



**Universidad
Rey Juan Carlos**

Grado en Ingeniería en Sistemas Audiovisuales y Multimedia

Escuela Técnica Superior de Ingeniería de Telecomunicación

Curso académico 2016-2017

Trabajo Fin de Grado

Prácticas docentes de desarrollo web

Autor: Walter Rene Cuenca Guachamin
Tutor: José María Cañas Plaza

Madrid 2017

Dedicatoria

*'El genio es 1 % de inspiración y
99 % de transpiración'.*

By Tomas Edison

Agradecimientos

*A todo mi familia pero en especial a mi mama,
por ser madre y padre para mi,
y apoyarme incondicionalmente todo este tiempo.
y por creer en mi.*

*Y a mis amigos, por ser
amigos de verdad.*

Gracias.

Resumen

La utilización de tecnologías y aplicaciones Web presenta un gran crecimiento debido a sus características logrando remplazar en muchos casos a las aplicaciones convencionales de escritorio. Dentro de sus características destaca su fácil acceso, la no necesidad de instalar ni actualizar componentes en el cliente y ser independiente del sistema operativo.

Este TFG sirve como apoyo en las prácticas de la asignatura de LTAW del grado de Ingeniería de sistemas audiovisuales y multimedia. Se pretende cubrir algunos puntos tecnológicos que por tiempo no llegan a ser cubiertos e introducir una mejora sustancial sobre las prácticas de años anteriores. Por ello se realizará un barrido por las tecnologías de cliente, servidor, BBDD y mecanismos de comunicación entre cliente-servidor para seleccionar los más adecuados para el desarrollo del TFG.

La memoria consta de los siguientes capítulos:

- **Introducción:** Presenta el contexto general de TFG.
- **Objetivos:** Presenta los objetivos que quieren cubrir.
- **Infraestructura:** Detalla cada una de las tecnologías utilizadas.
- **Prácticas(cap.4-7):** Cada uno de los capítulos detalla el desarrollo de la práctica correspondiente. El primero se centra en tecnologías del cliente, el segundo en tecnologías de comunicación bidireccional, el tercero en tecnologías del servidor con acceso A BBDD y el último en comunicación audiovisual.
- **Conclusiones:** Evalúa si las prácticas desarrolladas cubren los requisitos marcados al inicio del TFG.

Índice general

Índice de figuras	7
1. Introducción	1
1.1. Paradigma Cliente-Servidor	1
1.2. Tecnologías Web	2
1.3. Estado del arte	3
1.4. Antecedentes	7
2. Objetivos	9
2.1. Metodología	9
2.2. Plan de trabajo	9
3. Infraestructura	11
3.1. HTML5	11
3.1.1. Canvas	11
3.1.2. Media	15
3.1.3. WebSockets	16
3.1.4. API File	18
3.2. JavaScript	19
3.2.1. Propiedades	19
3.3. JQuery	20
3.3.1. Características	20
3.4. Bootstrap	20
3.5. NodeJS	20
3.5.1. Características	21
3.6. BBDD	22
3.7. Django	23
3.7.1. Patrón de diseño MVC	23
3.8. WebRTC	26
3.8.1. Protocolos	26
3.8.2. MediaStream APIs	30
3.8.3. RTCPeerConection APIs	30
3.8.4. RTCDATAChannel APIs	31
3.8.5. Servidor Señalización	32
3.9. Web Services	33
3.9.1. Características	33
3.9.2. Tipos WebServices	34
3.9.3. WebServices Google Maps	34

4. Comecocos Web	36
4.1. Enunciado	36
4.2. Diseño	37
4.3. Desarrollo	38
4.3.1. Game Area	38
4.3.2. Pacman	42
4.3.3. Fantasmas	46
4.3.4. Movimiento	50
4.4. Pruebas	51
5. Comecocos Multijugador	53
5.1. Enunciado	53
5.2. Diseño	54
5.3. Desarrollo Servidor	54
5.3.1. Modulo Game	55
5.3.2. Lógica del Servidor	57
5.4. Desarrollo Cliente	62
5.4.1. Lógica de comunicación	62
5.5. Pruebas	71
6. Tienda Web	72
6.1. Enunciado	72
6.1.1. Requisitos	72
6.1.2. Tecnologías Necesarias	73
6.2. Desarrollo	74
6.3. Back-End	74
6.3.1. Conexión Django-BBDD	75
6.3.2. Models	75
6.3.3. URL's	76
6.3.4. Formularios	77
6.3.5. Vistas	78
6.4. Front-End	86
6.4.1. Barra de Navegación	86
6.4.2. Home	91
6.4.3. Cantantes	91
6.4.4. Eventos	94
6.4.5. Carrito de la Compra	98
6.5. Pruebas	99
7. VideoConferencia usando WebRTC	100
7.1. Enunciado	100
7.2. Diseño	101
7.3. Desarrollo Servidor Señalizaion	101
7.4. Desarrollo Cliente Peer-to-Peer	104
7.5. Pruebas	113
8. Conclusiones	115
Bibliografía	116

Índice de figuras

1.1.	Esquema Cliente-Servidor	1
1.2.	Navegadores de 1 ^a generación.	4
1.3.	Ejemplo Web 1 ^a generación	4
1.4.	Página Web:Wikipedia.	5
1.5.	Página Web:YouTube.	6
1.6.	Página Web:Amazon.	6
1.7.	Comparativa del uso de los navegadores	6
1.8.	Esquema Surveillance 5.1	7
1.9.	Interfaz Surveillance 5.1.	8
1.10.	Esquema JdeRobotWebClients	8
1.11.	Interfaz JdeRobotWebClients	8
2.1.	Metodología en espiral	10
3.1.	Ejemplo del dibujo de un trazo	12
3.2.	Ejemplo rectángulos canvas	12
3.3.	Ejemplo texto canvas	14
3.4.	Ejemplo escalado de imagen canvas.	14
3.5.	Ejemplo recorte imagen canvas.	15
3.6.	Comparación Polling-WebSockets	17
3.7.	Esquema Base de Datos.	23
3.8.	Esquema MVC Django.	26
3.9.	Ejemplo Protocolo SDP	27
3.10.	Ejemplo petición-respuesta STUN	28
3.11.	Ejemplo petición-respuesta TURN	29
3.12.	Capas de SCTP	29
3.13.	Proceso del Servidor Señalización	33
3.14.	Comparativa Soap y Rest	35
4.1.	Aspecto clásico Pacman	36
4.2.	Diseño comecocos web	37
4.3.	Apariencia elementos del juego	40
4.4.	Apariencia información del juego	42
4.5.	Colisión Pacman-Obstáculo	44
4.6.	Colisión Pacman-Cocos	45
4.7.	Apariencia comportamiento Fantasmas	49
4.8.	Inicio Juego	51
4.9.	Pause/Start juego.	52

4.10. Game/Loose juego	52
5.1. Ejemplo Aspecto multijugador Pacman	53
5.2. Esquema Pacman multijugador	54
5.3. Ejecución del servidor	55
5.4. ServidorSeñalizacion	58
5.5. ServidorSeñalizacion	58
5.6. Petición de partida usuario.	58
5.7. Respuesta de partida usuario.	59
5.8. Envió elementos del juego.	59
5.9. Envio actualización del juego.	61
5.10. Envio actualización del juego.	61
5.11. Envió coco comido al usuario.	62
5.12. Pagina Inicio Pacman-Online	62
5.13. Petición/Respuesta sala 1 ^a jugador.	63
5.14. Petición/Respuesta sala 2 ^a jugador.	64
5.15. Envió petición partida.	64
5.16. Recepción petición partida.	65
5.17. Recepción respuesta a la petición de partida.	66
5.18. Recepción parámetros iniciales 1 ^a usuario.	67
5.19. Recepción parámetros iniciales 2 ^a usuario.	68
5.20. Recepción NewPos_Ghost 1 ^a usuario.	70
5.21. Recepción NewPos_Ghost 2 ^a usuario.	70
5.22. Aspecto clásico Pacman	71
6.1. Portada Paginas Web.	72
6.2. Create BBDD	74
6.3. Create tablas App	75
6.4. Interfaz admin Django	76
6.5. Respuesta WebServices Geolocalizacion	81
6.6. Respuesta WebServices Servicios	82
6.7. Barra de navegación	87
6.8. Buscador de la aplicación	87
6.9. Pagina Contacta	88
6.10. Pagina Registro	89
6.11. Formulario Login	90
6.12. Perfil Usuario	90
6.13. Pagina Principal	91
6.14. Cantante panel Información	92
6.15. Cantante panel Información	92
6.16. Cantante panel Discos	93
6.17. Cantante pestaña imágenes	95
6.18. Evento panel Información	95
6.19. Evento panel ubicación	96
6.20. Evento panel Cantantes	98
6.21. Evento pestaña Entradas	98
6.22. Detalle Carrito	99
6.23. Orden Compra	99

7.1.	Ejemplo sitio Web	100
7.2.	ServidorSeñalización	101
7.3.	Request/Replay Salas Servidor	102
7.4.	Request/Replay conexión Servidor	103
7.5.	Request/Replay conexión Servidor	103
7.6.	Request/Replay conexión Servidor	103
7.7.	Request/Replay conexión Servidor	104
7.8.	Petición/Recepción salas.	105
7.9.	Petición/Respuesta establecimiento Conexión.	106
7.10.	Creación canal de datos cliente.	107
7.11.	Creación oferta cliente.	108
7.12.	Creación canal de recepción datos.	109
7.13.	Creación de la respuesta.	109
7.14.	Conexión Peer-to-Peer final.	110
7.15.	Envío de caracteres del chat por RTCDataChannel.	111
7.16.	Envío información del fichero con RTCDataChannel.	113
7.17.	Recepción y reconstrucción del fichero	114

Capítulo 1

Introducción

Este Trabajo Fin de Grado se centra en el campo de las tecnologías Web. En este capítulo vamos a introducir los conceptos de básicos de la comunicación entre los navegadores y las aplicaciones web. Ademas se ofrece una visión general de aplicaciones que emplean tecnologías Web, que dan contexto a este TFG.

1.1. Paradigma Cliente-Servidor

La red (Internet) ofrece un servicio básico de comunicación (transferencia de bits). El software de comunicaciones (implementación de TCP/IP) de las máquinas no inicia comunicaciones con otras máquinas sino que son las aplicaciones. Si nos centramos en el punto de vista funcional, se puede definir como una arquitectura distribuida que permite a los usuarios finales obtener acceso a información de forma transparente por medio de peticiones HTTP a un servidor, figura 1.1.

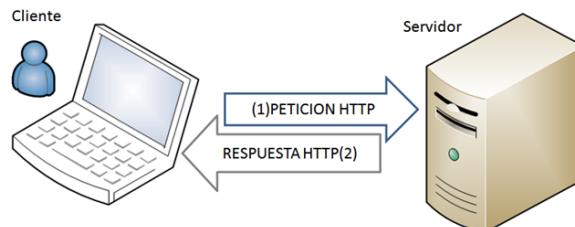


Figura 1.1: Esquema Cliente-Servidor.

La forma estándar de uso de sistemas Cliente/Servidor es a través de interfaces gráficas de usuario; mientras que la administración de datos y su seguridad e integridad se deja a cargo de los servidores.

Cliente

Es el proceso que permite a los usuarios formular peticiones al servidor, a este proceso se le conoce como **front-end**.

Las principales funciones que maneja son :

- Administrar la interfaz de usuario.

- Interactuar con el usuario.
- Procesar la lógica de la aplicación y hacer validaciones locales.
- Generar requerimientos de bases de datos.
- Recibir resultados del servidor.
- Formatear resultados.

Servidor

Es el proceso encargado de atender a múltiples clientes que hacen peticiones de algún recurso administrado por él. A este se le conoce con el término **back-end**. Las principales funciones que maneja son:

- Aceptar los requerimientos de bases de datos que hacen los clientes.
- Procesar requerimientos de bases de datos.
- Formatear datos para trasmitirlos a los clientes.
- Procesar la lógica de la aplicación y realizar validaciones a nivel de bases de datos.

1.2. Tecnologías Web

Protocolo HTTP

Como se ha mencionado el modelo C/S se basa en peticiones entre cliente-servidor bajo el protocolo HTTP (**Hyper Text Transfer Protocol**). Este protocolo fue desarrollado por el *World Wide Web Consortium¹* y la *Internet Engineering Task Force*.

Se compone de dos mensajes, el primer mensaje es originado por el cliente y es el requerimiento inicial de la comunicación. Este requerimiento llamado **Request** está dividido en una cabecera y un mensaje de texto. En la cabecera el cliente envía información de la página solicitada al servidor y de la aplicación cliente que estamos utilizando en el **User-Agent**, entre otras cosas.

```
GET /index.html HTTP/1.1
Host: www.example.com
User-Agent: HTTPTool/1.0
Connection: close
Linea en blanco
```

Listing 1.1: Petición HTTP.

Una vez recibido el requerimiento el servidor devuelve una respuesta llamada **Response** la cual también está compuesta por una cabecera y un mensaje.

```
HTTP/1.1 200 OK
Date: Fri, 31 Dec 2003 23:59:59 GMT
Content-Type: text/html
Content-Length: 1221
```

¹Web: <http://www.w3.org/>

```
<html><body>
<h1>Pagina principal</h1>
...
</body></html>
```

Listing 1.2: Respuesta HTTP.

Es un protocolo sin estado, es decir, que no guarda ninguna información sobre conexiones anteriores.

HTML

Son las iniciales de la expresión en inglés HyperText Markup Language. Se trata de un conjunto de etiquetas que se van intercalando entre el texto de forma que los navegadores sepan qué es lo que tienen que mostrar cuando accedemos a una página y cómo deben presentarlo en la pantalla.

El W3C (World Wide Web Consortium)² es el fórum internacional que se encarga desarrollar nuevas tecnologías relacionadas con la WEB dictando las normas que constituyen el estándar HTML entre otros. A lo largo de sus diferentes versiones, se han incorporado y suprimido diversas características, con el fin de hacerlo más eficiente y facilitar el desarrollo de páginas web. La versión más extendida y estable fue 4.0 que permitía utilizar textos, sonidos, imágenes, y lo más importante, enlaces a otras páginas enriqueciendo de esta forma la información de la página.

Pero a medida que este lenguaje se extendía se vio la necesidad de actualizar la versión del estándar. Es aquí, donde surge la versión 5³ de HTML que pretende cubrir varias necesidades que habían surgido hasta el momento. Dentro de estas nuevas características destacamos:

- **Contenido multimedia:** Habilita la posibilidad de incluir contenido multimedia de forma nativa sin necesidad de plugins o software de terceros.
- **Animación:** Incluye la etiqueta canvas que permite crear contenido 2d y 3d dentro de la web.
- **Comunicación:** con la aparición de aplicaciones en tiempo real, era necesario proveer de mecanismos que permitan intercambiar datos de forma ligera por lo que se incluye WebSockets y WeRTC.
- **Animación:** PENDIENTE.

1.3. Estado del arte

Las fuentes⁴ establecen tres generaciones de sitios Web. Actualmente conviven las distintas generaciones, aunque la creación de sitios web siguen el modelo de la segunda generación.

²Web: <http://www.w3.org/>

³<http://www.w3.org/TR/2014/REC-html5-20141028/>.

⁴Fuente:<http://www.revistadintel.es/Revista/Numeros/Numeros16/Zona/Firmas/userosRedesSociales.pdf>

Primera generación

Esta primera generación abarca desde el nacimiento de la Web(1992) hasta mediados de 1994 y es conocida como Web 1.0.

Sus paginas eran simples documentos constituidos por texto e interpretadas por los navegadores existentes en ese momento. Algunos de los navegadores existentes eran Erwise(figura 1.2(a)⁵) y ViolaWWW(figura 1.2(b)⁶)

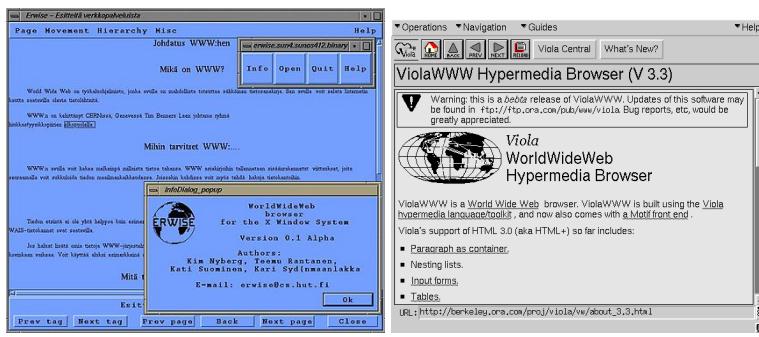


Figura 1.2: Navegadores de 1^a generación.

Posteriormente, con la aparición del lenguaje HTML, el concepto inicial de la Web 1.0 evolucionó convirtiéndose en un conjunto de documentos en lenguaje HTML que se encontraban interconectados por medio de enlaces. Estos documentos los generaba una única persona que se encargaba del diseño y de la obtención de los datos necesarios presentando como obstáculo que los documentos solo eran de lectura por lo que el usuario no tenía la capacidad de interactuar con el contenido de la página,figura 1.3⁷.

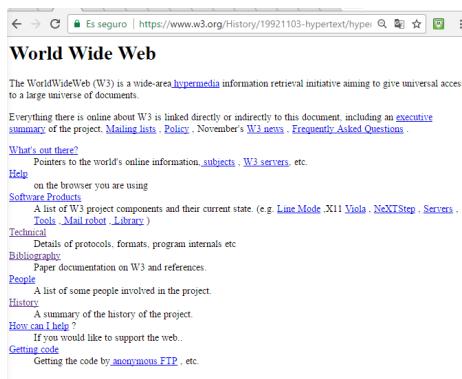


Figura 1.3: Ejemplo Web 1^a generación.

⁵fuente:<https://en.wikipedia.org/w/index.php?curid=24471738>

⁶fuente: <https://en.wikipedia.org/w/index.php?curid=6098546>

⁷fuente:<https://www.w3.org/History/19921103-hypertext/hypertext/WWW/TheProject.html>

Segunda generación

Con la llegada de las “compañías ciberneticas” (año 2001) Internet da un giro de 180 grados. El éxito de estas compañías dependía de webs mucho más dinámicas y para ello era necesario huir de sitios estáticos y poco actualizados, y servir páginas HTML dinámicas creadas al vuelo desde una actualizada base de datos. Los CMS (Content Management System o Sistema Gestor de Contenidos) entraron en acción.

Esta nueva generación recibe el nombre Web 2.0 .Este termino aparece por primera vez en el año 2004 cuando Dale Dougherty (vicepresidente de O'Reilly Media) utilizó este término en una conferencia en la que hablaba del renacimiento y evolución de la web.

Aunque no existe una definición consensuada, en el año 2005 Tim O'Reilly (fundador de O'Reilly Media) definió el concepto de Web 2.0 como *“una serie de aplicaciones y páginas de Internet que utilizan la inteligencia colectiva para proporcionar servicios interactivos en red dando al usuario el control de sus datos”*.

A diferencia de la Web 1.0, la información y los contenidos se producen (directa o indirectamente) por los usuarios del sitio web y adicionalmente puede ser compartida por varios portales web de estas características.

Algunos ejemplos de la Web 2.0 son las las wikis(figura 1.4⁸), los servicios multimedia (figura 1.5⁹) y en general sitios web que interactuan con los usuarios (figura 1.6¹⁰)

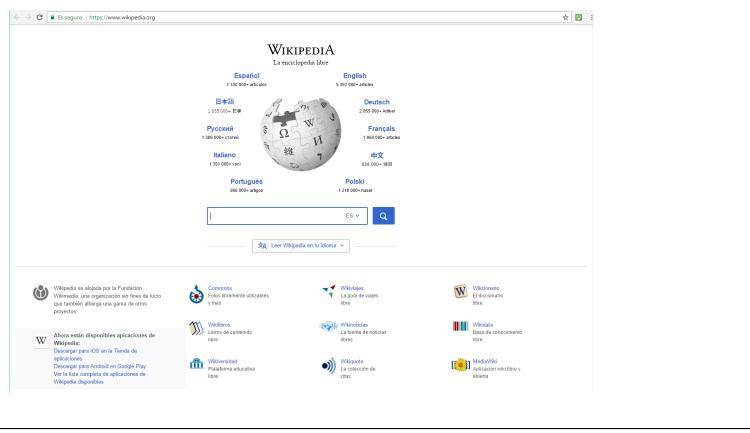


Figura 1.4: Pagina Web:Wikipedia.

