la tercera generación de Raspberry Pi. Sus especificaciones son las siguientes:

- Una CPU ARMv8 quad-core de 64 bits de 64 bits y 1.2 GHz.
- LAN inalámbrica 802.11n.
- Bluetooth 4.1.
- Bluetooth baja energía (BLE).
- 1 GB de RAM.
- 4 puertos USB.
- 40 conexiones GPIO.
- Puerto HDMI.
- Puerto Ethernet.
- Conector de audio combinado de 3,5 mm y vídeo compuesto.
- Interfaz de la cámara (CSI).
- Interfaz de pantalla (DSI).
- Ranura para tarjeta Micro SD.
- VideoCore IV núcleo de gráficos 3D.



Figura 3.2: Imagen de una Raspberry Pi 3 Model B

3.3.2. Controladora de motores L298N

El módulo controlador de motores L298N H-bridge nos permite controlar la velocidad y la dirección de dos motores de corriente continua o un motor paso a paso de una forma muy sencilla, gracias a los 2 los dos H-bridge que dispone.

básicamente un puente-H o H-bridge es un componente formado por 4 transistores que nos permite invertir el sentido de la corriente, y de esta forma podemos invertir el sentido de giro del motor.

El rango de tensiones en el que trabaja este módulo va desde 3V hasta 35V, y una intensidad de hasta 2A. A la hora de alimentarlo hay que tener en cuenta que la electrónica del módulo consume unos 3V, así que los motores reciben 3V menos que la tensión con la que alimentemos el módulo.

Además el L298N incluye un regulador de tensión que nos permite obtener del módulo una tensión de 5V, perfecta para alimentar nuestro Arduino. Eso sí, este regulador sólo funciona si alimentamos el módulo con una tensión máxima de 12V.

Es un módulo que se utiliza mucho en proyectos de robótica, por su facilidad de uso y su reducido precio, lo cual ha servido para que sea seleccionado para el presente proyecto.

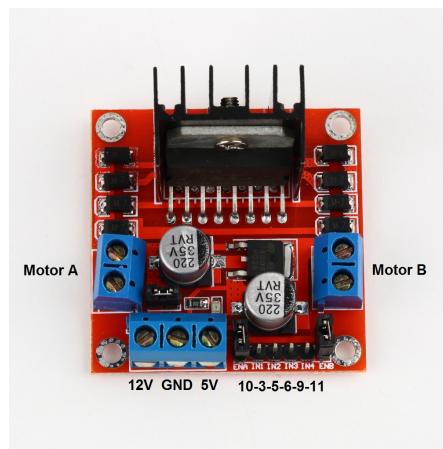


Figura 3.3: Imagen de la controladora de motores L298n utilizada.

3.3.3. Batería LiPo

Para alimentar los motores y su controladora se ha empleado una batería LiPo de 1000mAh a 3,7V. Las baterías de polímero de iones de litio, son pilas recargables (células de secundaria), compuestas generalmente de varias células secundarias idénticas en paralelo para aumentar la capacidad de la corriente de descarga. Siendo ideales para este tipo de usos.



Figura 3.4: Imagen de la batería LiPo utilizada.

3.3.4. Tarjeta de expansión con batería de Litio para Raspberry Pi

Para la alimentación de la placa se ha optado por un módulo de potencia diseñado especialmente para la Raspberry Pi 3 Model B, permitiendo que la placa maestra trabaje sin conexión hasta 9 horas de forma ininterrumpida.

Por otra parte, esta placa dispone de 2 puertos USB adicionales: uno suministra energía para la Raspberry Pi y el otro para una posible pantalla LCD, resultando interesante para otros proyectos.

Sus características principales son las siguientes:

1. Capacidad de la batería: 3800mAH.
2. Corriente de descarga máxima: 1.8A.
3. Tensión de salida sin carga: $5.1V \pm 0.1V$.
4. Corriente / voltaje de carga estándar: 1.0A / 5.0V.
5. Tensión de corte de la carga completa de la batería de iones de litio: 4.18V - 4.2V.

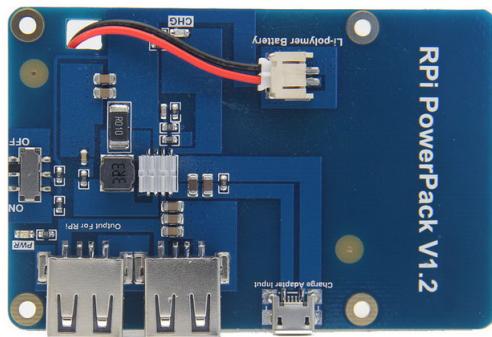


Figura 3.5: Imagen de la tarjeta de expansión con batería de Litio utilizado.

3.3.5. Cámara USB de alta definición

Cámara USB para su conexión en la Raspberry para la emisión de imágenes. La cámara seleccionada dispone de las siguientes características:

- 2 megapíxeles de resolución.
- Ángulo de visión de 170 grados.
- Interfaz USB 2.0 de alta velocidad, refresco de 60 fps en resolución 1280X720, 30 fps en resolución 1920X1080.
- Tamaño reducido y perfil delgado ideal para aplicaciones embebidas.



Figura 3.6: Imagen de la cámara USB utilizada.

Los motivos de su elección ha sido principalmente su facilidad de puesta en funcionamiento ya que al tratarse de una cámara USB. Tan sólo debemos conectarla para comenzar con su funcionamiento ya que es detectada por prácticamente todas las distribuciones Linux.

Capítulo 4

Desarrollo software

4.1. Metodología de desarrollo

Este proyecto ha sido elaborado empleando una metodología de desarrollo basada en el modelo de desarrollo incremental para la parte software referente a todos los subsistemas web y una metodología de desarrollo en cascada para el desarrollo de la parte software referente al robot de pruebas.

El modelo de desarrollo incremental proporciona una serie de características que lo hacen idóneo para este proyecto. Dicho modelo se basa en la filosofía de construir e ir incrementando las funcionalidades del sistema mediante el desarrollo de los diferentes módulos. Esto permite ir aumentando gradualmente las capacidades del software.

Dicha metodología de desarrollo resulta especialmente útil en las siguientes situaciones:

- Facilita el desarrollo permitiendo a cada miembro del equipo desarrollar un módulo particular. En el caso del presente proyecto me ha permitido desarrollar un módulo tras otro de una manera secuencial.
- Es similar al ciclo de vida en cascada aplicándose un ciclo en cada nueva funcionalidad del programa.
- A final de cada ciclo se entrega el software al cliente. En el caso que compete a este proyecto se mantenía una reunión con el director del proyecto para su aprobación.

Centrándonos nuevamente en el desarrollo del proyecto, los motivos que llevaron a cabo la elección de un modelo de desarrollo incremental viene dada por la necesidad de simplificar e ir desarrollando de una forma gradual y modularizada debido a la extensión del proyecto. Más si cabe que el equipo de desarrollo solo consta de una persona.

Por otro lado, para el desarrollo del vehículo de pruebas y por su simplicidad, se ha optado por un desarrollo en cascada. El modelo de desarrollo en cascada resulta adecuado en situaciones en las que:

- Se dispone de unos requisitos claros y precisos.
- El sistema a desarrollar es de pequeña envergadura.
- Las tecnologías utilizadas son conocidas por los desarrolladores.

Siendo precisamente éstas las características del proyecto del vehículo a desarrollar puesto que se trata de un desarrollo de pequeño tamaño y las herramientas empleadas ya me resultaban conocidas tras la realización de otros trabajos previos.

Por tanto el proyecto queda distribuido en los siguientes subsistemas:

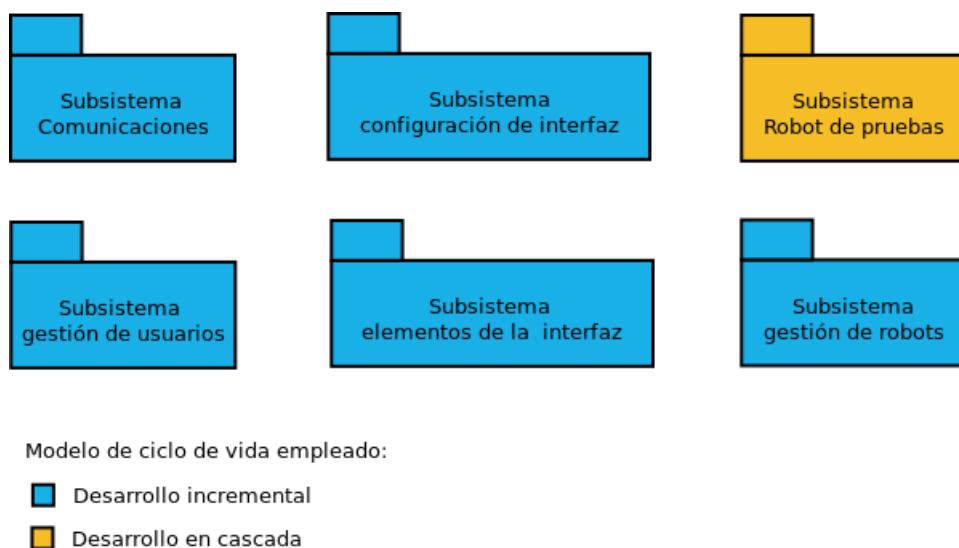


Figura 4.1: Subsistemas existentes en el proyecto junto con el modelo de ciclo de vida utilizado para su desarrollo.

4.2. Recolección de requisitos

4.2.1. Requisitos funcionales

Para la elaboración de este proyecto se ha utilizado la metodología Métrica V3 así como el estándar ISO/IEC 12207. Métrica es una metodología de planificación, desarrollo y mantenimiento de los sistemas de información desarrollada por el Ministerio de Administraciones Públicas del Gobierno de España. Tiene como objetivo proporcionar una guía para la sistematización de actividades del ciclo de vida de los proyectos software

en el ámbito de las administraciones públicas.

Métrica V3 [7] está basada en el modelo de procesos del ciclo de vida de desarrollo ISO/IEC 12207 (Information Technology - Software Life Cycle Processes) así como en la norma ISO/IEC 15504 SPICE (Software Process Improvement And Assurance Standards Capability Determination).

En la página oficial de Métrica V3, sus desarrolladores indican que puede ser utilizada libremente con la única restricción de citar la fuente de su propiedad intelectual, la del Ministerio de Administraciones Públicas.

En esta etapa del modelado de requisitos se captura el propósito general del sistema:

- Se analiza qué debe hacer el sistema.
- Se obtiene una versión contextualizada del sistema.
- Identifica y delimita el sistema.
- Se determinan las características, cualidades y restricciones que debe satisfacer el sistema.

Por tanto, Los requisitos funcionales que se han obtenido después del proceso de obtención de requisitos son:

- Definir los pasos para dar de alta un dispositivo robótico en el sistema.
- Una vez configurado el dispositivo, configurar la interfaz de control con las acciones de control específicas.
- Realizar un sistema de monitorización y visualización para los usuarios y los dispositivos robóticos en tiempo real.
- Sistema de gestión de base de datos en donde se encuentren los datos de la aplicación recogidos.
- Disponer de un panel de administración donde visualizar la información de los usuarios conectados y dispositivos en uso en tiempo real.
- Proporcionar un sistema de streaming de vídeo para la difusión de imágenes a los usuarios espectadores procedentes de los robots dispongan de cámara.
- Deberá ser una herramienta multiplataforma.

4.2.2. Requisitos no funcionales

Los requisitos no funcionales son aquellos que describen cualidades o restricciones del sistema que no se relacionan de forma directa con el comportamiento funcional del mismo. A continuación se especifican los más importantes del sistema:

- No requiere un conocimiento específico del sistema una vez puesto en funcionamiento.
- La aplicación tendrá manual de uso.
- La base de datos estará implementada en un lenguaje objeto no relacional como MongoDB.
- La aplicación estará realizada en el lenguaje de programación Python.
- La interfaz debe reflejar claramente la distinción entre las distintas partes del sistema.
- El sistema se desplegará sobre una versión GNU Linux Debian 8 Jessie.
- El código fuente de la aplicación seguirá un estilo uniforme y normalizado para todos los módulos del mismo.

4.3. Diagramas de casos de uso

Un caso de uso es una descripción de los pasos o actividades que deberán realizarse para llevar a cabo algún proceso. Los objetivos de los casos de uso son los siguientes:

- Obtener los requisitos funcionales del sistema y expresarlos desde un punto de vista más cercano al usuario.
- Proporcionar una guía de todo el proceso de desarrollo del sistema de información. Los casos de uso proporcionan, por tanto, un modo claro y preciso de comunicación entre lo que desea el cliente y el desarrollador proporcionando desde el punto de vista del cliente una visión de “caja negra” del sistema eliminando los detalles de su construcción o desarrollo. Para los desarrolladores es utilizado como punto de partida y el eje sobre el que se apoya todo el desarrollo del sistema en los procesos de análisis y diseño.

Los diagramas de caso de uso muestran una representación del comportamiento ofrecido por el sistema de información desde el punto de vista del usuario. El comportamiento del sistema es representado como un conjunto de transacciones ejecutadas entre el sistema y los actores junto con la descripción sobre las de las relaciones de comunicación existentes entre un actor y el sistema.

En la figura 4.4 podemos observar los diagramas de casos de uso resultantes tras la recolección de requisitos:

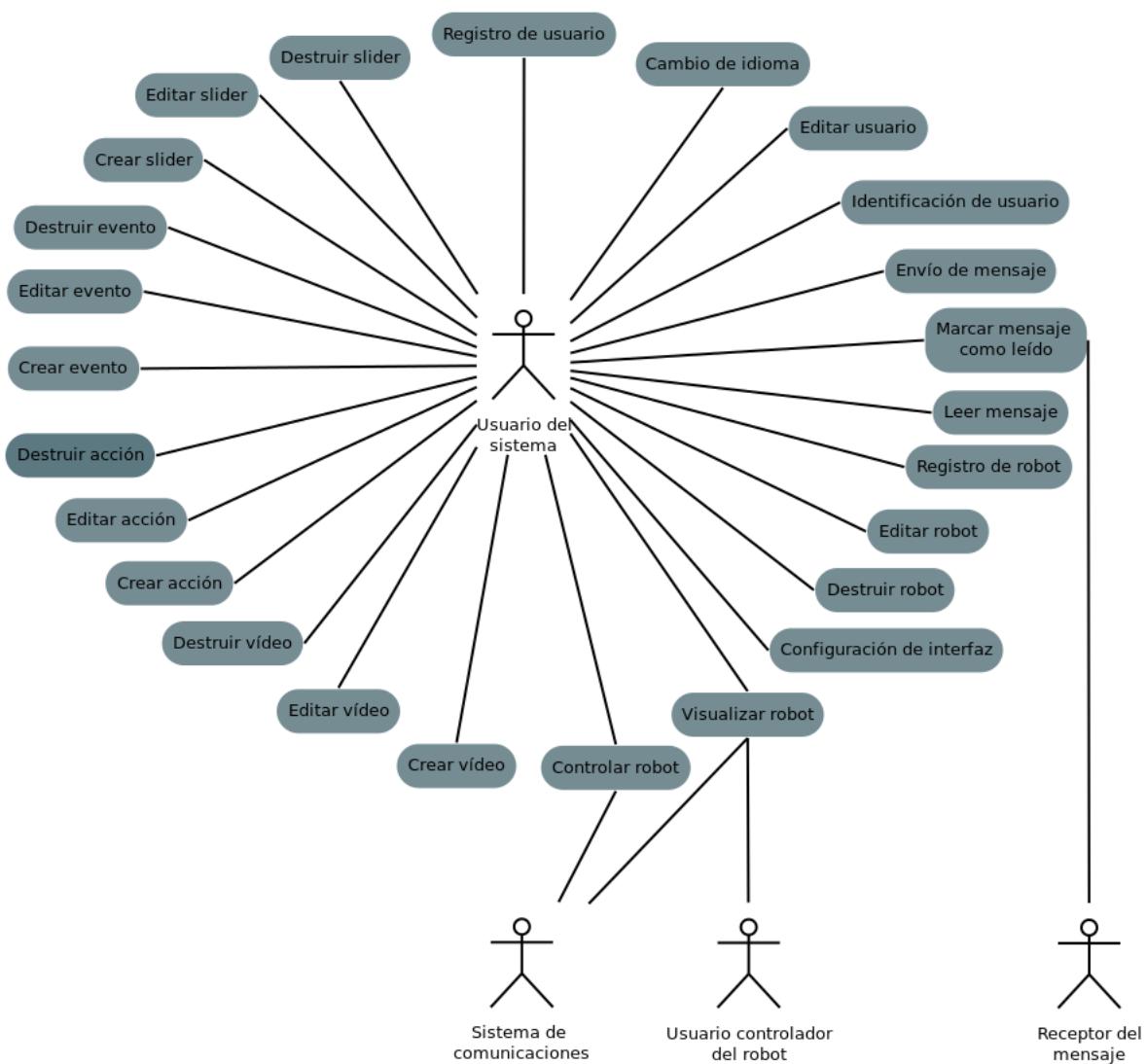


Figura 4.2: Diagrama de casos de uso para la interacción con la aplicación.

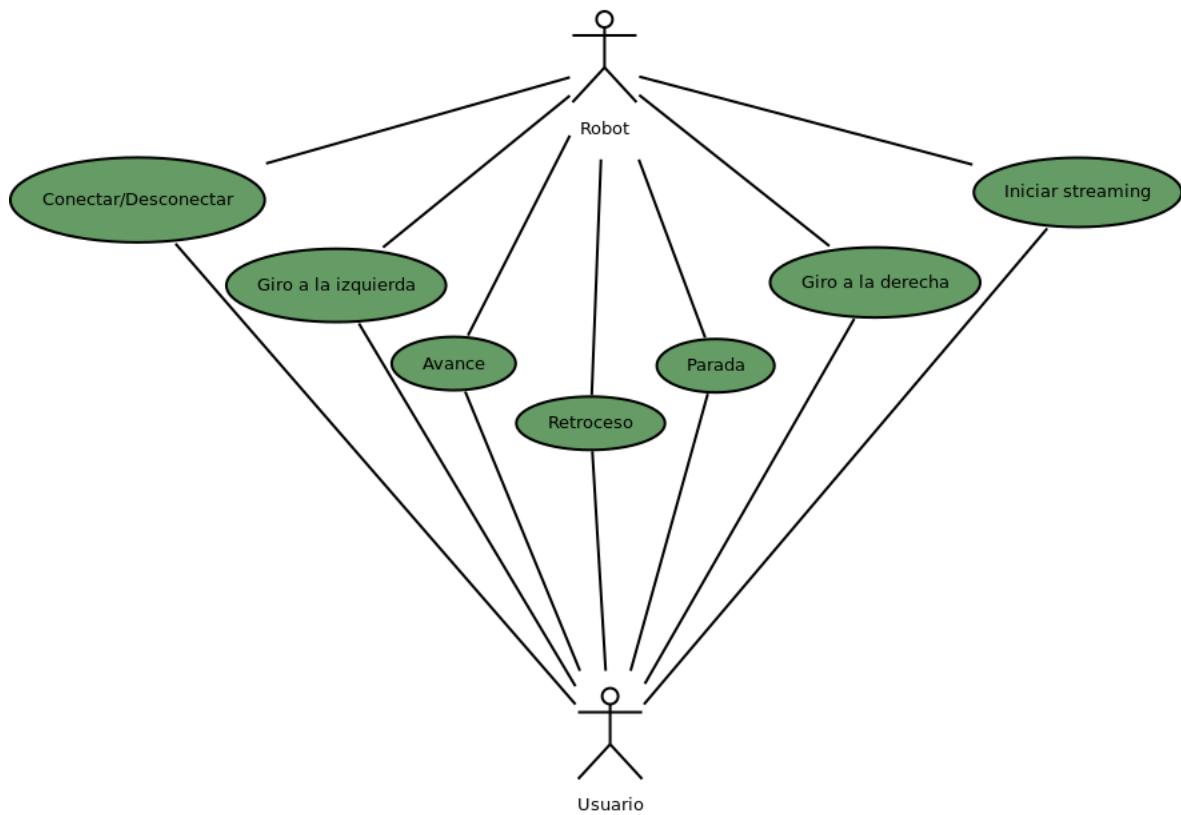


Figura 4.3: Diagrama de casos de uso para la interacción con el robot de pruebas.

4.4. Descripción de los casos de uso

A continuación se proporciona la especificaciones de cada uno de los casos de uso de la figura 4.4:

Caso de uso:	Registro de usuario.
Descripción:	El sistema deberá actuar como describe en este caso de uso cuando el usuario solicita el registro en el sistema.
Actor principal:	Usuario del sistema.
Actor secundario:	-
Precondiciones:	-
Flujo principal:	<ol style="list-style-type: none"> 1. El actor usuario del sistema solicita el registro en el sistema. 2. El sistema solicita los datos correo electrónico, nombre de usuario, avatar, contraseña, confirmación de contraseña e idioma. 3. El usuario proporciona al sistema al menos los siguientes datos, correo electrónico, nombre de usuario, contraseña y confirmación de contraseña. 4. El sistema realiza el registro de un nuevo usuario e informa de que el registro se ha realizado con éxito.
Postcondición	Se realiza el registro de un usuario en el sistema.
Excepciones:	<ol style="list-style-type: none"> 1. Si existe un usuario con el correo electrónico introducido: <ul style="list-style-type: none"> • El sistema informa de que no se puede realizar el registro. • Se cancela el caso de uso. 2. Si la contraseña y la confirmación no coinciden: <ul style="list-style-type: none"> • El sistema informa de que no se puede realizar el registro. • Se cancela el caso de uso.

Tabla 4.1: Descripción del caso de uso: Registro de usuario.

Caso de uso:	Identificación de usuario.
Descripción:	El sistema deberá actuar como describe en este caso de uso cuando el usuario solicita la Identificación en el sistema.
Actor principal:	Usuario del sistema.
Actor secundario:	-
Precondiciones:	El usuario debe estar registrado en el sistema
Flujo principal:	<ol style="list-style-type: none"> 1. El actor usuario del sistema solicita la Identificación en el sistema. 2. El sistema solicita los datos correo electrónico y contraseña. 3. El usuario proporciona al sistema el correo electrónico y contraseña. 4. El sistema realiza el logueo del usuario y crea una sesión.
Postcondición	Se realiza la identificación del usuario en el sistema, se crea una sesión y se carga el idioma establecido por el usuario.
Excepciones:	<ol style="list-style-type: none"> 1. Si no existe un usuario con el correo electrónico introducido: <ul style="list-style-type: none"> • El sistema informa de que no existe ningún usuario con ese correo electrónico. • Se cancela el caso de uso. 2. Si la contraseña es incorrecta: <ul style="list-style-type: none"> • El sistema informa de que la contraseña es incorrecta. • Se cancela el caso de uso.

Tabla 4.2: Descripción del caso de uso: Identificación de usuario.

Caso de uso:	Cambio de idioma.
Descripción:	El sistema deberá actuar como describe en este caso de uso cuando el usuario solicita cambiar de idioma la página.
Actor principal:	Usuario del sistema.
Actor secundario:	-
Precondiciones:	-
Flujo principal:	<ol style="list-style-type: none"> 1. El actor usuario del sistema selecciona un idioma disponible. 2. El sistema establece el idioma seleccionado.
Postcondición	Se realiza el cambio de idioma en la aplicación.
Excepciones:	-

Tabla 4.3: Descripción del caso de uso: Cambio de idioma.

Caso de uso:	Editar usuario.
Descripción:	El sistema deberá actuar como describe en este caso de uso cuando el usuario solicita editar usuario.
Actor principal:	Usuario del sistema.
Actor secundario:	-
Precondiciones:	El usuario debe estar registrado en el sistema
Flujo principal:	<ol style="list-style-type: none"> 1. El actor usuario del sistema selecciona editar su perfil. 2. El sistema solicita los nuevos datos de usuario (correo electrónico, nombre de usuario, avatar, contraseña, confirmación de contraseña e idioma). 3. El usuario modifica al menos uno de los datos anteriores. 4. El sistema realiza la modificación de los datos al usuario.
Postcondición	Se realiza la modificación de los datos al usuario.
Excepciones:	<ol style="list-style-type: none"> 1. Si existe un usuario con el correo electrónico introducido: <ul style="list-style-type: none"> • El sistema informa de que no se puede realizar el registro. • Se cancela el caso de uso. 2. Si la contraseña y la confirmación no coinciden: <ul style="list-style-type: none"> • El sistema informa de que no se puede realizar el registro. • Se cancela el caso de uso.

Tabla 4.4: Descripción del caso de uso: Edición de usuario.

Caso de uso:	Envío de mensaje usuario.
Descripción:	El sistema deberá actuar como describe en este caso de uso cuando el usuario solicita enviar un mensaje.
Actor principal:	Usuario del sistema.
Actor secundario:	Receptor del mensaje
Precondiciones:	El usuario debe estar registrado e identificado en el sistema
Flujo principal:	<ol style="list-style-type: none"> 1. El actor usuario del sistema solicita comenzar el proceso de envío de un mensaje. 2. El sistema solicita los nuevos datos del mensaje (título, destinatario y contenido del mensaje). 3. El usuario proporciona los datos anteriores y ordena mandar el mensaje. 4. El sistema realiza el envío del mensaje al destinatario e informa al receptor de que dispone un nuevo mensaje en su bandeja de entrada.
Postcondición	Se realiza el envío de un mensaje a un usuario.
Excepciones:	<ol style="list-style-type: none"> 1. Si el usuario no se encuentra logueado en el sistema: <ul style="list-style-type: none"> • El sistema informa de que es necesario identificarse. • Se cancela el caso de uso. 2. Si el usuario no ha introducido título, contenido o destinatario: <ul style="list-style-type: none"> • El sistema informa de que no se puede realizar el envío. Faltan datos obligatorios. • Se regresa al punto 2.

Tabla 4.5: Descripción del caso de uso: Envío de mensaje.

Caso de uso:	Leer mensaje.
Descripción:	El sistema deberá actuar como describe en este caso de uso cuando el usuario solicita leer un mensaje.
Actor principal:	Usuario del sistema.
Actor secundario:	-
Precondiciones:	El usuario debe estar registrado, logueado y disponer de al menos un mensaje en su bandeja de entrada.
Flujo principal:	<ol style="list-style-type: none"> 1. El actor usuario del sistema solicita leer un mensaje de su bandeja de entrada. 2. El sistema muestra contenido del mensaje y lo establece como leído.
Postcondición	Se muestra el contenido del mensaje.
Excepciones:	<ol style="list-style-type: none"> 1. Si el usuario no se encuentra logueado en el sistema: <ul style="list-style-type: none"> • El sistema informa de que es necesario identificarse. • Se cancela el caso de uso.