Al igual que la web evoluciona los navegadores¹¹ también los hicieron. En la actualidad disponemos de los siguientes navegadores:

1. **Internet Explorer:** creado en 1995 aunque no fue hasta el año 2000 que domino absolutamente el mercado. Pero con la llegada de Firefox su uso global descendió de una forma notable.
2. **Firefox:** creado por la fundación Mozilla y es la continuación al navegador Mozilla. Se publicó en 2004 con el objetivo de permitir que la web sea pública, abierta y accesible.

⁸fuente: <https://www.wikipedia.org/>

⁹fuente:<https://www.youtube.com/>

¹⁰fuente:<https://www.amazon.es/>

¹¹fuente:http://www.mclibre.org/consultar/htmlcss/otros/otros_historia_navegadores.html#navegadores

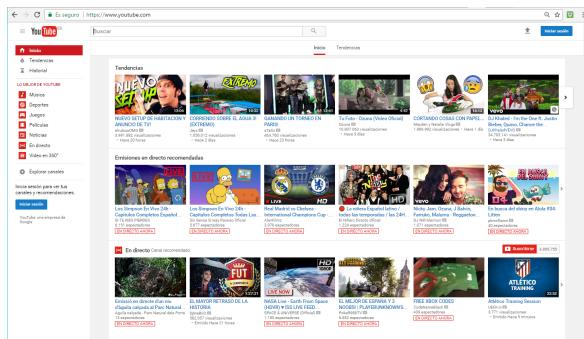


Figura 1.5: Pagina Web:YouTube.



Figura 1.6: Pagina Web:Amazon.

3. **Safari:** creado en 2003 para el sistema operativo Mac de Apple ya que antes no disponía de un navegador propio.
 4. **Chrome:** creado en 2008 por Google a partir de WebKit. Destaca por su interfaz minimalista y por la velocidad de ejecución del código JavaScript, lo que obligó a sus competidores a ponerse las pilas en estos aspectos.

A continuación se muestra una comparativa¹² del porcentaje de uso de cada uno de los navegadores en la actualidad.

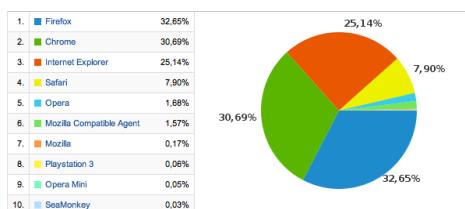


Figura 1.7: Comparativa del uso de los navegadores.

¹²fuente:<http://identidadgeek.com/por-que-el-navegador-opera-no-es-tan-popular/>

Tercera generación

En el intento de comprender la propia Web 2.0 se vislumbran futuras etapas de la Web, sobre todo orientadas a mejorar la interactividad y la movilidad entre/de los usuarios.

El término Web 3.0 está asociado al concepto de Web Semántica, desarrollado bajo la tutela del creador de la Web Tim Berners Lee.

Básicamente, toda la información publicada en las diferentes páginas web no es entendible por los ordenadores, teniendo únicamente significado para las personas. La idea consiste en añadir información adicional a la información “visible”, de tal manera que pueda ser entendida por los ordenadores.

En conclusión, se trata de dotar de significado a las páginas web

1.4. Antecedentes

Como se ha mencionado a lo largo del capítulo las tecnologías web permiten crear interfaces para que los usuarios interactúan con las prestaciones de la aplicación. A continuación se muestra como ejemplo de esto dos TFG que incorporan tecnologías Web en su desarrollo ya sea para la creación de un interfaz como para intercambiar información entre el cliente y el servidor de la aplicación.

Surveillance 5.1 (URJC)

Surveillance 5.1 [3][4] fue desarrollado por Edgar Barrero como trabajo fin de grado. La aplicación web ofrece un flujo de vídeo desde una cámara web, un flujo de imagen de profundidad procedente de un sensor Kinect y su representación en 3D. También ofrece el acceso a un sensor de humedad y un actuador como ejemplos de dispositivos domóticos. Su desarrollo se creó por medio de Ruby on Rails, un entorno de código abierto para el desarrollo de aplicaciones web. El servidor web se conecta a componentes JdeRobot que ofrecen interfaces ICE de objetos distribuidos. De esta forma obtiene datos de los distintos sensores y actuadores de la aplicación. En el lado cliente, el navegador refresca estos datos realizando peticiones AJAX.

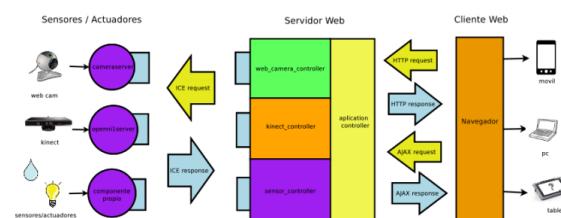


Figura 1.8: Esquema Surveillance 5.1.

JdeRobotWebClients (URJC)

JdeRobotWebClients[5][6]fue desarrollado por Aitor Martínez Fernández como trabajo fin de grado. La aplicación Web ofrece la posibilidad de crear seis versiones web de herramientas utilizadas JdeRobot(CameraViewJS,RGBDViewerJS,KobukiViewerJS,...) y que

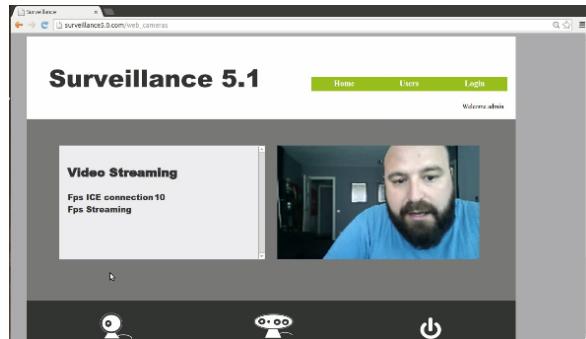


Figura 1.9: Interfaz Surveillance 5.1.

estaban programadas en C++ o Python con su propio interfaz gráfico provocando que sean ejecutables solo en Linux. Estas nuevas versiones pretenden ser multiplataforma (Linux, Android, IOS, Windows,...), y accesibles desde un navegador web como interfaz gráfico permitiendo acceder a los sensores y actuadores sin un servidor intermedio.

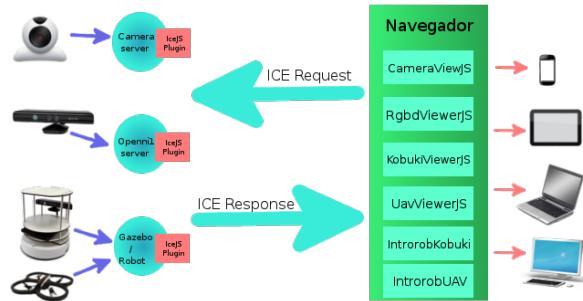


Figura 1.10: Esquema JdeRobotWebClients.

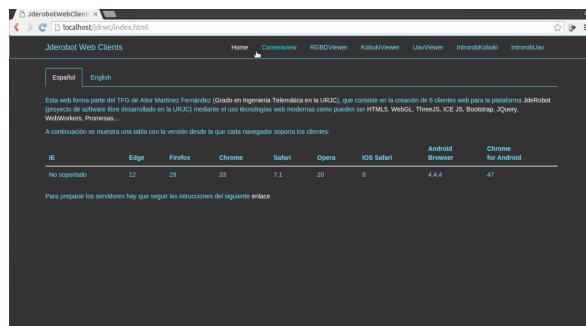


Figura 1.11: Interfaz JdeRobotWebClients.

Capítulo 2

Objetivos

Una vez presentado el contexto en el que se basa este trabajo, pasamos a definir los objetivos que se cubren en el TFG.

El objetivo principal es desarrollar un conjunto de prácticas basadas en tecnologías web que sirva como modelo a los alumnos de la asignatura de LTAW. Cada una de estas prácticas se trata como un subobjetivo en la siguiente lista.

1. Crear un juego en la web basado en tecnologías del cliente.
2. Crear un juego multijugador basado en tecnologías de comunicación bidireccional en tiempo real.
3. Crear un sitio Web de eventos y cantantes basada en tecnologías de servidor con manejo de base de datos.
4. Crear una aplicación de Videoconferencia Peer-to-Peer entre navegadores basada en tecnologías de comunicación audiovisual.

2.1. Metodología

La realización de todo proyecto necesita una metodología a seguir con el que se planifica las tareas necesarias para llegar a nuestro objetivo. Por ello se ha seleccionado el modelo de desarrollo en espiral, figura 2.1, que se aplica habitualmente en ingeniería de software. Este modelo define una serie de ciclos que se repiten continuamente hasta finalizar el proyecto, dividiéndolo en subtareas más sencillas en las que se establece un punto de control al final de cada una para evaluar el resultado y establecer nuevas tareas.

Durante el tiempo que ha durado el proyecto se acordaron reuniones semanales con el tutor de forma presenciales o por Video-Conferencia en las que se revisaba los objetivos semanas y se definían los nuevos objetivos.

Los avances más relevantes se añadían a la mediawiki [7] de JdeRobot y a través de repositorio de GitHub [8] guardábamos el código.

2.2. Plan de trabajo

Para finalizar este capítulo explicamos las distintas partes en las que se ha dividido el plan del proyecto.

1. **Aprendizaje de tecnologías web:** Estudiar y conocer distintas tecnologías web del servidor, cliente, BBDD y tecnologías de intercambio de información entre el cliente y el servidor.
2. **Selección de prácticas:** Con los conocimientos adquiridos realizamos una propuesta que se ajuste a los requisitos marcados.
3. **Desarrollo de prácticas:** El diseño y desarrollo de cada práctica ha sido de forma incremental, es decir, se ha buscado obtener la funcionalidad exigida en cada una para luego centrarnos en dar una apariencia más atractiva.
4. **Pruebas:** Cada una de las prácticas han sido ejecutadas en distintos navegadores para evaluar su funcionalidad.

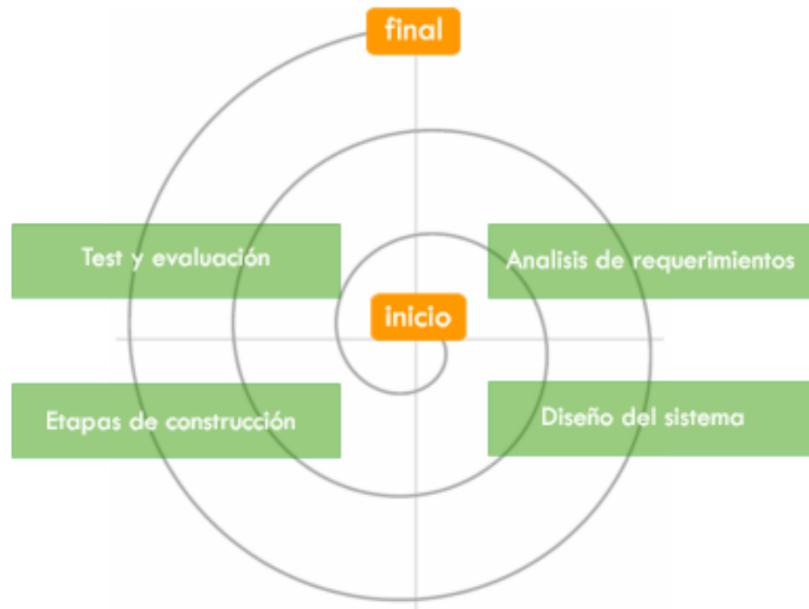


Figura 2.1: Metodología en espiral.

Capítulo 3

Infraestructura

3.1. HTML5

Este nuevo estándar incorpora una serie de APIs accesibles desde JavaScript que surgen para dotar de funcionalidad nativa a la Web sin necesidad de utilizar plugins externos. Los distintos ingredientes asociados a HTML5[9] los describimos en las siguientes secciones.

3.1.1. Canvas

Se trata de una nueva etiqueta, <canvas>[10], que permite trabajar con gráficos dentro de la web sin la necesidad de emplear programas externos. Dispone de varios métodos que permiten realizar múltiples tareas de carácter gráfico que explicamos a continuación.

Context 2d

Es el punto de partida para utilizar las propiedades de canvas. Por ello, es necesario acceder a la etiqueta canvas existente en el documento para obtener el contexto.

El contexto puede ser 2D y 3D dependiendo del tipo de elementos que se quieren dibujar.

```
var canvas = document.getElementById('tutorial');
var ctx = canvas.getContext('2d');
```

Listing 3.1: Acceso al contexto de Canvas.

Formas Disponibles

Por medio del contexto tenemos acceso a un conjunto de métodos que permiten dibujar primitivas predefinidas, que se explican a continuación.

1. Trazos

- **beginPath()**: Marcan el inicio de un nuevo trazo.
- **closePath()**: Marcan el final del trazo definido.
- **stroke()**: Dibuja el contorno de la forma.
- **fill()**: Dibuja una forma sólida rellenando el área del trazo.

2. Líneas: La función `lineTo(x,y)` permite dibujar líneas. Toma como punto de partida el ultimo punto conocido y como punto final la coordenada(x,y) que se le pasa .
3. Movimiento : La función `moveTo(x, y)` permite moverse a un punto del lienzo para empezar a dibujar a partir de el.

```

1 function draw() {
2   var canvas = document.getElementById('canvas');
3   if (canvas.getContext) {
4     var ctx = canvas.getContext('2d');
5
6     // Filled triangle
7     ctx.beginPath();
8     ctx.moveTo(25, 25);
9     ctx.lineTo(105, 25);
10    ctx.lineTo(25, 105);
11    ctx.fill();
12
13    // Stroked triangle
14    ctx.beginPath();
15    ctx.moveTo(125, 125);
16    ctx.lineTo(125, 45);
17    ctx.lineTo(45, 125);
18    ctx.closePath();
19    ctx.stroke();
20  }
21}

```



Figura 3.1: Ejemplo del dibujo de un trazo.

4. Rectángulos

- **`fillRect(x, y, width, height)`:** Dibuja un rectángulo relleno.
- **`strokeRect(x, y, width, height)`:** Dibuja el contorno de un rectángulo.
- **`clearRect(x, y, width, height)`:** Borra el área rectangular especificada.

```

1 function draw() {
2   var canvas = document.getElementById('canvas');
3   if (canvas.getContext) {
4     var ctx = canvas.getContext('2d');
5
6     ctx.fillRect(25, 25, 100, 100);
7     ctx.clearRect(45, 45, 60, 60);
8     ctx.strokeRect(50, 50, 50, 50);
9   }
10}

```

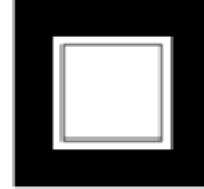


Figura 3.2: Ejemplo rectángulos canvas.

5. Arcos: Por medio de la función `arc(x,y,radio,angInit,angFin,true)` dibujamos una circunferencia o media circunferencia. Para realizar este proceso establece como centro

las coordenadas (x,y) ,su tamaño depende del radio que se especifique y por ultimo establecemos el angulo inicial y final que queremos.

Estilos y Colores

Para poder aplicar colores a los dibujos que se crean tenemos a nuestra disposición dos propiedades que podemos usar:

- **fillStyle:** Establece el color de relleno de la figura.
- **strokeStyle:** Establece el color del contorno de la figura.

Además de dibujar formas opacas en el lienzo,también podemos dibujar formas semitransparentes (o translúcidas). Esto se realiza estableciendo la propiedad globalAlpha o asignando un color semitransparente al estilo de trazo y / o de relleno.

- **globalAlpha:** Aplica el valor de transparencia especificado a todas las formas futuras del lienzo.El valor debe estar entre 0,0 (totalmente transparente) y 1,0 (completamente opaco).

Por otra parte podemos establecer el estilo de las lineas que componen las figuras,por medio las siguientes propiedades:

- **lineWidth:** Establece el ancho de las lineas.
- **lineCap:** Establece el aspecto de los extremos de las lineas.Estos atributos pueden ser: butt(extremos cuadrado),round(extremos redondeados) y square()
- **lineJoin:** Establece el aspecto de las esquinas donde se encuentran las lineas.Estos atributos pueden ser: round() , bevel() y miter().

Texto

Permite generar cadenas de texto dentro de canvas por medio del método **fillText(text,x,y)** que recibe como parámetro el texto y la coordenada donde se dibujaran,figura 3.3.

- **fillText(text,x,y):** Dibuja el texto dado en las coordenadas(x,y) del lienzo.
- **strokeText(text,x,y):** Dibuja el texto dado en las coordenadas(x,y) del lienzo sin relleno.

Ademas nos permite establecer el estilo del texto por medio de las siguientes propiedades:

- **font:** Estilo del texto que se utiliza al dibujar el texto.Los valores permitidos son similares a las propiedades CSS font.
- **textAlign:** Alinea el texto.Los valores disponibles son: start,end,left,rigth o center.
- **direction:** Direccionalidad del texto.

```

1 | function draw() {
2 |   var ctx = document.getElementById('canvas').getContext('2d');
3 |   ctx.font = '48px serif';
4 |   ctx.fillText('Hello world', 10, 50);
5 |

```

Hello world

Figura 3.3: Ejemplo texto canvas.

Imágenes

Para importar imágenes en el lienzo es necesario obtener la referencia de un objeto `HTMLImageElement` como fuente y dibujarla en el lienzo por medio de la función `drawImage()`. Para cargar una imagen externa es necesario generar un objeto `HTMLImageElement` por medio de JavaScript. Para ello, utilizamos el constructor `Image()` en el que definimos la dirección de la imagen en el atributo `src` y por medio del método `load` nos aseguraremos que la carga se ha completado.

Tras obtener la referencia a nuestra imagen podemos utilizar los métodos disponibles para cargar las imágenes en el lienzo. A continuación, se muestra las variantes del método ya que se encuentra sobrecargado.

- **`drawImage(img,x,y)`:** Dibuja la imagen especifica en las coordenadas(x,y) del lienzo.
- **`drawImage(image, x, y, width, height)`:** Dibuja la imagen especifica en las coordenadas(x,y) estableciendo la escala en la que se dibuja por medio de los parámetros width y height, figura 3.4.

```

1 | function draw() {
2 |   var ctx = document.getElementById('canvas').getContext('2d');
3 |   var img = new Image();
4 |   img.onload = function() {
5 |     for (var i = 0; i < 4; i++) {
6 |       for (var j = 0; j < 3; j++) {
7 |         ctx.drawImage(img, j * 50, i * 38, 50, 38);
8 |       }
9 |     };
10 |   };
11 |   img.src = 'https://mdn.mozilla.org/files/5397/rhino.jpg';
12 |

```



Figura 3.4: Ejemplo escalado de imagen canvas.

- **`drawImage(image, sx, sy, sWidth, sHeight, dx, dy, dWidth, dHeight)`:** Dada una imagen, esta función toma el área de la imagen de origen por el rectángulo cuya esquina superior izquierda es (sx,sy) y cuya anchura y la altura son sWidth y sHeight, colocándolo en el lienzo en las coordenadas(dx,dy) y escalado por medio de dWidth y dHeight, figura 3.5.

Transformaciones

Antes de indicar las transformaciones disponibles es necesario conocer los métodos `save()` y `restore()`. El primero se utiliza para guardar el estado del lienzo antes de aplicar un

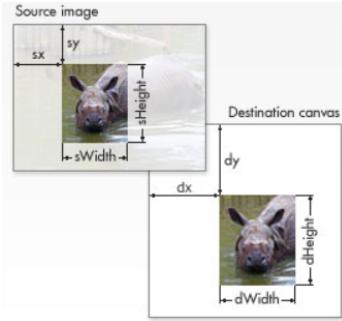


Figura 3.5: Ejemplo recorte imagen canvas.

cambio en el lienzo mientras que el otro se encarga de recuperar el estado del lienzo guardado.

Ahora es momento de presentar las distintas transformaciones que se pueden utilizar

- **translate(x,y):** Mueve el lienzo y su origen a la coordenada (x,y).
- **rotate(angle):** Gira el lienzo en sentido a las agujas del reloj hasta encontrar el angulo (radianes) indicado.
- **scale(x,y):** Escala x e y unidades del lienzo. Se tratan de números reales, si son menores a la unidad disminuyen el tamaño encaso contrario aumentan el tamaño.

Se emplea esta etiqueta en el desarrollo de las prácticas del **ComeCocos Web** y del **Come-Cocos Multijugador**.