Tabla 4.6: Descripción del caso de uso: Leer mensaje.

Caso de uso:	Registro de un robot.
Descripción:	El sistema deberá actuar como describe en este caso de uso cuando el usuario solicita el registro de un nuevo robot en el sistema.
Actor principal:	Usuario del sistema.
Actor secundario:	-
Precondiciones:	Usuario registrado e identificado en el sistema.
Flujo principal:	<ol style="list-style-type: none"> 1. El actor usuario del sistema solicita el registro de un robot en el sistema. 2. El sistema solicita los datos del robot, nombre, dirección IP, puerto, descripción, imagen, usuarios espectadores, usuarios controladores y si es de control abierto o visualización abierta . 3. El usuario proporciona al sistema al menos los siguientes datos, nombre, dirección Ip, puerto, y permisos. 4. El sistema realiza el registro de un nuevo robot e informa de que la creación se ha realizado con éxito.
Postcondición	Se realiza el registro de un robot en el sistema ligado al usuario que lo ha creado.
Excepciones:	<ol style="list-style-type: none"> 1. Si el usuario no se encuentra logueado en el sistema: <ul style="list-style-type: none"> • El sistema informa de que es necesario identificarse. • Se cancela el caso de uso. 2. Si faltan datos obligatorios: <ul style="list-style-type: none"> • El sistema informa de que no se puede realizar el registro. • Vuelve al paso 2. 3. Si la dirección IP es inválida: <ul style="list-style-type: none"> • El sistema informa de que no se puede realizar el registro. • Vuelve al paso 2. 4. Si el puerto no es un valor numérico: <ul style="list-style-type: none"> • El sistema informa de que no se puede realizar el registro. • Vuelve al paso 2.

Caso de uso:	Editar robot.
Descripción:	El sistema deberá actuar como describe en este caso de uso cuando el usuario solicita la edición de un robot en el sistema.
Actor principal:	Usuario del sistema.
Actor secundario:	-
Precondiciones:	Usuario registrado e identificado en el sistema y propietario del robot.
Flujo principal:	<ol style="list-style-type: none"> 1. El actor usuario del sistema solicita la edición de un robot en el sistema. 2. El sistema solicita los datos del robot, nombre, dirección IP, puerto, descripción, imagen, usuarios espectadores, usuarios controladores y si es de control abierto o visualización abierta . 3. El usuario proporciona al sistema al menos los siguientes datos, nombre, dirección Ip, puerto, y permisos. 4. El sistema realiza la modificación de los datos del robot e informa de que la edición se ha realizado con éxito.
Postcondición	Se realiza la edición de un robot en el sistema.
Excepciones:	<ol style="list-style-type: none"> 1. Si el usuario no se encuentra logueado en el sistema: <ul style="list-style-type: none"> • El sistema informa de que es necesario identificarse. • Se cancela el caso de uso. 2. Si el robot no pertenece al usuario: <ul style="list-style-type: none"> • El sistema informa de que no tiene permisos. • Se cancela el caso de uso. 3. Si faltan datos obligatorios: <ul style="list-style-type: none"> • El sistema informa de que no se puede realizar la edición. • Vuelve al paso 2. 4. Si la dirección IP es inválida: <ul style="list-style-type: none"> • El sistema informa de que no se puede realizar el registro. 40 • Vuelve al paso 2. 5. Si el robot ya no existe: <ul style="list-style-type: none"> • El sistema informa de que no se puede realizar la edición. • Vuelve al paso 2.

4.5. Diagrama de clases

En esta sección de la arquitectura es dónde se define la parte de la interacción con la base de datos y que tablas se han definido y con que relaciones.

La clase o tabla que contendrá el peso de toda la jerarquía de base de datos será la clase Robot. Esta tabla contendrá referencias externas o “foreign keys” a cada tabla que haga participe en su definición, es decir:

- Interfaz
- Usuario propietario
- Usuarios conductores
- Usuarios espectadores
- Room

Otra de las clases con mayor presencia en el modelo es la clase Interfaz, la cual está directamente relacionada con el robot y que incorpora las siguientes relaciones:

- Acción
- Vídeo
- Evento
- Slider

Por último destacar la clase Usuario que incorpora las siguientes relaciones:

- Robot en propiedades
- Robot con permisos de conducción
- Robots con permisos de visualización
- Mensajes enviados
- Mensajes recibidos
- Sesiones abiertas
- Iconos

4.5. DIAGRAMA DE CLASES

CAPÍTULO 4. DESARROLLO

La jerarquía de clases queda de la siguiente manera:

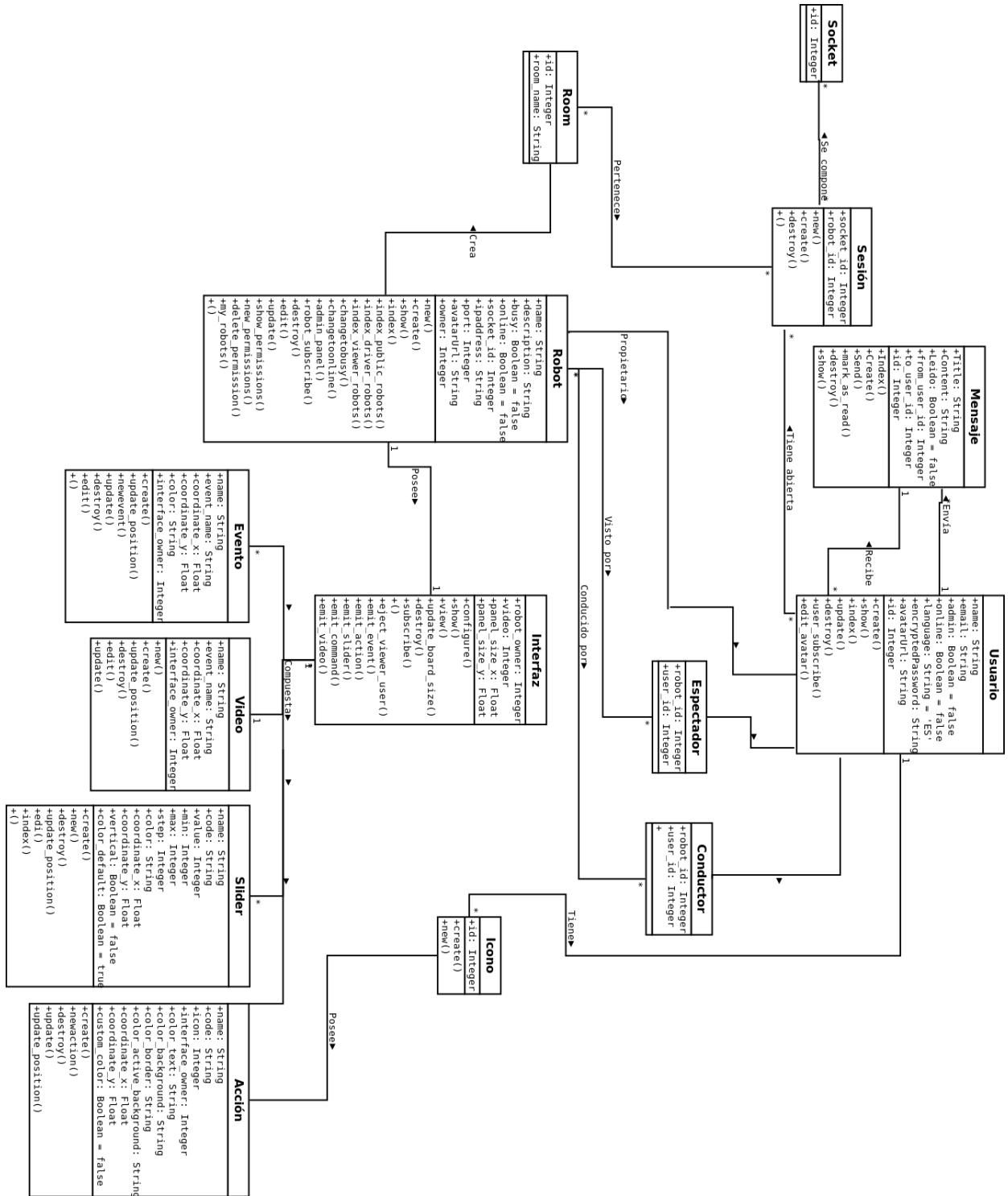


Figura 4.4: Diagrama de clases para la BD.

Capítulo 5

Desarrollo de la interfaz

5.1. Elementos de la interfaz

Para la elaboración de parte referente la capa de presentación¹ de la aplicación se ha empleado Bootstrap, un potente framework CSS que fue creado por Twitter para simplificar el proceso de maquetación. Este framework CSS presenta multitud de ventajas:

- Permite crear interfaces adaptables a los diferentes navegadores, tanto de escritorio como tablets o móviles con distintas escalas y resoluciones.
- Se integra perfectamente con las principales librerías Javascript, entre ellas JQuery.
- Ofrece un diseño sólido usando LESS y estándares como CSS3/HTML5.
- Es un framework ligero y fácilmente integrable en la mayoría de proyectos.
- Funciona con los navegadores más populares.
- Dispone de distintos layouts predefinidos con estructuras fijas a 940 píxeles de distintas columnas o diseños fluidos.

Por otra parte, dada la cantidad de efectos dinámicos necesarios en la aplicación junto con múltiples llamadas Ajax, se ha utilizando jQuery que es un framework Javascript que facilita toda esta labor haciéndola compatible con los navegadores más comúnmente utilizados.

Se consideró antes de comenzar a definir los diferentes estilos y características desarrollo de la interfaz gráfica, una serie de elementos mínimos imprescindibles que dicha interfaz debería incorporar.

¹ Terminología también conocida como Front-End. Son todas aquellas tecnologías que corren del lado del cliente, es decir, todas aquellas tecnologías que corren del lado del navegador web, generalizándose más que nada en tres lenguajes, Html , CSS Y JavaScript, normalmente el FrontEnd se encarga de estilizar la página de tal manera que la página pueda quedar cómoda para la persona que la utiliza transmitiendo una experiencia de usuario cómoda. En contraposición con el back-end que engloba el motor de la aplicación.

Dada las características del proyecto, era necesario proporcionar a la aplicación de un entorno gráfico sencillo, intuitivo pero a la vez funcional teniendo como principal objetivo ofrecer al usuario, tras una visión rápida, la localización de los diferentes elementos para control del robot y deducir el funcionamiento general de la aplicación.

Las vistas principales desarrolladas en la aplicación son los siguientes:

- Vista principal de la aplicación.
- Panel de administración para el visionado de los usuarios conectados en tiempo real.
- Panel de administración para el visionado de los robots conectados en tiempo real.
- Formularios de registro de usuarios y robots.
- Panel de creación y edición de interfaces de control.
- Panel de control de robots.
- Panel de visualización de robots.
- Índice de robots disponibles.

El desarrollo de todos los componentes anteriormente mencionados se han ido realizando paralelamente al desarrollo de la parte Backend² correspondiente.

²El BackEnd, en contraposición al FrontEnd es el área dedicada a la parte lógica de un sitio web, el BackEnd engloba toda la parte no visible para el usuario, excluyendo toda la parte de diseño o de elementos gráficos.

Capítulo 6

Comunicaciones

En el presente capítulo comenzaremos con una introducción sobre el funcionamiento y los fundamentos teóricos sobre cómo se gestionan las diferentes conexiones y eventos de cualquier aplicación Sails para, posteriormente, centrarnos específicamente en la descripción de un caso práctico desarrollado en el presente proyecto con la finalidad de comprender mejor su funcionamiento y poder afianzar conocimientos.

6.1. Introducción

En esta sección comenzaremos destacando aquellos aspectos teóricos sobre cómo Sails, framework Node JS utilizado en el desarrollo, trabaja con Websocket y Socket.io.

Como sabemos, un servidor HTTP no puede enviar datos a menos que un cliente los haya solicitado mediante una petición. Los Websockets, en cambio, presentan la particularidad de que permiten que un servidor envíe datos a un cliente sin la necesidad de que éstos sean solicitados, al menos de una manera inmediata. Estas solicitudes, realizadas con anterioridad, se formalizan mediante suscripciones, concepto que iremos constantemente haciendo referencia a lo largo del presente capítulo y que debemos recordar.

Anteriormente comentamos que el servidor HTTP no puede enviar datos a menos que el cliente lo haya solicitado y los Websockets permiten que un servidor envíe datos no solicitados una vez que se hace una conexión inicial (suscripción) desde el lado del

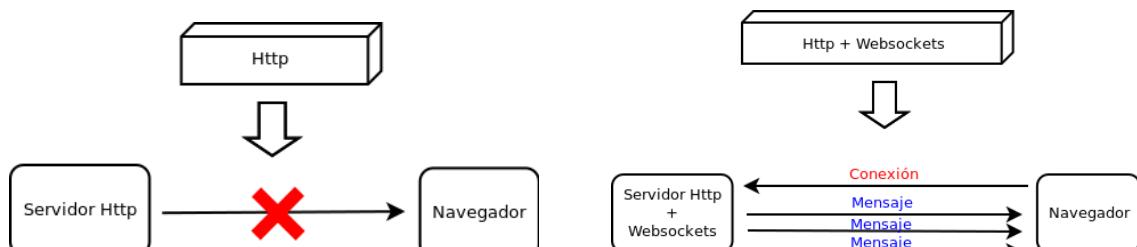


Figura 6.1: Peticiones entre un navegador y un servidor http con y sin el empleo de websockets.

cliente. Una aplicación Sails típica queda dividida en dos partes. La primera, en el lado del servidor, consta de un servidor HTTP junto con un nodo central que actúa como un servidor de sockets.

En segundo lugar, en el lado del cliente, se dispone de los diferentes códigos HTML y funciones JavaScript para realizar las conexiones con el servidor. Estas conexiones cliente-servidor permiten que cualquier otro cliente conectado pueda enviar mensajes al servidor para que a su vez éstos sean emitidos a los clientes que se encuentren conectados en la aplicación en ese instante.

Además cada socket posee un identificador único que identifica de manera inequívoca el cliente o, en nuestro caso el navegador que está accediendo a la página.

Destacar también el concepto de salas o rooms de Socket.io. Las salas nos permite agrupar los sockets de modo que en lugar de mensajes que son enviados a todos los sockets conectados, se puede enviar mensajes sólo a los sockets que están asociados con una sala. De esta podremos mandar mensajes específicos para un cliente concreto, todos ellos o un conjunto de los mismos.

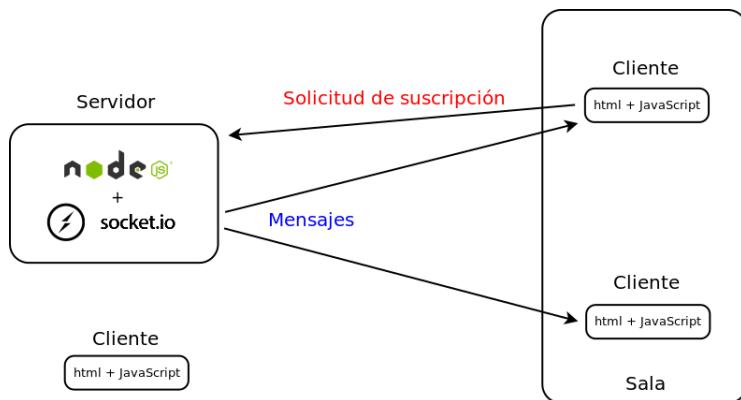


Figura 6.2: Representación de una sala compuesta por dos clientes.

De todo lo anterior podemos deducir que Sails es un framework especialmente potente, el cual proporciona infinidad de posibilidades a desarrollar. A continuación se cita un fragmento tal y como podemos extraer de la página oficial de Sails:

Sails.js facilita el desarrollo de aplicaciones Node.js empresariales. Ha sido diseñado para imitar el patrón MVC de frameworks como Ruby on Rails, pero con soporte para los requisitos de aplicaciones modernas: data-driven APIs con una arquitectura escalable y service-oriented. Es especialmente bueno para el desarrollo de chats, cuadros de mando en tiempo real o juegos multijugadores.

En este caso menciona diferentes aplicaciones tipo que pueden desarrollarse con Sails, desde un chat, cuadros de mando en tiempo real o juegos multijugadores. RobotUI

resulta como una combinación de las anteriores puesto que incorpora cuadros de mandos, sistema de mensajería y también es considerado como un juego ya que también está enfocado al entretenimiento.

6.2. Conexión y suscripción

Tras la comprensión de los fundamentos descritos en el punto 6.1 donde se recogen los principios de comunicación con websockets pasamos a trasladarlos a un caso práctico describiendo una de las funcionalidades del proyecto donde éstos son aplicados de tal manera que se facilite su comprensión.

Como sabemos, Sails nos ayuda a incorporar funcionalidades en tiempo real a nuestra aplicación web gestionando una serie de eventos aplicándolos sobre los modelos. Para el caso a describir nos centraremos en el modelo *Usuario* de RobotUI. Uno de los atributos de este modelo es un booleano *online*, el cual representa si un usuario se encuentra logueado en el sistema o no. El objetivo es saber cuando el usuario cambia el estado para poder proporcionar una actualización en tiempo real de la página de gestión de usuarios 6.3 de manera que cuando usuario inicie sesión o salga de la aplicación se alterne entre las imágenes online: y offline: sin necesidad de refrescar la página manualmente.

The screenshot shows the 'Usuarios' (Users) section of the RobotUI administration interface. The page has a dark header with the ROBOTUI logo, navigation links for 'Administración', 'Mensajes', 'Robots', and language selection ('ES'). Below the header is a light-colored card titled 'Usuarios'. The card contains a table with columns: 'Online', 'Nombre', 'Email', 'Role', and 'Acciones'. There are four rows of data:

Online	Nombre	Email	Role	Acciones
	User4	user4@user.com		[Añadir robot] [Mostar robots] [Editar] [Eliminar]
	User3	user3@user.com		[Añadir robot] [Mostar robots] [Editar] [Eliminar]
	User2	user2@user.com		[Añadir robot] [Mostar robots] [Editar] [Eliminar]
	Manuel	user1@user.com		[Añadir robot] [Mostar robots] [Editar] [Eliminar]

At the bottom of the card, there is a note: 'Autor: Manuel López Urbina' followed by social media icons for Facebook, Twitter, Google+, and Email. A link 'Acerca de RobotUI' is also present.

Figura 6.3: Página de gestión de usuarios actualizable en tiempo real.

Para capturar los eventos correspondientes al modelo *Usuario* o cualquier otro debemos realizar una suscripción. Sails incorpora dos modalidades de suscripción. suscripción a clase y suscripción a modelo.

¿Cuál es la diferencia entre suscribirse a una “clase” frente a la suscripción a una “instancia” de un modelo? Al suscribirse a la “clase” del modelo, el socket podrá escuchar la creación de nuevas instancias de modelo mediante un método denominado publishCreate(). Mientras que la suscripción a “instancia” permite al socket escuchar los cambios de modelos a través de los métodos publishUpdate y/o publishDestroy de una instancia o conjunto de instancias en concreto.

Por tanto, centrándonos en el ejemplo de los usuarios, debemos realizar la suscripción mediante la definición de una acción en el controlador de usuario. La acción se denominará *user_subscribe*.

En dicha acción podremos acceder al socket solicitante vía req.socket. Así que, primeiramente se realiza una suscripción a la sala de “clase” del usuario mediante User.watch pasando req.socket como argumento. De esta manera nos hemos suscrito a la clase del modelo de usuario.

Seguidamente se realiza la suscripción a las instancias del modelo de usuario mediante el método User.subscribe pasando como parámetro el socket solicitante obtenido a través de req.socket seguido de un segundo argumento, que se corresponde con las instancias existentes de usuarios. Las instancias de los usuarios existentes son recuperadas mediante un método de búsqueda para que, posteriormente, su salida sea pasada al método de suscripción como segundo argumento. Con esto ya nos hemos suscrito a las salas de instancias del modelo de usuario.

A continuación mostramos el código que realiza la suscripción al modelo Usuario en sus dos modalidades, suscripción a clase de modelo y suscripción a instancias de modelo descritas anteriormente:

```

1 user_subscribe: function (req, res, next) {
2   if (req.isSocket) {
3     //Update, destroy...
4     User.find(function foundUsers(err, users) {
5       if (err) return res.badRequest(err);
6
7       User.subscribe(req.socket, users);
8     });
9
10
11    //Create
12    User.watch(req);
13    console.log('User ' + req.session.User.id + 'with socket id ' +
14      sails.sockets.id(req) + ' is now subscribed to the model
15      class \'User\'.' );
16  } else {
17    res.view();
18 }
```

```
16    }
17 }
```

Ahora bien, ya nos encontramos suscritos a un modelo, pero... ¿Qué ocurre con publishCreate, publishUpdate y publishDestroy?, ¿Dónde se realiza la llamada a estos métodos?

Para ello solamente debemos llamar al método en la acción deseada, por ejemplo, al crear una sesión, tan solo debemos actualizar el usuario como *online* y mandar el mensaje correspondiente mediante el método publishUpdate:

```
1
2     //Cambio de estado a online
3     User.update(user.id, {online: true, longitude: long, latitude:
4         lat}, function (err){
5             if (err) return next(err);
6
7             //Informar a otros clientes (sockets abiertos) que el
8             //usuario esta logueado
9             User.publishUpdate(user.id, {
10                 loggedIn: true,
11                 id: user.id
12             });
13         });
14     });

15 }
```

Tras ello se notificará a todos los clientes que han sido suscritos a las instancias del modelo Usuario de que un determinado usuario ha cambiado su estado.

Del mismo modo para *publishCreate* o *publishDestroy*.

¿Y desde el lado del cliente? ¿Cómo realizamos las llamadas a las diferentes acciones de suscripción y cómo capturamos los eventos?

Para la realización de las suscripciones y captura de eventos las siguientes funciones ejecutables en el cliente resultan de vital importancia.

Primeramente tenemos la función *savesocket* la cual realiza la llamada al método *saveSocketID* del controlador *Session*. Dicho método crea un registro en base de datos ligando el identificador del socket abierto con el usuario que se encuentra identificado en el sistema, de tal manera que siempre sabremos qué socket corresponde con qué usuario. Este almacenamiento se debe a que al realizarse una nueva conexión de un usuario en el sistema el servidor debe conocer y registrar a dicho cliente con la finalidad de seguir su comportamiento, sus acciones y movimientos por toda la aplicación.

Dicho código JavaScript se encuentra localizado en el layout principal de la aplicación de tal manera que independientemente de qué vista resulte cargada siempre se ejecutará el siguiente fragmento de código realizando el almacenamiento del socket abierto por el usuario:

```

1
2
3<script type="text/javascript">
4
5 $.when(savesocket()).done(function(){
6     if (typeof subscribeAndListen == 'function') {
7         subscribeAndListen();
8     }
9     listenMessages();
10 });
11
12 //Funcion para tener en todo momento almacenado en la tabla session
13 // los sockets conectados junto con el usuario al que pertenece
14 function savesocket() {
15     io.socket.get("/session/saveSocketID");
16 }
</script>
```

Función `emphsaveSocketID`, la cual es llamada en cliente y que se encuentra implementada en el controlador `session` en el lado del servidor. Esta función realiza el registro de un nuevo cliente en la base de datos tras recibir el identificador correspondiente al nuevo socket creado.

El código inferior muestra la implementación en el lado del servidor de la función `saveSocketID`:

```

1
2
3//Almacenamiento en la base de datos la sesion de cada usuario de la
4//pagina
5saveSocketID: function(req, res) {
6    if (!req.isSocket) return res.badRequest();
7
8    var socketId = sails.sockets.id(req);
9    // => "BetX2G-2889Bg22xi-jy"
10
11    var sessionObj = {
12        socket_id: socketId,
13        user_id: req.session.User != undefined ? req.session.User.id :
14            null
15    };
16
17    Session.create(sessionObj).exec( function (err, session) {
18        if (err) return res.badRequest();
19
20        console.log('\n.....');
21        console.log('Conectando a Sails js... ');
22        console.log('Cliente conectado - id del socket: ' + socketId);
23        console.log('.....');
24
25        return;
26    });
27}
```

Una vez tenemos almacenado el par socket-usuario en base de datos buscamos la función `subscribeAndListen`, la cual será específica según la vista en la que nos encontremos, por ejemplo, si nos encontramos en el panel de administración de usuarios, la función `subscribeAndListen` será específica para cada funcionalidad que queramos inicializar o a qué eventos nos deseemos suscribir. Si nos encontramos en la vista de índice de robots, la función se suscribirá a los eventos de conexión y desconexión de robots. O si estamos visualizando el funcionamiento de un robot entonces la función capturará los eventos que se correspondan con la captación de los diferentes comandos o datos transmitidos.

En el código inferior mostramos la función `subscribeAndListen` que implementa la suscripción a los eventos correspondiente a la conexión, desconexión y borrado de usuarios del panel de administración descrito en el apartado 6.1. Esta función está localizada en el *partial*¹ que implementa la vista correspondiente con el citado panel:

```

1
2
3 <script type="text/javascript">
4
5     function subscribeAndListen() {
6         io.socket.get('/user/user_subscribe');
7
8         io.socket.on('user', function (obj) {
9             if (obj.verb == 'created') {
10                 var user = obj.data;
11                 $.ajax({
12                     url: '/user/render',
13                     type: 'GET',
14                     data: {id: user.id},
15                     success: function(data){
16                         $("#user_list").append(data);
17                         toastr.info('<%= i18n("user_created") %>', 'RobotUI')
18                     }
19                 });
20             }
21
22             if (obj.verb == 'updated') {
23                 var data = obj.data;
24                 change_img_state(data.id, data.loggedIn, '<=%
25                             i18n("connect")%>', '<=%
26                             i18n("disconnect")%>');
27
28                 if (data.id != '<%= req.session.User.id %>') {
29                     if (data.loggedIn) {
30                         toastr.info('<%= i18n("user_connected") %>', 'RobotUI');
31                     } else {
32                         toastr.info('<%= i18n("user_disconnected") %>', 'RobotUI');
33                     }
34                     console.log('User has been updated to online:' +
35                         data.loggedIn);
36                 }
37             }
38         });
39     }
40
41     $(document).ready(function () {
42         subscribeAndListen();
43     });
44
45     $(window).on('beforeunload', function () {
46         io.socket.emit('user', {verb: 'destroy'});
47     });
48
49     $(document).on('click', '#user_list tr td .destroy', function () {
50         var id = $(this).attr('data-id');
51         var $tr = $(this).closest('tr');
52
53         $.ajax({
54             url: '/user/delete/' + id,
55             type: 'DELETE',
56             success: function (data) {
57                 $tr.remove();
58             }
59         });
60     });
61
62     $(document).on('click', '#user_list tr td .edit', function () {
63         var id = $(this).attr('data-id');
64
65         $.ajax({
66             url: '/user/edit/' + id,
67             type: 'GET',
68             success: function (data) {
69                 $('#user_form').html(data);
70                 $('#user_form').modal('show');
71             }
72         });
73     });
74
75     $(document).on('submit', '#user_form', function (e) {
76         e.preventDefault();
77
78         var $form = $(this);
79         var $table = $form.closest('table');
80
81         $.ajax({
82             url: $form.attr('action'),
83             type: $form.attr('method'),
84             data: $form.serialize(),
85             success: function (data) {
86                 $table.append(data);
87                 $('#user_form').modal('hide');
88             }
89         });
90     });
91
92     $(document).on('click', '#user_list tr td .view', function () {
93         var id = $(this).attr('data-id');
94
95         $.ajax({
96             url: '/user/view/' + id,
97             type: 'GET',
98             success: function (data) {
99                 $('#user_view').html(data);
100                $('#user_view').modal('show');
101            }
102        });
103    });
104
105    $(document).on('click', '#user_view .close', function () {
106        $('#user_view').modal('hide');
107    });
108
109    $(document).on('click', '#user_view .update', function () {
110        var id = $(this).attr('data-id');
111
112        $.ajax({
113            url: '/user/edit/' + id,
114            type: 'PUT',
115            data: $(this).closest('form').serialize(),
116            success: function (data) {
117                $('#user_view').modal('hide');
118                $('#user_list').append(data);
119            }
120        });
121    });
122
123    $(document).on('click', '#user_view .destroy', function () {
124        var id = $(this).attr('data-id');
125
126        $.ajax({
127            url: '/user/delete/' + id,
128            type: 'DELETE',
129            success: function (data) {
130                $('#user_view').modal('hide');
131                $('#user_list').append(data);
132            }
133        });
134    });
135
136    $(document).on('click', '#user_list tr td .refresh', function () {
137        var id = $(this).attr('data-id');
138
139        $.ajax({
140            url: '/user/render/' + id,
141            type: 'GET',
142            success: function (data) {
143                $('#user_list').append(data);
144            }
145        });
146    });
147
148    $(document).on('click', '#user_list tr td .log', function () {
149        var id = $(this).attr('data-id');
150
151        $.ajax({
152            url: '/user/log/' + id,
153            type: 'GET',
154            success: function (data) {
155                $('#user_log').html(data);
156                $('#user_log').modal('show');
157            }
158        });
159    });
160
161    $(document).on('click', '#user_log .close', function () {
162        $('#user_log').modal('hide');
163    });
164
165    $(document).on('click', '#user_log .refresh', function () {
166        var id = $(this).attr('data-id');
167
168        $.ajax({
169            url: '/user/render/' + id,
170            type: 'GET',
171            success: function (data) {
172                $('#user_log').html(data);
173            }
174        });
175    });
176
177    $(document).on('click', '#user_log .destroy', function () {
178        var id = $(this).attr('data-id');
179
180        $.ajax({
181            url: '/user/delete/' + id,
182            type: 'DELETE',
183            success: function (data) {
184                $('#user_log').modal('hide');
185                $('#user_log').html(data);
186            }
187        });
188    });
189
190    $(document).on('click', '#user_log .refresh', function () {
191        var id = $(this).attr('data-id');
192
193        $.ajax({
194            url: '/user/render/' + id,
195            type: 'GET',
196            success: function (data) {
197                $('#user_log').html(data);
198            }
199        });
200    });
201
202    $(document).on('click', '#user_log .destroy', function () {
203        var id = $(this).attr('data-id');
204
205        $.ajax({
206            url: '/user/delete/' + id,
207            type: 'DELETE',
208            success: function (data) {
209                $('#user_log').modal('hide');
210                $('#user_log').html(data);
211            }
212        });
213    });
214
215    $(document).on('click', '#user_log .refresh', function () {
216        var id = $(this).attr('data-id');
217
218        $.ajax({
219            url: '/user/render/' + id,
220            type: 'GET',
221            success: function (data) {
222                $('#user_log').html(data);
223            }
224        });
225    });
226
227    $(document).on('click', '#user_log .destroy', function () {
228        var id = $(this).attr('data-id');
229
230        $.ajax({
231            url: '/user/delete/' + id,
232            type: 'DELETE',
233            success: function (data) {
234                $('#user_log').modal('hide');
235                $('#user_log').html(data);
236            }
237        });
238    });
239
240    $(document).on('click', '#user_log .refresh', function () {
241        var id = $(this).attr('data-id');
242
243        $.ajax({
244            url: '/user/render/' + id,
245            type: 'GET',
246            success: function (data) {
247                $('#user_log').html(data);
248            }
249        });
250    });
251
252    $(document).on('click', '#user_log .destroy', function () {
253        var id = $(this).attr('data-id');
254
255        $.ajax({
256            url: '/user/delete/' + id,
257            type: 'DELETE',
258            success: function (data) {
259                $('#user_log').modal('hide');
260                $('#user_log').html(data);
261            }
262        });
263    });
264
265    $(document).on('click', '#user_log .refresh', function () {
266        var id = $(this).attr('data-id');
267
268        $.ajax({
269            url: '/user/render/' + id,
270            type: 'GET',
271            success: function (data) {
272                $('#user_log').html(data);
273            }
274        });
275    });
276
277    $(document).on('click', '#user_log .destroy', function () {
278        var id = $(this).attr('data-id');
279
280        $.ajax({
281            url: '/user/delete/' + id,
282            type: 'DELETE',
283            success: function (data) {
284                $('#user_log').modal('hide');
285                $('#user_log').html(data);
286            }
287        });
288    });
289
290    $(document).on('click', '#user_log .refresh', function () {
291        var id = $(this).attr('data-id');
292
293        $.ajax({
294            url: '/user/render/' + id,
295            type: 'GET',
296            success: function (data) {
297                $('#user_log').html(data);
298            }
299        });
300    });
301
302    $(document).on('click', '#user_log .destroy', function () {
303        var id = $(this).attr('data-id');
304
305        $.ajax({
306            url: '/user/delete/' + id,
307            type: 'DELETE',
308            success: function (data) {
309                $('#user_log').modal('hide');
310                $('#user_log').html(data);
311            }
312        });
313    });
314
315    $(document).on('click', '#user_log .refresh', function () {
316        var id = $(this).attr('data-id');
317
318        $.ajax({
319            url: '/user/render/' + id,
320            type: 'GET',
321            success: function (data) {
322                $('#user_log').html(data);
323            }
324        });
325    });
326
327    $(document).on('click', '#user_log .destroy', function () {
328        var id = $(this).attr('data-id');
329
330        $.ajax({
331            url: '/user/delete/' + id,
332            type: 'DELETE',
333            success: function (data) {
334                $('#user_log').modal('hide');
335                $('#user_log').html(data);
336            }
337        });
338    });
339
340    $(document).on('click', '#user_log .refresh', function () {
341        var id = $(this).attr('data-id');
342
343        $.ajax({
344            url: '/user/render/' + id,
345            type: 'GET',
346            success: function (data) {
347                $('#user_log').html(data);
348            }
349        });
350    });
351
352    $(document).on('click', '#user_log .destroy', function () {
353        var id = $(this).attr('data-id');
354
355        $.ajax({
356            url: '/user/delete/' + id,
357            type: 'DELETE',
358            success: function (data) {
359                $('#user_log').modal('hide');
360                $('#user_log').html(data);
361            }
362        });
363    });
364
365    $(document).on('click', '#user_log .refresh', function () {
366        var id = $(this).attr('data-id');
367
368        $.ajax({
369            url: '/user/render/' + id,
370            type: 'GET',
371            success: function (data) {
372                $('#user_log').html(data);
373            }
374        });
375    });
376
377    $(document).on('click', '#user_log .destroy', function () {
378        var id = $(this).attr('data-id');
379
380        $.ajax({
381            url: '/user/delete/' + id,
382            type: 'DELETE',
383            success: function (data) {
384                $('#user_log').modal('hide');
385                $('#user_log').html(data);
386            }
387        });
388    });
389
390    $(document).on('click', '#user_log .refresh', function () {
391        var id = $(this).attr('data-id');
392
393        $.ajax({
394            url: '/user/render/' + id,
395            type: 'GET',
396            success: function (data) {
397                $('#user_log').html(data);
398            }
399        });
400    });
401
402    $(document).on('click', '#user_log .destroy', function () {
403        var id = $(this).attr('data-id');
404
405        $.ajax({
406            url: '/user/delete/' + id,
407            type: 'DELETE',
408            success: function (data) {
409                $('#user_log').modal('hide');
410                $('#user_log').html(data);
411            }
412        });
413    });
414
415    $(document).on('click', '#user_log .refresh', function () {
416        var id = $(this).attr('data-id');
417
418        $.ajax({
419            url: '/user/render/' + id,
420            type: 'GET',
421            success: function (data) {
422                $('#user_log').html(data);
423            }
424        });
425    });
426
427    $(document).on('click', '#user_log .destroy', function () {
428        var id = $(this).attr('data-id');
429
430        $.ajax({
431            url: '/user/delete/' + id,
432            type: 'DELETE',
433            success: function (data) {
434                $('#user_log').modal('hide');
435                $('#user_log').html(data);
436            }
437        });
438    });
439
440    $(document).on('click', '#user_log .refresh', function () {
441        var id = $(this).attr('data-id');
442
443        $.ajax({
444            url: '/user/render/' + id,
445            type: 'GET',
446            success: function (data) {
447                $('#user_log').html(data);
448            }
449        });
450    });
451
452    $(document).on('click', '#user_log .destroy', function () {
453        var id = $(this).attr('data-id');
454
455        $.ajax({
456            url: '/user/delete/' + id,
457            type: 'DELETE',
458            success: function (data) {
459                $('#user_log').modal('hide');
460                $('#user_log').html(data);
461            }
462        });
463    });
464
465    $(document).on('click', '#user_log .refresh', function () {
466        var id = $(this).attr('data-id');
467
468        $.ajax({
469            url: '/user/render/' + id,
470            type: 'GET',
471            success: function (data) {
472                $('#user_log').html(data);
473            }
474        });
475    });
476
477    $(document).on('click', '#user_log .destroy', function () {
478        var id = $(this).attr('data-id');
479
480        $.ajax({
481            url: '/user/delete/' + id,
482            type: 'DELETE',
483            success: function (data) {
484                $('#user_log').modal('hide');
485                $('#user_log').html(data);
486            }
487        });
488    });
489
490    $(document).on('click', '#user_log .refresh', function () {
491        var id = $(this).attr('data-id');
492
493        $.ajax({
494            url: '/user/render/' + id,
495            type: 'GET',
496            success: function (data) {
497                $('#user_log').html(data);
498            }
499        });
500    });
501
502    $(document).on('click', '#user_log .destroy', function () {
503        var id = $(this).attr('data-id');
504
505        $.ajax({
506            url: '/user/delete/' + id,
507            type: 'DELETE',
508            success: function (data) {
509                $('#user_log').modal('hide');
510                $('#user_log').html(data);
511            }
512        });
513    });
514
515    $(document).on('click', '#user_log .refresh', function () {
516        var id = $(this).attr('data-id');
517
518        $.ajax({
519            url: '/user/render/' + id,
520            type: 'GET',
521            success: function (data) {
522                $('#user_log').html(data);
523            }
524        });
525    });
526
527    $(document).on('click', '#user_log .destroy', function () {
528        var id = $(this).attr('data-id');
529
530        $.ajax({
531            url: '/user/delete/' + id,
532            type: 'DELETE',
533            success: function (data) {
534                $('#user_log').modal('hide');
535                $('#user_log').html(data);
536            }
537        });
538    });
539
540    $(document).on('click', '#user_log .refresh', function () {
541        var id = $(this).attr('data-id');
542
543        $.ajax({
544            url: '/user/render/' + id,
545            type: 'GET',
546            success: function (data) {
547                $('#user_log').html(data);
548            }
549        });
550    });
551
552    $(document).on('click', '#user_log .destroy', function () {
553        var id = $(this).attr('data-id');
554
555        $.ajax({
556            url: '/user/delete/' + id,
557            type: 'DELETE',
558            success: function (data) {
559                $('#user_log').modal('hide');
560                $('#user_log').html(data);
561            }
562        });
563    });
564
565    $(document).on('click', '#user_log .refresh', function () {
566        var id = $(this).attr('data-id');
567
568        $.ajax({
569            url: '/user/render/' + id,
570            type: 'GET',
571            success: function (data) {
572                $('#user_log').html(data);
573            }
574        });
575    });
576
577    $(document).on('click', '#user_log .destroy', function () {
578        var id = $(this).attr('data-id');
579
580        $.ajax({
581            url: '/user/delete/' + id,
582            type: 'DELETE',
583            success: function (data) {
584                $('#user_log').modal('hide');
585                $('#user_log').html(data);
586            }
587        });
588    });
589
590    $(document).on('click', '#user_log .refresh', function () {
591        var id = $(this).attr('data-id');
592
593        $.ajax({
594            url: '/user/render/' + id,
595            type: 'GET',
596            success: function (data) {
597                $('#user_log').html(data);
598            }
599        });
600    });
601
602    $(document).on('click', '#user_log .destroy', function () {
603        var id = $(this).attr('data-id');
604
605        $.ajax({
606            url: '/user/delete/' + id,
607            type: 'DELETE',
608            success: function (data) {
609                $('#user_log').modal('hide');
610                $('#user_log').html(data);
611            }
612        });
613    });
614
615    $(document).on('click', '#user_log .refresh', function () {
616        var id = $(this).attr('data-id');
617
618        $.ajax({
619            url: '/user/render/' + id,
620            type: 'GET',
621            success: function (data) {
622                $('#user_log').html(data);
623            }
624        });
625    });
626
627    $(document).on('click', '#user_log .destroy', function () {
628        var id = $(this).attr('data-id');
629
630        $.ajax({
631            url: '/user/delete/' + id,
632            type: 'DELETE',
633            success: function (data) {
634                $('#user_log').modal('hide');
635                $('#user_log').html(data);
636            }
637        });
638    });
639
640    $(document).on('click', '#user_log .refresh', function () {
641        var id = $(this).attr('data-id');
642
643        $.ajax({
644            url: '/user/render/' + id,
645            type: 'GET',
646            success: function (data) {
647                $('#user_log').html(data);
648            }
649        });
650    });
651
652    $(document).on('click', '#user_log .destroy', function () {
653        var id = $(this).attr('data-id');
654
655        $.ajax({
656            url: '/user/delete/' + id,
657            type: 'DELETE',
658            success: function (data) {
659                $('#user_log').modal('hide');
660                $('#user_log').html(data);
661            }
662        });
663    });
664
665    $(document).on('click', '#user_log .refresh', function () {
666        var id = $(this).attr('data-id');
667
668        $.ajax({
669            url: '/user/render/' + id,
670            type: 'GET',
671            success: function (data) {
672                $('#user_log').html(data);
673            }
674        });
675    });
676
677    $(document).on('click', '#user_log .destroy', function () {
678        var id = $(this).attr('data-id');
679
680        $.ajax({
681            url: '/user/delete/' + id,
682            type: 'DELETE',
683            success: function (data) {
684                $('#user_log').modal('hide');
685                $('#user_log').html(data);
686            }
687        });
688    });
689
690    $(document).on('click', '#user_log .refresh', function () {
691        var id = $(this).attr('data-id');
692
693        $.ajax({
694            url: '/user/render/' + id,
695            type: 'GET',
696            success: function (data) {
697                $('#user_log').html(data);
698            }
699        });
700    });
701
702    $(document).on('click', '#user_log .destroy', function () {
703        var id = $(this).attr('data-id');
704
705        $.ajax({
706            url: '/user/delete/' + id,
707            type: 'DELETE',
708            success: function (data) {
709                $('#user_log').modal('hide');
710                $('#user_log').html(data);
711            }
712        });
713    });
714
715    $(document).on('click', '#user_log .refresh', function () {
716        var id = $(this).attr('data-id');
717
718        $.ajax({
719            url: '/user/render/' + id,
720            type: 'GET',
721            success: function (data) {
722                $('#user_log').html(data);
723            }
724        });
725    });
726
727    $(document).on('click', '#user_log .destroy', function () {
728        var id = $(this).attr('data-id');
729
730        $.ajax({
731            url: '/user/delete/' + id,
732            type: 'DELETE',
733            success: function (data) {
734                $('#user_log').modal('hide');
735                $('#user_log').html(data);
736            }
737        });
738    });
739
740    $(document).on('click', '#user_log .refresh', function () {
741        var id = $(this).attr('data-id');
742
743        $.ajax({
744            url: '/user/render/' + id,
745            type: 'GET',
746            success: function (data) {
747                $('#user_log').html(data);
748            }
749        });
750    });
751
752    $(document).on('click', '#user_log .destroy', function () {
753        var id = $(this).attr('data-id');
754
755        $.ajax({
756            url: '/user/delete/' + id,
757            type: 'DELETE',
758            success: function (data) {
759                $('#user_log').modal('hide');
760                $('#user_log').html(data);
761            }
762        });
763    });
764
765    $(document).on('click', '#user_log .refresh', function () {
766        var id = $(this).attr('data-id');
767
768        $.ajax({
769            url: '/user/render/' + id,
770            type: 'GET',
771            success: function (data) {
772                $('#user_log').html(data);
773            }
774        });
775    });
776
777    $(document).on('click', '#user_log .destroy', function () {
778        var id = $(this).attr('data-id');
779
780        $.ajax({
781            url: '/user/delete/' + id,
782            type: 'DELETE',
783            success: function (data) {
784                $('#user_log').modal('hide');
785                $('#user_log').html(data);
786            }
787        });
788    });
789
790    $(document).on('click', '#user_log .refresh', function () {
791        var id = $(this).attr('data-id');
792
793        $.ajax({
794            url: '/user/render/' + id,
795            type: 'GET',
796            success: function (data) {
797                $('#user_log').html(data);
798            }
799        });
800    });
801
802    $(document).on('click', '#user_log .destroy', function () {
803        var id = $(this).attr('data-id');
804
805        $.ajax({
806            url: '/user/delete/' + id,
807            type: 'DELETE',
808            success: function (data) {
809                $('#user_log').modal('hide');
810                $('#user_log').html(data);
811            }
812        });
813    });
814
815    $(document).on('click', '#user_log .refresh', function () {
816        var id = $(this).attr('data-id');
817
818        $.ajax({
819            url: '/user/render/' + id,
820            type: 'GET',
821            success: function (data) {
822                $('#user_log').html(data);
823            }
824        });
825    });
826
827    $(document).on('click', '#user_log .destroy', function () {
828        var id = $(this).attr('data-id');
829
830        $.ajax({
831            url: '/user/delete/' + id,
832            type: 'DELETE',
833            success: function (data) {
834                $('#user_log').modal('hide');
835                $('#user_log').html(data);
836            }
837        });
838    });
839
840    $(document).on('click', '#user_log .refresh', function () {
841        var id = $(this).attr('data-id');
842
843        $.ajax({
844            url: '/user/render/' + id,
845            type: 'GET',
846            success: function (data) {
847                $('#user_log').html(data);
848            }
849        });
850    });
851
852    $(document).on('click', '#user_log .destroy', function () {
853        var id = $(this).attr('data-id');
854
855        $.ajax({
856            url: '/user/delete/' + id,
857            type: 'DELETE',
858            success: function (data) {
859                $('#user_log').modal('hide');
860                $('#user_log').html(data);
861            }
862        });
863    });
864
865    $(document).on('click', '#user_log .refresh', function () {
866        var id = $(this).attr('data-id');
867
868        $.ajax({
869            url: '/user/render/' + id,
870            type: 'GET',
871            success: function (data) {
872                $('#user_log').html(data);
873            }
874        });
875    });
876
877    $(document).on('click', '#user_log .destroy', function () {
878        var id = $(this).attr('data-id');
879
880        $.ajax({
881            url: '/user/delete/' + id,
882            type: 'DELETE',
883            success: function (data) {
884                $('#user_log').modal('hide');
885                $('#user_log').html(data);
886            }
887        });
888    });
889
890    $(document).on('click', '#user_log .refresh', function () {
891        var id = $(this).attr('data-id');
892
893        $.ajax({
894            url: '/user/render/' + id,
895            type: 'GET',
896            success: function (data) {
897                $('#user_log').html(data);
898            }
899        });
900    });
901
902    $(document).on('click', '#user_log .destroy', function () {
903        var id = $(this).attr('data-id');
904
905        $.ajax({
906            url: '/user/delete/' + id,
907            type: 'DELETE',
908            success: function (data) {
909                $('#user_log').modal('hide');
910                $('#user_log').html(data);
911            }
912        });
913    });
914
915    $(document).on('click', '#user_log .refresh', function () {
916        var id = $(this).attr('data-id');
917
918        $.ajax({
919            url: '/user/render/' + id,
920            type: 'GET',
921            success: function (data) {
922                $('#user_log').html(data);
923            }
924        });
925    });
926
927    $(document).on('click', '#user_log .destroy', function () {
928        var id = $(this).attr('data-id');
929
930        $.ajax({
931            url: '/user/delete/' + id,
932            type: 'DELETE',
933            success: function (data) {
934                $('#user_log').modal('hide');
935                $('#user_log').html(data);
936            }
937        });
938    });
939
940    $(document).on('click', '#user_log .refresh', function () {
941        var id = $(this).attr('data-id');
942
943        $.ajax({
944            url: '/user/render/' + id,
945            type: 'GET',
946            success: function (data) {
947                $('#user_log').html(data);
948            }
949        });
950    });
951
952    $(document).on('click', '#user_log .destroy', function () {
953        var id = $(this).attr('data-id');
954
955        $.ajax({
956            url: '/user/delete/' + id,
957            type: 'DELETE',
958            success: function (data) {
959                $('#user_log').modal('hide');
960                $('#user_log').html(data);
961            }
962        });
963    });
964
965    $(document).on('click', '#user_log .refresh', function () {
966        var id = $(this).attr('data-id');
967
968        $.ajax({
969            url: '/user/render/' + id,
970            type: 'GET',
971            success: function (data) {
972                $('#user_log').html(data);
973            }
974        });
975    });
976
977    $(document).on('click', '#user_log .destroy', function () {
978        var id = $(this).attr('data-id');
979
980        $.ajax({
981            url: '/user/delete/' + id,
982            type: 'DELETE',
983            success: function (data) {
984                $('#user_log').modal('hide');
985                $('#user_log').html(data);
986            }
987        });
988    });
989
990    $(document).on('click', '#user_log .refresh', function () {
991        var id = $(this).attr('data-id');
992
993        $.ajax({
994            url: '/user/render/' + id,
995            type: 'GET',
996            success: function (data) {
997                $('#user_log').html(data);
998            }
999        });
1000    });
1001
1002    $(document).on('click', '#user_log .destroy', function () {
1003        var id = $(this).attr('data-id');
1004
1005        $.ajax({
1006            url: '/user/delete/' + id,
1007            type: 'DELETE',
1008            success: function (data) {
1009                $('#user_log').modal('hide');
1010                $('#user_log').html(data);
1011            }
1012        });
1013    });
1014
1015    $(document).on('click', '#user_log .refresh', function () {
1016        var id = $(this).attr('data-id');
1017
1018        $.ajax({
1019            url: '/user/render/' + id,
1020            type: 'GET',
1021            success: function (data) {
1022                $('#user_log').html(data);
1023            }
1024        });
1025    });
1026
1027    $(document).on('click', '#user_log .destroy', function () {
1028        var id = $(this).attr('data-id');
1029
1030        $.ajax({
1031            url: '/user/delete/' + id,
1032            type: 'DELETE',
1033            success: function (data) {
1034                $('#user_log').modal('hide');
1035                $('#user_log').html(data);
1036            }
1037        });
1038    });
1039
1040    $(document).on('click', '#user_log .refresh', function () {
1041        var id = $(this).attr('data-id');
1042
1043        $.ajax({
1044            url: '/user/render/' + id,
1045            type: 'GET',
1046            success: function (data) {
1047                $('#user_log').html(data);
1048            }
1049        });
1050    });
1051
1052    $(document).on('click', '#user_log .destroy', function () {
1053        var id = $(this).attr('data-id');
1054
1055        $.ajax({
1056            url: '/user/delete/' + id,
1057            type: 'DELETE',
1058            success: function (data) {
1059                $('#user_log').modal('hide');
1060                $('#user_log').html(data);
1061            }
1062        });
1063    });
1064
1065    $(document).on('click', '#user_log .refresh', function () {
1066        var id = $(this).attr('data-id');
1067
1068        $.ajax({
1069            url: '/user/render/' + id,
1070            type: 'GET',
1071            success: function (data) {
1072                $('#user_log').html(data);
1073            }
1074        });
1075    });
1076
1077    $(document).on('click', '#user_log .destroy', function () {
1078        var id = $(this).attr('data-id');
1079
1080        $.ajax({
1081            url: '/user/delete/' + id,
1082            type: 'DELETE',
1083            success: function (data) {
1084                $('#user_log').modal('hide');
1085                $('#user_log').html(data);
1086            }
1087        });
1088    });
1089
1090    $(document).on('click', '#user_log .refresh', function () {
1091        var id = $(this).attr('data-id');
1092
1093        $.ajax({
1094            url: '/user/render/' + id,
1095            type: 'GET',
1096            success: function (data) {
1097                $('#user_log').html(data);
1098            }
1099        });
1100    });
1101
1102    $(document).on('click', '#user_log .destroy', function () {
1103        var id = $(this).attr('data-id');
1104
1105        $.ajax({
1106            url: '/user/delete/' + id,
1107            type: 'DELETE',
1108            success: function (data) {
1109                $('#user_log').modal('hide');
1110                $('#user_log').html(data);
1111            }
1112        });
1113    });
1114
1115    $(document).on('click', '#user_log .refresh', function () {
1116        var id = $(this).attr('data-id');
1117
1118        $.ajax({
1119            url: '/user/render/' + id,
1120            type: 'GET',
1121            success: function (data) {
1122                $('#user_log').html(data);
1123            }
1124        });
1125    });
1126
1127    $(document).on('click', '#user_log .destroy', function () {
1128        var id = $(this).attr('data-id');
1129
1130        $.ajax({
1131            url: '/user/delete/' + id,
1132            type: 'DELETE',
1133            success: function (data) {
1134                $('#user_log').modal('hide');
1135                $('#user_log').html(data);
1136            }
1137        });
1138    });
1139
1140    $(document).on('click', '#user_log .refresh', function () {
1141        var id = $(this).attr('data-id');
1142
1143        $.ajax({
1144            url: '/user/render/' + id,
1145            type: 'GET',
1146            success: function (data) {
1147                $('#user_log').html(data);
1148            }
1149        });
1150    });
1151
1152    $(document).on('click', '#user_log .destroy', function () {
1153        var id = $(this).attr('data-id');
1154
1155        $.ajax
```

```

34      }
35
36      if (obj.verb == 'destroyed') {
37          $("#user_" + obj.id).remove();
38          toastr.info('<%= i18n('user_destroyed') %>', 'RobotUI');
39      }
40
41  });
42
43 }
44
45 </script>

```

Por otro lado se realiza la llamada a la función *listenMessages* ya que independientemente de la vista en la que nos encontremos siempre nos mantendremos a la escucha de los mensajes recibidos a nuestra bandeja de entrada por parte de otros usuarios.

```

1 //Evento a la espera de recibir mensajes y notificar al usuario
2 function listenMessages(){
3     io.socket.on('user', function messageReceived(message) {
4         switch (message.verb) {
5             case 'messaged':
6                 var pathname = window.location.pathname;
7                 if(pathname == '/message/index'){
8                     location.reload();
9                 }else{
10                     new_msg_num_update('<%= i18n('messages') %>');
11                 }
12                 toastr.info('<%= i18n('new_message') %>' + '<a
13                     href="/message/index"> <%= i18n('open_here') %> </a>' ,
14                     'RobotUI');
15                 break;
16             default:
17                 break;
18         }
19     });
}

```

6.3. Desconexión

En este apartado describiremos los puntos de mayor relevancia a la hora de la desconexión de alguna de las diferentes comunicaciones establecidas comenzando con la desconexión de un usuario.