3.1.2. Media

El nuevo estándar incorpora etiquetas que permiten incorporar contenido multimedia[11] en la web de forma nativa ya que antes era necesario utilizar pugins externos. La etiqueta `<video>` permite incrustar un vídeo en la web e incorpora la posibilidad de establecer múltiple formatos ya que no todos los formatos de vídeo son compatibles en cada uno de los navegadores.

- Normalmente, un contenedor WebM empaqueta audio Ogg Vorbis con vídeo VP8 / VP9. Esto se apoya principalmente en Firefox y Chrome.
- Un contenedor MP4 suele empaquetar audio AAC o MP3 con vídeo H.264. Esto se apoya principalmente en Internet Explorer y Safari.
- El contenedor Ogg más antiguo tiende a ir con Ogg Vorbis audio y vídeo Ogg Theora. Esto fue apoyado principalmente en Firefox y Chrome, pero básicamente ha sido reemplazado por el mejor formato WebM.

Las principales características de las que dispone la etiqueta `<video>` son las siguientes:

- **width/height:** Tamaño del vídeo
- **autoplay:** Indica que una vez se ha cargado el elemento empiece la reproducción automáticamente.

- **loop:** Se crea un bucle en el que se repite indefinidamente el vídeo.
- **muted:** Desactiva el sonido.
- **poster:** Toma la dirección de una imagen que se muestra antes de empezar la reproducción del vídeo.
- **src:** Contiene la ruta de acceso al vídeo
- **controls:** Permite a los usuarios pausar , reproducir o ajustar el volumen del vídeo.

Ademas tenemos la etiqueta `<audio>` que permite incrustar audio en la web, en cuanto a su funcionalidad es similar al de la etiqueta `<video>` aunque no presenta un interfaz gráfico. Estos elementos se emplean en el desarrollo de todas las practicas.

3.1.3. WebSockets

Internet se ha creado a partir del paradigma solicitud/respuesta de HTTP. Un cliente carga una página web, se cierra la conexión y no ocurre nada hasta que el usuario hace clic en un enlace o envía un formulario.

Hace algún tiempo que existen tecnologías que intentan simular este comportamiento, como por ejemplo **Comet**. Uno de los trucos más comunes para crear la ilusión de una conexión iniciada por el servidor se denomina Long Polling.

Con el Long Polling, el cliente abre una conexión HTTP con el servidor, el cual la mantiene abierta hasta que se envíe una respuesta. Cada vez que el servidor tenga datos nuevos, enviará la respuesta.

La siguiente lista de características permiten que WebSockets[13] posea ese comportamiento sin necesidad de utilizar trucos como en caso de **Comet**, figura 3.6.

1. **Conexión bidireccional:** Esta conexión se produce en tiempo real y se mantiene permanentemente abierta hasta que se cierre de manera explícita.
2. **Gran rendimiento y escalabilidad:** Si un socket esta abierto, el servidor puede enviar datos a todos los clientes conectados al socket, sin tener que estar constantemente procesando peticiones.
3. **Latencia:** Como el socket está siempre abierto y escuchando, los datos son enviados inmediatamente desde el servidor al navegador.
4. **Transmisión de datos:** Los datos a transmitir se reducen también de manera drástica, pasando de un mínimo de 200-300 bytes en peticiones Ajax, a 10-20 bytes.

Funcionamiento API

El cliente establece una conexión WebSockets[14] a través de un proceso conocido como **Handshake** WebSocket. Este proceso se inicia con una solicitud HTTP normal al servidor que incluye el campo **Upgrade** para informar al servidor que se desea establecer una conexión WebSocket.

```
GET ws://websocket.example.com/ HTTP/1.1
Origin: http://example.com
Connection: Upgrade
```

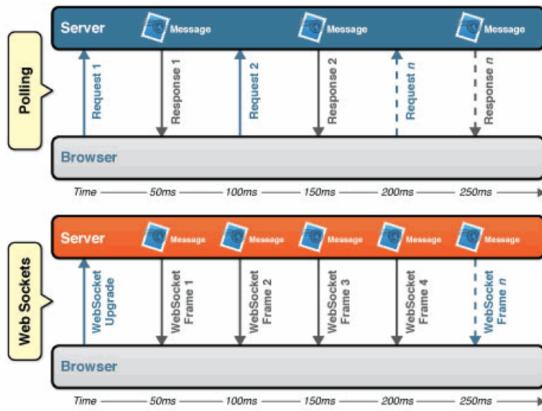


Figura 3.6: Comparación Polling-WebSockets.

```
Host: websocket.example.com
Upgrade: websocket
```

Listing 3.2: Petición conexión WebSockets.

Si el servidor soporta el protocolo WebSockets, lo comunica a través del campo **Upgrade** en la respuesta.

```
HTTP/1.1 101 WebSocket Protocol Handshake
Date: Wed, 16 Oct 2013 10:07:34 GMT
Connection: Upgrade
Upgrade: WebSocket
```

Listing 3.3: Respuesta conexión WebSockets.

Ahora que el proceso **Handshake** se ha completado, la conexión inicial de HTTP se sustituye por una conexión WebSockets que utiliza la misma conexión TCP / IP subyacente. En este punto, cualquiera de las partes puede iniciar el envío de datos.

Librería Socket.io

Socket.IO[?] es una librería que facilita el desarrollo de aplicaciones basadas en Websockets tanto en el cliente como en el servidor.

En lado servidor(NodeJS) se importa la librería por medio de la sentencia **require()**

```
var io = require('socket.io')(server);
```

Listing 3.4: Llamada libreria Socket.IO Server.

Para iniciar la conexión WebSockets se emplea el evento **io.on('connection',function())**

```
io.on('connection', function(socket) {
  console.log('Un cliente se ha conectado');
});
```

Listing 3.5: Comprobación conexión WebSockets Server.

En el lado cliente es necesario incluir el script **socket.io.js** en el fichero HTML de proyecto.

```
<script src="/socket.io/socket.io.js"></script>
```

Listing 3.6: Instancia libreria Socket.IO cliente.

Para establecer la conexión Websocket utilizamos el método **io.connect** al que se le pasa la url donde se encuentra el servidor con el que se quiere establecer la conexión.

```
var socket = io.connect('http://localhost:8080');
```

Listing 3.7: Llamada libreria Socket.IO cliente.

Una vez establecida la conexión entre el cliente y servidor pueden enviar mensajes¹ utilizamos el método **emit**, que recibe el nombre del mensaje y los datos que se envían. El receptor del mensaje utiliza el método **on** para definir un evento con el nombre del mensaje que espera recibir y un callback que se encarga de obtener los datos del mensaje.

```
//cliente
socket.emit('new-message', 'ola');

//server
socket.on('new-message', function(data) {
  console.log(data)
});
```

Listing 3.8: Ejemplo Envió-Recepción mensaje con WebSockets.

La versión 1.0 de la librería se utiliza en el desarrollo de las prácticas del **ComeCocos Multiplayer** y de **VideoConferencia con WebRTC**.

3.1.4. API File

El tratamiento de ficheros por parte de los navegadores de forma nativa no fue posible hasta la aparición de API File[12].

La API requiere de una etiqueta **<input>** para seleccionar el archivo con el que se quiere trabajar,a continuación JavaScript se encargara de obtener la información del documento a través del objeto 'FileReader()'.

```
<!DOCTYPE html>
<head>
<meta charset="utf-8">
<title>File API</title>
<script type="text/javascript">
  function processFiles(file) {
    var files = file[0];
    var reader = new FileReader();
    reader.onload = function (e) {
      /*
        e.result tiene el resultado de la
        lectura del fichero
      */
    };
  }
</script>
```

¹<https://socket.io/>

```

    reader.readAsArrayBuffer(files);
}
</script>
</head>
<body>
  Select a text file:
  <input type="file" id="fileInput" onchange="processFiles(this.files)">
</body>
</html>

```

Listing 3.9: Ejemplo API File.

Es necesario especificar el modo en el que la API va a leer la información, a continuación se resumen los métodos disponibles.

- **reader.readAsText()**: Permite leer archivos de texto y recibe dos parámetros. El primer parámetro es el objeto file o Blob que se va a leer y el segundo parámetro se utiliza para especificar la codificación del archivo.
- **reader.readAsDataURL()**: Permite leer un File o Blob y generar una URL de datos . Esto es básicamente una cadena base64 de los datos del archivo. Se puede utilizar esta URL datos para cosas como el establecimiento de la propiedad src de una imagen.
- **reader.readAsBinaryString()**: Permite leer cualquier tipo de archivo. El método devuelve los datos binarios sin formato.
- **reader.readAsArrayBuffer()**: Permite leer un File o Blob y obtener como resultado un ArrayBuffer,es decir,un buffer de datos binarios de longitud fija.

Esta API se utiliza en el desarrollo de la práctica de **VideoConferencia con WebRTC** .

3.2. JavaScript

JavaScript[16] es un lenguaje de programación que permite crear script con eventos, clases y acciones para el desarrollo de aplicaciones Internet del lado del cliente.Los usuarios no leerán únicamente las páginas sino que adquieren un carácter interactivo permitiendo cambiar las páginas dentro de una aplicación: poner botones, cuadros de texto, código para hacer una calculadora, un editor de texto, un juego, o cualquier otra cosa.

3.2.1. Propiedades

- Se interpreta por el ordenador que recibe el programa, no se compila.
- Tiene una programación orientada a objetos. El código de los objetos está predefinido y es expandible. No usa clases ni herencia.
- El código está integrado (incluido) en los documentos HTML.
- No se declaran los tipos de variables.
- Ejecución dinámica: los programas y funciones no se chequean hasta que se ejecutan.
- Los programas de JavaScript se ejecutan cuando sucede un evento.

3.3. JQuery

JQuery[17][18] es un framework Javascript que sirve como base para la programación avanzada de aplicaciones del lado del cliente aportando una serie de funciones o códigos para realizar tareas habituales.

3.3.1. Características

- efectos dinámicos.
- aplicaciones que hacen uso de Ajax.
- manipular el árbol DOM.
- manejo de eventos.
- desarrollar animaciones
- simplifica la manera de interactuar con los documentos HTML

La versión 3.2.0 de la librería se emplea en el desarrollo de todas las prácticas ya que nos permite simplificar ciertas tareas.

3.4. Bootstrap

Bootstrap[19][20] es un Framework (front-end) de twitter para desarrollo de aplicaciones web. Se basa en un sistema de grid de 12 columnas que escalan adecuadamente a medida que aumenta el tamaño del dispositivo o la ventana de visualización, además contiene elementos de diseño básicos de HTML y CSS como pueden ser barras de navegación, plantillas predefinidas, botones, desplegables entre otras extensiones de JavaScript.

Características

- Sencillo y ligero
- Basado en los últimos estándares de desarrollo de Web
- Curva de aprendizaje baja
- Compatible con todos los navegadores habituales
- Responsive web design.

La versión 3.3.7 del framework se utiliza para diseñar la apariencia de cada práctica.

3.5. NodeJS

Es un proyecto creado por Ryan Dahl a principios de 2009 orientado a la creación de aplicaciones para Internet, principalmente Web.

La idea empezó a gestarse a partir de otro proyecto para el framework Ruby on Rails, un pequeño y rápido servidor web llamado Ebb, que evolucionó a una librería en C.

Una de las razones de la evolución del proyecto desde Ruby a C, y luego de C a JavaScript fue el objetivo de realizar un sistema en que la Entrada/Salida fuera enteramente no bloqueante que es esencial para obtener un alto rendimiento. Estos dos lenguajes presentaban una parte del sistema bloqueante pero JavaScript se ajustaba a este requisito ya que esta diseñado para ejecutarse por medio de un bucle de eventos.

Si necesitamos definir a NodeJS[21] podemos hacer referencia a la definición que aparece en la web:

"Node.js es una plataforma construida encima del entorno de ejecución javascript de Chrome para fácilmente construir rápidas, escalables aplicaciones de red. Node.js usa un modelo de E/S no bloqueante dirigido por eventos que lo hace ligero y eficiente, perfecto para aplicaciones data-intensive en tiempo real"

3.5.1. Características

Fácil desarrollo y escalabilidad

Una de las ventajas de emplear JavaScript como lenguaje para las aplicaciones de *Node* es que ,al tratarse de un lenguaje con una curva de aprendizaje pronunciada permite desarrollar aplicaciones rápidamente con solo tener unas nociones básicas de las características de este lenguaje.

Uno de los aspectos con mayor impacto en cuanto a la escalabilidad es el diseño del sistema. Este es uno de los punto fuertes de *Node* ya que su arquitectura y la forma de desarrollar sus aplicaciones hacen que se cumplan los principios básicos de escalabilidad, algunos ejemplos son:

- **Sin esperas:** el tiempo que un proceso espera a que un recurso esté disponible es tiempo que otro proceso no emplea para ejecutarse
- **Lucha por los recursos:** : *Node* gestiona internamente de manera eficiente los recursos del sistema para que todas las operaciones que se demandan en el código estén satisfechas sin que se abuse de ellas.

E/S no bloqueante por eventos

Uno de los puntos críticos es el cuello de botella que afecta en alto grado al rendimiento de cualquier sistema, en especial aquellos que hacen un uso intensivo de operaciones de Entrada/Salida con ficheros y dispositivos.

Para solventar este problema *Node* utiliza un modelo de concurrencia basado en eventos. Las implicaciones que conlleva utilizar este modelo son las siguientes:

- Necesidad de un bucle de procesado de eventos que se tratará como un único proceso y que solo ejecutara un manejador ,o *callback*, a la vez.
- Emplear un lenguaje que se adapte a este modelo como es el caso de JavaScript ya que su interprete se basa un modelo idéntico.

Ligero y Eficiente

Node es una fina capa de software entre el sistema operativo y la aplicación escrita ya que con su arquitectura se persigue velocidad y eficiencia.

Centrado en este propósito descarta emplear un modelo *multithread* para manejar las distintas conexiones ya que el coste es muy elevado. Se busca entonces una solución de alto rendimiento que permita realizar operaciones Entrada/Salida no bloqueantes delegando en el sistema operativo y coordinarlo a través de uno o varios bucles de eventos.

Perfecto para aplicaciones en tiempo real

Node encaja con los requisitos que exigen las aplicaciones en tiempo real flexibles. De acuerdo a su capacidad de manejar un alto numero de conexiones y procesar un enorme numero de operaciones de Entrada/Salida muy rápido se puede afirmar que *Node* encaja perfectamente si se requiere:

- **Interfaces ligeros REST/JSON:** su modelo de Entrada/Salida para atender peticiones *REST* junto al soporte nativo *JSON* lo hacen óptimo como capa superior de fuente de datos como base de datos.
- **Aplicaciones monopágina:** la interacción del cliente con el servidor se realiza por medio de peticiones *Ajax*. El uso de *Ajax* puede producir una avalancha de peticiones que el servidor tiene que ser capaz de procesar es aquí donde *Node* entra en acción.
- **Datos por streaming:** al tratarse de conexiones HTTP como streams, permite procesar ficheros al vuelo.
- **Comunicación:** aplicaciones de mensajería instantánea o web en tiempo real, e incluso, juegos multijugador.

La versión 6.10 se utiliza en el desarrollo de las prácticas del **ComeCocos Multijugador** y de **VideoConferencia con WebRTC**.

3.6. BBDD

El objetivo principal de las bases de datos[22] es unificar los datos que se manejan y los programas o aplicaciones que los manejan. Antiguamente los programas se codificaban junto con los datos lo que desembocaba en una dependencia de los programas respecto a los datos. Además, cada aplicación utiliza ficheros que pueden ser comunes a otras sectores de la misma aplicación lo que producía redundancia en la información.

Con las bases de datos, se busca independizar los datos y las aplicaciones. Los datos residen en memoria y los programas mediante un sistema gestor de bases de datos, manipulan la información. El sistema gestor de bases de datos recibe la petición por parte del programa para manipular los datos y se encarga de recuperar la información de la base de datos y devolvérsela al programa que la solicitó.

Por lo tanto una base de datos pretende conseguir a través del Sistema Gestor de Bases de Datos(SGBD):

- **Independencia de datos:** Cambios en la estructura de la Base de Datos no modifican las aplicaciones.
- **Integridad de los datos:** Los datos han de ser siempre correctos. Se establecen una serie de restricciones (reglas de validación) sobre los datos.
- **Seguridad de los datos:** Control de acceso a los datos para evitar manipulaciones no deseadas.



Figura 3.7: Esquema Base de Datos.

3.7. Django

Consideremos el diseño de una aplicación Web escrita usando el estándar Common Gateway Interface (CGI), una forma popular de escribir aplicaciones Web alrededor del año 1998. En esa época, cuando escribías una aplicación CGI se desarrollaba todas las tareas por uno mismo. Este enfoque es válido si la aplicación solo utiliza un fichero pero a medida que una aplicación Web crece este enfoque se desmorona debido a una serie de problemas:

- ¿Qué sucede cuando múltiples páginas necesitan conectarse a la base de datos? El código de conexión a la base de datos no debería estar en cada uno de los script de CGI ya que la mejor forma de hacerlo es refactorizarlo en una función compartida.
- ¿Qué sucede cuando este código es reutilizado en múltiples entornos, cada uno con una base de datos y contraseñas diferentes? En ese punto, se vuelve esencial alguna configuración específica del entorno.
- ¿Qué sucede cuando este código es reutilizado en múltiples entornos, cada uno con una base de datos y contraseñas diferentes? En ese punto, se vuelve esencial alguna configuración específica del entorno.
- ¿Qué sucede cuando un diseñador Web que no tiene experiencia programando y desea rediseñar la página? Lo ideal sería que la lógica de la página esté separada del código HTML de la página, de modo que el diseñador pueda hacer modificaciones sin afectar la lógica del programa.

Precisamente estos son los problemas que un framework Web intenta resolver. Un framework Web provee una infraestructura de programación para las aplicaciones que permite concentrarse en escribir código limpio y de fácil mantenimiento.

En nuestro caso utilizamos como framework Django[25].

3.7.1. Patrón de diseño MVC

El diseño general de este patrón consta de los siguientes ficheros:

Models

Los modelos se representan como una clase Python dentro del fichero *models.py* que es independiente del motor de base de datos definido en la aplicación.

```
# models.py (las tablas de la base de datos)
from django.db import models

class Artista(models.Model):
    name = models.CharField(max_length=100, null=True)
    videos = models.ManyToManyField(Videos)
    galeria = models.ManyToManyField(Imagenes)
```

Listing 3.10: Ejemplo de un Modelo.

Para definir cada elemento de la clase utilizamos el objeto *models*² para definir el tipo de dato de cada elemento. En este punto destacamos la posibilidad de establecer relación entre los distintos modelos existentes por medio de los siguientes atributos:

- **models.ForeignKey:** relación 1 a 1.
- **models.ManyToManyField:** relación 1 a n.

Con la información del fichero se generan tantas tablas como clases existan aunque si en un clase existe relación con otra se construye tablas intermedias.

Vistas

Las vistas se representan como una función en Python dentro del fichero *view.py* que contiene la lógica para interactuar con la base de datos en busca de la información solicitada y responder con el resultado.

```
# views.py (la parte logica)
from django.http import HttpResponse

def EventSelect(request, evento):
    event = Evento.objects.filter(name__startswith=evento)
    return render(request, 'IndexEvent.html', {'event':event})

def IndexView(request):
    list_video=Videos.objects.all()
    return render(request, 'fullVideo.html', {'list_video':list_video})
```

Listing 3.11: Ejemplo de vistas.

Las funciones se pueden dividir en dos grupos:

- **Dinámicas:** son aquellas que reciben un parámetro adicional a través de la url configurada.
- **Estáticas:** son aquellas que reciben solo el parámetro request.

²fuente: <https://docs.djangoproject.com/en/1.10/ref/models/fields/>

Controlador

El Controlador se representa como una dupla en Python dentro del fichero *urls.py* que define las distintas rutas de acceso de la aplicación y su correspondiente vista asociada.

```
# urls.py (la configuracion URL)
from django.conf.urls.defaults import patterns, url
from .views import ...

urlpatterns = patterns('',
    url(r'^index/$', views.MainPage),
    url(r'^eventos/(?P<evento>\w{1,50})/$', views.EventSelect),
)
```

Listing 3.12: Ejemplo de url's.

Al igual que en las vistas las URLs pueden ser de dos tipos:

- **Dinámicas:** son aquellos que incluyen un parámetro dentro del path que influye en respuesta.
- **Estáticas:** son aquellas definen el path con texto único.