Cuando un usuario realiza una desconexión, bien deslogueándose de la página o cerrando una de las ventanas abiertas, automáticamente se lanza una llamada a la función *afterDisconnect* localizada en el fichero *config/Socket.js* del servidor. Función que será ejecutada cada vez que un socket es desconectado del sistema.

Dicha función primeramente localiza qué sesión en la base de relacionada está relacionada con identificador del socket desconectado. Si este socket estaba usando algún robot, éste se libera cambiando el estado del robot a *libre*. Y se informa a todos los sockets abiertos que el robot queda libreado y accesible al resto de usuarios.

```

1  Robot.update({id: session.robot_id}, {busy: false}, function
2    robotUpdated(err) {
3      if (err) return next(err);
4
5      // Informar a otros clientes (sockets abiertos) que el robot queda
6      // liberado
7      Robot.publishUpdate(session.robot_id, {
8        busy: false,
9        id: session.robot_id
10   });
11 });

```

A continuación, se comprueba si el usuario no dispone de más sockets abiertos. Si fuera el caso de que no los disponga, entonces se procede a cambiar su estado a desconectado y se emite o se informa al resto de clientes (sockets abiertos) que el usuario ya no se encuentra en el sistema. Llamada al método *User.publishUpdate*:

```

1  User.publishUpdate(session.user_id, {
2    loggedIn: false,
3    id: session.user_id
4  });

```

Se comprueba si el usuario desconectado se encontraba en alguna *room*. Pongamos como ejemplo que el usuario ha abandonado la visualización de un robot por lo que se debe informar a todos los sockets integrantes de esa *room* que un usuario con un identificador determinado ha abandonado con la función *sails.sockets.broadcast*. A continuación se muestra el fragmento de código:

```

1 // Emite a cada room que que se encuentre la sesión que un usuario la
2 // ha abandonado
3 session.rooms.forEach(function (room) {
4   sails.sockets.broadcast(room.room_name, {type: 'exit', msg:
5     {user_id: session.user_id}});
6 });

```

Finalmente se destruye la sesión:

```

1   Session.destroy(session.id, function sessionDestroyed(err) {
2     if (err) return cb();
3     return cb();
4   });

```

A continuación mostramos el código completo de la función *afterDisconnect*:

```

1  afterDisconnect: function(session, socket, cb) {
2      console.log('Disconnected - id del socket: ' + socket.id);
3
4
5      //Session del socket cerrado,
6      Session.findOne({socket_id:
7          socket.id}).populate('rooms').exec(function (err, session) {
8          if (err) return cb();
9          if (!session) return cb();
10
11         //Comprobar si el soscket estaba usando algun robot para
12         //liberarlo:
13         if (session.robot_id) {
14             console.log('Socked was using a robot');
15
16             Robot.update({id: session.robot_id}, {busy: false}, function
17                 robotUpdated(err) {
18                     if (err) return next(err);
19
20                     //Informar a otros clientes (sockets abiertos) que el robot
21                     //queda liberado
22                     Robot.publishUpdate(session.robot_id, {
23                         busy: false,
24                         id: session.robot_id
25                     });
26                 });
27         }
28
29
30
31
32         //Comprobar si el usuario tiene m s sockets abiertos:
33         Session.count({user_id: session.user_id}).exec(function
34             countUserSessions(error, n_sessions) {
35                 console.log('There are ' + n_sessions + ' users ' +
36                     session.user_id);
37
38                 //Cambiamos el usuario a offline si solo ten a una ventana o
39                 //conexi n abierta.
40                 if (n_sessions == 1) {
41                     User.update(session.user_id, {online: false}, function (err)
42                     {
43                         if (err) return cb(err);
44
45                         //Informar a otros clientes (sockets abiertos) que el
46                         //usuario ya NO se encuentra logueado
47                         User.publishUpdate(session.user_id, {
48                             loggedIn: false,
49                             id: session.user_id
50                         });
51                     });
52                 }
53             });
54         });
55     });
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94

```

```

45      });
46    });
47  }
48
49  Session.destroy(session.id, function sessionDestroyed(err) {
50    if (err) return cb();
51    return cb();
52  });
53});
54});
55}

```

En este punto ya conocemos cómo la aplicación RobotUI gestiona las diferentes conexiones y cómo se realiza el almacenamiento en base de datos de la información necesaria para determinar qué sockets se corresponde con qué usuarios en todo momento. Todo ello además ha sido explicado para la gestión de conexiones y desconexiones de usuarios notificando los cambios de estados entre *online* y *offline*.

Para la conexión y desconexión de los robots se aplica la metodología anterior de manera análoga. Pero... en el caso particular de los robots. ¿Cómo se realizan las comunicaciones entre el cliente (navegador) y el robot?, ¿Cómo es controlado un robot?, ¿Cómo se difunden los comandos lanzados a un robot al resto de usuarios espectadores?. En los sucesivos puntos iremos describiendo cómo se ha implementado estos comportamientos.

6.4. Envío de comandos al robot

Cuando un usuario ha recibido la notificación de que un determinado dispositivo se encuentra disponible, entonces procede a acceder a su interfaz de control realizándose una conexión con el mismo. Una vez establecida dicha conexión el usuario puede proceder al lanzamiento de comandos para el control del robot. En ese momento se realiza un procedimiento para el envío de la información que consta de tres puntos:

1. Envío del comando indicado al robot.
2. Envío de notificación al servidor del comando lanzado.
3. Envío de la notificación por parte del servidor a todos los clientes suscritos al robot de que un comando ha sido enviado.

EL gráfico 6.5 muestra el flujo de datos generado cada vez que un comando es enviado al robot:

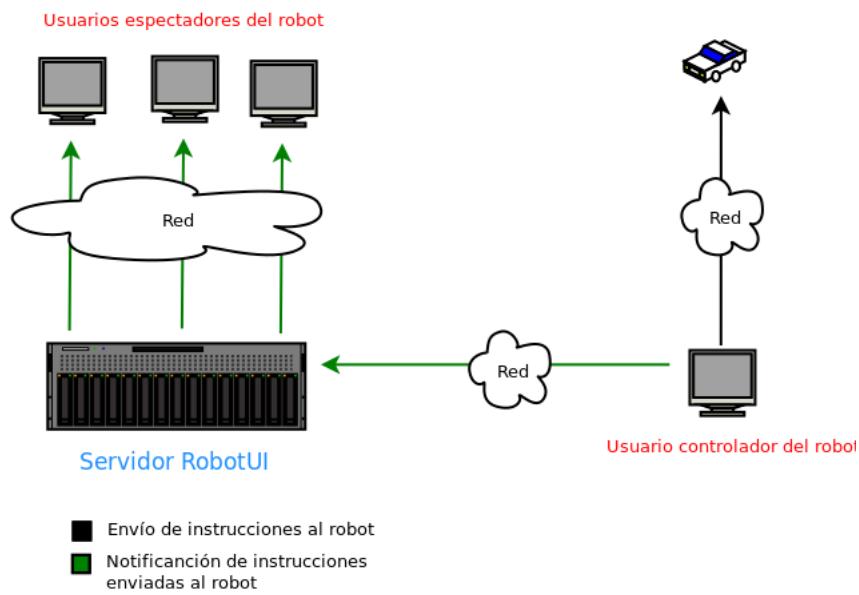


Figura 6.4: Esquema representativo del flujo de datos originado por un usuario controlador de un robot.

El código inferior realiza el envío del comando al robot para su interpretación y al servidor para su posterior difusión a los clientes suscritos:

```

1 //Presionar boton accion!
2 $('[id^=button_]').click(function (e) {
3   e.preventDefault();
4   var button_pressed_id = this.id.replace('button_custom_', '');
5   var code = document.getElementById(this.id).value
6
7   //Comunicacion al servidor del comando para difundir a los
8   //usuarios espectadores
9   io.socket.get('/interface/emit_action/', {robot: '<%=robot.id
10 %>', id: button_pressed_id, msg: code })
11
12   if (code != ''){
13     $('#chat').prepend('_ ' + code + '<br/>');
14     //Mandar instrucion al robot
15     socket.emit('action', code);
16     console.log('Emitting comand: ' + code );
17   }
}) ;

```

6.5. Captura de datos del robot

Ahora bien, ya sabemos el procedimiento realizado a la hora de lanzar comandos al robot pero, esta comunicación tendría poco sentido si no capturamos sus respuestas.

Estas respuestas pueden ser los datos correspondientes a los diferentes frames obtenidos por la cámara de vídeo del robot o aquellos datos correspondientes a la medición de los diferentes sensores de los que pueda disponer el robot.

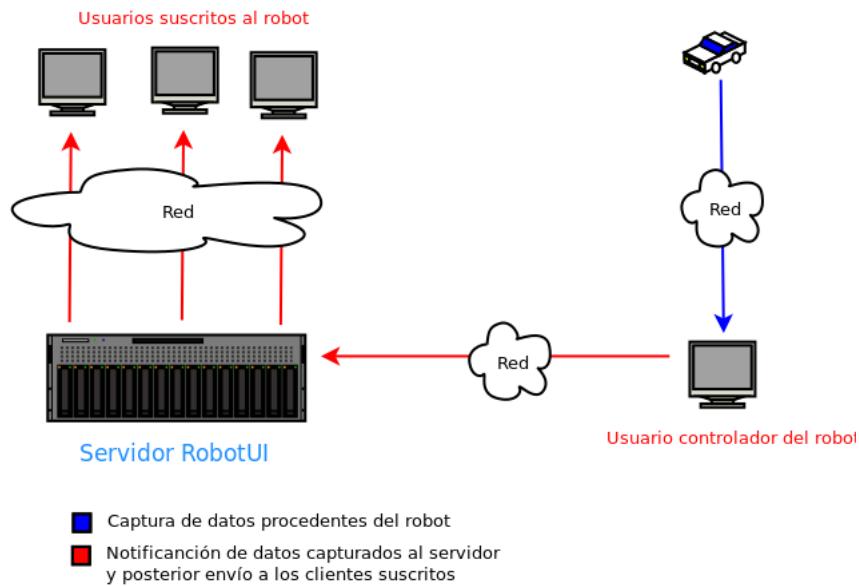


Figura 6.5: Esquema representativo del flujo de datos originado por un usuario tras la captura de datos procedentes del robot.

El código inferior muestra cómo se realiza la captura de los frames de vídeos procedentes del robot. Dicho código envía al servidor el frame para su posterior difusión con los clientes suscritos a robot y “dibuja” el frame capturado en el canvas de la interfaz de control del robot:

```

1 //Captura de video
2 socket.on('<%= video.event_name %>', function(data) {
3     io.socket.get('/interface/emit_video/', {robot: '<%=robot.id %>',
4         id: '<%= video.id %>', msg: data });
5     draw_image_to_canvas('play-<%= video.id %>', data)
}) ;

```

Código de la acción *emit_video* localizada en el servidor y llamada desde el cliente (código superior), la cual emite el frame capturado a todos los clientes que se encuentran suscritos al robot (broadcast a todos los clientes suscritos a la sala cuyo nombre se corresponde con el identificador del robot):

```

1 //Emision de las acciones a los vivitantes de una interfaz
2 emit_video: function(req, res){
3     if (!req.isSocket) return res.badRequest();
4     sails.sockets.broadcast(req.param('robot'), {type: 'video', id:
5         req.param('id'), msg: req.param('msg')});
}

```


Capítulo 7

Robot de pruebas

Con la finalidad de demostrar y hacer un primer uso de la aplicación RobotUI se ha decidido abordar la elaboración de un vehículo de pruebas. Dicho vehículo será utilizado de modelo o guía para el resto de personas que quieran crear un robot para su integración en la aplicación o bien para programar un robot del que ya dispongan previamente.

En el presente capítulo se detalla los diferentes pasos que se han seguido a la hora de la construcción y programación del vehículo desarrollado. Este capítulo tiene como objetivo proporcionar al usuario una guía básica a partir de la cual poder desarrollar sus propios robots e integrarlos en la aplicación RobotUI.

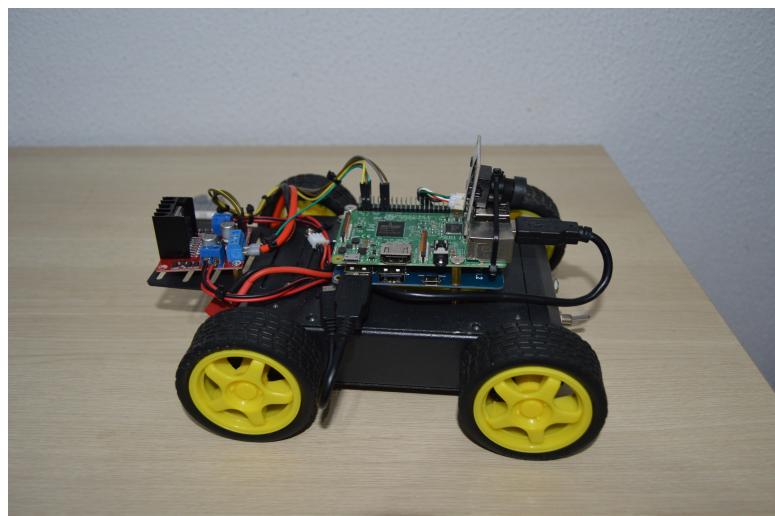


Figura 7.1: Imagen del robot de pruebas desarrollado.

7.1. Análisis

7.1.1. Análisis de requerimientos hardware

Se ha optado por la construcción de un pequeño robot móvil dotado de un chasis de 4 ruedas donde cada una de ellas está accionada por un motor. Los motores seleccionados deberán funcionar a corriente continua de manera que en función de la polarización de los terminales haga girar las ruedas en una dirección o la contraria (marcha adelante o atrás).

El chasis utilizado deberá permitir añadir multitud de componentes necesarios para construir el robot, deberá disponer de paneles para instalar las diferentes placas electrónicas y sensores.

El robot además necesita de una cámara para la obtención de vídeo y su transmisión siendo necesaria una cámara de pequeñas dimensiones de alta definición.

Por otra parte, todo robot necesita de una unidad de central de procesamiento donde se localizará el programa de control. Este programa tendrá la función de interpretar las diferentes señales recibidas, control de sensores y dispositivos conectados. Además, esta placa es la encargada de distribuir la alimentación por los diferentes componentes hardware que lo necesiten y recibir las señales de los sensores y enviarla a los motores. Utilizando para ello la placa Raspberry Pi modelo B cuya descripción se encuentra en la sección [3.3.1](#).

Este modelo de placa dispone de una serie de pines denominados GPIO (General Purpose Input/Output) que son, como su propio nombre indica, un sistema de E/S (Entrada/Salida) de propósito general, es decir, una serie de conexiones que se pueden usar como entradas o salidas para usos múltiples. Estos pines están incluidos en todos los modelos de Raspberry Pi, con la finalidad de ser utilizados en diferentes proyectos de una manera similar a la que se haría con Arduino¹.

7.1.2. Análisis de requerimientos software

Habiendo detallado los requerimientos hardware para la construcción del robot, pasamos al análisis de los diferentes requerimiento software para la correcta gestión de los diferentes elementos hardware y para su correcto funcionamiento.

En cuanto a la programación del robot, uno de los requisitos fundamentales es el de disponer de una vía de comunicación bidireccional.

Para ello se ha seleccionado el entorno de ejecución Node.Js, el cual permite escribir programas en JavaScript además de disponer de una gran cantidad de bibliotecas

¹Arduino es una plataforma de prototipos electrónica de código abierto (open-source) basada en hardware y software flexibles y fáciles de usar.

hechas por toda una comunidad que la respalda. Además de que permite escribir programas relativamente complejos en tan sólo unas pocas líneas de código.

se ha realizado haciendo uso de Websockets los cuales presentan algunas ventajas sobre simples solicitudes http:

- **Velocidad:** Una petición http normal tiene que establecer una conexión antes de comenzar las transacciones, la cual toma bastante tiempo. Los Websockets, una vez establecida la conexión, siempre están abiertos y listos para enviar o recibir datos. Esto significa que el retraso puede ser tan bajo como su ping, en torno a un milisegundo o dos en la mayoría de los casos.
- **Bidireccional:** Los Websockets permiten transmisión de datos en ambas direcciones permitiendo la activación de eventos en el cliente y viceversa.

Como podemos ver las propiedades anteriormente descritas resultan esenciales para nuestro proyecto que, como cabe recordar, queremos realizar lectura de sensores y envío de órdenes desde un servidor externo. Además de la transmisión de vídeo.

Estados del robot

Otro de los requerimientos es que el robot deberá de responder a una serie de estados de tal modo que en la aplicación se conozca las diferentes situaciones en la que se pueda encontrar un robot. En el autómata diseñado encontramos un modo desactivado, modo de espera y modo de conexión. El autómata representativo es el siguiente:

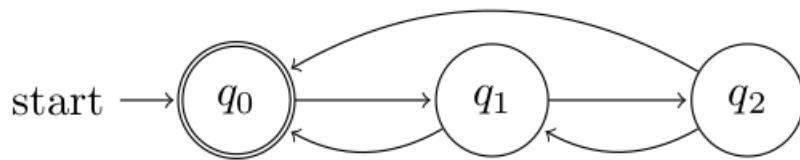


Figura 7.2: Autómata representativo de los diferentes estados del robot.

Donde:

- Estado q_0 inicial y final se corresponde con el estado apagado.

- Estado q1 se corresponde con el estado en escucha.
- Estado q2 se corresponde con el estado en funcionamiento.

7.2. Montaje

En esta sección se recogen todas las descripciones y procedimientos seguidos y que han resultado de mayor interés a la hora de la construcción del robot y sus diferentes interconexiones.

7.2.1. Interconexión de elementos

En el presente punto, se recoge todos aquellos puntos de interés referentes a la interconexión de elementos utilizados, los cuales quedaron descritos en la sección *Tecnologías hardware 3.3* junto con sus especificaciones.

Comenzando por el módulo central del sistema, la placa Raspberry Pi Model B+, dispone de una serie de pines denominados GPIO (General Purpose Input/Output) es, como su propio nombre indica, un sistema de E/S (Entrada/Salida) de propósito general, es decir, una serie de conexiones que se pueden usar como entradas o salidas para usos múltiples. Estos pines se encuentran en todos los modelos de Raspberry Pi.

Los GPIO representan la interfaz entre la Raspberry Pi y el mundo exterior. En este caso para el control de los motores del vehículo.

Primeramente, antes de comenzar a utilizarlos debemos conocer sus características y como están distribuidos en nuestra placa. los pines GPIO para la Raspberry Pi 3 Model B, se encuentran distribuidos de la siguiente manera:



Figura 7.3: Esquema GPIO de una Raspberry Pi Model B+.

Los pines empleados son los siguientes para la construcción del robot han sido los siguientes:

GPIO	Modo	Control
2	OUTPUT	Motores lado izquierdo
3	OUTPUT	Motores lado izquierdo
17	OUTPUT	Motores lado derecho
27	OUTPUT	Motores lado derecho

Tabla 7.1: Tabla con los puertos GPIO utilizados, su configuración establecida y su utilización.

Pero existe una problemática y es que no podemos conectar directamente los pines de salida de nuestra placa Raspberry Pi directamente a los motores. Esto es debido a que la placa no dispone de potencia suficiente para mover actuadores. De hecho, la función de la placa no debe ser ejecutar acciones sino mandar ejecutar acciones a drivers que realicen el trabajo pesado.

Para ello empleamos el L298N, un controlador (driver) de motores, el cual nos permite encender y controlar dos motores de corriente continua desde una Raspberry Pi, Arduino o cualquier placa similar, variando tanto la dirección como la velocidad de giro.

La corriente máxima que el L298N puede suministrar a los motores es, en teoría, 2A por salida (hasta 3A de pico) y una tensión de alimentación de 3V a 35V. Sin embargo, el L298N tiene una eficiencia baja. La electrónica supone una caída de tensión de unos 3V, es decir, la tensión que recibe el motor es unos 3V inferior a la tensión de alimentación.

Estas pérdidas se disipan en forma de calor lo que se traduce en que, a efectos prácticos, es difícil que podamos obtener más de 0.8-1A por fase sin exceder el rango de temperatura de funcionamiento.

El L298N incorpora protecciones contra efectos que pueden producirse al manejar motores de corriente continua. Dispone de protecciones contra sobre intensidad, sobre temperatura, y diodos de protección contra corrientes inducidas.

El módulo seleccionado, el cual incorpora el controlador L298N, cuenta con todos los componentes necesarios para funcionar sin necesidad de elementos adicionales, entre ellos diodos de protección y un regulador **LM7805** que suministra 5V a la parte lógica del integrado L298N. Además cuenta con jumpers de selección para habilitar cada una de las salidas del módulo (A y B). La **salida A** esta conformada por **OUT1** y **OUT2** y la **salida B** por **OUT3** y **OUT4**. Los pines de habilitación son **ENA** y **ENB** respectivamente.

En la figura 7.4 se muestra el módulo empleado con sus diferentes pines al detalle.

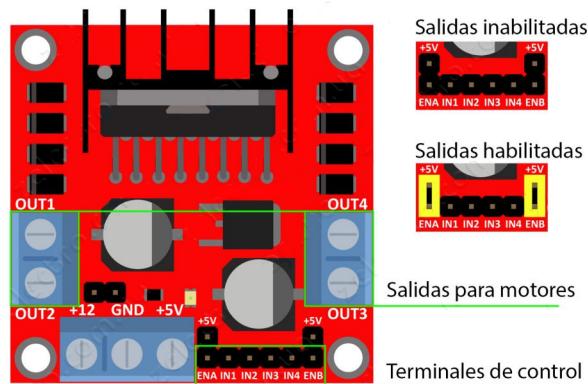


Figura 7.4: Pines de entrada/salida del módulo L298N empleado.

Alimentación

Para la alimentación del conjunto se ha optado por la utilización de dos baterías. Una para dotar de energía a la placa de control y otra para la alimentación de los motores debido a la gran cantidad de energía que éstos demandan y su elevado consumo.

Para la alimentación de la placa Raspberry Pi, se ha optado por la utilización de una batería de Litio desarrollada específicamente para su uso con este modelo de placas el cual permite una integración perfecta. Dicho módulo de alimentación queda descrito en la subapartado correspondiente de herramientas utilizadas [3.3.4](#).

Imagen de la Raspberry Pi junto con su módulo de expansión de batería:



Figura 7.5: Conjunto Raspberry Pi y módulo de expansión de alimentación.

Imagen de la batería LiPo para la alimentación de los motores:



Figura 7.6: Batería LiPo que alimenta los motores.

Esquema de conexiones

El siguiente gráfico 7.7 muestra las conexiones de todo el conjunto:

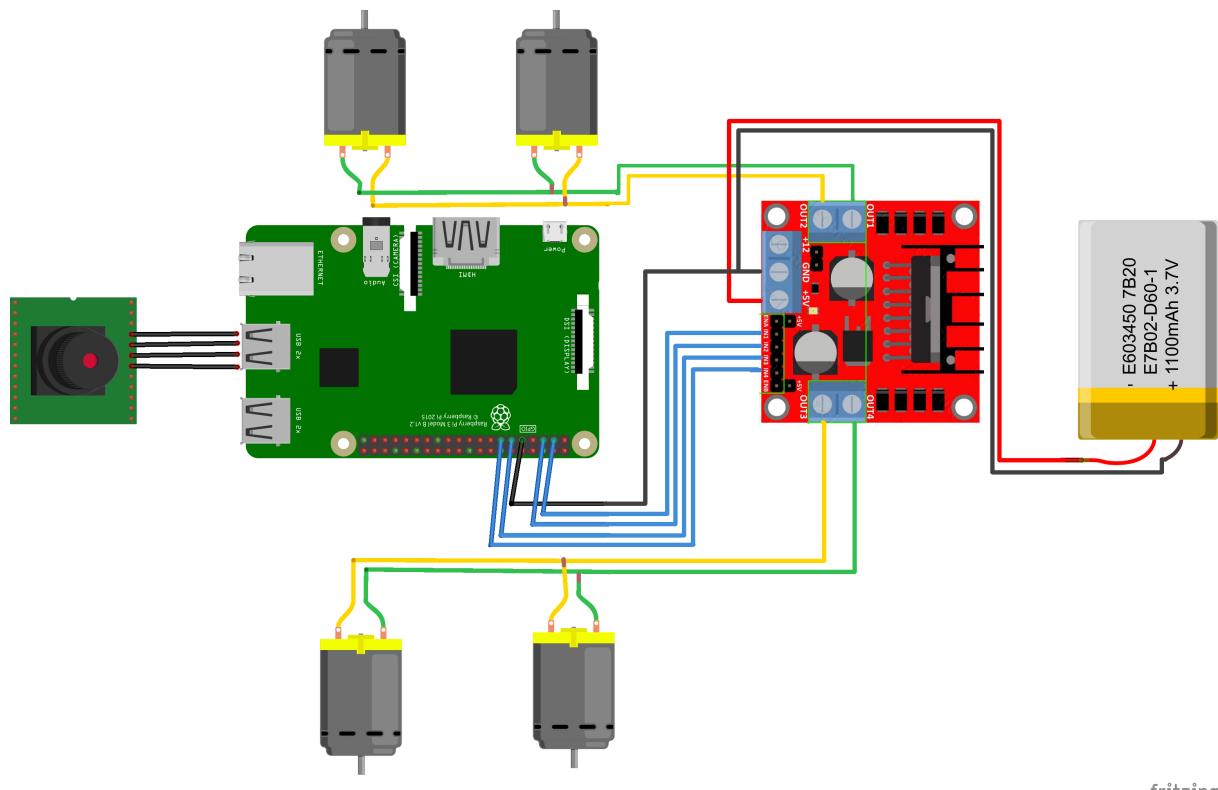


Figura 7.7: Esquema de conexiones del robot de pruebas.