Plantilla

Se tratan de ficheros html en que se renderiza con información de vistas correspondiente a cada petición. Para plasmar esta información utilizamos el lenguaje de plantillas³ que es similar a un lenguaje de programación mas limitado ya que posee bucles, sentencias condicionales entre otras características.

```
# latest_books.html (la plantilla)
<!DOCTYPE html>
<html>
<head>
<title>Hora actual</title>
</head>
<body>
<h1>Bienvenidos</h1>
<p>Dia y hora actual: {{ fechahora\_actual }}</p>
{ % for cantante in evento.artistas.all %}
<div>
<a href="pulpitrock.jpg" class="thumbnail">{{cantante.name}}</a>
</div>
{ % endfor %}
{ % block info %}
{ % endblock %}
</body>
</html>
{ % extends 'IndexApp.html' %}
```

Listing 3.13: Ejemplo plantilla .

En el lenguaje de plantillas se definen los siguientes modos de acceso:

³<https://docs.djangoproject.com/en/1.10/ref/templates/>

- El contenido encerrado entre {{ }} muestra el valor de una variable enviada por la vista correspondiente.
- El contenido encerrado entre { % %} hace referencia a las etiquetas de plantilla permitiendo dotar a las plantillas de algún mecanismo simple de programación.
- Se puede establecer un bloque a través de la etiqueta { % block name sección %} y { % endblock %} que se utilizan al establecer herencia entre las plantillas.

Relacionadas en su conjunto, estas piezas se aproximan al patrón de diseño Modelo-Vista-Controlador(MVC).Este patrón permite desarrollar software en la que el código que interacciona con los datos (el modelo) este separado de la asignación de rutas (el controlador) y a su vez separado del interfaz del usuario (la vista).

Una ventaja clave de este enfoque es que los componentes tienen un acoplamiento débil entre sí. Eso significa que cada pieza de la aplicación Web que funciona sobre Django tiene un único propósito por lo que puede ser modificada independientemente sin afectar a las otras piezas.



Figura 3.8: Esquema MVC Django.

La versión 1.9 de este frameWork se utiliza para desarrollar la práctica de **Sitio Web de una tienda**.

3.8. WebRTC

Permite configurar conexiones peer-to-peer entre navegadores web permitiendo transmitir fácilmente contenido de audio y vídeo a millones de personas.Para construir una aplicación de este tipo desde cero, sería necesario una gran cantidad bibliotecas que se ocupan de problemas típicos como la pérdida de datos, caída de conexiones y la NAT traversal pero WebRTC[15] incorpora de forma nativa las soluciones a estos problemas.

3.8.1. Protocolos

Eso También necesita ser fácilmente transportable. En pocas palabras, necesitamos un perfil basado en cadenas con información sobre el dispositivo del usuario. Aquí es donde entra SDP.

SDP

SDP(The Session Description Protocol) es una parte importante del WebRTC. Es un protocolo que pretende describir las sesiones de comunicación ya que no entrega los datos de los medios sino que se utiliza para la negociación de codecs de audio y vídeo, topologías de red y otra información del dispositivo.

SDP es un método bien conocido para establecer conexiones con los medios de comunicación de los 90s. Se ha utilizado en una gran cantidad de otros tipos de aplicaciones antes de WebRTC como teléfono y conversaciones basadas en texto.

Se puede definir SDP como una cadena de datos que contiene conjuntos de pares clave-valor, separados por saltos de línea:*Key = value*. Este protocolo es la primera parte de la conexión entre nodos ya que dicha información se tiene que intercambiar por medio del canal de señalización para finalmente establecer la conexión, figura 3.9.

```

descripción de la sesión
v=0
o=mozilla...THIS_IS_SOPARTA-46.0.1 4867651871611090737 0 IN IP4 0.0.0.0
s=-
t=0 0
a=fingerprint:sha-256
22f79181:15:03:1f:8a:FF:07:11:17:79:9f:BA:A9:7A:C3:E6:33:98:0E:3C:30:41:40:EF:5A:7D:3E:70:CE:A6
a=group:BUNDLE sdp0_a_sdparts_1
a=ice-options:trickle
a=msid-semantic:WSN
m=video 9 UDP/TLS/RTP/AVP 128 126 97
c=IN IP4 0.0.0.0
a=sendrecv
a=fmtp:128 profile-level-id=42e0bf;level-asymmetry-allowed=1;packetization-mode=1
a=fmtp:197 profile-level-id=42e0bf;level-asymmetry-allowed=1
a=fmtp:128 max-fs=12288;max-fr=60
a=ice-pwd:2a534092a4f34a5ba94da255ee4984f
a=ice-ufrag:cd2dc9ac
a=mid:sdp0_a_sdparts_0
a=msid:(e0db156c-5c3d-44b1-a42e-2d757d297b92) (7a01e077-b75f-4b77-93ff-6d5ea6d20e2b)
a=rtpmap:97:128 nack
a=rtpmap:97:128 nack pli
a=rtpmap:97:128 ccm fir
a=rtpmap:97:128 nack
a=rtpmap:97:128 nack
a=rtpmap:97:128 nack pli
a=rtpmap:97:128 ccm fir
a=rtpmap:97:128 nack
a=rtpmap:126 VP8/90000
a=rtpmap:126 H264/90000
a=rtpmap:97 H264/90000
a=rtpmap:97:126 nack
a=ssrc:3784998849 cname:(a79cf374+f9bd-4378-9226-3d0f54982415)
m=application 9 DTLS/SCTP 5000
c=IN IP4 0.0.0.0
a=sendrecv
a=ice-pwd:2a534092a4f34a5ba94da255ee4984f
a=ice-ufrag:cd2dc9ac
a=mid:sdp0_a_sdparts_1
a=sctpmap:5000 webrtc-datachannel 256
a=setup:actpass
a=$ssrc:1269130962 cname:(a79cf374+f9bd-4378-9226-3d0f54982415)
```

Figura 3.9: Ejemplo Protocolo SDP.

STUN

STUN(Session Traversal Utilities for NAT) ayuda a identificar a cada usuario y encontrar una buena conexión entre ellos. En primer lugar realiza una solicitud a un servidor, habilitado con el protocolo STUN el cual devuelve la dirección IP del cliente.

El cliente ahora puede identificarse con esta dirección IP para establecer conexión. Así que básicamente hay dos pasos que seguir, figura 3.10⁴.

TURN

En ocasiones hay un firewall que no permite ningún tráfico basado en STUN con el otro usuario. Aquí es donde TURN(Traversal Using Relays around NAT) sale como un método diferente de conectar con otro usuario.

TURN funciona como un distribuidor entre los clientes. El usuario obtiene entonces sus

⁴fuente: https://www.tutorialspoint.com/webrtc/webrtc_tutorial.pdf

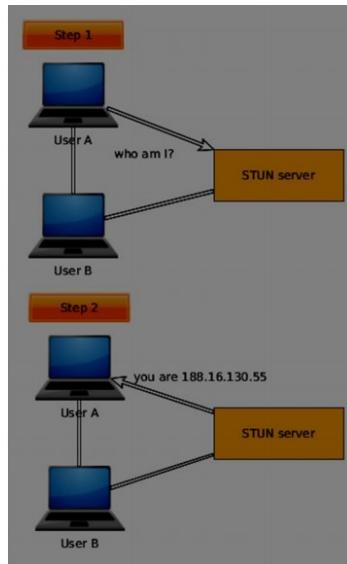


Figura 3.10: Ejemplo petición-respuesta STUN.

datos del servidor TURN. A continuación, el servidor TURN obtendrá y redirigirá cada paquete de datos que se envía a él para cada usuario. Por eso, es el último recurso cuando no hay alternativas, figura 3.11⁵.

ICE

Ahora podemos ver cómo STUN y TURN se unen a través de ICE (Interactive Connectivity Establishment) para proporcionar una conexión peer to peer exitosa. ICE encuentra y prueba en orden un rango de direcciones que funcionarán para ambos usuarios.

Cuando ICE comienza, no sabe nada sobre la red de cada usuario por lo que a través de un conjunto de etapas de forma incremental descubre cómo se configura la red de cada cliente. La tarea principal es encontrar suficiente información sobre cada red para poder tener éxito en la conexión.

STUN y TURN se usan para encontrar a cada candidato de ICE. ICE utilizará el servidor STUN para encontrar una IP externa. Si la conexión falla, intentará utilizar el servidor TURN.

SCTP

Tras establecer la conexión, tenemos la capacidad de enviar rápidamente datos de vídeo y audio. El protocolo SCTP(Stream Control Transmission Protocol) se utiliza para enviar datos(blob) en la parte superior de nuestra conexión al utilizar el objeto RTCDataChannel. SCTP se basa en el protocolo DTLS (Datagram Transport Layer Security) que se implementa para cada conexión WebRTC. Todo esto se sitúa encima del protocolo UDP que es el protocolo de transporte base para todas las aplicaciones WebRTC, figura 3.12⁶.

⁵fuente: https://www.tutorialspoint.com/webrtc/webrtc_tutorial.pdf

⁶fuente: https://www.tutorialspoint.com/webrtc/webrtc_tutorial.pdf

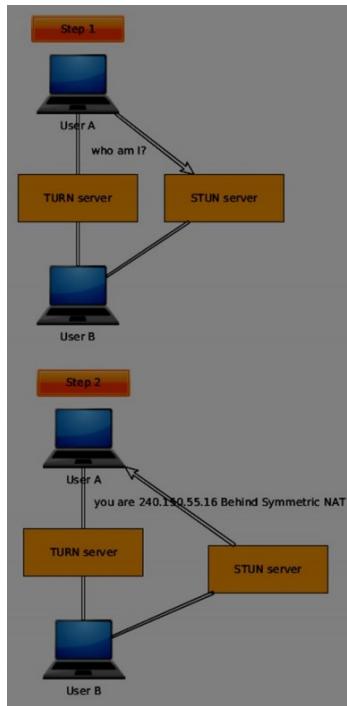


Figura 3.11: Ejemplo petición-respuesta TURN.

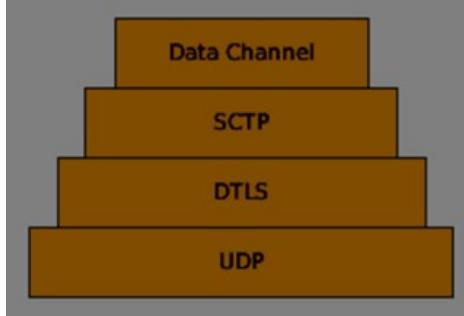


Figura 3.12: Capas de SCTP.

Los desarrolladores de WebRTC sabían que cada aplicación sería única al usar el canal de datos. Algunos podrían querer el alto rendimiento de UDP, mientras que otros pueden necesitar la entrega fiable de TCP. Es por eso que se creó el protocolo SCTP que posee las siguientes características:

- Existen dos modos de transporte: fiable y no fiable.
- La capa de transporte está protegida
- Cuando se transportan mensajes de datos, se permite que se descompongan y vuelvan a montarse en el otro lado.

- El control del flujo y de la congestión se proporciona a través de la capa de transporte.

3.8.2. MediaStream APIs

Es una API diseñada para acceder fácilmente a los flujos de datos de las cámaras y micrófonos locales. El método getUserMedia() es la forma principal para acceder al flujo de datos de los dispositivos. La API tiene las siguientes características:

1. Un flujo de stream en tiempo real está representado por un objeto stream en forma de vídeo o audio.
2. Proporciona un nivel de seguridad ya que pide permiso a los usuarios para acceder a los elementos.

3.8.3. RTCPeerConection APIs

RTCPeerConection es el núcleo de la conexión peer-to-peer entre cada uno de los navegadores. Para crear el objeto RTCPeerConnection se realiza la siguiente llamada.

```
var pc = new RTCPeerConnection(config);
```

Listing 3.14: Instancia RTCPeerConnection.

Donde el argumento config contiene al menos la clave, iceServers, que es una matriz de objetos URL que contiene información sobre los servidores STUN y TURN utilizados durante la búsqueda de los candidatos ICE.

Propiedades

1. **RTCPeerConnection.localDescription(read only)**: Devuelve un objeto RTCSessionDescription que describe la sesión local.
2. **RTCPeerConnection.remoteDescription(read only)**: Devuelve un objeto RTCSessionDescription que describe la sesión remota.

Controladores de eventos

1. **RTCPeerConnection.onaddstream**: Se activa cada vez que se añade un objeto MediaStream por el par remoto.
2. **RTCPeerConnection.ondatachannel**: Se activa cuando se incluye el canal de datos en el par remoto.
3. **RTCPeerConnection.onicecandidate**: Se activa cuando se agrega un objeto RTCIceCandidate.
4. **RTCPeerConnection.onremovestream**: Se activa cuando se quita un objeto MediaStream de la conexión.

Métodos

1. **RTCPeerConnection()**: Devuelve un nuevo objeto evento RTCPeerConnection.
2. **RTCPeerConnection.createOffer(HandlerOffer, HandlerError, Options)**: Crea una solicitud de oferta para encontrar un peer remoto. Los dos primeros parámetros de este método son los retornos de la función de éxito y error. El tercer parámetro es opcional.
3. **RTCPeerConnection.createAnswer(HandlerAnswer, HandlerError, Options)**: Crea una respuesta a la oferta recibida por el peer remoto durante el proceso de negociación (oferta/respuesta). Los dos primeros parámetros de este método son los retornos de la función de éxito y error. El tercer parámetro es opcional.
4. **RTCPeerConnection.setLocalDescription()**: Cambia la descripción de la conexión local. El método toma tres parámetros, el objeto RTCSessionDescription y los retornos de la función de éxito y fallo.
5. **RTCPeerConnection.setRemoteDescription()**: Cambia la descripción de la conexión remota. El método toma tres parámetros, el objeto RTCSessionDescription y los retornos de la función de éxito y fallo.
6. **RTCPeerConnection.addStream()**: Añade un objeto MediaStream como una fuente local de vídeo o audio.
7. **RTCPeerConnection.addIceCandidate()**: proporciona un candidato remoto al agente ICE.
8. **RTCPeerConnection.createDataChannel()**: crea un nuevo objeto RTCDATAChannel.
9. **RTCPeerConnection.close()**: finaliza la conexión.

3.8.4. RTCDATAChannel APIs

WebRTC no sólo es bueno para transferir secuencias de audio y video, sino cualquier dato que tengamos. Aquí es donde entra en juego el objeto RTCDATAChannel.

Propiedades

1. **RTCDATAChannel.label(read only)**: Devuelve el nombre del canal de datos.
2. **RTCDATAChannel.protocol(read only)**: Devuelve una cadena con el nombre de sub-protocolo utilizado para este canal.
3. **RTCDATAChannel.readyState(read only)**: Devuelve el estado de la conexión. Los posibles valores:
 - connecting: Indica que la conexión aún no está activa.
 - open: Indica que la conexión se está ejecutando
 - closing: Indica que la conexión no se pudo establecer o se ha cerrado

Controladores de eventos

1. **RTCDataChannel.onopen:** Se activa cuando se ha establecido la conexión de datos.
2. **RTCDataChannel.onmessage:** Se activa cuando está disponible un mensaje en el canal de datos.
3. **RTCDataChannel.onclose:** Se activa cuando la conexión se ha cerrado.
4. **RTCDataChannel.onerror:** Se activa cuando se produce un fallo.

Métodos

1. **RTCDataChannel.close():** Cierra el canal de datos.
2. **RTCDataChannel.send():** Envía los datos pasados como parámetro a través del canal. Los datos pueden ser blob, cadenas de texto, un ArrayBuffer o un ArrayBufferView.

3.8.5. Servidor Señalización

Para conectarse con otro usuario debe saber la dirección IP de su dispositivo como se ha mencionado anteriormente de esta tarea se encarga RTCPeerConnection. Tan pronto como los dispositivos saben encontrarse a través de Internet, comienza el intercambio de datos sobre qué protocolos y códecs soporta cada dispositivo.

El proceso de conexión con el otro usuario también se conoce como señalización y negociación. Consta de unos pocos pasos:

1. Cree una lista de candidatos potenciales para una conexión entre iguales.
2. El usuario o una aplicación selecciona un usuario con el que establecer una conexión.
3. La capa de señalización notifica a otro usuario que alguien quiere conectarse con él. Él puede Aceptar o rechazar.
4. Se notifica al primer usuario la aceptación de la oferta.
5. Ambos usuarios intercambian información de software y hardware a través del servidor de señalización.
6. Ambos usuarios intercambian información de ubicación.
7. La conexión tiene éxito o falla.

La especificación WebRTC no contiene ningún estándar sobre el intercambio de información, así que se puede utilizar cualquier protocolo o tecnología para crear el servidor de señalización, figura 3.13.

Se utiliza la versión 1.0 en el desarrollo de la práctica de **Videoconferencia con WebRTC**.

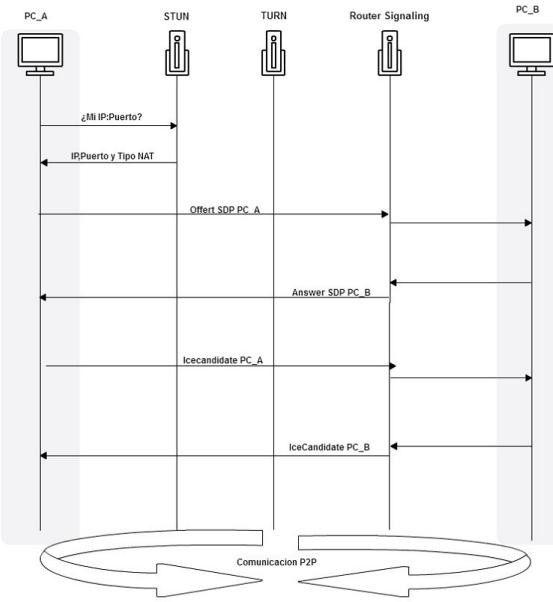


Figura 3.13: Proceso del Servidor Señalización.

3.9. Web Services

Es un estándar de comunicación entre procesos diseñado para ser multiplataforma y multilenguaje, es decir, no importa en qué lenguaje este diseñado el *Web Services*[23], o en qué plataforma se ejecuta ya que es accesible y utilizable por otras aplicaciones desarrolladas en otras plataformas o lenguajes.

Antiguamente se utilizaban otros tipos de estándares como **DCOM** (*Distrinuted Component Object Model*) y **CORBA** (*Common Object Request Broker Architecture*). Estos estándares presentaban graves problemas de configuración en entornos que se encontraban *Firewall* de por medio ya que era imposible habilitar cierto puertos por temas de seguridad. Por lo que se prefería utilizar el puerto 80 de *HTTP*, que normalmente se encontraba habilitado debido al uso de navegadores y servidores Web. De esta forma *HTPP* se convirtió en el protocolo preferido para el transporte de mensajes de los *Web Services*.

3.9.1. Características

1. **Combinación:** Las operaciones de un servicio web pueden utilizar otros servicios web para sus operaciones.

2. Patrones de comunicación

- **Petición-respuesta síncrona:** Invocamos al servicio y esperamos la respuesta a la petición.
- **Comunicación asíncrona:** Se envía la petición y se continúa la ejecución.
- **Mediante eventos:** El cliente se suscribe a eventos ofrecidos por el servicio.

3. **Desacoplamiento:** Se refiere a minimizar las dependencias entre los servicios para ofrecer una mayor flexibilidad en la arquitectura.
4. **Representación de mensajes**
 - **Textual:** SOAP representa los servicios y los mensajes en XML.
 - **Binario:** Los datos ocupan menos espacio aunque son ilegibles.
5. **Referencia y activación del servicio:** Los servicios se referencia generalmente mediante una URL, que se conoce como punto final (endpoint). El servicio Web puede ejecutarse en la máquina de punto final, o en servidores secundarios.
6. **Transparencia:** Protege al programador de los detalles de la representación de los datos y asemeja una petición local a una remota.

3.9.2. Tipos WebServices

SOAP[24] (Simple Object Access Protocol) es el protocolo base de los Web Services. Este protocolo está basado en XML y no se encuentra atado a ninguna plataforma o lenguaje de programación.

Si bien es un protocolo, este no es un protocolo de comunicación entre mensajes como lo es HTTP. Básicamente, SOAP son documentos XML que necesitan utilizar algún otro protocolo para ser transmitidos como puede ser HTTP o cualquier otro tipo de protocolo. Consta de tres componentes principales:

1. **WSDL:** lenguaje de descripción del servicio.
2. **HTTP/SMTP:** protocolo de comunicación.
3. **XML:** lenguaje de peticiones y respuestas.