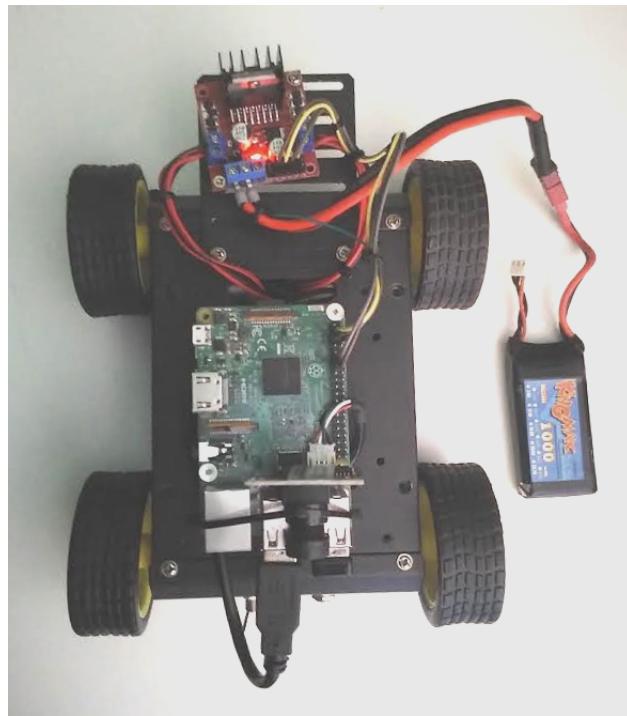


Figura 7.8: Vista superior del vehículo.

7.3. Software de control

Para la programación del robot se ha empleado el lenguaje de programación JavaScript en un entorno de ejecución Node.js. A continuación se describirá aquellos aspectos más importantes referentes al código desarrollado para el control del robot.

Primeramente se ha realizado la carga de librerías necesarias, entre ellas encontramos:

- *pigpio*: Módulo para la comunicación y control de los pines GPIO.
- *child process*: El módulo *child_process* proporciona la capacidad de generar procesos secundarios. Se ha empleado para la captura de vídeo mediante el lanzado de comandos *ffmpeg*.
- *socket.io*: Biblioteca que establece enlaces bidireccionales en tiempo real y en comunicación basada por eventos.

7.3.1. Entrada/Salida

En segundo lugar se han definido los diferentes pines GPIO a utilizar y si serán empleados como pines de entrada o de salida:

GPIO	Modo	Control
2	OUTPUT	Motores lado izquierdo
3	OUTPUT	Motores lado izquierdo
17	OUTPUT	Motores lado derecho
27	OUTPUT	Motores lado derecho

Tabla 7.2: Configuración establecida para los puertos GPIO.

Si para el vehículo que deseemos programar resultan necesarios más pines para la utilización de servos, sensores o cualquier otro elemento, tan solo debemos inicializarlos e indicar si van a ser pines de entrada o de salida. Si existen dudas al respecto se puede acceder a la documentación de la biblioteca *pigpio* en el siguiente enlace: <https://www.npmjs.com/package/pigpio>. Para el caso de este proyecto, la inicialización de los pines se ha realizado mediante las siguientes instrucciones:

```

1 // Carga del módulo.
2 var Gpio = require('pigpio').Gpio;
3
4 // Pines utilizados. Motores izquierdos: 2 y 3, motores derechos: 17
5 // y 27
6 var gpio2 = new Gpio(2, {mode: Gpio.OUTPUT}),
7     gpio3 = new Gpio(3, {mode: Gpio.OUTPUT}),
8     gpio17 = new Gpio(17, {mode: Gpio.OUTPUT}),
9     gpio27 = new Gpio(27, {mode: Gpio.OUTPUT});

```

7.3.2. Comunicaciones

En el punto anterior hemos visto las diferentes configuraciones de Entrada/Salida establecidas para el control de los diferentes motores y sensores. En el presente y sucesivos puntos describiremos los diferentes canales de comunicación abiertos y los flujos de información existentes. En la figura 7.9 se muestra un gráfico representativo de los diferentes flujos de datos existentes:

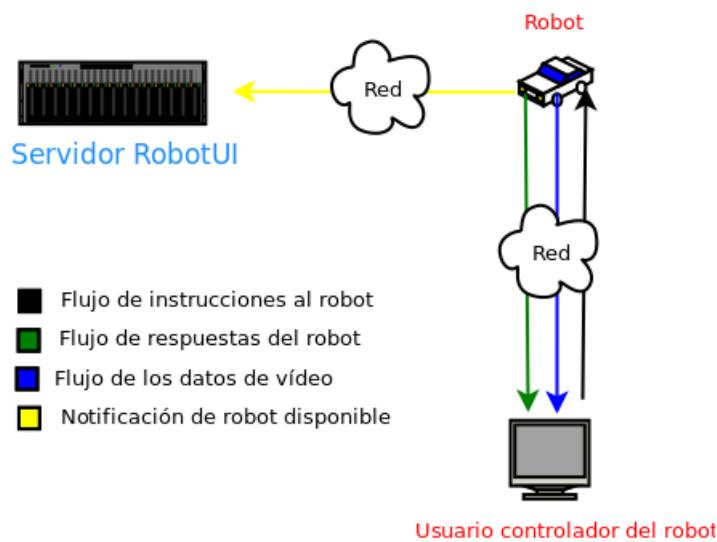


Figura 7.9: Canales de comunicación abiertos por el robot.

Primeramente al accionar el robot, éste envía una notificación al servidor indicando que se encuentra en modo *online* y permanece a la espera de que algún usuario de la aplicación decida conectarse para su control. Canal de comunicación representado en amarillo en la figura 7.9.

Un usuario ve disponible el robot y decide conectarse. Entonces se establece un enlace entre el robot, servidor, y el usuario, cliente. Flujo de comunicación representado en negro, figura 7.9.

El usuario envía comandos al robot a través del canal abierto y las respuestas son enviadas a al cliente, transmisiones representados en verde para datos y en azul para el vídeo, figura 7.9.

Las tres vías de comunicación entre el robot y el usuario discurren dentro de un mismo canal de comunicación (socket). En el siguiente punto se describirá más detalladamente el procedimiento.

Sockets

Ahora bien, una vez definidos los pines que activarán nuestros motores y los canales de comunicación necesitamos que éstos sean activados cuando desde una red externa lo indiquemos haremos uso de la biblioteca Socket.io. Comenzamos el código incluyendo las librerías necesarias:

```
1 var io_client = require('./node_modules/socket.io-client');
2 var sails_client = require('./node_modules/sails.io.js');
```

Para la comunicaciones usuario y dispositivo robótico se ha empleado la biblioteca socket.io-client.

Para las comunicaciones directas con el servidor se ha empleado el SDK² proporcionado por el framework para comunicarse con Sails a través de sockets desde una aplicación Node.js o desde el propio navegador.

El siguiente paso una vez cargadas las librerías es establecer las diferentes conexiones. El primer comportamiento deseado es, por parte del robot, lanzar un mensaje al servidor indicando su disponibilidad al servidor con la finalidad de enviar las diferentes notificaciones a los usuarios que estén usando la aplicación. Para ello establecemos la conexión y enviamos un mensaje al servidor con el identificador del robot junto con su estado online igual a true. A continuación mostramos el código:

```
1 var io_server = sails_client(io_client);
2 io_server.sails.url = 'http://46.101.102.33:80';
3 io_server.socket.get('/robot/changetoonline/', {robot:
    '59188631c8e94ba54f7a4bdc', online: true});
```

Una vez realizado el paso anterior, tenemos un robot que ha indicado a la aplicación web principal de que se encuentra en estado online pero nada más. El siguiente paso sería establecer alguna comunicación que se mantuviera a la escucha a la espera de la llegada de nuevas conexiones. Para la creación del socket basta con la siguiente instrucción, la cual recibe un puerto que utilizará para mantenerse a la escucha:

```
1 var io = require('./node_modules/socket.io').listen(8085, { log:
    false });
```

Con la finalidad de ir capturando los diferentes eventos, se han definido las siguientes funciones para la conexión y desconexión de los clientes:

```
1
2 io.sockets.on('connection', function (socket)
3 {
4
5     //Almacenamiento del número total de clientes conectados.
6     sockets[socket.id] = socket;
7     console.log("Total clientes conectados : ",
8         Object.keys(sockets).length);
9
10    //Envío de un saludo.
11    socket.emit('robotmsg', {msg: "!!!!HOLA!!!!"});
12
13    //Salida de un cliente.
14    socket.on('disconnect', function() {
15        console.log('Bye!');
16        stopStreaming(socket);
17    });
18}
```

²Un kit de desarrollo o SDK (siglas en inglés de software development kit) es generalmente un conjunto de herramientas de desarrollo de software que le permite al programador o desarrollador de software crear aplicaciones para un sistema concreto, por ejemplo ciertos paquetes de software, frameworks, etc.

```
18 }
19 }
```

Cuando un evento *action* es recibido se activada la función que procesa el comando recibido y activa las salidas correspondientes, la cual establece los pines necesarios a los valores 1 o 0 según el parámetro establecido. La tabla 7.3 muestra las diferentes combinaciones de salidas y su acción correspondiente:

Acción	GPIO 2	GPIO 3	GPIO 17	GPIO 27
UP	1	0	1	0
DOWN	0	1	0	1
LEFT	1	0	0	0
RIGHT	0	0	1	0
STOP	0	0	0	0

Tabla 7.3: Combinaciones de salida para los puertos GPIO y su acción correspondiente.

A continuación se muestra el ejemplo desarrollado para la activación de los motores en sentido de giro y dirección según la tabla anterior:

```
1 // Escucha de comandos.
2 socket.on('action', function (data){
3
4     console.log('Comando recibido: ' + data);
5
6     switch(data) {
7         case 'UP':
8             gpio2.digitalWrite(1);
9             gpio3.digitalWrite(0);
10            gpio17.digitalWrite(1);
11            gpio27.digitalWrite(0);
12            console.log('UP');
13            break;
14
15        case 'RIGHT':
16            gpio2.digitalWrite(0);
17            gpio3.digitalWrite(0);
18            gpio17.digitalWrite(1);
19            gpio27.digitalWrite(0);
20            console.log('UP');
21            break;
22
23        case 'LEFT':
24            gpio2.digitalWrite(1);
25            gpio3.digitalWrite(0);
26            gpio17.digitalWrite(0);
27            gpio27.digitalWrite(0);
28            console.log('UP');
29            break;
30
31        case 'DOWN':
```

```

32     gpio2.digitalWrite(0);
33     gpio3.digitalWrite(1);
34     gpio17.digitalWrite(0);
35     gpio27.digitalWrite(1);
36     console.log('UP');
37     break;
38
39     case 'STOP':
40     gpio2.digitalWrite(0);
41     gpio3.digitalWrite(0);
42     gpio17.digitalWrite(0);
43     gpio27.digitalWrite(0);
44     console.log('UP');
45     break;
46
47     default:
48     console.log('command not found');
49   }
50 }
51 )

```

Streaming de vídeo

Para realizar la transferencia de vídeo desde el robot hacia el cliente se ha empleado la librería FFmpeg haciendo uso de su herramienta de línea de comandos.

El procedimiento de captura de vídeo y su posterior transmisión es realizado mediante la siguiente instrucción de Ffmpeg:

```

1 ffmpeg -f video4linux2 -i /dev/video0 -s 300x150 -f mjpeg pipe:1
      -b:v 28k -bufsize 28k

```

En los puntos sucesivos analizaremos qué es lo que realiza la instrucción anterior y por qué resulta clave en todo el proceso de difusión, comprendido desde la captura del vídeo hasta su posterior transmisión al usuario que está controlando el robot.

Nada prodríamos transmitir si no disponemos inicialmente de los datos que queremos difundir. De ahí que inicialmente debamos realizar la captura de las diferentes imágenes a partir de la cámara USB conectada a la Raspberry Pi. Para ello se utiliza la API de captura de vídeo video4linux2³ (o simplemente v4l2) la cual dispone las bibliotecas de Ffmpeg. Tan solo debemos especificar el dispositivo de captura.

El nombre del dispositivo de captura es un nodo de dispositivo de archivo, por lo general los sistemas Linux tienden a crear automáticamente estos nodos cuando el dispositivo está conectado al sistema, y tiene un nombre del tipo /dev/videoN, donde N es un

³ Video4Linux o V4L es una API de captura de video para Linux. Muchas webcams USB, sintonizadoras de tv, y otros periféricos son soportados. Video4Linux está integrado con el núcleo Linux. V4L está en su segunda versión (V4L2). El V4L original fue incluido en el ciclo 2.1.X de desarrollo del núcleo Linux. Video4Linux2 arregla algunos fallos y apareció en los núcleos 2.5.X.

número asociado al dispositivo.

Para proceder a la captura de las imágenes nos bastaría con introducir el siguiente comando:

```
1  ffmpeg -f video4linux2 -i /dev/video0 -s 300x150 -f mjpeg
   video_out.mpeg
```

Ahora bien, el comando anterior toma las imágenes de la cámara y las almacena en el archivo especificado *video_out.mpeg* especificando una resolución de salida de 300x150 píxeles empleando la opción *-s*. Pero nosotros no deseamos exactamente ese comportamiento. Debemos canalizar esos datos capturados hacia el socket creado con la finalidad de ir transmitiendo los diferentes frames y no almacenándolos en disco tal y como realiza la instrucción anterior.

Para resolver este problema podemos emplear el sistema de tuberías que implementan los sistemas UNIX⁴.

En cualquier sistema Unix se puede hacer que la salida de una determinada orden sea la entrada estándar de otra, lo que le confiere a las órdenes Unix una enorme potencia. Para realizar dicha “canalización” debemos utilizar las siguientes opciones:

```
1  pipe:1 -b:v 28k -bufsize 28k
```

Con la opción *pipe:1* accedemos al protocolo pipe de UNIX, el cual lee y escribe de las *tuberías* UNIX siendo el número 1 la tubería correspondiente a la salida estándar *stdout* (0 para *stdin* y 2 para *stderr*), la cual podría ser omitida puesto que es la salida por defecto.

La opción *-b:v 28k* establece la tasa de transferencia, en nuestro caso una tasa de 28 kbit/s.

La opción *-bufsize 28k* establece un tamaño de buffer⁵ de 28 kbits.

A continuación mostramos el código de transmisión de vídeo al completo junto con la captura de los diferentes eventos activados cuando se produce la salida de datos por cada una de las salidas estándar:

```
1
2  function startStreaming(socket) {
3      //ffmpeg -f video4linux2 -i /dev/video0 -s 300x150 -f mjpeg pipe:1
4      -b:v 28k -bufsize 28k
```

⁴ Una tubería (pipe, cauce o '|') consiste en una cadena de procesos conectados de forma tal que la salida de cada elemento de la cadena es la entrada del próximo. Permiten la comunicación y sincronización entre procesos. Es común el uso de buffer de datos entre elementos consecutivos.

⁵ Un buffer de datos es un espacio de la memoria en un disco o en un instrumento digital reservado para el almacenamiento temporal de información digital, mientras que está esperando ser procesada.

```

5   if (running_camera == false){
6     console.log('Starting streaming....');
7     var args = ["-f", "video4linux2", "-i", "/dev/video0", "-s",
8       "300x150", "-f", "mpeg", "pipe:1", "-b:v 28k", "-bufsize 28k"]
9     ffmpeg_command = require('child_process').spawn("ffmpeg", args);
10    running_camera = true
11  }
12
13  ffmpeg_command.on('error', function(err, stdout, stderr) {
14    console.log("ffmpeg stdout:\n" + stdout);
15    console.log("ffmpeg stderr:\n" + stderr);
16    running_camera = false
17  });
18
19  ffmpeg_command.on('close', function (code) {
20    console.log('ffmpeg exited' + code );
21    running_camera = false
22  });
23
24
25  ffmpeg_command.stderr.on('data', function (data) {
26    //console.log('stderr: ' + data);
27  });
28
29  ffmpeg_command.on('end', function() {
30    console.log('Finished');
31    running_camera = false
32  });
33
34  ffmpeg_command.stdout.on('data', function (data) {
35    //console.log('stdout: ' + data);
36    var frame = new Buffer(data).toString('base64');
37    socket.emit('canvas',frame);
38  });
39}

```

Código de ejemplo completo

Finalmente se muestra el código completo para el robot de pruebas desarrollado. Dicho código puede emplearse como guía de referencia o plantilla para futuros proyectos con la idea de integrarlos en la aplicación RobotUI.

```

1 var io_client = require('./node_modules/socket.io-client');
2 var sails_client = require('./node_modules/sails.io.js');
3 var io_server = sails_client(io_client);
4 io_server.sails.url = 'http://46.101.102.33:80';
5 io_server.socket.get('/robot/changetoonline/', {robot:
6   '59188631c8e94ba54f7a4bdc', online: true});
7
8 // Inicia servidor socket.io en el puerto 8085.
9 var io =io_client.listen(8085, { log: false });

```

```
10 // Carga de módulos necesarios.
11 var ffmpeg_command, running_camera = false, child_process =
12     require('child_process');
13
14 var Gpio = require('pigpio').Gpio;
15 // Pines utilizados. Motores izquierdos: 2 y 3, motores derechos: 17 y
16     27
17 var gpio2 = new Gpio(2, {mode: Gpio.OUTPUT}),
18     gpio3 = new Gpio(3, {mode: Gpio.OUTPUT}),
19     gpio17 = new Gpio(17, {mode: Gpio.OUTPUT}),
20     gpio27 = new Gpio(27, {mode: Gpio.OUTPUT});
21
22
23 console.log('Esperando conexión...');

24 var sockets = {};
25
26 io.sockets.on('connection', function (socket)
27 {
28
29     sockets[socket.id] = socket;
30     console.log("Clientes totales conectados: ",
31         Object.keys(sockets).length);
32
33     socket.on('disconnect', function() {
34         console.log('Adios !');
35         //stopStreaming(socket);
36     });
37
38     socket.on('start-stream', function() {
39         startStreaming(socket);
40     });
41
42     socket.emit('robotmsg', {msg: "Bienvenido !!!"});
43     console.log('emitiendo: ' + "Bienvenido !!!");
44
45     socket.on('action', function (data){
46
47         console.log('Comando recibido: ' + data);
48
49         switch(data) {
50             case 'UP':
51                 gpio2.digitalWrite(1);
52                 gpio3.digitalWrite(0);
53                 gpio17.digitalWrite(1);
54                 gpio27.digitalWrite(0);
55                 console.log('UP');
56                 break;
57
58             case 'RIGHT':
59                 gpio2.digitalWrite(0);
60                 gpio3.digitalWrite(0);
61                 gpio17.digitalWrite(1);
62                 gpio27.digitalWrite(0);
```

```

62         console.log('UP');
63         break;
64
65     case 'LEFT':
66         gpio2.digitalWrite(1);
67         gpio3.digitalWrite(0);
68         gpio17.digitalWrite(0);
69         gpio27.digitalWrite(0);
70         console.log('UP');
71         break;
72
73     case 'DOWN':
74         gpio2.digitalWrite(0);
75         gpio3.digitalWrite(1);
76         gpio17.digitalWrite(0);
77         gpio27.digitalWrite(1);
78         console.log('UP');
79         break;
80
81     case 'STOP':
82         gpio2.digitalWrite(0);
83         gpio3.digitalWrite(0);
84         gpio17.digitalWrite(0);
85         gpio27.digitalWrite(0);
86         console.log('UP');
87         break;
88
89     default:
90         console.log('command not found');
91     }
92
93 })
94 });
95
96 function stopStreaming(socket) {
97     delete sockets[socket.id];
98     // no more sockets, kill the stream
99     if (Object.keys(sockets).length == 0) {
100         if (ffmpeg_command){
101             ffmpeg_command.kill();
102             running_camera = false;
103             console.log('Stop streaming');
104         }
105     }
106 }
107
108 function startStreaming(socket) {
109     //ffmpeg -f video4linux2 -i /dev/video0 -s 300x150 -f mjpeg pipe:1
110     // -b:v 28k -bufsize 28k
111     if (running_camera == false){
112         console.log('Starting streaming....');
113         var args = ["-f", "video4linux2", "-i", "/dev/video0", "-s",
114             "300x150", "-f", "mjpeg", "pipe:1", "-b:v 28k", "-bufsize 28k"]
115         ffmpeg_command = child_process.spawn("ffmpeg", args);

```

```
115     running_camera = true
116 }
117
118 ffmpeg_command.on('error', function(err, stdout, stderr) {
119     console.log("ffmpeg stdout:\n" + stdout);
120     console.log("ffmpeg stderr:\n" + stderr);
121     running_camera = false
122 });
123
124
125 ffmpeg_command.on('close', function (code) {
126     console.log('ffmpeg exited' + code );
127     running_camera = false
128 });
129
130
131 ffmpeg_command.stderr.on('data', function (data) {
132     //console.log('stderr: ' + data);
133 });
134
135 ffmpeg_command.on('end', function() {
136     console.log('Fin');
137     running_camera = false
138 });
139
140 ffmpeg_command.stdout.on('data', function (data) {
141     //console.log('stdout: ' + data);
142     var frame = new Buffer(data).toString('base64');
143     socket.emit('canvas',frame);
144 });
145
146 }
```

Para la ejecución del código introducimos el siguiente comando:

```
1 sudo node raspberry.js
```

Siendo *raspberry.js* el nombre del archivo que contiene nuestro código.

Capítulo 8

Organización temporal

La planificación general del proyecto siguiendo un modelo SCRUM, empleando para ello el panel de tareas Trello; un gestor de proyectos que permite aplicar una metodología de desarrollo ágil.

El panel se encuentra accesible en el siguiente enlace: <https://trello.com/b/SpIbFI7k/robotui>.

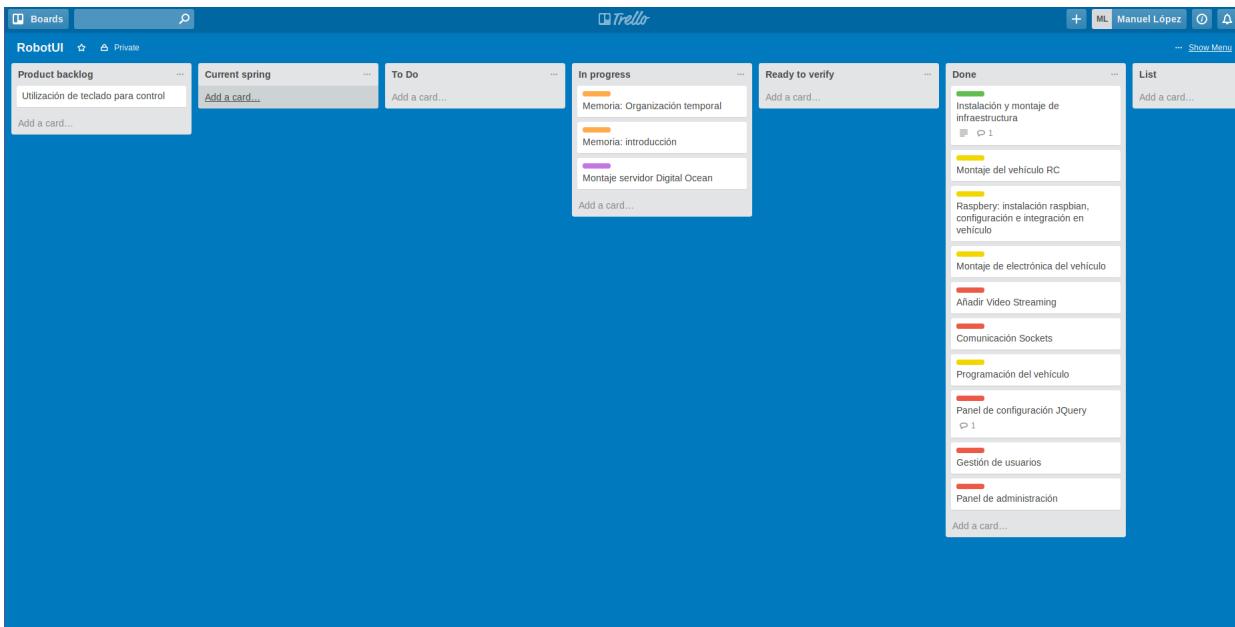


Figura 8.1: Panel de actividades - Trello

Cabe destacar que para el desarrollo de RobotUI ha sido necesario emplear varias herramientas, utilidades y bibliotecas. Algunas de ellas ya habían sido utilizadas en ciertas ocasiones, bien sea en el ámbito estudiantil o profesional. Sin embargo, otras han requerido un periodo de formación previo, en el que se han adquirido los conocimientos necesarios para poder desarrollar el presente proyecto.

La mayor parte del proceso de investigación fue dedicado al estudio de las diferentes tecnologías existentes para la programación web en tiempo real. Todo ello ha implicado

un esfuerzo bastante considerable en el uso, aprendizaje e investigación de las diferentes tecnologías existentes y comprobar su potencial.

Una vez determinadas las diferentes herramientas a utilizar se comenzó con la implementación de la aplicación, siendo seleccionada como herramienta principal el framework Sails.js. Un framework MVC en tiempo real para Node.js, el cual está muy enfocado al propósito de este proyecto.

Finalmente, tras la necesidad de probar la aplicación en un entorno real, se optó por elaborar un robot de pruebas, un pequeño vehículo elaborado con una Raspberry Pi con la finalidad de probar, testear y hacer demostraciones de la aplicación.

La figura 8.2 y 8.3 muestran una visión de las diferentes tareas desarrolladas para la elaboración del proyecto junto con la descomposición de cada una de ellas:

Los puntos más importantes del proyecto se han dividido en hitos, así como entregas que se definieron en cada reunión que se realizaba con el director del proyecto. También se ha definido una planificación temporal del desarrollo del proyecto mediante un diagrama de Gantt con la duración de las tareas recogidas en el panel de actividades de Trello.