REST(Representational State Transfer) intentan emular al protocolo HTTP o protocolos similares mediante la restricción de establecer el interfaz en un conjunto conocido de operaciones estándar (por ejemplo GET, PUT,...). Por tanto, este estilo se centra más en interactuar con recursos con estado, que con mensajes y operaciones.

Cabe destacar que REST no es un estándar, ya que es tan solo un estilo de arquitectura. Aunque REST no es un estándar, está basado en estándares:

- HTTP
- URL
- Representación de los recursos: XML/HTML/GIF/JPEG/...
- Tipos MIME: text/xml, text/html, ...

3.9.3. WebServices Google Maps

En el desarrollo de la práctica **Sitio Web de una tienda** vamos a implementar el WebServices de Google Maps⁷ que sigue la estructura REST. Consta de un conjunto de interfaces HTTP que proporcionan datos geográficos para aplicaciones que utilizan mapas.

⁷Descripción WebServices: <https://developers.google.com/maps/web-services/overview>

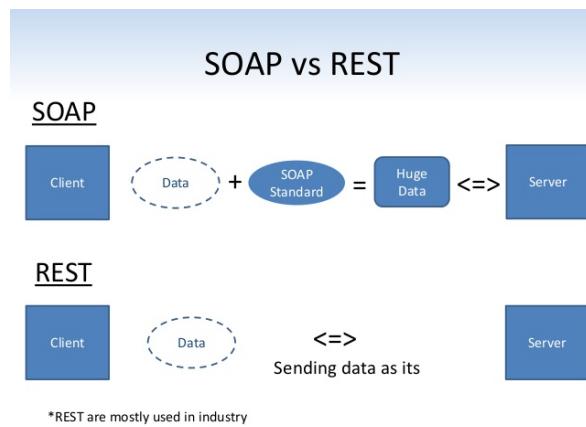


Figura 3.14: Comparativa Soap y Rest.

Antes de establecer la comunicación con el WebServices es necesario obtener una clave⁸ asociado a la aplicación que implementa las peticiones. De los numerosos servicios de los que dispone nos centramos en utilizar dos.

1. **Geocode**⁹: permite conocer las coordenadas geográficas de un lugar a partir del nombre. La petición se realiza a la URL definida en la que se incluye el formato de los datos(json o xml),el dato que se envía y la clave asociada a la aplicación.

```
https://maps.googleapis.com/maps/api/geocode/json?address=Madrid&key=YOUR_API_KEY
```

Listing 3.15: Instancia RTCPeerConnection.

2. **Place**¹⁰: permite obtener información asociada a un servicio (hoteles, restaurante,...) tomando como punto de referencia una coordenada geográfica. La petición se realiza a la URL definida en la que se incluye el formato de los datos(json o xml),el dato que se envía y la clave asociada a la aplicación.

```
https://maps.googleapis.com/maps/api/place/nearbysearch/json?location=-33.8670,151.1957&radius=500&types=food&key=YOUR_API_KEY
```

Listing 3.16: Instancia RTCPeerConnection.

⁸<https://developers.google.com/maps/documentation/geocode/get-api-key>

⁹Descripción API Geocode :<https://developers.google.com/maps/documentation/geocoding/>

¹⁰Descripción API Place :<https://developers.google.com/places/web-service/search>

Capítulo 4

Comecocos Web

4.1. Enunciado

En la actualidad gracias a las novedades introducidas en la Web han aumentando los juego desarrollado sin la necesidad de pugins o software de terceros para su ejecución. Por ello en esta primera practica se pide desarrollar un juego con herramientas nativas de la web.Dentro de las varias opciones existentes el juego seleccionado es Pac-Man(comecocos).



Figura 4.1: Aspecto clásico Pacman.

Requisitos Se presenta los distintos elemento que forman parte del juego y la función que deben desarrollar.

1. **Escenario:** Contiene todos los elementos estáticos del juego con los que interactua Pacman como pueden ser los obstáculos y el contorno del escenario que no pueden ser sobrepasados y los cocos que es la comida de Pacman.Ademas informara al usuario del tiempo y puntos a medida que progrese la partida.
2. **Pacman:** se trata del personaje principal del juego que controle el usuario.

3. **Fantasmas:** son los enemigos que persiguen a Pacman durante la partida. En total serán 4 fantasmas¹ que ha diferencia de Pacman no interactúan con los elementos del juego ya que solo basan su comportamiento en la posición de Pacman, esto es conocido como modo de persecución por lo que es necesario aplicar un algoritmo básico de Inteligencia Artificial.

Tecnologías Sera necesario emplear las etiquetas canvas y audio ademas de la API LocalStorage de HTML5, JavaScript e inteligencia artificial en juegos.

4.2. Diseño

Una vez se ha planteado la práctica pasamos a definir el esquema que se va a seguir en la etapa del desarrollo.

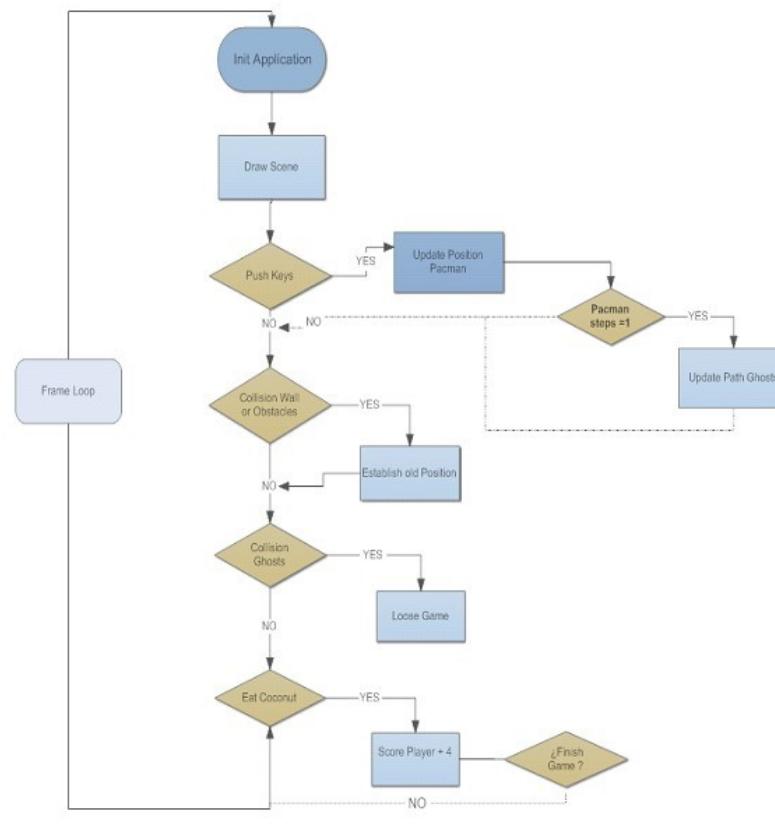


Figura 4.2: Diseño comecocos web.

¹fuente:<http://gameinternals.com/post/2072558330/understanding-pac-man-ghost-behavior>

4.3. Desarrollo

Como se ha dicho a lo largo de este desarrollo la lógica del juego recae sobre JavaScript. Por esto se han creado tres tipos de objetos (GameArea, Pacman, Ghost) con el objetivo de hacer más compacto el código y menos repetitivo ya que hay muchas funciones que tienden a repetirse.

4.3.1. Game Area

Este objeto es el encargado de generar y dibujar los elementos del juego. El objeto inicializa una serie de variables que hacen referencia a dichos elementos.

```
value_cuad : 40,
filas:19,
columnas:18,
canvas : document.createElement("canvas"),
image :new Image(),
img_loose : new Image(),
img_win : new Image(),
/* time */
seconds : 0,
min : 0,
horas :0,
time : '00'+':'+'00'+':'+'00',
state : 0,
score :0,
start_crono : false,
lifePlayer : 1,
shape_1 : [...],
shape_2 : [...],
list_cocos : [...],
list_obstaculos : [...],
var map: [...]
```

Listing 4.1: Iniciacion de variables del Obj.Game

Tras la creación de las variables es necesario dotar al objeto de una serie de funciones que permitan dibujar los elementos.

Start

Inicializa el tamaño del lienzo y obtiene el contexto de *canvas* para poder dibujar los elementos dentro de el. Ademas obtenemos la referencia a las imágenes y fuentes de audio que vamos a utilizar.

```
start : function() {
  this.img_loose.src ='game_over.jpeg';
  this.img_win.src = 'game_win.jpg';
  this.image.src = 'pacman_fruit.png';
```

```

this.canvas.width = this.value_cuad*this.filas;
this.canvas.height = this.value_cuad*(this.columnas+4);
this.context = this.canvas.getContext("2d");
document.body.insertBefore(this.canvas,
document.body.childNodes[2]);
this.AudioGame = document.getElementById('musica');
this.AudioDied = document.getElementById('hitPacman');
this.AudioEat = document.getElementById('eating');
},

```

Listing 4.2: Iniciacion de variables del Obj.Game

Escenario y elementos

La función **shape_scene** dibuja el escenario del juego por medio de un array con los comandos que se necesitan ejecutar.

```

shape_scene : function(shape) {
this.context.save();
this.context.beginPath();
for (var i = 0; i < shape.length; i++) {
  var elemento = shape[i];
  for (var propiedad in elemento) {
    var x = elemento[propiedad];
    if(x.moveTo) {
      this.context.moveTo(this.value_cuad*x.moveTo[0],
        this.value_cuad*x.moveTo[1]);
    }else if(x.lineTo) {
      this.context.lineTo(x.lineTo[0]*this.value_cuad,
        this.value_cuad*x.lineTo[1]);
    }
  }
}
this.context.strokeStyle = 'blue';
this.context.lineWidth = 2;
this.context.stroke();
this.context.restore();
},

```

Listing 4.3: Visualizacion escenario.

Para dibujar los obstáculos definimos la función **draw_obstaclos** que lee el contenido de la variable **list_obstaculos**. Cada elemento esta formado por las coordenadas y dimensiones que se multiplican por el tamaño que tiene cada cuadro del escenario.

```

draw_obstaclos : function() {
  for(var i=0;i<this.list_obstaculos.length;i++) {
    var elemento = this.list_obstaculos[i];
    this.context.fillRect(elemento.x*value_cuad,elemento.y*value_cuad,
      elemento.width*value_cuad,elemento.height*value_cuad);
  }
},

```

Listing 4.4: Visualizacion obstaculos.

Los últimos elementos son los cocos que por medio de la función `draw_cocos` se dibujan. La función lee el contenido de la variable `list_cocos`, donde cada elemento tiene la misma estructura que los elementos de la variable `list_obstaculos`.

```
draw_cocos : function() {
    if(this.list_cocos.length > 0) {
        for(var i=0;i<this.list_cocos.length;i++) {
            var elemento = this.list_cocos[i];
            this.context.fillStyle = 'white';
            this.context.beginPath();
            this.context.arc((elemento.x*40)+20,(elemento.y*40)+20,
                elemento.radio,0,(Math.PI/180),true);
            this.context.fill();
        }
    }
},
```

Listing 4.5: Visualizacion cocos.

En la imagen 4.3 se muestra el aspecto que tiene el escenario al utilizar las funciones anteriores.

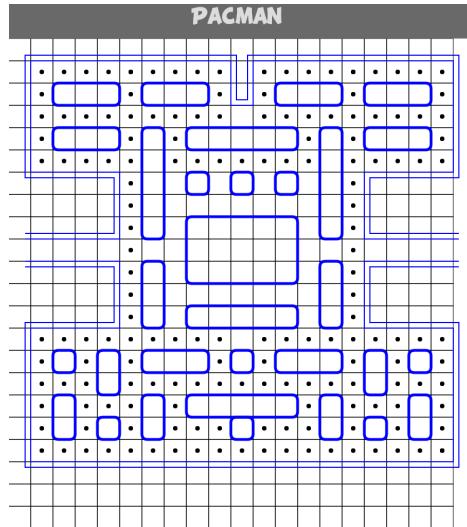


Figura 4.3: Apariencia elementos del juego.

Información de la partida

Otra característica que presenta el objeto es mostrar información sobre el estado de la partida.

La función `time_game` se encarga de dibujar el cronómetro del juego. Para ello dibuja el valor que contiene la variable `time`. El valor de esta variable se actualiza por medio de la función `CronoTime` que se explica mas adelante.

```
time_game : function() {
    this.context.save();
    this.context.font = "30px BDCartoonShoutRegular";
    roundedRect(this.context, 8*40, 20*40, 5*40, 1*40, 10);
    this.context.fillStyle = "#00FA9A";
    this.context.fillText(this.time, 8.75*40, 20.80*40);
    this.context.restore();
},
```

Listing 4.6: Visualizacion cronometro.

También dibuja la información del marcador del usuario por medio de la función **score_user** que utiliza la información de variable **score**.

```
score_user : function() {
    this.context.save();
    this.context.fillStyle = "white";
    this.context.font = "30px BDCartoonShoutRegular";
    this.context.fillText('score: '+this.score, 1*40, 20*40);
    this.context.restore();
},
```

Listing 4.7: Visualizacion Marcador.

Por ultimo, la función **life_user** se encarga de dibujar el contenido de la variable **life**.

```
life_user : function() {
    this.context.save();
    this.context.fillStyle = "white";
    this.context.font = "30px BDCartoonShoutRegular";
    this.context.fillText('life: '+this.life, 1*40, 21*40);
    this.context.restore();
},
```

Listing 4.8: Visualizacion vidas.

En la imagen 4.4 se muestra el aspecto que tiene el escenario al utilizar las funciones anteriores.

Fin de Partida

La partida puede terminar cuando Pacman ha logrado comerse todos los cocos de la partida lo que produce la ejecución de la función **win_game** que carga una imagen indicando al usuario que ha ganado y permite guardar la información de la partida a través de una función **PosMouseClick** que se explica mas adelante.

```
win_game : function() {
    var finish = false;
    if(this.list_cocos == 0) {
        this.context.globalAlpha = 0.9;
        this.context.save();
        this.context.drawImage(this.img_win, 0, 0,
            this.canvas.width,this.canvas.height);
        roundedRect(this.context, this.canvas.width/2-(4*40), 13*40,
```

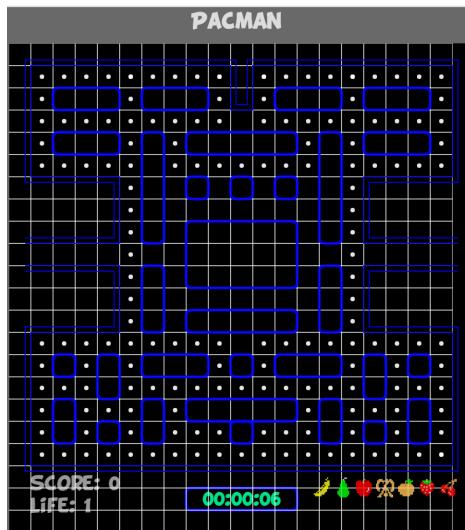


Figura 4.4: Apariencia información del juego.

```

    8*40,1.5*40,10);
this.context.fillStyle = "#00FA9A";
this.context.font = "30px BDCartoonShoutRegular";
this.context.fillText('Save Score',
    this.canvas.width/2-(2.5*40),14*40);
this.context.restore();
finish = true;
}
return finish
},

```

Listing 4.9: Visualizacion Partida Ganada.

Otro forma de terminar la partida se produce cuando los fantasmas capturan a Pacman lo que produce la ejecución de la función `lose_game` que carga una imagen indicando al usuario que ha perdido.

```

lose_game : function() {
this.context.save();
this.context.globalAlpha = 0.6;
this.context.drawImage(this.img_loose,0,0,
    this.canvas.width,this.canvas.height);
this.context.restore();
},

```

Listing 4.10: Visualizacion Partida Perdida.

4.3.2. Pacman

Es el protagonista del juego y por ende el usuario interactúa con el moviéndolo por todo el escenario. En su lógica tenemos que tener en cuenta diversos factores que afectan a

su progreso por el juego.

Detectar colisiones

Tenemos que tener en cuenta el entorno del juego con respecto a la posición en la que se encuentra Pacman. Para ello la función **hitt_Counter** evalúa que las coordenadas(x,y) no sobrepasen el contorno del escenario.

```
this.hitt_counter = function() {
    var hitt_counter = false;
    if(this.x < 0 || this.x > GameArea.filas) {
        hitt_counter = true;
    } else if(this.y < 0 || this.y > GameArea.columnas) {
        hitt_counter = true;
    }
    return hitt_counter;
}
```

Listing 4.11: Detección de colisiones con el escenario.

Otro tipo de colisión a tener en cuenta es con los obstáculos, de esto se encarga la función **hitObject**. La función obtiene los vértices del objeto y de Pacman para comprobar si alguno de los vértices de Pacman se encuentran dentro del área que forman los vértices del objeto.

```
this.hitObject = function(list) {
    var hitt = false;
    for(var i = 0;i<list.length;i++) {
        var obstaculo = list[i];
        var Aobstaculo = {x:obstaculo.x,y:obstaculo.y};
        var Bobstaculo = {x:obstaculo.x+obstaculo.width,
                          y:obstaculo.y};
        var Cobstaculo = {x:obstaculo.x,
                          y:obstaculo.y+obstaculo.height};
        var Dobstaculo = {x:obstaculo.x+obstaculo.width,
                          y:obstaculo.y+obstaculo.height};
        var Apac = {x:this.x,y:this.y};
        var Bpac = {x:this.x+this.width,y:this.y};
        var Cpac = {x:this.x,y:this.y+this.height};
        var Dpac = {x:this.x+this.width,y:this.y+this.height};
        if(Apac.x > Aobstaculo.x && Apac.x < Bobstaculo.x &&
           Apac.y > Aobstaculo.y && Apac.y < Cobstaculo.y) {
            hitt = true;
            break;
        }
        if(Bpac.x > Aobstaculo.x && Bpac.x < Bobstaculo.x &&
           Bpac.y > Aobstaculo.y && Bpac.y < Cobstaculo.y) {
            hitt = true;
            break;
        }
        if(Cpac.x > Aobstaculo.x && Cpac.x < Bobstaculo.x &&
           Cpac.y > Aobstaculo.y && Cpac.y < Cobstaculo.y) {
            hitt = true;
            break;
        }
    }
}
```

```

if(Dpac.x > Aobstaculo.x && Dpac.x < Bobstaculo.x &&
Dpac.y > Aobstaculo.y && Dpac.y < Cobstaculo.y) {
    hitt= true;
    break
}
}
if(hitt == true) {
    this.move = false;
}
}
}

```

Listing 4.12: Detección de colisiones con objetos del juego.

En la imagen 4.5 se muestra la colisión con un objeto impidiendo a Pacman avanzar.

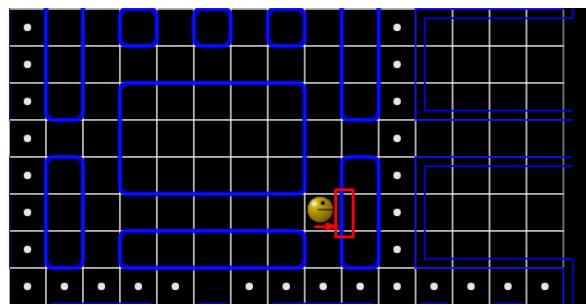


Figura 4.5: Colisión Pacman-Obstáculo.

Comer Cocos

Los cocos se encuentran por todo el escenario por lo que es necesario comprobar si hay colisión con ellos.

La función **eat_doit** calcula los vértices de pacman para evaluar si la posición del coco esta dentro del área formada por los vértices anteriores. En caso afirmativo se elimina este elemento de la variable **GameArea.score** para no volverlo a ser dibujado.

```

this.eat_doit = function(list_cocos) {
    var eat = false;
    for(var i=0;i<list_cocos.length;i++) {
        var coco = list_cocos[i];
        var Apac = {x:this.x,y:this.y};
        var Bpac = {x:this.x+this.width,y:this.y};
        var Cpac = {x:this.x,y:this.y+this.height};
        var Dpac = {x:this.x+this.width,y:this.y+this.height};
        if (coco.x+0.5 > Apac.x && coco.x+0.5 < Bpac.x &&
            coco.y+0.5 > Apac.y && coco.y+0.5 < Cpac.y ) {
            list_cocos.splice(i,1);
            eat = true
            GameArea.score = GameArea.score +4 ;
            break;
        }
    }
}

```

```

    return eat;
}

```

Listing 4.13: Detección de colisión con los cocos.

En la imagen 4.6 se muestra la colisión con un coco lo que provoca que dicho coco desaparezca.

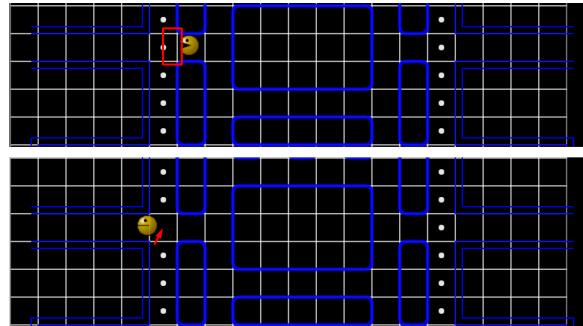


Figura 4.6: Colisión Pacman-Cocos.

Actualizar posición Canvas

Para actualizar la posición de Pacman la función **new_position** suma los nuevos valores que han sido asignados al haber pulsado una tecla en las variables **x_speed** e **y_speed**.

```

this.new_position = function () {
  this.x += this.speed_x;
  this.y += this.speed_y;
}

```

Listing 4.14: Actualizar posición en canvas.

Tras esto, la función **add_steps** actualiza el valor de la variable **pasos** que es un contador definido para evaluar si su valor es +/-1 lo que indica que se ha desplazado a una nueva casilla del mapa.

```

this.add_steps = function() {
  if(this.speed_x != 0 || this.speed_y != 0) {
    if(this.type_move == 'y_pos' || this.type_move == "y_neg") {
      this.pasos += this.speed_y;
    } else{
      this.pasos += this.speed_x;
    }
  }
}

```

Listing 4.15: Actualización de pasos dados.

Actualizar posición Mapa

Actualizamos el valor de las variables **x_map** e **y_map** según corresponda y restablecemos el valor de la variable **pasos**.

```
this.new_path = function() {
    if(this.x_map < 22 && this.x_map >= 0) {
        if(this.y_map < 19 && this.y_map >= 0) {
            if(this.type_move == 'y_pos') {
                this.y_map += 1;
            }else if(this.type_move == 'x_pos') {
                this.x_map += 1;
            }else if(this.type_move == 'y_neg') {
                this.y_map -= 1;
            }else{
                this.x_map -= 1;
            }
        this.pasos = 0;
        this.new_pos = true;
    }
}
```

Listing 4.16: Actualización coordenada del mapa.

Dibujar

Finalmente, para dibujar a Pacman utilizamos la función **drawImagen()** de canvas que carga el trozo de imagen correspondiente a Pacman del spreedsheet. Posee dos estados de dibujo para crear la sensación de animación.

```
this.draw = function() {
    GameArea.context.save();
    if(this.state_draw == 0) {
        GameArea.context.drawImage(this.image, 320, this.yDraw, 32, 32
            , this.x*40, this.y*40, 35, 35);
        this.state_draw = 1;
    }else{
        GameArea.context.drawImage(this.image, 320+32, this.yDraw,
            32, 32, this.x*40, this.y*40, 35, 35);
        this.state_draw = 0;
    }
    GameArea.context.restore();
}
```

Listing 4.17: Dibujar Pacman.

4.3.3. Fantasmas

Definimos el objeto **Ghost** que tiene la lógica del comportamiento de los cuatro fantasmas que forman parte del juego.

En el momento de instanciar el objeto **Ghost(x_map,y_map,name,speed,initMoment)** le

pasamos las coordenadas (x_map,y_map) donde se dibujara,el nombre,su velocidad y el numero de cocos que Pacman tiene que haber comido para salir a perseguirlo.

Persecución Pacman

La función **init** se encarga de evaluar si se tiene que activar el fantasma, en caso afirmativo lo coloca fuera de la casa y empieza la persecución.

```
this.init = function(score) {
    if(score == this.initMoment && !this.move) {
        this.x_map =11;
        this.y_map =7;
        this.move = true;
    }
}
```

Listing 4.18: Actualizar posicion del los Fantasmas.

Actualizar objetivo

La función **new_path** aplica IA a cada fantasmas, por medio del mapa del juego obtiene el camino que cada fantasma tiene que seguir. Para ello tomamos como punto de inicio la posición actual del fantasma y la posición de Pacman formado por las variables **x_map** e **y_map** como punto de destino .A continuación explicamos el modo de persecución de cada fantasma.

- *Blinky*: su comportamiento es seguir a Pacman en la misma dirección, entonces el punto de inicio es la posición actual de fantasma y el punto final es la posición actual de Pacman.
- *Speedy*: al igual que Blinky persigue a Pacman, pero tiene en cuenta su dirección. En este caso el cálculo se realiza tomando como punto de partida la posición de Speedy y se consulta la dirección de Pacman , al resultado le sumamos/restamos 4 posiciones según corresponda para obtener el punto final.
- *Clyde*: tiene dos tipos de comportamientos que dependen de la distancia entre él y Pacman, si dicha distancia es mayor a ocho sigue en modo persecución en caso contrario deja de seguirlo y se aleja de él tomando como nuevo objetivo una de las esquinas.

El resultado de cada operación se guarda en la variable **result** correspondiente.

```
this.new_path = function(graph,pacman_x,pacman_y,direccion) {
    var start = graph.grid[this.x_map][this.y_map];
    if(this.name == 'blinky'){
        var end = graph.grid[pacman_x][pacman_y];
    }else if(this.name == 'speedy'){
        if(pacman_x+4 < 21 && direccion == 'x_pos'){
            pacman_x += 4;
        }else if(pacman_x-4 > 0 && direccion == 'x_neg'){
            pacman_x -= 4;
        }else if(pacman_y+4 < 19 && direccion == 'y_pos'){
            pacman_y += 4;
        }else if(pacman_y-4 > 0 && direccion == 'y_neg'){

        }
    }
}
```

```

pacman_y -= 4;
}
var end = graph.grid[pacman_x][pacman_y];
}else if (this.name == 'clyde'){
var end = graph.grid[pacman_x][pacman_y];
this.result = astar.search(graph,start,end,false);
if(this.result.length < 8){
var end = graph.grid[this.cuad_static.x][this.cuad_static.y];
}
}
this.result = astar.search(graph,start,end,false);
}

```

Listing 4.19: Actualizar direccion hacia el objetivo

Actualizar posición

Para actualizar la posición de los fantasma vinculamos la función **nexStepGhost** por medio de un **timer** de forma que dicha función se ejecute según el valor definido en la variable **speed** de cada fantasmas.

```

function nexStepGhost(ghost) {
ghost.flag = 1;
if(ghost.result.length > 0){
ghost.x = ghost.result[0].x;
ghost.y = ghost.result[0].y;
ghost.x_map = ghost.result[0].x;
ghost.y_map = ghost.result[0].y;
ghost.result.splice(0,1);
}
ghost.interval = setTimeout(function() {
nexStepGhost(ghost)
},ghost.speed);
}

```

Listing 4.20: Actualizar posicion del los Fantasmas.

Dibujar

Para dibujar los fantasmas lo hacemos igual que Pacman, pero al tener diferentes fantasmas por medio del nombre asignado en su creación obtenemos la imagen correspondiente a cada uno.

```

this.draw = function(){
GameArea.context.save();
if(this.name == 'blinky'){
if(this.state_draw == 0){
GameArea.context.drawImage(this.image,0,0,32,32,this.x_map*40,this.y_map*40,35,35);
}else{
GameArea.context.drawImage(this.image,32,0,32,32,this.x_map*40,this.y_map*40,35,35);
}
}

```

```

}else if(this.name == 'clyde') {
    if(this.state_draw == 0) {
        GameArea.context.drawImage(this.image, 64, 0, 32, 32, this.x_map*40, this.y_map*40, 35, 35);
    }else{
        GameArea.context.drawImage(this.image, 94, 0, 32, 32, this.x_map*40, this.y_map*40, 35, 35);
    }
}else if(this.name == 'inky') {
    if(this.state_draw == 0) {
        GameArea.context.drawImage(this.image, 192, 0, 32, 32, this.x_map*40, this.y_map*40, 35, 35);
    }else{
        GameArea.context.drawImage(this.image, 222, 0, 32, 32, this.x_map*40, this.y_map*40, 35, 35);
    }
}else if(this.name == 'speedy') {
    if(this.state_draw == 0) {
        GameArea.context.drawImage(this.image, 128, 0, 32, 32, this.x_map*40, this.y_map*40, 35, 35);
    }else{
        GameArea.context.drawImage(this.image, 160, 0, 32, 32, this.x_map*40, this.y_map*40, 35, 35);
    }
}
this.state_draw = ( this.state_draw === 1 ) ? 0 : 1;
GameArea.context.restore();
}

```

Listing 4.21: Dibujar Fantasma.

En la imagen 4.7 se muestra un ejemplo del comportamiento de los fantasmas..

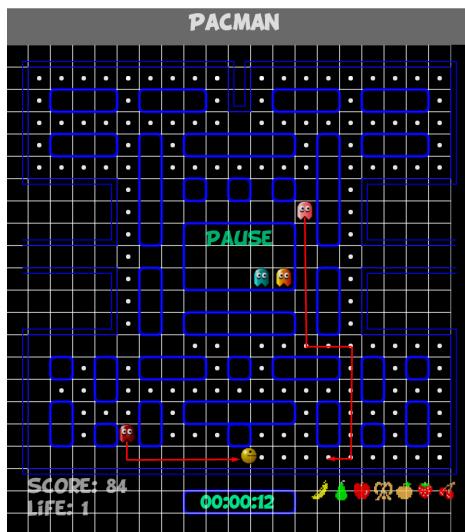


Figura 4.7: Apariencia comportamiento Fantasmas.

4.3.4. Movimiento

Para dotar de movimiento al juego a la función **updateGameArea** se vincula a un timer que se ejecuta cada 100ms y de esta forma mostrar las cambios de cada objeto.

```
function updateGameArea() {
    /** GameArea **/
    GameArea.clear();
    GameArea.AudioGame.play();
    GameArea.time_game();
    GameArea.score_user();
    GameArea.life_user();
    GameArea.shape_scene();
    GameArea.draw_doints(list_cocos);
    GameArea.draw_obstacles(list_obstaculos);
    GameArea.draw_fruit();
    /** Fantasma **/
    for(var i = 0; i < list_Ghost.length;i++) {
        var _ghost = list_Ghost[i];
        _ghost.init(GameArea.score);
        if (_ghost.move ) {
            _ghost.new_path(graph,Pac.x_map,Pac.y_map,Pac.type_move);
        }
    }
    _ghost.draw();
    /** Pacman **/
    Pac.new_position();
    Pac.hitObject(list_Ghost);
    if (!Pac.move) {
        /*Refibujamos todo el conteido del canvas diciendo que a perdido */
        GameArea.AudioGame.pause();
        GameArea.context.globalAlpha = 0.9;
        GameArea.context.save();
        GameArea.lose_game();
        GameArea.context.restore();
        GameArea.AudioDied.play();
        GameArea.stop();
    } else {
        var x = Pac.hitt_counter();
        Pac.hitObject(list_obstaculos);
        if (Pac.move != true || x == true) {
            /* si hay choque hay que resetear el contenido para dibujar */
            Pac.reset();
        } else {
            Pac.add_steps();
            if (Pac.pasos == 1 || Pac.pasos == -1 ) {
                Pac.new_path();
            }
        }
    }
    /* se comprueba si hemos comido algo para poner el sonido*/
    var eatPil = Pac.eat_doit(list_cocos);
    if (eatPil) {
        GameArea.AudioGame.volume=0;
        GameArea.AudioEat.play();
```

```

        GameArea.AudioGame.volume=0;
    }
Pac.reset_speed();
Pac.draw();
GameArea.win_game(list_cocos);
if(list_cocos.length == 0) {
    GameArea.stop();
}
}
}

```

Listing 4.22: Renderizado del juego.

4.4. Pruebas

Para realizar las pruebas necesitamos disponer de un portátil en el que se encuentre el fichero de la practica, ademas de tener instalado varios navegadores como pueden ser Chrome, FireFox e Internet Explorer.

Se ejecuta el fichero **game_pacman.html** en cada navegador provocando que se visualice el juego, figura 4.8.



Figura 4.8: Inicio juego Pacman.

Para iniciar la aplicación el usuario pulsa la **tecla L** provocando el movimiento de los fantasmas y el inicio del cronometro. A partir de este momento el usuario puede manipular a Pacman por medio de las flechas del teclado. Ademas se permite al usuario pausar el juego pulsado la **tecla P**, figura 4.9.

La partida termina cuando uno de los fantasmas capture a Pacman provocando que se visualice **Game Over** o cuando Pacman se ha comido todos los cocos provocando que se visualice **Win Game**, figura 4.10

A continuación, se muestra una tabla resumen del resultado de las pruebas en los distintos navegadores.

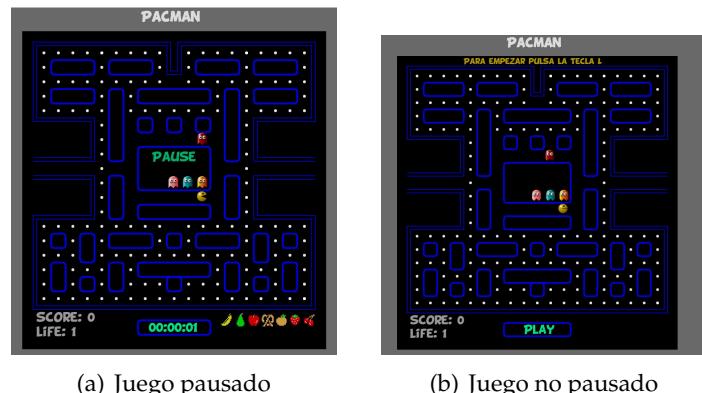


Figura 4.9: Pause/Start juego.



Figura 4.10: Game/Loose juego.

Cuadro 4.1: My caption

	Chrome	Firefox	Internet Explore
Funcionamiento APP	OK	OK	OK

Capítulo 5

Comecocos Multijugador

5.1. Enunciado

En la comunidad de internet han aumentado el desarrollo de aplicaciones en tiempo real entre distintos usuarios que se encuentran en diferentes partes del mundo. Esta segunda practica es una extensión de la primera ya que ahora el juego se va a ser multijugador online por medio de interfaz Web.

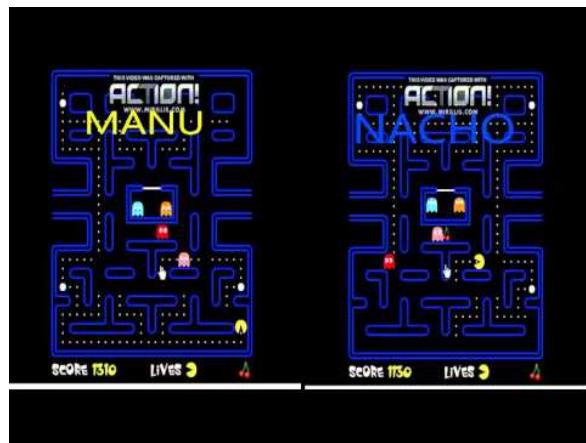


Figura 5.1: Ejemplo Aspecto multijugador Pacman.

Requisitos El comportamiento del juego sera el desarrollado en la primera practica pero sera necesario organizarlo de tal forma que el cliente se encargue de ciertas tareas y el servidor de otras con el objetivo de intercambiar informacion por medio de WebSockets, a continuación se explica en detalle cada una.

1. Cliente

- Conexión WebSockets con el servidor.
- Dibujar el escenario del juego,fantasmas,jugadores e informacion de la partida.
- Gestión del movimiento del personaje por medio de los eventos del teclado.

2. Servidor

- Entregar el fichero que contiene la aplicación.
- Conexión WebSockets con los clientes.
- Creación de un modulo que contenga los valores iniciales del juego como los cocos,obstáculo entre otros y un objeto de tipo **Player** que contenga la posición del usuario y del fantasma correspondiente.
- Se encargara de la lógica de los fantasmas aplicada en la primera practica. Hay que remarcar que en esta versión solo existe un fantasma por usuario.
- Por medio de la conexión WebSockets gestiona la entrada en sala,inicio del juego y la actualización de los distintos elementos del juego.

Tecnologías Sera necesario emplear la API WebSockets y JavaScript en el cliente mientras que en el servidor utilizaremos NodeJS.

5.2. Diseño

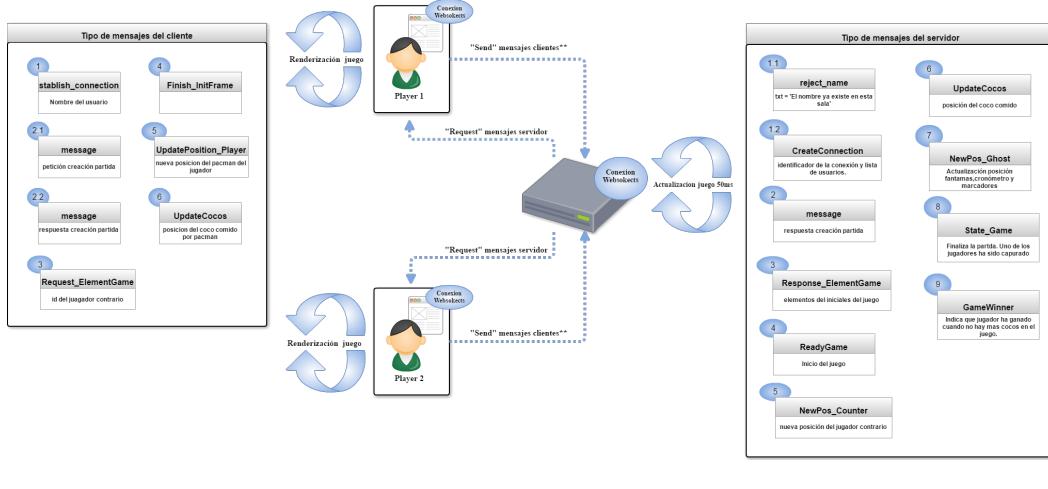


Figura 5.2: Esquema Pacman multijugador.

5.3. Desarrollo Servidor

El servidor de la aplicación lo desarrollamos con NodeJS por lo que se importa las librerías **node-static** y **http** para la creación del servidor.

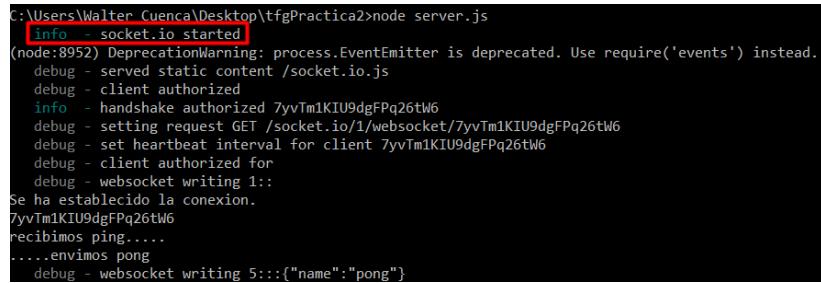
Con esto el servidor entrega el fichero inicial a los usuarios que se conecten pero queda importar la librería **socket.io** para crear la conexión WebSockets en el momento de recibir una petición.

```
var static = require('node-static');
var http = require('http');
var file = new(static.Server)();
```

```
var app = http.createServer(function (req, res) {
  file.serve(req, res);
  console.log('Server listening 8181');
}).listen(8181);
/* inst. socket.io */
var io = require('socket.io').listen(app);
```

Listing 5.1: Definición del servidor.

La imagen 7.2 muestra la conexión de un cliente al servidor y el inicio de la conexión Web-Sockets.



```
C:\Users\Walter_Cuenca\Desktop\tfgPractica2>node server.js
info - socket.io started
(node:8952) DeprecationWarning: process.EventEmitter is deprecated. Use require('events') instead.
  debug - served static content /socket.io.js
  debug - client authorized
    info - handshake authorized 7yvTm1KIU9dgFPq26tW6
  debug - setting request GET /socket.io/1/websocket/7yvTm1KIU9dgFPq26tW6
  debug - set heartbeat interval for client 7yvTm1KIU9dgFPq26tW6
  debug - client authorized for
  debug - websocket writing 1::Se ha establecido la conexión.
7yvTm1KIU9dgFPq26tW6
recibimos ping.....
.....enviamos pong
  debug - websocket writing 5::{"name":"pong"}
```

Figura 5.3: Ejecución del servidor.