CAPÍTULO 8. ORGANIZACIÓN TEMPORAL

WBS	Nombre	Trabajo
1	▼ RobotUI	135d 3h
1.1	Conocimiento del proyecto	22d
1.2	Planificación y estudio del proyecto	5d
1.3	▼ Análisis de herramientas existentes	17d
1.3.1	Estudio de Node.js	5d
1.3.2	Estudio de Sails.js	5d
1.3.3	Estudio de Socket.io	2d
1.3.4	Códigos y pruebas	3d
1.3.5	Estudio de MongoDB	2d
1.4	▼ Definición de requisitos	5d 6h
1.4.1	Arquitectura BBDD análisis	1d
1.4.2	Diseño de diagrama UML	2d
1.4.3	Requisitos funcionales	1d
1.4.4	Requisitos no funcionales	6h 45min
1.4.5	Reunión de planificación con director del proyecto	1d
1.5	▼ Desarrollo aplicación	30d 4h
1.5.1	Sistema de autenticación - registro	2d
1.5.2	Alta de dispositivos robóticos	2d
1.5.3	Internacionalización	1d
1.5.4	Módulo de mensajes entre usuarios	3d
1.5.5	Implementación de políticas de permisos	4d
1.5.6	Reunión de seguimiento con director del proyecto	3d 3h
1.5.7	▼ Elementos de la interfaz	5d 5h
1.5.7.1	Acciones	1d 1h
1.5.7.2	Sliders	1d
1.5.7.3	Labels	1d 7h
1.5.7.4	Video	1d 4h
1.5.8	▼ Personalización de la interfaz	9d 4h
1.5.8.1	Personalización de acciones	1d 4h
1.5.8.2	Personalización de sliders	2d
1.5.8.3	Personalización de labels	2d
1.5.8.4	Edición de la interfaz	4d
1.6	▼ Módulo de comunicaciones	18d
1.6.1	▼ Conexión cliente - robot	4d 7h
1.6.1.1	Envío de órdenes al robot	2d
1.6.1.2	Captura de vídeo del robot	2d 7h
1.6.2	▼ Conexión cliente - servidor	3d
1.6.2.1	Envío de vídeo capturado al servidor	1d 4h
1.6.2.2	Envío de órdenes lanzadas al servidor	1d 4h
1.6.3	Desarrollo de tests funcionales	8d
1.6.4	Frontend - detalles de estilos	2d

Figura 8.2: Descomposición de las tareas implicadas en el desarrollo del proyecto (Primera Parte).

1.7	▼ Conexión servidor - cliente	4d
1.7.1	Difusión de vídeo a espectadores	2d
1.7.2	Difusión de comandos a espectadores	2d
1.8	▼ Montaje del vehículo	3d
1.8.1	Búsqueda de elementos hardware	2d
1.8.2	Montaje y conexiones	1d
1.9	▼ Programación del vehículo	4d 7h
1.9.1	Gpio	2d
1.9.2	Video streaming	1d 3h
1.9.3	Fase de pruebas	1d 4h
1.10	Despliegue de la aplicación en producción	1d
1.11	▼ Documentación	23d
1.11.1	Memoria	19d
1.11.2	Resumen	2d
1.11.3	Presentación	2d
1.12	Reunión de seguimiento con el director del proyecto	1d

Figura 8.3: Descomposición de las tareas implicadas en el desarrollo del proyecto (Segunda parte).

8.1. Planificación temporal de tareas

A continuación se definirán los diferentes hitos que componen el diagrama de Gantt divididos en las diferentes subtareas principales de cada uno de ellos:

8.1.1. Hito 1: Planificación y análisis

En esta primera etapa de desarrollo del proyecto final de carrera se realizaron los estudios previos necesarios para abordar cualquier proyecto de cierta envergadura.

Este hito se descompone en las siguientes tareas principales:

1. Planificación y estudio del proyecto. En esta fase se centró en la elaboración de un documento, a modo borrador, con la idea a desarrollar, objetivos del proyecto y su alcance.
2. Análisis de herramientas existentes, de las tecnologías a implementar, la arquitectura del sistema, las tecnologías de BD, visualización, para la selección de las herramientas más adecuadas para afrontar el desarrollo con garantías y no sea necesaria una “vuelta atrás” por necesidad imperiosa de cambio de tecnología. En definitiva se buscaba una herramienta libre, con un respaldo de una comunidad importante y que resuelva la problemática o necesidad de trabajar con eventos en tiempo real.

8.1.2. Hito 2: Definición de requisitos

Este segundo hito queda dividido en las siguientes etapas:

1. Elaboración de un documento formal con la propuesta de proyecto definiendo sus objetivos y alcance, para la aprobación por parte del director del proyecto.
2. Se definen las clases del sistema, el modelo de la base de datos y la definición de requisitos funcionales y no funcionales.

8.1.3. Hito 3: Comienzo de desarrollo de la aplicación

En este tercer hito, uno de los de mayor magnitud, se comienza con el desarrollo de la aplicación, el cual queda dividido en las siguientes módulos.

1. Implementación del módulo *Usuario* con las funciones de registro, autenticación, configuraciones de idioma, entre otras.
2. Implementación del módulo central de la aplicación *Robot*
3. Implementación del módulo de mensajes.
4. Implementación del módulo de políticas de permisos.

8.1.4. Hito 4: Desarrollo de la aplicación, módulo componentes

En este hito, se trabaja en el desarrollo de los diferentes elementos que compondrán la interfaz de control. Tanto su parte de configuración y personalización. Cada elemento integrante de la interfaz lo denominamos componentes.

En definitiva, el hito queda dividido en el desarrollo de los siguientes componentes:

1. *Acciones*
2. *Sliders*
3. *Labels*
4. *Vídeo*

8.1.5. Hito 5: Desarrollo de la aplicación, módulo interfaz

En este hito, se realiza la elaboración de la interfaz de control. Tanto su parte de configuración y personalización como la de visualización para su control. Este hito quedó dividido en las siguientes etapas:

1. Implementación de la funcionalidad de configuración.
2. Implementación de la funcionalidad de control.

8.1.6. Hito 6: Desarrollo del módulo de comunicaciones

El presente hito, clasificado como crítico debido a su importancia. Debía realizar la integración de todos los módulos anteriores y dotarlos de la funcionalidad principal para la que han sido diseñados. Estar interconectados entre sí además de la elaboración de los test funcionales.

1. Implementación de la conexión cliente - robot.
2. Implementación de la conexión cliente - servidor.

8.1.7. Hito 7: Construcción del vehículo de pruebas

Se realiza la construcción y montaje e instalación software del vehículo de pruebas y comprobación de las diferentes conexiones.

8.1.8. Hito 8: Programación del vehículo de pruebas

Se realiza la programación del vehículo y se realizan pruebas de todo el conjunto junto con las correcciones necesarias.

8.1.9. Hito 9: Documentación

Se finaliza la memoria para la revisión por parte del director del proyecto y su posterior impresión. Se prepara la presentación para la defensa ante tribunal.

8.2. Diagrama de Gantt

A continuación se muestra el diagrama de Gantt donde quedan reflejados los diferentes hitos descritos en el punto anterior.

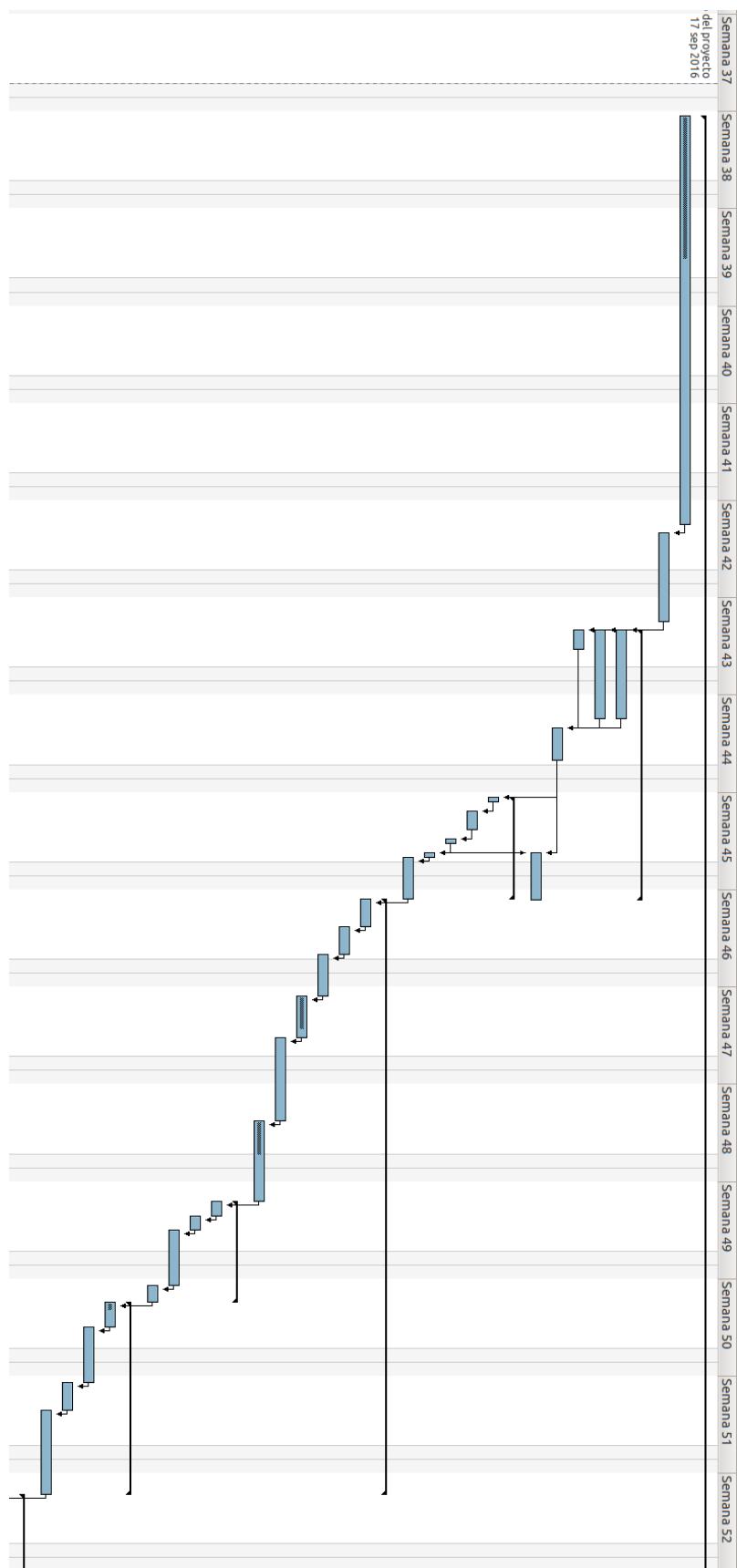


Figura 8.4: Diagrama de Gantt 1. Desarrollo del proyecto.

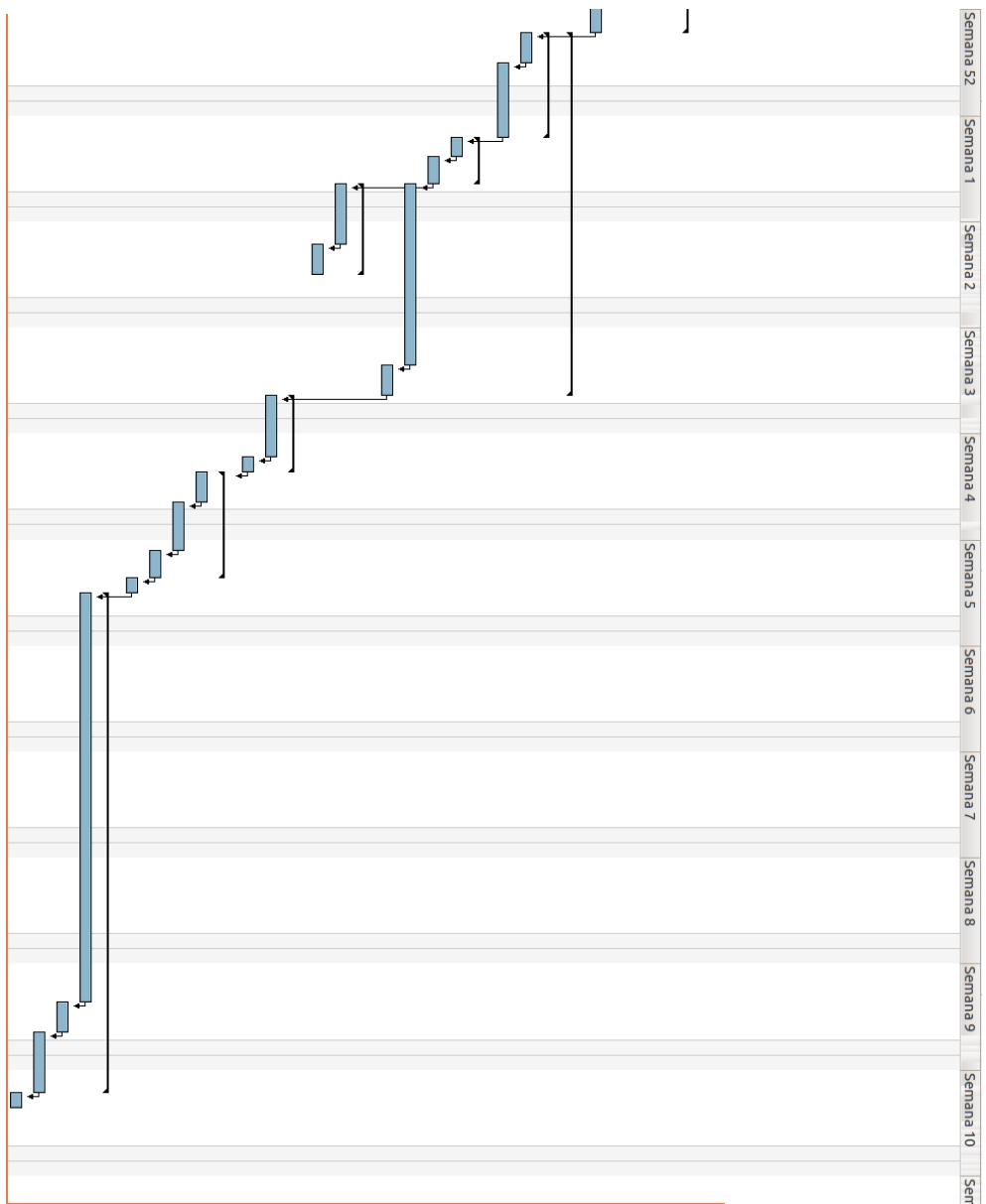


Figura 8.5: Diagrama de Gantt 2. Desarrollo del proyecto.

Capítulo 9

Guía de Usuario

La aplicación RobotUI ha sido creada por Manuel López Urbina para el proyecto fin de carrera de la titulación de Ingeniería Informática de la Universidad de Cádiz.

Resulta de vital importancia consultar esta guía antes y/o durante la utilización de los diferentes elementos tanto hardware, robot de pruebas desarrollado, como software del presente proyecto ya que le proporcionará una guía paso a paso en el manejo correcto de la aplicación.

La resolución recomendada para la aplicación debe ser superior o igual a 1024x768 (Estándar XGA), aunque es adaptable a cualquier resolución debido a su diseño web responsive.

RobotUI ha sido elaborado con un claro propósito; el de proporcionar a los usuarios un medio donde compartir sus dispositivos robóticos con el resto de usuarios. Esto es posible gracias a una serie de herramientas desarrolladas para que, sin necesidad de tener grandes conocimientos en programación, puedan configurar un entorno para el manejo de sus proyectos robóticos y tener posibilidad de compartir sus dispositivos y experiencias con el resto de usuarios.

La particularidad de RobotUI es que el usuario propietario del robot tiene la posibilidad de permitir el manejo de sus dispositivos robóticos al resto de usuarios que él mismo considere de una manera controlada o, por otra parte, permitir que otros usuarios visualicen, como si de espectadores se tratase, el control que un determinado usuario realiza de un determinado robot. Todo ello en tiempo real.

Por tanto, tras esta breve introducción en el ámbito de la aplicación, en este manual se describen los diferentes pasos a realizar para configurar sus dispositivos correctamente en el sistema y abrirlo a toda una comunidad de usuarios. Además de tener abierto el acceso a otros muchos dispositivos de otras personas.

9.0.1. Objetivo de esta guía

Esta guía tiene como objetivo proporcionar al usuario un soporte de ayuda e iniciación a la utilización de RobotUI.

Esta sección comprende:

- Introducción.
- Guía de acceso al código fuente de la aplicación.
- Guía de uso de la aplicación.
- Guía para la puesta en marcha y programación de un robot.

9.0.2. Dirigido a

Esta guía esta dirigida al usuario final del proyecto RobotUI. Tiene la finalidad de proporcionar una guía descriptiva de los procedimientos de creación, configuración y utilización de los diferentes dispositivos robóticos en sus dos modalidades disponibles, la de control y la de visualización.

9.0.3. Obtener RobotUI

El código fuente junto con la presente memoria se encuentra disponible en el repositorio GitHub en el enlace <https://github.com/lopi87/SAILS-RobotUI> o usando la herramienta Git, escribiendo en la consola el siguiente comando:

```
1 git clone git@github.com:lopi87/SAILS-RobotUI.git
```

9.1. Uso de RobotUI

9.1.1. Acceso a la aplicación

Para acceder a la aplicación, el usuario deberá accedes al siguiente enlace: www.robotui.com.

Al acceder podrá ver el portal de entrada a la aplicación. En él puede acceder al resto de funcionalidades identificándose con sus credenciales y acceder a los formularios de registro de usuario.

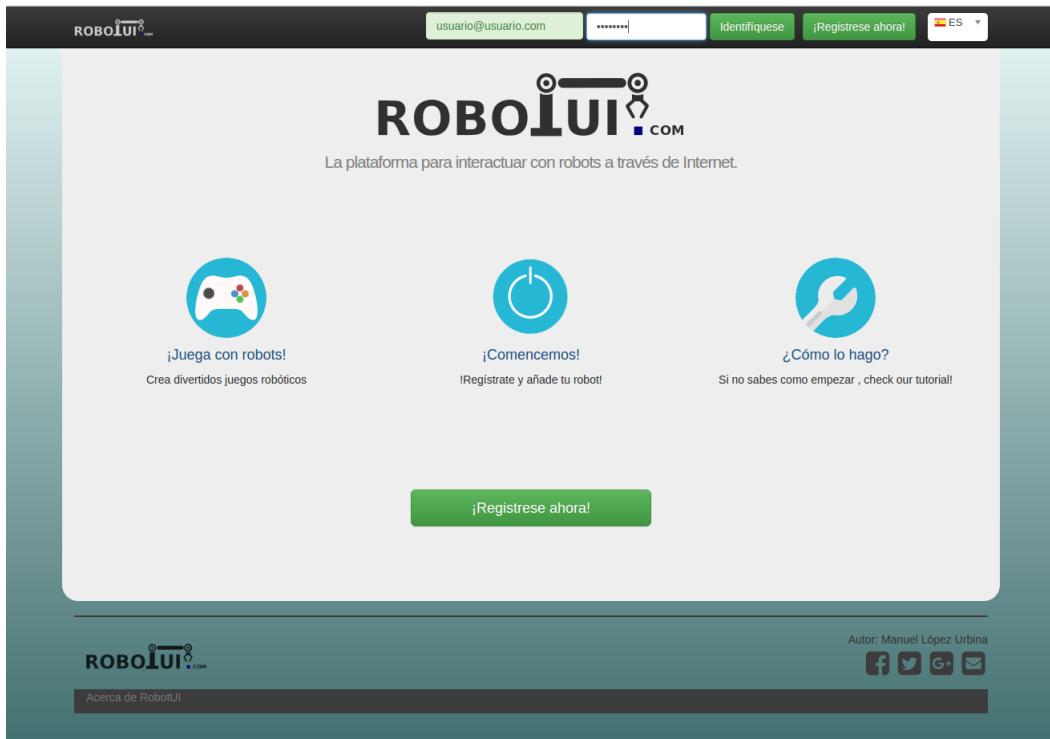


Figura 9.1: Página principal RobotUI.

9.1.2. Registro de usuario

Una vez en la página principal, figura 9.18, podemos observar la barra superior, la cual posee dos estados, la de usuario sin identificar y la de usuario identificado.



Figura 9.2: Vista de la barra superior en modalidad usuario no identificado en el sistema.



Figura 9.3: Vista de la barra superior en modalidad usuario identificado en el sistema.

Para iniciar el proceso de registro de usuario hacemos clic sobre el botón de la barra superior **¡Registrese ahora!** mediante el cual accedemos al formulario 9.4.

The form is titled 'Crear cuenta'. It includes the following fields:

- Nombre * :** Input field containing 'Nombre'.
- Contraseña * :** Input field containing 'Contraseña'.
- Email * :** Input field containing 'Email'.
- Confirmacion de contraseña * :** Input field containing 'Confirmacion de contraseña'.
- Por favor proporcione un archivo menor que 42 kb:** Input field with a placeholder 'Seleccionar un archivo'.
- Idioma * :** A dropdown menu showing 'ES'.

A large blue button labeled 'Crear cuenta' is centered at the bottom.

Figura 9.4: Formulario de registro de usuario.

Para proceder con el registro necesitaremos los siguientes datos:

1. Nombre: nombre de usuario.
2. Email: dirección de correo electrónico utilizada para acceder al sistema.
3. Contraseña: deberá proporcionar una contraseña para poder acceder al sistema.
4. Confirmación de contraseña: campo para comprobar que la contraseña es la deseada por el usuario.
5. Imagen: imagen avatar del usuario.
6. Idioma: selección del idioma por defecto del usuario.

Una vez introducidos los datos correctamente podremos hacer clic en *crear cuenta*. Si los datos introducidos son correctos dispondremos de un nuevo usuario en la aplicación redirigiéndonos a la página de información del nuevo usuario [9.5](#).

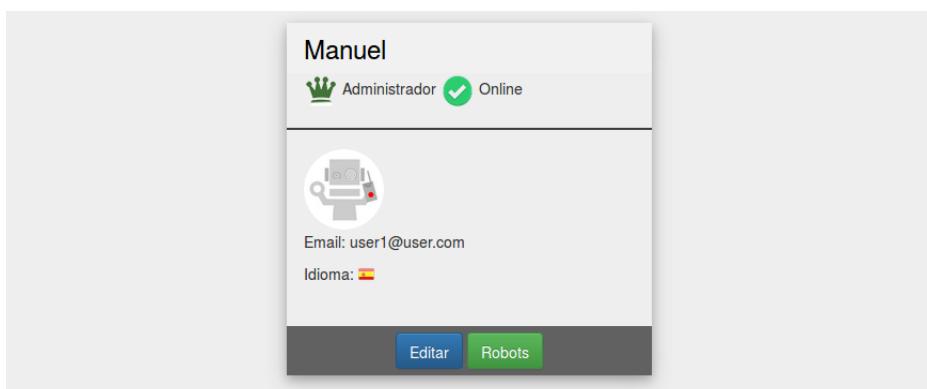


Figura 9.5: Información de un usuario.

9.1.3. Ingreso al sistema

Una vez registrados, procedimiento descrito en [9.1.2](#), podremos acceder al sistema mediante la introducción de nuestro usuario y contraseña en los campos disponibles en la barra superior localizado en la aplicación y haciendo clic en  localizados en la barra superior.

9.1.4. Registro de un Robot

Para el iniciar el proceso de creación de un robot haremos clic en *Robots → Mis robots* del menú superior accediendo al formulario de creación, figura [9.8](#).

En este caso registraremos nuestro robot de pruebas descrito en el capítulo [7](#). Para ello introducimos los siguientes datos:

1. **Nombre:** nombre de nuestro robot.
2. **Dirección IP:** dirección donde se encontrará accesible nuestro robot.
3. **Puerto:** puerto donde se encontrará nuestro robot a la escucha de conexiones.
4. **Descripción:** campo opcional en el que podemos añadir una descripción de nuestro robot y que será visible al resto de usuarios.
5. **Imagen:** campo opcional en el que podremos subir una imagen de nuestro robot.
6. **Usuarios espectadores:** selector en el que podemos especificar qué usuarios del sistema tendrán permisos para visualizar el funcionamiento de nuestro robot.
7. **Usuarios controladores:** selector para especificar qué usuarios podrán tomar control de nuestro robot.
8. **Público:** checkboxes para indicar si por el contrario, el robot que abierto a todos los usuarios para su control o abierto para su visualización si marcamos el primero o el segundo checkbox respectivamente y haciendo por tanto inválidos los parámetros de los selectores anteriores.

Una vez introducidos los datos para el nuevo usuario pulsamos en  .

Robot

Nombre *:

Por favor proporcione un archivo menor que 42 kb:

Cambiar Eliminar

Dirección IP*:

Usuarios espectadores:

Puerto*:

Usuarios controladores:

Descripción:

Público *:

Control: Ver:

Crear

Figura 9.6: Formulario de creación de un robot.

Si los campos introducidos son correctos la aplicación nos redireccionará a la página informativa de nuestro nuevo robot:

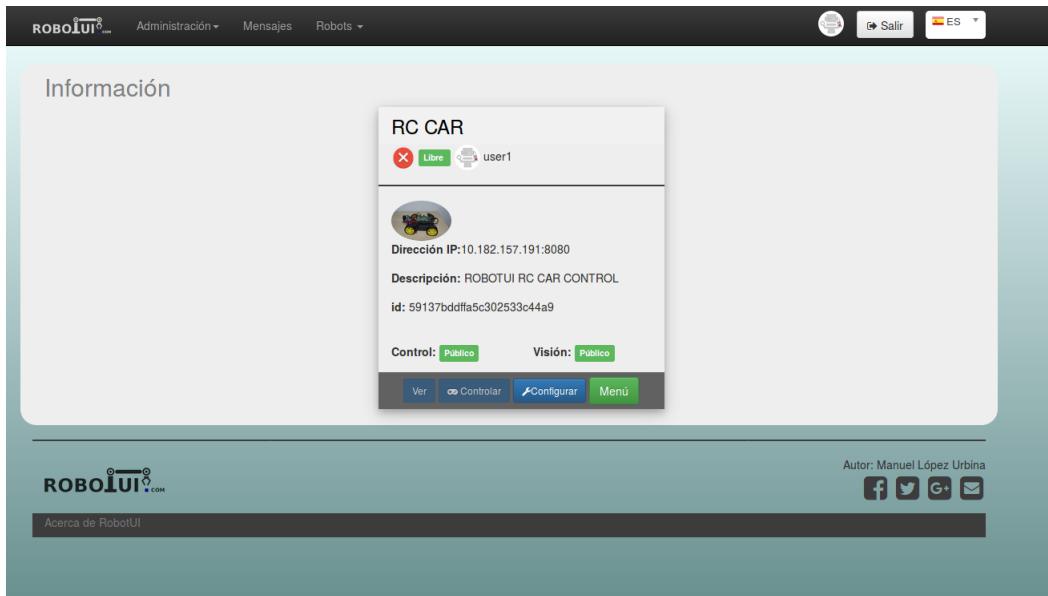


Figura 9.7: Vista informativa de un robot.