5.3.1. Modulo Game

Se crea el modulo **CoreServer.js** en NodeJs para separar la lógica y las variables que definen los elementos del juego. Este modulo primero define las variables que tienen relación con el aspecto del juego como son los obstáculos, cocos y propiedades del juego.

```
var map = [...];
/* Jugadores */
var player1 = {'x':5,'y':15,'typePacman':1,numPasos:0};
var player2 = {'x':11,'y':3,'typePacman':2,numPasos:0};
/* */
var list_user=[];
var name_room = 'hallGame';

/* Cronometro */
var seconds = 0;
var min = 0;
var horas = 0;
var time = '00' + ':' + '00' + ':' + '00';
var _timer;
var list_cocos = [
];
var list_obstaculos = [
];
var properGame={
  'wCuad':40,
```

```

    'hCuad':40,
    'nColum':18,
    'nFila':19,
}
var shape_1 = [....
];
var shape_2 = [....
];

```

Listing 5.2: Definición variables del Obj.Game

Por otro lado define el objeto **Player** para representar a cada jugador. El objeto contiene la posición y pasos del personaje e incluye la posición y lista de nodos del fantasma correspondiente.

```

function Player(x,y,typePacman,xGhost,yGhost) {
  this.info= {'x':x,
  'y':y,
  'typePacman':typePacman,
  'numPasos':0,
  'fantasma': {'x':xGhost,
    'y':yGhost,
  },
};
this.path = [];
}

```

Listing 5.3: Definición Player Obj.Game

Para que el contenido del modulo sea accesible desde el servidor es necesario declarar los elementos como **module.exports**.

```

module.exports = {
  'map':map,
  'shape_1':shape_1,
  'shape_2':shape_2,
  'properGame':properGame,
  'list_obstaculos':list_obstaculos,
  'list_cocos':list_cocos,
  'name_room':name_room,
  '_timer;:_timer,
  'seconds':seconds,
  'min':min,
  'horas':horas,
  'time':time,
  'Player':Player
}

```

Listing 5.4: Export elementos del Obj.Game

Tras esto volvemos al fichero **server.js** donde se declara el acceso al modulo a través de la variable **Game**. A partir de ella creamos la instancia de los jugadores y calculamos la lista de nodos de su correspondiente fantasma por medio de la función auxiliar **CreatePath()**.

```
var Game = requiere('./ghost.js')
```

```

/* posiciones iniciales */
var player1 = new Game.Player(5,15,1,1,1);
var player2 = new Game.Player(14,15,2,18,1);
player1.path = CreatePath(player1.info,player1.info.fantasma);
player2.path = CreatePath(player2.info,player2.info.fantasma);

```

Listing 5.5: Instancia Jugadores del Servidor.

5.3.2. Lógica del Servidor

Hasta este momento solo se ha definido los elementos del juego pero no se ha visto como el servidor gestiona cada uno de los estados del juego hasta que este finaliza. Aquí es donde **WebSockets** interviene por medio de la librería **socket.io**, ya que permite al servidor gestionar los mensajes que recibirá. A continuación, se explican los tipos de mensajes y su tratamiento.

Sala de juego

Definimos el evento **stablish_connection** que recibe el nombre del usuario para validar la entrada a la sala.

Primero comprueba el numero de usuario en la sala ya que esta restringido a dos. Si aun no esta completa comprueba que el nombre del usuario no exista por medio de la función **seekUser(name)**. Si existe el nombre envía un mensaje **reject_name** en caso contrario envía un mensaje **CreateConnection** con el id de la conexión y la lista de usuario dentro de la sala.

Finaliza vinculando la conexión del usuario a la sala del juego y actualizando la lista de usuarios.

```

socket.on('stablish_connection',function(name) {
  var numClients = io.sockets.clients(Game.name_room).length;
  var existeName = seekUser(name);
  if(existeName) {
    var txt = 'El nombre ya existe en esta sala';
    socket.emit('reject_name',txt);
  }else{
    socket.emit('CreateConnection',socket.id,list_user);
    socket.username = name;
    socket.room = Game.name_room;
    socket.join(Game.name_room);
    if(list_user.length == 0) {
      var user = {'user':name,'room':Game.name_room,'id':socket.id,'posGame
        ':player1.info,'pos':1,'score':0};
    }else{
      var user = {'user':name,'room':Game.name_room,'id':socket.id,'posGame
        ':player2.info,'pos':2,'score':0};
    }
    list_user.push(user);
    socket.broadcast.to(Game.name_room).emit('New_Joined',socket.id);
  }
});

```

Listing 5.6: Definición evento establish_connection

La figura 5.4 muestra la conexión del primer jugador mientras la figura 5.5 trata la del segundo jugador.

```
Número de clientes:0
false
walter
[]
debug - websocket writing 5:::{ "name": "CreateConnection", "args": [ "7yvTm1KIU9dgFPq26tW6", [] ] }
debug - broadcasting packet
```

Figura 5.4: Petición sala 1^a jugador.

```
Número de clientes:1
False
PEDRO
[ { user: 'walter',
  room: 'hallGame',
  id: 'zYjIS6M4xgkcoVC8d3d',
  posGame: { x: 5, y: 15, typePacman: 1, numPasos: 0, fantasma: [Object] },
  pos: 1,
  score: 0 } ]
debug - websocket writing 5:::{ "name": "CreateConnection", "args": [ "AQYWhyjpA2T5bb9u8d3e", { "user": "walter", "room": "hallGame", "id": "zYjIS6M4xgkcoVC8d3d", "posGame": { "x": 5, "y": 15, "typePacman": 1, "numPasos": 0, "fantasma": [ { "x": 5, "y": 15 } ], "score": 0 } } ] }
debug - broadcasting packet
debug - websocket writing 5:::{ "name": "New Joined", "args": [ "AQYWhyjpA2T5bb9u8d3e" ] }
```

Figura 5.5: Petición sala 2º jugador.

Petición de partida

La petición y respuesta a la creación de una partida la realizan los usuarios por lo que el servidor en este caso sera transparente para ellos. Por ello definimos el evento **message** que encamina los mensajes entre los usuarios por medio del valor **idDestino**.

```
socket.on('message', function(message) {
  io.sockets.socket(message.idDestino).emit('message', message);
});
```

Listing 5.7: Definición evento message.

En la figura 5.7 se muestra como el servidor encamina los mensajes entre los usuarios.

```
Reenvío de mensaje al nodo destino.
{ typeMsg: 'Init',
  idOrigen: 'AQYWhyjpA2T5bb9u8d3e',
  texto: 'PEDRO: Jugamos una partida ??',
  idDestino: 'zYjIS6M4xgkcoVC8d3d' }
debug - websocket writing 5:::{ "name": "message", "args": [ { "typeMsg": "Init", "idOrigen": "AQYWhyjpA2T5bb9u8d3e", "texto": "PEDRO: Jugamos una partida ??", "idDestino": "zYjIS6M4xgkcoVC8d3d" } ] }
```

Figura 5.6: Petición de partida usuario.

Elementos del juego

Tras haber acordado jugar una partida ambos jugadores se define el evento **Request_ElementGame** para enviar los parámetros iniciales del juego.

```
Reenvio de mensaje al nodo destino.  
{ typeMsg: 'replayInit',  
  idOrigen: '1zLW7ebdeUpMjWg9JLb',  
  texto: 'si',  
  idDestino: 'QYYCeQlKup-2wpE9JLc' }  
debug - websocket writing 3: [{"name": "message", "args": [{"typeMsg": "replayInit", "idOrigen": "1zLW7ebdeUpMjWg9JLb", "texto": "si", "idDestino": "QYYCeQlKup-2wpE9JLc"}]}]
```

Figura 5.7: Respuesta de partida usuario.

Obtiene la información de los jugadores utilizando la función `getInfoUser(id)` pasándole cada uno de los identificadores.

Con la información anterior y las características del escenario crea la variable **elements** que se envía dentro del mensaje **Response_ElementGame**.

```
socket.on('Request_ElementGame',function(otherId){  
    var myInfo = getInfoUser(socket.id);  
    var counterInfo = getInfoUser(otherId);  
    var element = {  
        'shape_1':Game.shape_1,  
        'shape_2':Game.shape_2,  
        'cocos':Game.list_cocos,  
        'obstaculos':Game.list_obstaculos,  
        'properGame':Game.properGame,  
        'myInfo':myInfo,  
        'counterInfo':counterInfo  
    }  
    socket.emit('Response_ElementGame',element);  
});
```

Listing 5.8: Definicion del evento Request_ElementGame

La figura 5.8 muestra el envío de la información a los jugadores.

Figura 5.8: Envío elementos del juego.

Inicio del juego

Para iniciar la partida definimos el evento **Finish_InitFrame**. Se encarga de validar que la petición de inicio de partida este solicitada por los dos jugadores para enviar el mensaje **ReadyGame** a los clientes.

En este punto se establece el bucle de actualización del juego por parte del servidor a los clientes por medio del evento timer `setInterval(UpdateGhost,800)`.

```

socket.on('Finish_InitFrame',function() {
  cooardiar +=1;
  if(cooardiar == 2) {
    io.sockets.in(Game.name_room).emit('ReadyGame');
    setTimeout(function() {
      Game._timer = setInterval(UpdateGhost,800);
      CronoTime();
    },3000)
  }
});
```

Listing 5.9: Definición del evento Finish_InitFrame

La función **UpdateGhost** es la parte mas importante de la lógica del servidor ya que se encarga de actualizar la posición de cada fantasma y obtener la puntuación de cada usuario para enviar un mensaje **NewPos_Ghost**.

También comprueba si alguno de los usuarios a colisionado con el fantasma lo que provoca que se envié el mensaje **State_Game** a los usuarios.

```

function UpdateGhost(){
  var elementsGame = [];
  /* */
  var hitGhost1 = playerHitGhost(player1); //true o false
  var hitGhost2 = playerHitGhost(player2); // true o false
  if (!hitGhost1 && !hitGhost2){
    var coordGhost={'x':player1.path[0].x,'y':player1.path[0].y};
    setGhost_User(list_user[0].id,coordGhost);
    coordGhost={'x':player2.path[0].x,'y':player2.path[0].y};
    setGhost_User(list_user[1].id,coordGhost);
    elementsGame.push({'id':list_user[0].id,'x':player1.path[0].x,'y'
      :player1.path[0].y,'score':list_user[0].score});
    player1.path.splice(0,1);
    elementsGame.push({'id':list_user[1].id,'x':player2.path[0].x,'y'
      :player2.path[0].y,'score':list_user[1].score});
    player2.path.splice(0,1);
    io.sockets.in(Game.name_room).emit('NewPos_Ghost',fantasmas,Game.
      time,scores);
  }

  if (hitGhost1 || hitGhost2){
    var msgPlayer1 = (hitGhost1 === true ) ? 'KO' : 'OK';
    var msgPlayer2 = (hitGhost2 === true ) ? 'KO' : 'OK';
    io.sockets.socket(list_user[0].id).emit('State_Game', msgPlayer1)
    ;
    io.sockets.socket(list_user[1].id).emit('State_Game', msgPlayer2)
    ;
    clearInterval(Game._timer);
  }
}
```

Listing 5.10: Definición de la función UpdateGhost.

La figura 5.9 muestra el envío de la información actualizada del juego a los jugadores.

Figura 5.9: Envío actualización del juego.

Actualización elementos del juego

El movimiento del personaje de cada usuario es notificado al servidor por lo que se define el evento **UpdatePosition_Player**. Con la nueva posición actualiza la posición del personaje correspondiente por medio de la función **setPosition_User**. Ademas, evalúa el numero de pasos dado por el personaje ya que si el valor es 1 o -1 indica que se ha movido a una nueva casilla del mapa de juego y es necesario actualizar la lista de nodos que contiene la variable **path**.

Finalmente envía el mensaje **NewPos_Counter** con la nueva posición al usuario contrario.

```
socket.on('UpdatePosition_Player',function(newPosition){  
    setPosition_User(socket.id,newPosition);  
    if(newPosition.numPasos == 1 || newPosition.numPasos == -1){  
        for (var i = 0; i < list_user.length; i++) {  
            var user = list_user[i];  
            if(user.id == socket.id){  
                var myInfo = getInfoUser(socket.id)  
                if (user.pos == 1) {  
                    player1.path = CreatePath(myInfo,myInfo.fantasma);  
                }else{  
                    player2.path = CreatePath(myInfo,myInfo.fantasma);  
                }  
            }  
        }  
    }  
    socket.broadcast.to(Game.name_room).emit('NewPos_Counter',newPosition);  
});
```

Listing 5.11: Definición del evento UpdatePosition_Player.

La figura 5.10 muestra el envío de la nueva posición al usuario contrario.

```
{ x: 5.5, y: 15, typePacman: 1, numPasos: 0.5 }
  debug - broadcasting packet
  debug - websocket writing 5::>{"name": "NewPos_Counter", "args": [{"x": 5.5, "y": 15, "typePacman": 1, "numPasos": 0.5}]}
```

Figura 5.10: Envío nueva posición usuario.

El movimiento del personaje provoca interacción con los cocos por lo que definimos el evento **UpdateCocos_Player** que se encarga de eliminar el elemento de la lista de cocos y reenviar la información al otro jugador por medio del mensaje **UpdateCocos_Player**. Por ultimo, comprueba si no existen mas cocos en el juego ya que si esto ocurre se envía el mensaje **GameWinner** a los jugadores informando quien a ganado.

```
socket.on('UpdateCocos', function(cocoPosition) {
```

```

Game.list_cocos.splice(cocoPosition,1);
socket.broadcast.to(Game.name_room).emit('UpdateCocos_Player',
    cocoPosition);
setScore_User(socket.id,4);
if(Game.list_cocos.length == 0){
    ganador = gameFinish();
    io.sockets.in(Game.name_room).emit('GameWinner',ganador);
}
});

```

Listing 5.12: Definición del evento UpdateCocos.

La figura 5.10 muestra el envío del coco comido por un usuario al contrario.

```

debug - broadcasting packet
debug - websocket writing 5:::{ "name": "UpdateCocos_Player", "args": [99] }

```

Figura 5.11: Envió coco comido al usuario.

5.4. Desarrollo Cliente

Los jugadores que conectan a la url <http://localhost:8181/> en la que se encuentra el servidor que entrega el fichero `index.html`¹ al navegador estableciendo la conexión WebSockets con el servidor.

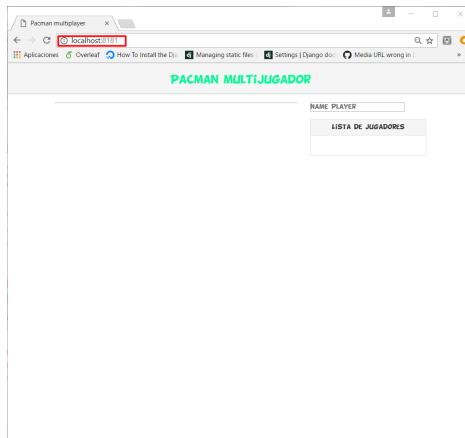


Figura 5.12: Pagina Inicio Pacman-Online.

5.4.1. Lógica de comunicación

Ahora se muestra los distintos mensajes y eventos definidos en el cliente para establecer la comunicación.

¹Apéndice A

Sala de juego

El usuario introduce el nombre en el **input** de la pagina, provocando que se envíe un mensaje **establish_connection** con el nombre para validar que no existe en la sala.

```
function sendConnection() {
    socket.emit('stablish_connection', $('#player').val());
    $('#player').attr('disabled', true);
}
```

Listing 5.13: Envió mensaje establish_connection.

Se genera el evento **Reject_name** para tratar la respuesta al mensaje en caso de que el nombre no sea valido.

```
socket.on('reject_name', function(info) {
  $('#player').attr('disabled', false);
});
```

Listing 5.14: Definición evento reject_name.

Por ultimo, se crea el evento **CreateConnection** para tratar en caso de ser afirmativa la respuesta al mensaje. Recibe el **id** de la conexión con el que se crea la instancia del objeto **Player** en la variable **myPacman** y la lista de usuario.

```
socket.on('CreateConnection',function(id,listUser){  
    listplayers = listUser;  
    myPacman = new Player(id);  
    for (var i = 0; i < listUser.length; i++) {  
        var player = listUser[i];  
        $('.list-group').append('<li id=user_'+i+' class=list-group-item>' +  
            player.user+'</li>');  
        $('#user_'+i).click(function(){requestGame(this)});  
    }  
});
```

Listing 5.15: Definición evento CreateConnection.

La figura 5.13 muestra la petición de sala del primer usuario mientras que la figura 5.14 del segundo.

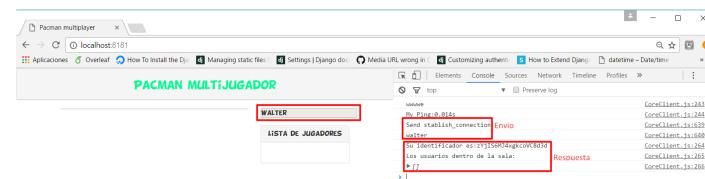
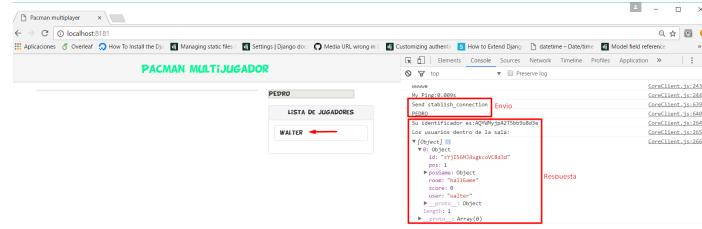


Figura 5.13: Petición/Respuesta sala 1^a jugador.

Figura 5.14: Petición/Respuesta sala 2^a jugador.

Petición de Partida

Tras entrar en la sala los usuarios pueden realizar una petición de partida a los usuarios existentes. Al seleccionar un nombre de la lista se active la función **RequestGame(this)**. Esta función se encarga de obtener el id de la conexión del cliente seleccionado por medio de la función **SeekUser(name)** y genera un mensaje **message** cuyo cuerpo contiene el **id_origen**, **id_destino**, el texto de la petición y como subtipo de mensaje **Init**.

```
function requestGame(elemento) {
    var text=$('#player').val()+' Jugamos una partida ??';
    var destino = seekID($(elemento).text());
    var message = {
        'typeMsg':'Init',
        'idOrigen':myPacman.id,
        'texto':text,
        'idDestino':destino
    }
    socket.emit('message',message)
    $('#li').off('click');
}
```

Listing 5.16: Definición función requestGame.

La imagen 5.15 muestra la petición de partida de un jugador a otro.

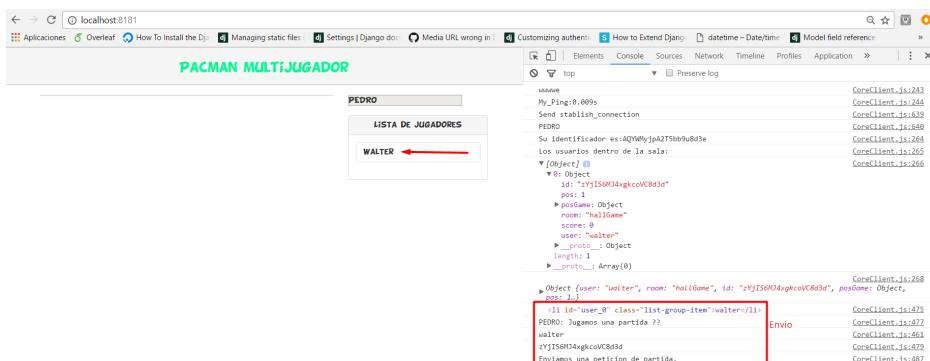


Figura 5.15: Envío petición partida.

Los mensajes de petición de partida se crean entre usuarios por lo que el servidor solo nos sirve como intermediario para encaminar el mensaje. Por ello es necesario definir el evento **message** para gestionar estos mensajes.

```
socket.on('message', function(message) {
  if(message.typeMsg == 'Init') {
    _message = message
    $('#peti').text(message.texto);
    $('#myModal').modal('show')
  } else if (message.typeMsg == 'replayInit') {
    if(message.texto == 'si') {
      $('#escena').show();
      CounterPacman = new Player(message.idOrigen);
      socket.emit('Request_ElementGame', CounterPacman.id);
    } else {
      $('#li').on('click', function() {requestGame(this)});
    }
  }
});
```

Listing 5.17: Definicion evento message.