En la ventana informativa de un robot, figura 9.7, podemos apreciar una serie de elementos representativos de los diferentes estados en los que se puede encontrar el dis-

positivo. La tabla 9.1 describe cada uno de los elementos:

Elemento	Significado
	Imagen indicativa de estado <i>offline</i> . El robot se encuentra sin conexión.
	Imagen indicativa de estado <i>online</i> . El robot se encuentra conectado, reemplazará a la imagen de estado <i>offline</i> .
Control: Público	Imagen representativa de robot de control público.
Control: Privado	Imagen representativa de robot de control privado.
Visión: Público	Imagen representativa de robot de seguimiento pública.
Seguimiento: Privado	Imagen representativa de robot de seguimiento privado.
Ocupado	Imagen representativa de robot en uso.

Tabla 9.1: Elementos representativos de los estados de un robot junto con su descripción.

9.1.5. Gestión de permisos

Como hemos mencionado en el procedimiento de registro de un robot, éste se puede crear con una serie de permisos. Existen dos modalidades en RobotUI, como sabemos, la de visualización o seguimiento y la de control. Funcionalidades que podemos hacer públicas o privadas, y, en el caso de que optemos por alguna modalidad privada, definir para qué usuarios estará disponible.

La gestión y visualización de los permisos configurados estarán disponibles en todo momento en la ventana de edición de un robot. En ella podremos visualizar los diferentes usuarios añadidos con los permisos de cada uno de ellos. Podremos eliminar usuarios o añadir otros nuevos estableciendo el acceso a alguna de las modalidades existentes como pública o privada.

Si el robot es establecido como público en alguna de sus dos modalidades (control o seguimiento) entonces se ignorará toda la configuración de usuarios establecida para esa modalidad aunque permanecerán registradas en el sistema por si se desea volver al modo privado tan solo haciendo clic en el checkbox correspondiente.

Permisos

Público * :

Control: Seguimiento:

Usuario	Nombre	Control	Seguimiento	Acciones
	user2	✓	✓	Eliminar
	user3	✓	✗	Eliminar

Usuarios * :

 user2
 user4

Añadir

Figura 9.8: Panel de configuración de los permisos de un robot.

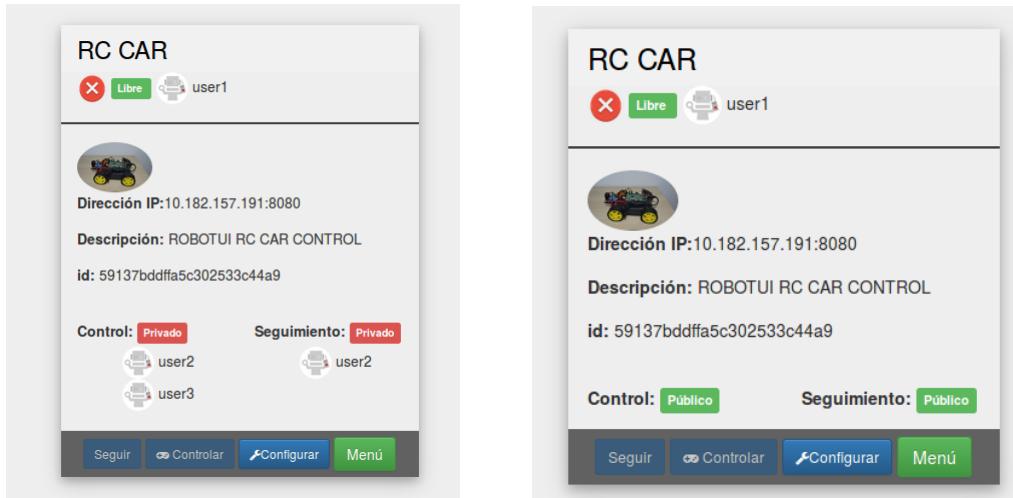


Figura 9.9: Panel informativo de un robot privado frente a uno público.

9.1.6. Configuración de la interfaz

Una vez creado nuestro robot en el sistema ya podremos configurar su interfaz. En la figura 9.10 podemos ver el canvas sobre el que iremos añadiendo los diferentes elementos que conformarán nuestro panel de control.

El diseño resultante de esta ventana servirá tanto para panel de control del robot como ventana de seguimiento a los usuarios que sigan el funcionamiento del dispositivo.

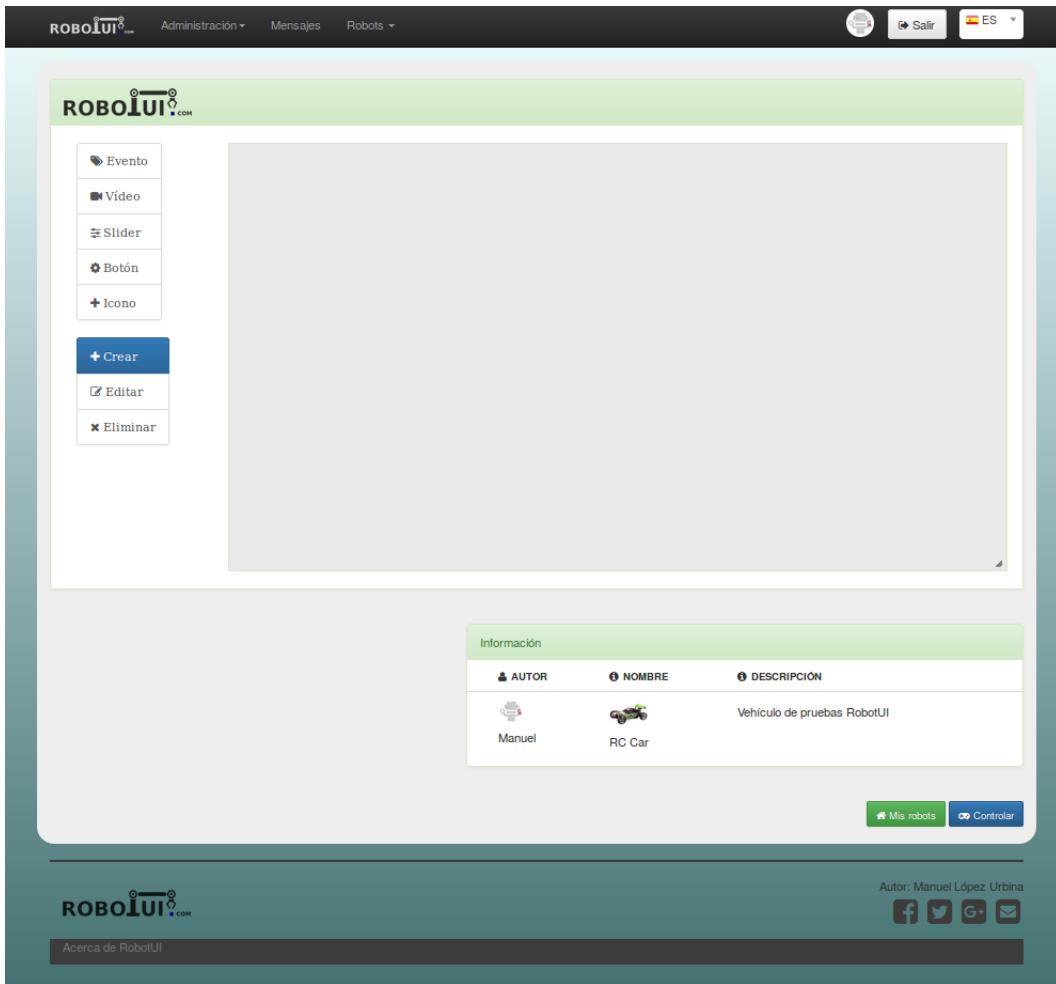


Figura 9.10: Panel de configuración de la interfaz sin elementos añadidos.

La figura 9.10 correspondiente al panel de configuración de una interfaz queda dividida en los siguientes elementos:

1. Panel de elementos disponibles (Evento, Vídeo, Slider, Botón e Icono).
2. Panel de acciones sobre elementos (Crear, editar y eliminar).
3. Panel canvas sobre el que posicionar los elementos.
4. Panel informativo de las características del robot.

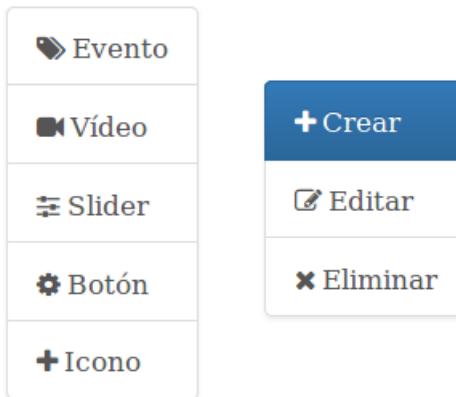


Figura 9.11: Panel de elemento (izquierda) y panel de acciones (derecha) para la configuración de la interfaz.

En los siguientes subapartados se describen los procedimientos para la creación, edición y borrados de elementos:

Nuevo elemento

Para la creación de elementos debemos tener seleccionado el modo creación en el panel de acciones y pulsar sobre el tipo de nuevo elemento deseado. La siguiente tabla muestra las diferentes combinaciones:

Botón	Modo	Acción
Botón	+ Crear	Apertura del formulario para la creación de un botón.
Vídeo	+ Crear	Apertura del formulario para la creación de un canvas de vídeo.
Slider	+ Crear	Apertura del formulario para la creación de un slider.
Evento	+ Crear	Apertura del formulario para la creación de una etiqueta.
Icono	+ Crear	Apertura del formulario para la subida de un ícono.

Tabla 9.2: Elementos añadibles a una interfaz.

Una vez completado el formulario y creado el nuevo elemento tan solo debemos arrastrar el nuevo elemento a la posición deseada.

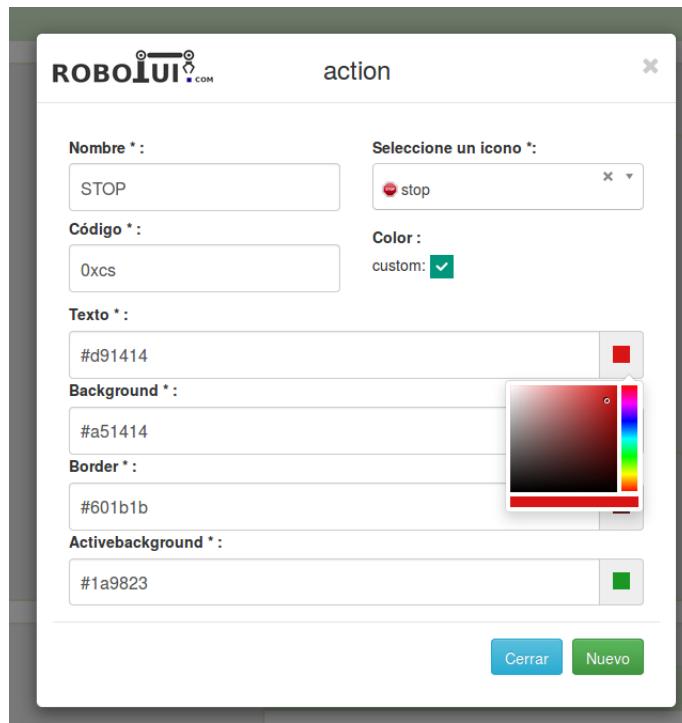


Figura 9.12: Formulario para la creación de un botón.

Del mismo modo para la creación del resto de tipos de elementos.

Editar elemento

Para la edición de elementos pulsaremos seleccionaremos el modo edición y posteriormente haremos clic sobre el elemento que deseemos editar.

Tras ello se abrirá el formulario correspondiente, modificamos los parámetros deseados y pulsamos guardar.

Eliminar elemento

Del mismo modo que en la edición de elementos, para borrar un elemento seleccionaremos el modo eliminar del panel de acciones y posteriormente haremos clic sobre el elemento que deseemos borrar.

9.1.7. Ejemplo

La figura ?? muestra una posible interfaz de control para un dispositivo robótico.



Figura 9.13: Vista tipo de una interfaz configurada.

9.1.8. Programa tu robot

NOTA: Esta guía comprende una serie de pasos para realizar la programación de un dispositivo robótico para la aplicación RobotUI. En este caso particular, el robot empleado posee como base una placa Raspberry Pi, la cual emplea para la conexión de sensores y motores haciendo uso de su sistema de Entrada/Salio Gpio.

El sistema es compatible con cualquier dispositivo, no solo limitado a las placas Raspberry. La única diferencia radica en que se deberán emplear a nivel de programación las bibliotecas adecuadas para la activación o lectura de las Entradas/Salidas correspondientes al modelo de placa utilizado.

Antes de proceder con la programación de nuestro robot debemos de realizar su creación en la aplicación RobotUI como queda descrito en el punto [9.1.4](#) y tomar nota del identificador único que la aplicación proporciona para dicho robot y que posteriormente necesitaremos.

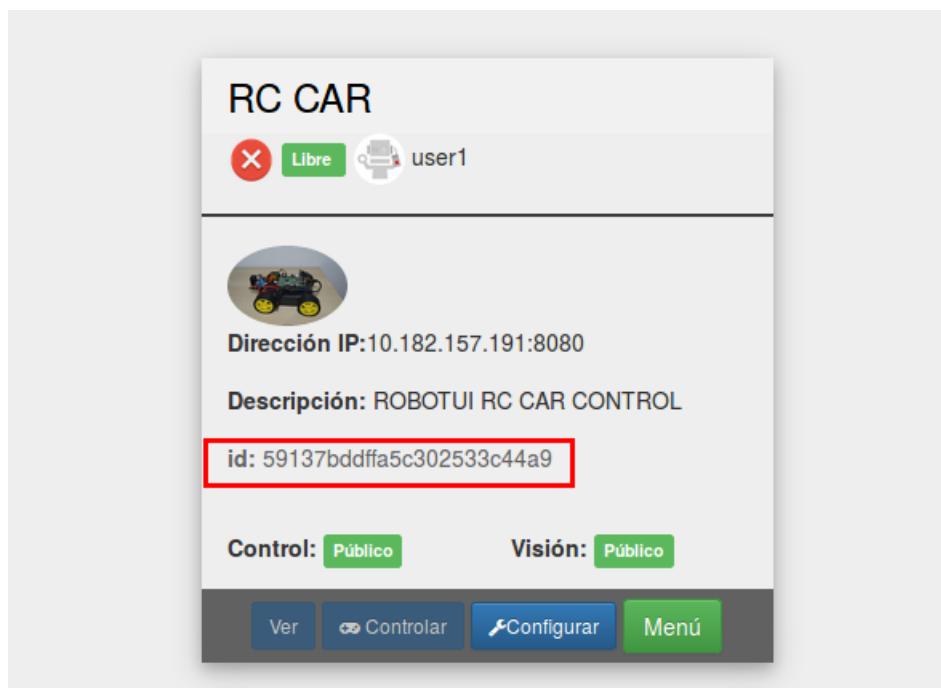


Figura 9.14: Panel informativo de un robot donde se aprecia su identificador.

Una vez creado el robot y configurada su interfaz, añadiendo los diferentes elementos disponibles como acciones, vídeo, etiquetas, etcétera, recogidos en el punto 9.1.6: Configuración de la interfaz. Podemos comenzar con la programación del robot.

Para ello debemos seguir una serie de pasos:

1. Generar una archivo de extensión .js, con el código base mostrado en el punto 9.1.8.
2. Reemplazar en el código la palabra *IDENTIFICADOR* por el identificador único de nuestro robot obtenido en la vista informativa del mismo. Por ejemplo: 59188631c8e94ba54f7a4bdc.
3. Indicar el puerto en el que el robot permanecerá a la escucha. Para ello debemos reemplazar palabra *PUERTO* del código inferior por el puerto deseado. Por ejemplo 8085.
4. El siguiente paso es determinar qué puertos GPIO necesitaremos y si van a ser utilizado en modo Entrada, Salida o Entrada/Salida dentro de la sección *PINES* de la plantilla. En este caso se ha empleando la biblioteca *pigpio* cuya documentación se encuentra disponible en el siguiente enlace: <https://www.npmjs.com/package/pigpio>.
5. Definir el funcionamiento de los diferentes eventos dentro de la sección *EVENTOS*.

código base

```
1 var io_client = require('./node_modules/socket.io-client');
2 var sails_client = require('./node_modules/sails.io.js');
3 var io_server = sails_client(io_client);
4 io_server.sails.url = 'http://46.101.102.33:80';
5 io_server.socket.get('/robot/changetoonline/', {robot:
6   'IDENTIFICADOR', online: true});
7
8 // Inicia servidor socket.io en el puerto PUERTO.
9 var io =io_client.listen( PUERTO, { log: false });
10
11 var ffmpeg_command, running_camera = false, child_process =
12   require('child_process');
13
14 var Gpio = require('pigpio').Gpio;
15
16 // SECCION PINES
17
18 // EJEMPLO:
19 //     var gpio2 = new Gpio(2, {mode: Gpio.OUTPUT}),
20 //     gpio3 = new Gpio(3, {mode: Gpio.OUTPUT}),
21 //     gpio17 = new Gpio(17, {mode: Gpio.OUTPUT}),
22 //     gpio27 = new Gpio(27, {mode: Gpio.OUTPUT});
23
24
25 // FIN SECCION PINES
26
27
28 console.log('Esperando conexión...');

29 var sockets = {};
30
31 io.sockets.on('connection', function (socket)
32 {
33
34   // SECCION EVENTOS
35
36   // FIN SECCION EVENTOS
37 });
38
39
40 function stopStreaming(socket) {
41   delete sockets[socket.id];
42   // no more sockets, kill the stream
43   if (Object.keys(sockets).length == 0) {
44     if (ffmpeg_command){
45       ffmpeg_command.kill();
46       running_camera = false;
47       console.log('Stop streaming');
48     }
49   }
50 }
```

```

52 function startStreaming(socket) {
53   if (running_camera == false){
54     console.log('Starting streaming....');
55     var args = ["-f", "video4linux2", "-i", "/dev/video0", "-s",
56                 "300x150", "-f", "mpeg", "pipe:1", "-b:v 28k", "-bufsize 28k"];
57     ffmpeg_command = child_process.spawn("ffmpeg", args);
58     running_camera = true
59   }
60
61   ffmpeg_command.on('error', function(err, stdout, stderr) {
62     console.log("ffmpeg stdout:\n" + stdout);
63     console.log("ffmpeg stderr:\n" + stderr);
64     running_camera = false
65   });
66
67   ffmpeg_command.on('close', function (code) {
68     console.log('ffmpeg exited' + code );
69     running_camera = false
70   });
71
72
73   ffmpeg_command.stderr.on('data', function (data) {
74     //console.log('stderr: ' + data);
75   });
76
77   ffmpeg_command.on('end', function() {
78     console.log('Fin');
79     running_camera = false
80   });
81
82   ffmpeg_command.stdout.on('data', function (data) {
83     //console.log('stdout: ' + data);
84     var frame = new Buffer(data).toString('base64');
85     socket.emit('canvas',frame);
86   });
87
88 }

```

A continuación mostramos dos posibles eventos para añadir al código base superior a modo orientativo:

El primer fragmento de código se activa al recibir un evento tipo *action* (evento lanzado desde la interfaz al presionar cualquier botón generado por el usuario), captura el comando recibido, y si es igual a *UP*, entonces habilita el pin *gpio1* con el valor 1. Pin inicializado previamente en la sección de pines del código base.

```

1
2 socket.on('action', function (data){
3   if (data == 'UP') {
4     gpio1.digitalWrite(1);
5   }
6 });

```

El segundo fragmento de código devuelve la lectura del pin *gpio2* cada vez que recibe el comando *READ* y mandando al cliente el valor de lectura obtenido:

```

1  socket.on('action', function (data){
2      if (data == 'READ') {
3          var temp = gpio2.digitalRead(1);
4          socket.emit('robot_temp', {msg: temp});
5      }
6  });
7

```

En la parte referente a la interfaz de control de la aplicación RobotUI, para lanzar o capturar los comandos correspondientes a los ejemplos superiores, en el primer caso, debemos crear un botón cuyo código a emitir sea *UP* y para el segundo, debemos añadir un botón cuyo código de emisión sea *READ* y una etiqueta cuyo nombre de evento se corresponda con *robot_temp*.

Una vez generado el código para nuestro robot debemos copiarlo a la placa Raspberry Pi o computador que actuará como Robot para nuestra aplicación y ejecutarlo. Para la ejecución del código introducimos el siguiente comando:

```

1  sudo node raspberry.js

```

Siendo *raspberry.js* el nombre del archivo que contiene nuestro código.

Obteniendo el siguiente resultado:

```

pi@raspberrypi:~/Desktop/SAILS-RobotUI/client
pi@raspberrypi:~/Desktop/SAILS-RobotUI/client $ sudo node raspberry.js
Esperando conexión...

|> Now connected to Sails.
\___/ For help, see: http://bit.ly/1DmTvgK
      (using sails.io.js node SDK @v1.1.12)

```

Figura 9.15: Robot a la espera de conexión entrante.

9.1.9. Control de robots

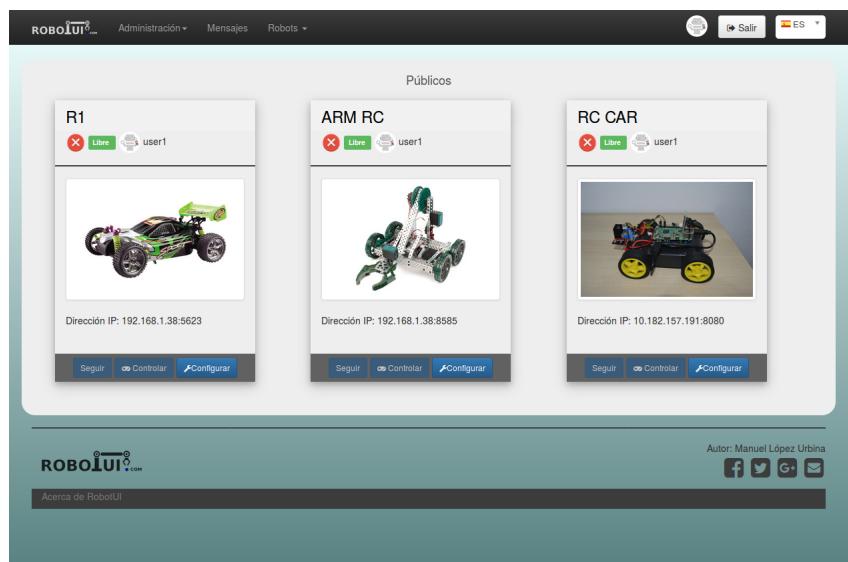


Figura 9.16: Índice robots públicos.

El panel de control de la interfaz podemos identificar los siguientes elementos:

Información		
AUTOR	NOMBRE	DESCRIPCIÓN
 Manuel	 RC Car	Vehículo de pruebas RobotUI

Figura 9.17: Panel informativo de las características del robot de la interfaz.



Figura 9.18: Panel informativo de las órdenes enviadas al robot junto con una entrada de comandos para enviar nuevas órdenes directamente.

9.1.10. Seguimiento de robots

Para realizar el seguimiento de un robot accedemos desde el menú superior en el desplegable *Robots* → *Públicos*. En la nueva ventana tendremos acceso a todos los robots públicos existentes pudiendo realizar el seguimiento en aquellos cuyo estado es *ocupado*.

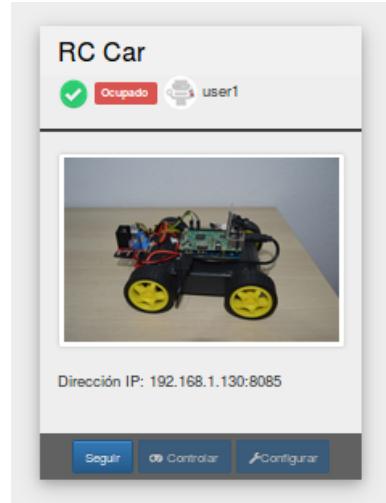


Figura 9.19: Panel informativo de un robot en su modalidad ocupado.

Para pulsamos sobre el botón *Seguir* que se encontrará habilitado accediendo a su interfaz en modo espectador. La página mostrada será de aspecto similar a la mostrada

en la figura 9.20:

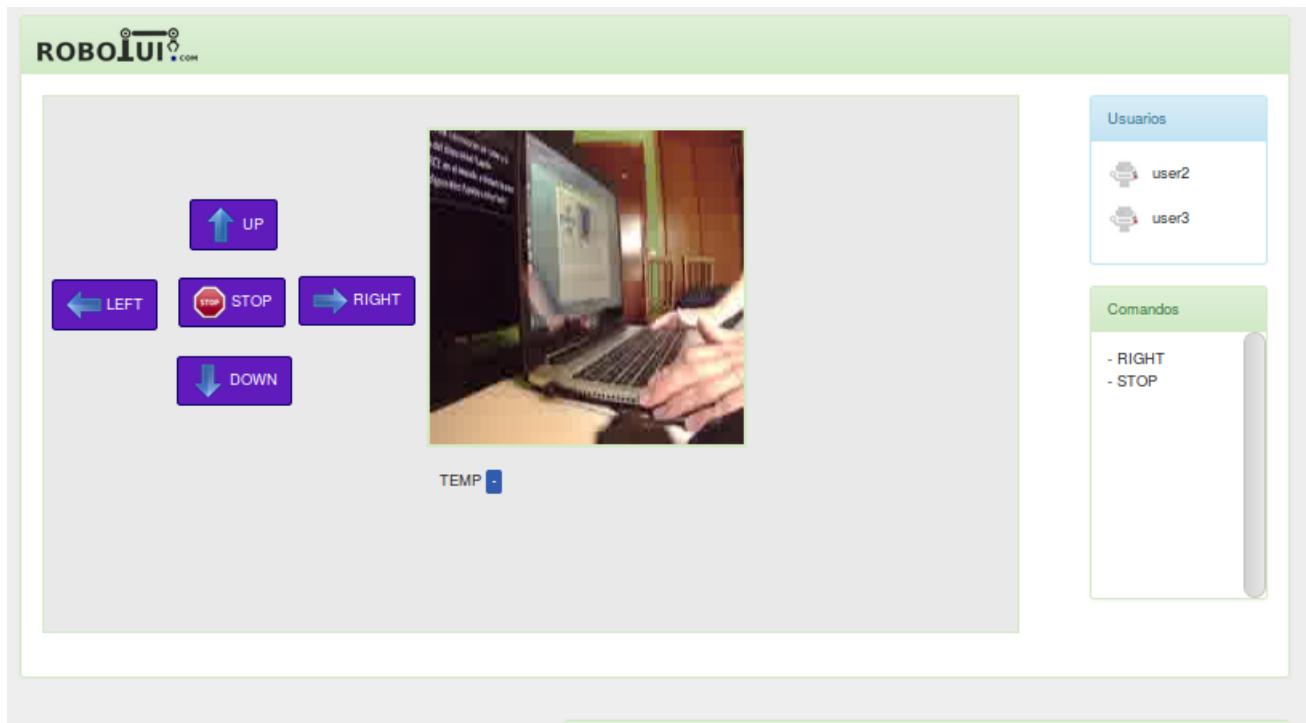


Figura 9.20: Vista del panel de seguimiento de un robot en tiempo real.

Mientras tanto, el usuario que dispone del control del robot será notificado de la entrada de un nuevo espectador. La figura 9.21 muestra la ventana de control de un robot junto con el listado de usuarios espectadores actualizado en todo momento.

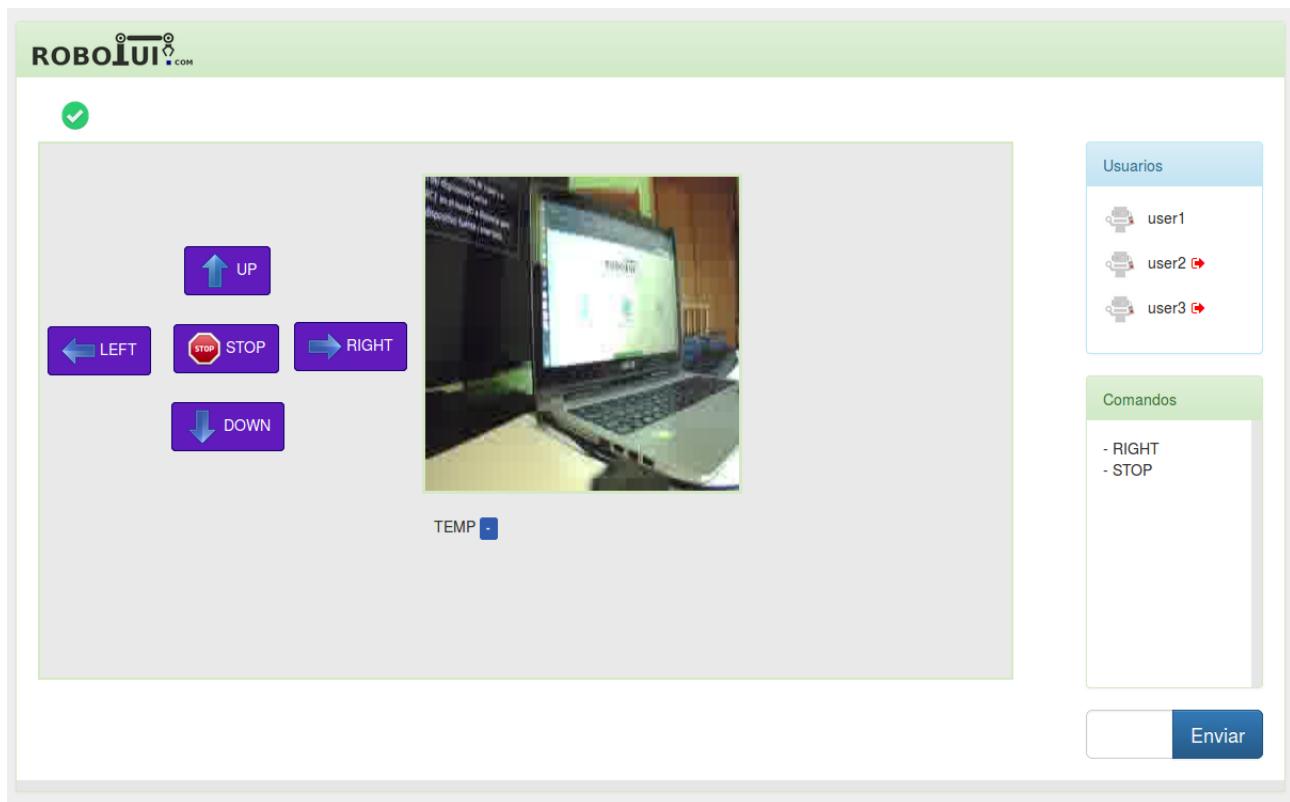


Figura 9.21: Vista del panel de control de un robot junto con el listado de usuarios realizando el seguimiento.

9.1.11. Mensajes

RobotUI incorpora un sistema de mensajería donde los usuarios de la aplicación se pueden enviar mensajes a otros usuarios y recibirlós. Para acceder a la página de mensajes pulsamos sobre *Mensajes* de barra superior.

En la página *Mensajes* tenemos acceso a la bandeja de entrada y los mensajes enviados.

Para mandar un mensaje pulsamos sobre **Redactar**, rellenamos el formulario con título, usuario destinatario y contenido del mensaje y pulsamos en **Nuevo**.

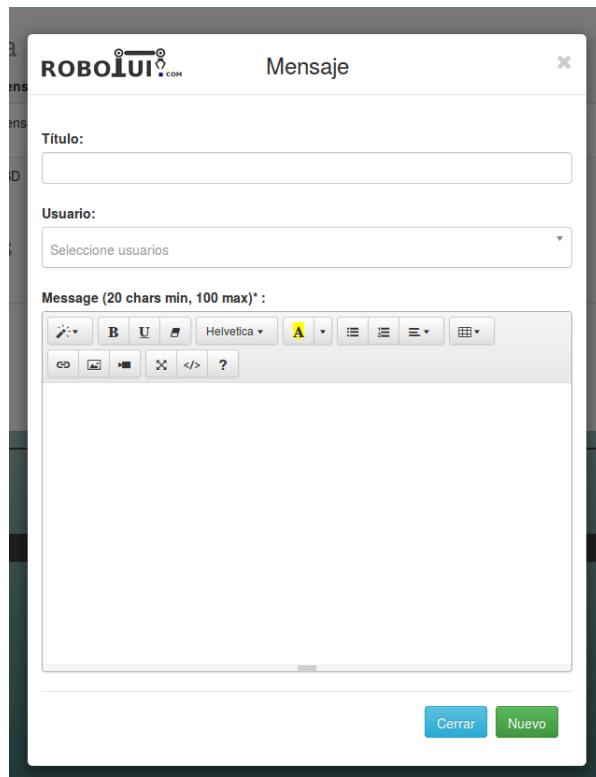


Figura 9.22: Formulario de redacción de un mensaje.

9.1.12. Panel de administración

Páginas disponibles únicamente para usuarios administradores. Accesible desde la barra superior *Administración* → *Usuarios* o *Administración* → *Robots*.

En ella podremos gestionar todos los usuarios o robots existentes en el sistema, editarlos, borrarlos o crear nuevos.

The screenshot displays two administrative panels within the RobotUI interface:

Usuarios (Users)

Online	Nombre	Email	Role	Acciones
✓	user1	user1@user1.com	Administrador	Añadir robot Mostrar robots Editar Eliminar
✗	user2	user2@user2.com	Administrador	Añadir robot Mostrar robots Editar Eliminar
✗	user3	user3@user3.com	Administrador	Añadir robot Mostrar robots Editar Eliminar
✗	user4	user4@user4.com	Administrador	Añadir robot Mostrar robots Editar Eliminar

Robots

Online	Estado	Nombre	Descripción	dirección	Propietario	Conducción	Visión	Acciones
✗	Libre	Buggy	Buggy	192.168.1.120:1234	user1	Único	Único	Interfaz Editar Eliminar
✗	Libre	ARM RC	ARM	10.182.157.191:4234	user1	Privado	Privado	Interfaz Editar Eliminar
✗	Libre	RC Car	RC ROBOTUI CAR	192.168.1.130:8085	user1	Único	Único	Interfaz Editar Eliminar

Figura 9.23: Paneles de administración para robots y usuarios.

Capítulo 10

Comentarios finales

10.1. Presupuesto

Descripción	Unidades	Precio € unidad	Total €
Raspberry Pi 3 Modelo B	1	38,70	38,70
Cámara USB alta definición	1	39,90	39,90
Tarjeta de expansión con batería de Litio para Raspberry Pi	1	16,99	16,99
Lipo batería (3.7v, 600mAh Lipo)	1	13,99	13,99
Indicador Tester de baterías Lipo	1	8,90	8,90
Cargador baterías Lipo	1	21,90	21,90
Chasis vehículo Radiocontrol	1	54	54
Droplet DigitalOcean	2 meses	5/mes	10
Horas de programación	350 horas	50/hora	17500

Total bruto: 17745,38 €

I.V.A. %: 21 %

Total presupuesto: 21471,91 €

10.2. Conclusiones

La elaboración de este proyecto ha resultado muy gratificante a nivel personal. Uno de los motivos principales ha sido la necesidad de trabajar en numerosas áreas de conocimiento entre las que encontramos, por un lado la programación web, haciendo uso del framework Sails js, junto con el empleo de una base de datos no relacional. Todo ello combinado con la robótica. Algunas de las plataformas mencionadas eran desconocidas al inicio del desarrollo de proyecto y han sido adquiridas tras una amplia labor de investigación.

Entre los elementos desarrollados se destaca:

- La elaboración de un vehículo de pruebas haciendo uso de una Raspberry Pi 3 Modelo B.
- Aprendizaje a la utilización del framework Sails.js.
- Aprendizaje al trabajo con eventos en tiempo real mediante el empleo de web-Sockets, tecnología nunca utilizada por mí hasta la fecha.
- Empleo de una base de datos no relacional como Mongo DB.
- Transmisión de gran cantidad de datos entre cliente servidor y servidor cliente. Streaming de vídeo y emisión de comandos entre otros datos.

Pienso que el resultado final del proyecto es ideal para aquellas personas aficionadas a la robótica y programación proporcionando una herramienta sea utilizable por la gran comunidad poseedora de cualquier proyecto robótico y que puedan compartirlo con el resto del mundo.

Una vez presentado podré continuar añadiendo mejoras y muchas cosas que tengo pensadas y que, posiblemente, se realicen para el proyecto del máster de Ingeniería de Sistemas y Computación que me encuentro realizando en la actualidad.

10.3. Mejoras futuras

La aplicación puede mejorarse en diversos aspectos. A continuación, se citan algunas de las mejoras que pueden llevarse a cabo:

- Permitir la definición de las teclas de control del teclado e incorporación de dispositivos tales como gamepads o joysticks.
- Elaborar un generador de código para la exportación a los dispositivos robóticos, reduciendo por tanto las labores de programación. Es decir, proporcionar un generador de código en la aplicación de tal manera que a partir de una interfaz elaborada genere el código ejecutable para el robot según su interfaz de control definida
- Permitir el streaming de audio además del de vídeo.
- Añadir autenticación por un sistema de certificados por clave pública y privada que garantice que la conectividad entre dispositivo robótico y usuario es la correcta.

Anexos

Anexos con las instrucciones para la instalación de todos los componentes software empleados en el desarrollo del proyecto.

.1. Instalación de Node.js

Instalación de los requisitos:

```
1 sudo apt-get install python-software-properties python g++ make
```

Si está utilizando Ubuntu 12.10, necesitará hacer lo siguiente:

```
1 sudo apt-get install software-properties-common
```

Añadimos el repositorio:

```
1 sudo add-apt-repository ppa:chris-lea/node.js
```

Actualizamos la lista de paquetes:

```
1 sudo apt-get update
```

Instalación de Node.js:

```
1 sudo apt-get install nodejs
```

.2. Instalación Sails.js

Esta guía proporciona las pautas necesarias para la configuración de un entorno de trabajo para el desarrollo de aplicaciones Sails. Esta guía no cubre la instalación en un entorno de producción.

.2.1. Prerrequisitos

Partiendo de que se encuentra Node.js correctamente instalado en una máquina con Ubuntu 16.04.2 LTS. Ubuntu es una plataforma muy popular y utilizada en el desarrollo de Sails.js, al igual que otros sistemas operativos basados en Unix, como Mac OS X. La instalación es relativamente fácil y existe multitud de información gracias a su amplia comunidad de desarrolladores.

- Una máquina con Ubuntu 16.04.2 LTS
 - Node.js instalado.

.2.2. Instalación

A continuación detallaremos los pasos para la instalación de Sails. Lo primero que haremos es instalar Sails haciendo uso de npm, el gestor de paquetes que viene con el propio Node. Para ello, lo que vamos a hacer es ir directamente a la terminal y e introducir lo siguiente:

```
1 sudo npm install sails -g
```

La opción `-g`, que significa global, lo que hace es instalar Sails a nivel global, la cual nos permitirá acceder a las funcionalidades de Sails que emplearemos para la creación de nuestros proyectos.

Es posible que necesite permisos de administrador para instalar Sails a nivel global.

Para comprobar que la instalación se realizó correctamente, crearemos un proyecto inicial y levantaremos el servidor. Para ello introducimos en la terminal:

```
1 | sails new my_first_app
```

Cambiamos de directorio (`cd`) al nuevo directorio creado, en el cual nos aseguraremos de que Sails está correctamente instalado. Para ello arrancaremos el servidor y comprobaremos en nuestro navegador en `localhost 1337` (`localhost: 1337`) que Sails está funcionando correctamente.

```
1 cd my_fisrt_app  
2 sails lift
```

Tras introducir en nuestro navegador `localhost: 1337` debemos obtener el siguiente resultado:

Figura 1: Iniciando Sails.

Y en nuestro navegador:

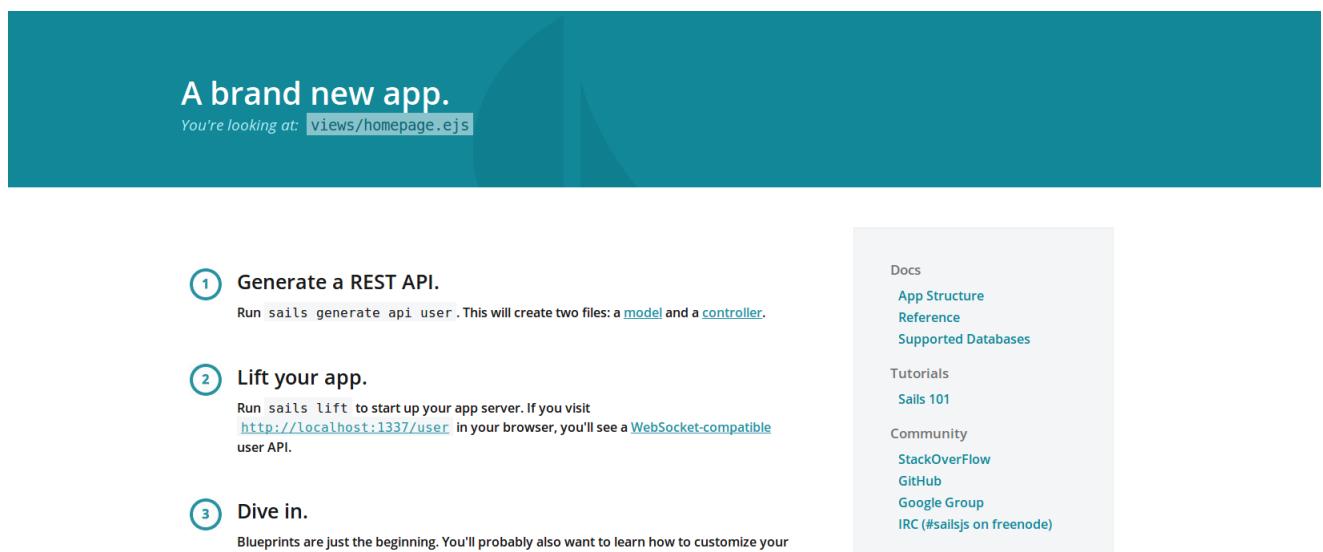


Figura 2: Sails en funcionamiento ¹.

.3. Instalación de MongoDB

MongoDB es una base de datos libre y de código abierto NoSQL utilizada comúnmente en aplicaciones web modernas. Esta guía le ayudará a configurar MongoDB en su máquina para un entorno de aplicación de producción.

.3.1. Prerrequisitos

Para seguir esta guía es necesario:

- Una máquina con Ubuntu 14.04.
- Un usuario con permisos de administrador (no root).

.3.2. Instalación

Para la instalación de MongoDB se puede optar con la versión disponible en los repositorios de paquetes de Ubuntu a pesar de que no sea la última versión disponible. Los repositorios propios de MongoDB proporcionan la versión más actualizada siendo la forma recomendada de instalación. Para la instalación desde repositorios externos,

Ubuntu garantiza la autenticidad de los paquetes de software al verificar que están firmados con las claves GPG, por lo que primero tenemos que importar la clave para el repositorio oficial de MongoDB.

Para ello, debemos ejecutar:

```
1 sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv  
7F0CEB10
```

Después de importar con éxito la clave, obtendremos el siguiente:

```
1 Gpg: Número total procesado: 1  
2 Gpg: importado: 1 (RSA: 1)
```

A continuación, tenemos que actualizar los detalles del repositorio de MongoDB para que APT sepa de dónde descargar los paquetes.

Introduzca el siguiente comando para crear un archivo de lista para MongoDB.

```
1 echo "deb http://repo.mongodb.org/apt/ubuntu "\$lsb_release  
-sc)/mongodb-org/3.0 multiverse" | sudo tee  
/etc/apt/sources.list.d/mongodb-org-3.0.list
```

Después de agregar los detalles del repositorio, necesitamos actualizar la lista de paquetes.

```
1 sudo apt-get update
```

Ahora podemos instalar el propio paquete MongoDB.

```
1 sudo apt-get install -y mongodb-org
```

Este comando instalará varios paquetes que contengan la última versión estable de MongoDB junto con útiles herramientas de administración para el servidor MongoDB.

Después de la instalación del paquete, MongoDB se iniciará automáticamente. Puede comprobarlo ejecutando el siguiente comando.

```
1 service mongodb status
```

Si MongoDB se está ejecutando, se mostrará una salida como la siguiente (con un ID de proceso diferente).

```
1 mongodb.service - An object/document-oriented database  
2   Loaded: loaded (/lib/systemd/system/mongodb.service; enabled;  
3           vendor preset: enabled)  
3   Active: active (running) since s b 2017-02-04 13:07:01 CET; 11h ago  
4     Docs: man:mongod(1)
```

```

5 Main PID: 807 (mongod)
6   Tasks : 10
7   Memory: 84.8M
8     CPU: 4min 7.494s
9   CGroup: /system.slice/mongodb.service
10           - 807 /usr/bin/mongod --config /etc/mongodb.conf

```

También es posible detener, iniciar y reiniciar MongoDB utilizando los siguientes comandos:

```

1 service mongodb stop
2 service mongodb start

```

.4. Instalación del control de versiones Git

Para seguir esta guía es necesario disponer de una máquina con Ubuntu 14.04.

La forma más sencilla de tener Git instalado y configurado para su utilización es mediante el uso de los repositorios predeterminados de Ubuntu. Este es el método más rápido, pero, por contra puede ser que la versión disponible no sea la más reciente. Si necesita la última versión, deberá seguir los pasos para compilar Git desde el origen.

Si no necesitamos disponer de la última versión podemos instalarlo desde el repositorio de Ubuntu. Para ello introducimos los siguientes comandos en una terminal:

```

1 sudo apt-get update
2 sudo apt-get install git

```

Esto descargará e instalará Git en el sistema. A continuación se describe los pasos para su configuración.

.4.1. Configuración

Una vez que disponemos de Git instalado, necesitamos realizar una serie de pasos para añadir nuestros datos de acceso de nuestro repositorio.

La forma más sencilla de hacerlo es a través del comando *git config* proporcionando nuestro nombre y dirección de correo electrónico. Esto es debido a que Git incorpora esta información en cada commit que hacemos. Por ejemplo, si nuestro nombre es *RobotUI* y nuestro email *email@robotui.com*, los comandos de configuración serían los siguientes:

```

1 git config --global user.name "RobotUI"
2 git config --global user.email "email@robotui.com"

```

Finalmente podemos comprobar todos los valores de configuración establecidos escribiendo:

```
1 git config --list
```

Existen muchas más opciones configurables pero estos son los dos esenciales necesarios.

.5. Despliegue de una aplicación Sails

Primeramente, debemos definir el entorno de producción para nuestra aplicación Sails. Para ello editamos el fichero */config/env/production.js* introduciendo los siguientes datos:

- Nombre de nuestra conexión de la base de datos la cual hemos definido en nuestro archivo *config/connections.js*
- Dirección IP de nuestro servidor.
- Puerto por el que queremos correr nuestra aplicación.

Un ejemplo de configuración es el siguiente:

```
1 /**
2  * Production environment settings
3  *
4  * This file can include shared settings for a production environment,
5  * such as API keys or remote database passwords. If you're using
6  * a version control solution for your Sails app, this file will
7  * be committed to your repository unless you add it to your .gitignore
8  * file. If your repository will be publicly viewable, don't add
9  * any private information to this file!
10 *
11 */
12
13 module.exports = {
14
15 /****************************************
16   * Set the default database connection for models in the production *
17   * environment (see config/connections.js and config/models.js )      *
18 ****************************************/
19
20  models: {
21    connection: 'MongodbServer',
22  },
23
24 /****************************************
25   * Set the port in the production environment to 80                  *
26 ****************************************/
27
28  port: 1337,
```

```

29
30  ****
31  * Set the log level in production environment to "silent" *
32  ****
33
34  log: {
35    level: "silent"
36  }
37
38  proxyHost: '46.101.102.33',
39  proxyPort: 1337
40
41 };

```

En el servidor de producción se ha utilizado Nginx ² con la siguiente configuración:

```

1
2 server {
3   listen      80;
4   server_name 46.101.102.33;
5
6
7   listen 443 ssl;
8
9   ssl_certificate /etc/nginx/ssl/nginx.crt;
10  ssl_certificate_key /etc/nginx/ssl/nginx.key;
11
12  location / {
13    proxy_pass          http://46.101.102.33:1337/;
14
15  proxy_http_version 1.1;
16  proxy_set_header Upgrade $http_upgrade;
17  proxy_set_header Connection "upgrade";
18  proxy_set_header Host $host;
19
20  }
21
22 }

```

Para arrancar la aplicación, se ha hecho uso del gestor de procesos Pm2, se introduce la siguiente orden:

```
1 sudo pm2 start app.js -x -- --prod
```

Para monitorizar la aplicación, Pm2 incorpora una herramienta de gestión de procesos. Para visualizarlo introducimos en una terminal:

```
1 sudo pm2 monit
```

Obteniendo un resultado similar al siguiente:

²Nginx es un servidor web/proxy inverso ligero de alto rendimiento y un proxy para protocolos de correo electrónico (IMAP/POP3)

```
λ ~ PM2 monitoring :  
λ testapp  
[0] [cluster_mode]  
[███████████] 66 %  
[███████████] 20.578 MB  
λ testapp  
[1] [cluster_mode]  
[███████████] 67 %  
[███████████] 19.789 MB  
λ
```

Figura 3: Utilización del gestor de procesos Pm2 en el entorno de producción.

Bibliografía

- [1] Douglas Crockford. *JavaScript: The Good Parts*. O'Reilly, 2008.
- [2] Digital Ocean. Digital Ocean. <https://www.digitalocean.com/>.
- [3] Ffmpeg. Documentación Ffmeg. <https://ffmpeg.org/documentation.html>.
- [4] Git Hub. Git Hub. <https://github.com>.
- [5] Pablo Hinojoda and JJ Merelo. *Aprende Git: ... y, de camino, GitHub*. Editorial Reviews, 2015.
- [6] Irl Nathan Mike McNeil. *Sails.js in Action*. Manning Publications, 2017.
- [7] Ministerio de Administraciones Pùblicas, Gobierno de Espaùa. Métrica v3. http://administracionelectronica.gob.es/pae_Home/pae_Documentacion/pae_Metodolog/pae_Metrica_v3.html#.WQ79Wic1GkB.
- [8] Irl Nathan. Activityoverlord, an application to learn sails.js. <https://github.com/irlnathan/activityoverlord>.
- [9] Rohit Rai. *Socket.IO Real-Time Web Application Development*. Packt, 2013.
- [10] Sails.js. Sails.js | Realtime MVC Framework for Node.js. <http://sailsjs.com/>.
- [11] Dr. Ovidiu Vermesan and Dr. Peter Friedss. *Internet of Things - From Research and Innovation to Market Deployment*. River Publishers Aalborg, 2014.
- [12] Wikibooks. The Book of LaTeX. <http://en.wikibooks.org/wiki/LaTeX>.
- [13] Stefan Kottwitz. *LaTeX Beginner's Guide*. Packt Publishing, 2011.
- [14] Official documentation. Sails JS Documentation . <https://sailsjs.com/documentation/reference>.
- [15] Frantisek Korbel. *FFmpeg Basics: Multimedia handling with a fast audio and video encoder*. CreateSpace, 2015.
- [16] Andrew Lombardi. *WebSocket: Lightweight Client-Server Communications*. O'Reilly Media, 2012.
- [17] Sandro Pasquali. *Deploying Node.js*. Packt Publishing, 2015.

- [18] Eben Upton and Gareth Halfacree. *Raspberry Pi User Guide*. <http://www.cs.unca.edu/~bruce/Fall14/360/RPiUsersGuide.pdf>.
- [19] Kristina Chodorow. *MongoDB: The Definitive Guide, 2nd Edition*. O'Reilly Media, 2013.

GNU Documentation Free License

Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc.

<<http://fsf.org/>>

Everyone is permitted to copy and distribute verbatim copies of this license document,
but changing it is not allowed.

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document “free” in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The “**Document**”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “**you**”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A “**Modified Version**” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “**Secondary Section**” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “**Invariant Sections**” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none. The “**Cover Texts**” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “**Transparent**” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “**Opaque**”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “**Title Page**” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

The “**publisher**” means any person or entity that distributes copies of the Document to the public.

A section “**Entitled XYZ**” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “**Acknowledgements**”, “**Dedications**”, “**Endorsements**”, or “**History**”.) To “**Preserve**

the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or non-commercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the

“History” section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

- K. For any section Entitled “Acknowledgements” or “Dedications”, Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled “Endorsements”. Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section to be Entitled “Endorsements” or to conflict in title with any Invariant Section.
- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section Entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements”.

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects. You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and

disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy’s public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

“Massive Multiauthor Collaboration Site” (or “MMC Site”) means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for

anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A “Massive Multiauthor Collaboration” (or “MMC”) contained in the site means any set of copyrightable works thus published on the MMC site.

“CC-BY-SA” means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

“Incorporate” means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is “eligible for relicensing” if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled “GNU Free Documentation License”.

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the “with ... Texts.” line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.