Para el mensaje tipo **Init** el evento se encarga de mostrar una ventana emergente con la petición recibida, figura 5.16.

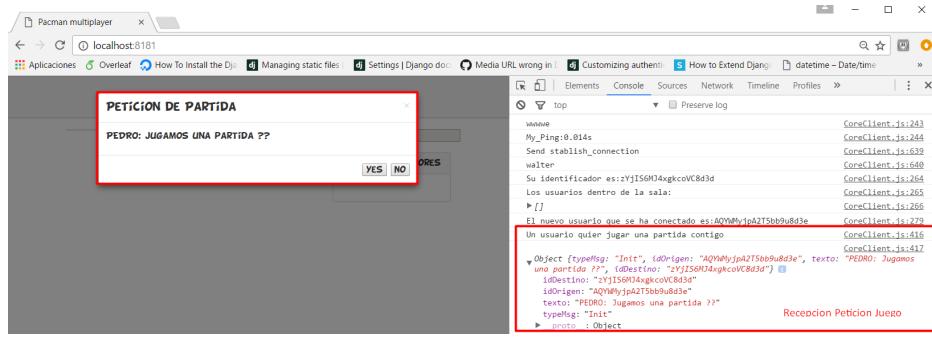


Figura 5.16: Recepción petición partida.

El usuario receptor tiene la opción de pulsar si o no provocando que la función **Replay-Game(si/no)** gestione la respuesta. La función independientemente de la opción genera un mensaje **message** que contiene el **id_origen**, **id_destino**, la respuesta a la petición y como subtipo de mensaje **Replay_Init**.

Solo si la respuesta es afirmativa crea el mensaje **Request_ElementGame** dirigido al servidor para pedir los parámetros iniciales del juego.

```
function replayGame(contesta) {
  $('#myModal').modal('hide')
  var message = {
    'typeMsg': 'replayInit',
    'idOrigen': myPacman.id,
    'texto': contesta,
    'idDestino': _message.idOrigen
```

```

};

socket.emit('message', message)
if(contesta == 'si'){
    $('#escena').show();
    CounterPacman = new Player(_message.idOrigen)
    socket.emit('Request_ElementGame', CounterPacman.id);
}
}
}

```

Listing 5.18: Definición función replayGame.

El usuario que envió el mensaje inicial recibe un mensaje **replayInit** para realizar las mismas tareas que el otro usuario, figura 5.17.



Figura 5.17: Recepción respuesta a la petición de partida.

Presentación del juego

Tras el ultimo mensaje enviado por los clientes es necesario definir el evento **Responde_ElementGame** para recibir el valor de los parámetros iniciales del juego. Los parámetros los obtiene de la variable **elements**, donde aquellos que tienen que ver con el escenario de juego se guardan en el objeto **GameArea** mientras que otros valores se guardan en **myPacman** y **CounterPacman** que corresponde al personaje del cliente y al del rival.

```

socket.on('Response_ElementGame', function(elementsGame) {
    GameArea.shape_1 = elementsGame.shape_1;
    GameArea.shape_2 = elementsGame.shape_2;
    GameArea.list_obstaculos=elementsGame.obstaculos;
    GameArea.list_cocos=elementsGame.cocos;
    GameArea.properGame=elementsGame.properGame;

    myPacman.setMyghostposition(elementsGame.myInfo.fantasma);
    delete elementsGame.myInfo.fantasma;
    myPacman.setMyPosition(elementsGame.myInfo);

    CounterPacman.setMyghostposition(elementsGame.counterInfo.fantasma);
}

```

```

delete elementsGame.counterInfo.fantasma;
CounterPacman.setMyPosition(elementsGame.counterInfo);

DrawScene();
socket.emit('Finish_InitFrame');
});

```

Listing 5.19: Definicion evento Response_ElementGame

Con esta información llamamos a la función **DrawScene()** que define las dimensiones del área del juego y dibujar el aspecto inicial del juego a través de la función **UpdateGame()**.

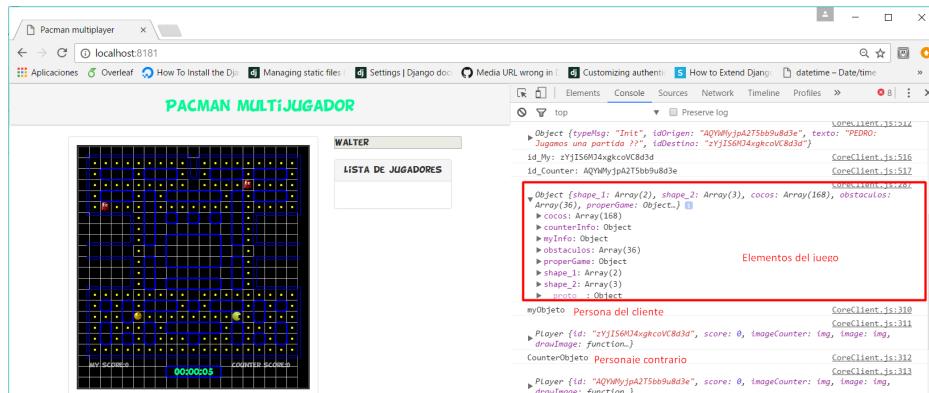
```

function DrawScene() {
    GameArea.canvas.width=(GameArea.properGame.nFila+2)*GameArea.properGame.wCuad;
    GameArea.canvas.height=(GameArea.properGame.nColum+4)*GameArea.properGame.hCuad;
    $('#pnCanvas').show();
    UpdateGame();
}

```

Listing 5.20: Definición función DrawScene.

La figura 5.18 muestra la recepción de los parámetros y la visualización del juego para el primero usuario mientras la figura 5.19 se muestra el del segundo usuario.

Figura 5.18: Recepción parámetros iniciales 1^a usuario.

Tras realizar estas operaciones envía un mensaje **Finish_initFrame** para indicar al servidor que el cliente está a la espera de empezar la partida.

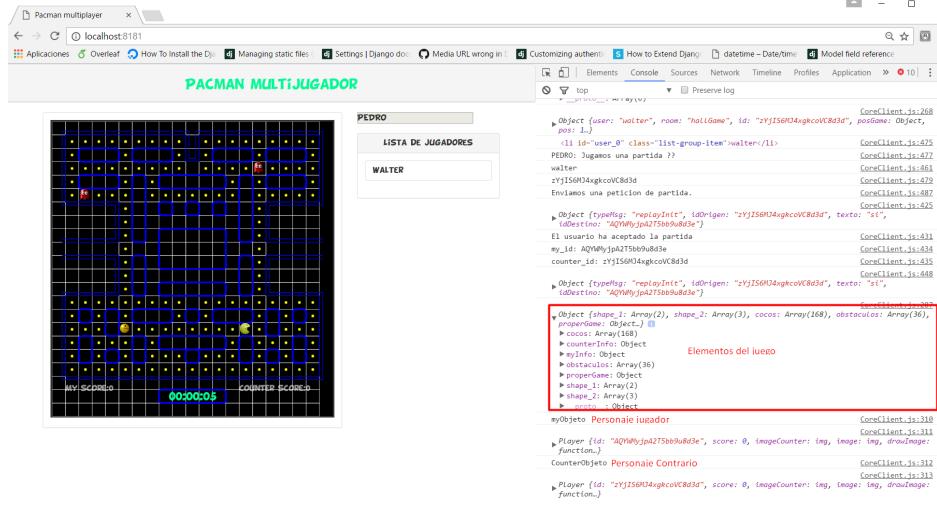
Inicio del juego

Se crea el evento **ReadyGame** para recibir el mensaje de inicio de juego. En este momento se genera un evento **timeInterval** que ejecuta función **UpdateGame()** cada 1000/60 s provocando el inicio del juego.

```

socket.on('ReadyGame', function() {
  x = true;
}

```

Figura 5.19: Recepción parámetros iniciales 2^a usuario.

```
GameArea.frame = setInterval(UpdateGame, 1000/60);
});
```

Listing 5.21: Definición evento ReadyGame.

Actualización del juego

El movimiento del personaje se realiza por medio de la función **NextPosPacman** que gestiona los eventos del teclado. Cada movimiento tiene que ser validado ya que esta nueva posición se envía al servidor.

Primero verifica que la nueva posición del jugador, variable **newCoord**, no sobrepase el contorno del juego por medio de la función **Game.hitt_counter(newCoord)** igual que con los obstáculos a través de función **GameArea.hitObject(newCoord)**.

Si estas dos validaciones son correctas, envía un mensaje **UpdatePosition_Player** con la nueva posición.

```
function NextPosPacman(e) {
  var newCoord = { 'x': 0, 'y': 0 };
  if (GameActive) {
    var pas = 0;
    newCoord.x = myPacman.myposition.x;
    newCoord.y = myPacman.myposition.y;
    if (e.code == 'ArrowDown') {
      newCoord.y += 0.5;
      pas += 0.5;
    } else if (e.code == 'ArrowUp') {
      newCoord.y += -0.5;
      pas += -0.5;
    } else if (e.code == 'ArrowRight') {
      newCoord.x += 0.5;
      pas += 0.5;
    }
  }
}
```

```

}else if (e.code == 'ArrowLeft') {
    newCoord.x += -0.5;
    pas += -0.5;
}
var HitContor = GameArea.hitt_counter(newCoord);
if(!HitContor) {
    var HitObstacle = GameArea.hitObject(newCoord);
    if(!HitObstacle) {
        myPacman.myposition.x = newCoord.x;
        myPacman.myposition.y = newCoord.y;
        myPacman.myposition.numPasos += pas;
        socket.emit('UpdatePosition_Player',myPacman.myposition);
        if(myPacman.myposition.numPasos == 1 || myPacman.myposition.numPasos
            == -1) {
            myPacman.myposition.numPasos = 0;
        }
    }
    var eat = GameArea.eatCoco(myPacman.myposition);
    if(eat >= 0) {
        socket.emit('UpdateCocos',eat);
    }
}
}
}

```

Listing 5.22: Definición función NextPosPacman.

Ademas, comprueba si existe colisión con algún coco por medio de la función **GameArea.eatCoco(newCoord)**. En caso afirmativo envía un mensaje **UpdateCoco** con la posición del coco con el que se ha colisionado.

Las acciones anteriores las realizan los dos usuarios por lo que es necesario definir una serie de eventos para actualizar el estado del jugador contrario.

El primer evento es `socket.on('NewPos_Counter',function())` que recibe la nueva posición del usuario contrario.

```
socket.on('NewPos_Counter', function(newPosition) {
  CounterPacman.myposition = newPosition;
});
```

Listing 5.23: Definición evento NewPos_Counter.

El siguiente evento a definir es `socket.on('UpdateCocos_Players',function())` que recibe la posicion del coco que ha comido el usuario contrario.

```
socket.on('UpdateCocos_Player',function(cocoPosition){  
  GameArea.list_cocos.splice(cocoPosition,1);  
});
```

Listing 5.24: Definición evento UpdateCocos_Player.

Por ultimo se define el evento `socket.on('NewPos_Ghost,function()')` que recibe actualizaciones periódicas del servidor con la posición de los fantasmas, el valor del cronómetro y la puntuación de cada usuario.

```
socket.on('NewPos_Ghost', function(fantasmas,timers,scores){
```

```
GameArea.time = timers;
for (var i = 0; i < fantasmas.length; i++) {
    ghost = fantasmas[i];
    core = scores[i];
    if(ghost.id == myPacman.id && core.id == myPacman.id) {
        myPacman.myGhost.x = ghost.x;
        myPacman.myGhost.y = ghost.y;
        myPacman.score = core.score;
    } else {
        CounterPacman.myGhost.x = ghost.x;
        CounterPacman.myGhost.y = ghost.y;
        CounterPacman.score = core.score;
    }
}
});
```

Listing 5.25: Definición evento NewPos_Ghost.

En las figuras 5.20, 5.21 se muestra la recepción de los elementos del juego por cada usuario.



Figura 5.20: Recepción NewPos_Ghost 1^a usuario.



Figura 5.21: Recepción NewPos_Ghost 2^a usuario.

Finalización del juego

Por ultimo, la finalización del juego lo determina el servidor ya que se encarga de comprobar si existe colisión con entre los jugadores y el fantasma. Por ello se define el evento **State_Game** que recibe **OK** o **KO** dependiendo si jugador que ha sido capturado por el fantasma o no.

```
socket.on('State_Game', function(msg) {
    GameArea.State_Game = msg;
});
```

Listing 5.26: Definición evento State_Game.

La figura 5.22 muestra como uno de los jugadores es capturado por el fantasma por lo que pierde la partida.

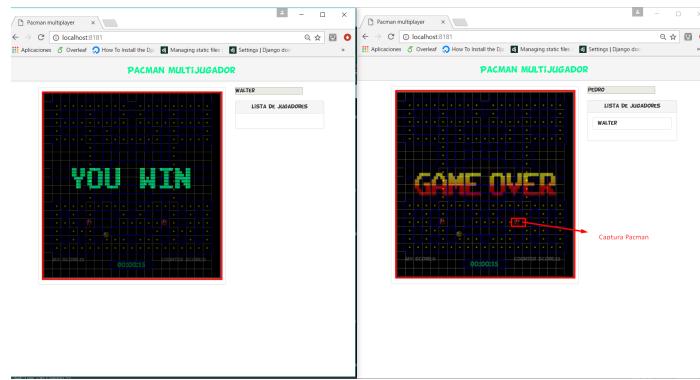


Figura 5.22: Aspecto clásico Pacman.

Otro evento necesario es **GameWinner** que recibe el ganador de la partida cuando todos los cocos del coco han sido comidos.

```
socket.on('GameWinner', function(userwinner) {
    GameArea.State_Game = userwinner;
});
```

Listing 5.27: Definición evento GameWinner.

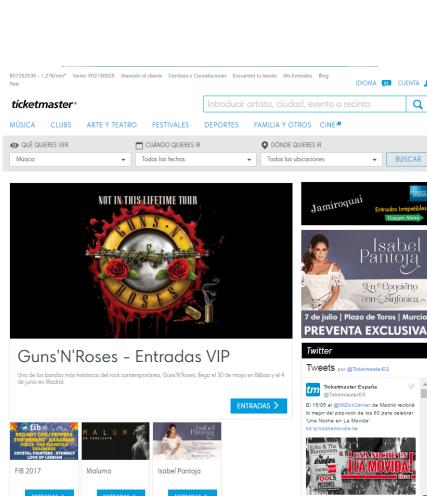
5.5. Pruebas

Capítulo 6

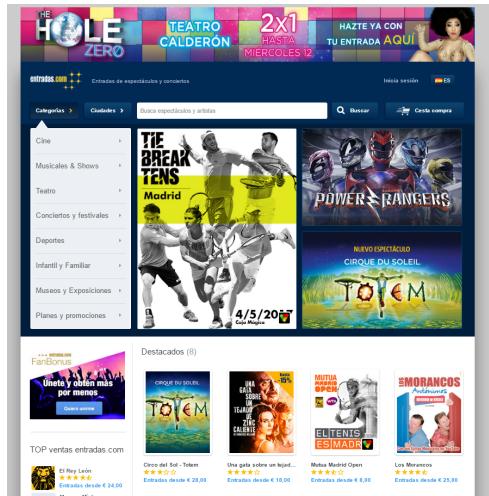
Tienda Web

6.1. Enunciado

En la actualidad los sitios web han empezado ha tener mayor presencia en Internet debido a la versatilidad y comodidad de los servicios que ofrecen a los usuarios por ejemplo TicketMaster.com o Entradas.com permiten a los usuarios adquirir entradas a distintos eventos sin necesidad de hacerlo personalmente ademas de estos ejemplos existen otros que prestan otro tipo servicio al usuario.



(a) Pagina TicketMaster.com



(b) Pagina Entradas.com

Figura 6.1: Portada Paginas Web.

Por ello, en esta tercera practica se pide desarrollar un sitio web Full-Stack,es decir, nos encargamos de crear la capa Front-End y Back-End del proyecto y conocer como interactuan entre si estas dos capas.

6.1.1. Requisitos

Se presenta los distintos puntos que tiene que tiene cubrir la practica en cuando a apariencia y funcionalidad.

Funcionalidad

Para gestionar el contenido de la Web sera necesario implementar como BBDD MySQL dejando de lado la BBDD de PostGrade que por defecto utiliza Django.

Tras esto los usuarios tienen que tener acceso a los distintos servicios de la aplicación dentro de los cuales ser necesario hacer uso de elementos multimedia (imágenes, audio y vídeo) para enriquecer la aplicación:

1. **Cantantes:** Se muestra información ,vídeo,discos e imágenes del cantante seleccionado.
2. **Eventos:** Muestra información,localización,cantantes y entradas del evento seleccionado.

Ademas la aplicación tiene que ser capas de gestionar los usuarios que la visiten. Las principales funciones que tienen ser capas de gestionar son las siguientes:

1. **Register:** Entrega un formulario al usuario para registrarse en la aplicación.
2. **Login:** Permite acceder al cliente siempre que se valide el contenido que rellene en el formulario de Login.
3. **Perfil Usuario:** Disponible para aquellos usuarios que hayan realizado el Login y debe mostrar la información que el usuario a llenado en su registro ademas de sus compras si las realizado.
4. **Logout:** Permite cerrar la sesión actual del usuario.

Por ultimo, es necesario proveer a la aplicación de un carrito de la compra basado en la sesión del usuario con el objetivo de tener un mecanismo de persistencia por lo que se tiene que implementar las siguientes funciones:

1. **Detalle del contenido:** Debe mostrar el contenido del carrito en una ventana individual donde se detalla cada uno de los productos.
2. **Actualizar contenido:** Se debe permitir la modificación del numero de productos añadidos.
3. **Eliminar contenido:** Se debe permitir eliminar un producto que seleccione el cliente.

Apariencia

Al tratarse de una aplicación es necesario darle un aspecto ordenado y cuidado aparte de las funcionalidad que se ha descrito con anterioridad. Para facilitar esta labor y que lleve el menor tiempo posible se recomienda utilizar la librería de Boostrap. Esta librería implementa procesos de CSS3 y JS.

6.1.2. Tecnologías Necesarias

En el desarrollo de la practica es necesario/obligatorio que se haga uso de las tecnologías que se listan a continuación:

1. FrameWork:Django.

2. BBDD : MySQL.
3. Comunicación cliente-servidor(sincrono) : Formularios.
4. Comunicación cliente-servidor(asíncrono) : Ajax.
5. WebServices : Google Maps.
6. Mapas Interactivos : Google Maps (JS)
7. Boostrap

6.2. Desarrollo

El desarrollo de la aplicación se divide en los dos próximas capas **Front-End** y **Back-End** para dividir el funcionamiento de una manera mas optima y se entienda como encaja cada uno de los elementos.

6.3. Back-End

Esta capa de la aplicación se encarga de buscar información en la BBDD de acuerdo a las peticiones que recibe y de esta forma entregar la información a los ficheros html correspondientes para que el navegador se encargue de cargarlos.

Primer vemos a instalar los controladores de MySQL que permiten a Django acceder al contenido de la BBDD así que a través de una consola ejecutamos los siguientes comandos:

1. `sudo apt -get install python-dev`
2. `sudo apt -get install libmysqlclient-dev`
3. `pip install MySQL-python`

El siguiente paso es crear la BBDD por lo que accedemos al prompt de MySQL y ejecutamos `create database AppBBDD`. Para comprobar que la BBDD se ha creado correctamente ejecutamos `show databases` y veríamos algo parecido a la figura 6.2.

```
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| AppBBDD |
| mysql |
| performance_schema |
| test |
+-----+
5 rows in set (0.01 sec)

mysql>
```

Figura 6.2: Creación de la BBDD.

Tras terminar la configuración previa de componentes externos a Django es momento de trabajar con sus componentes internos que explicamos a continuación.

6.3.1. Conexión Django-BBDD

La BBDD generada es AppBBDD que se incluye dentro del fichero **setting.py**, permitiendo de esta forma acceder a la BBDD.

```
DATABASES = {'default': {
    'ENGINE': 'django.db.backends.mysql',
    'NAME': 'appBBDD',
    'USER': 'root',
    'PASSWORD': '*****',
}}
```

Listing 6.1: añadimos la BBDD al entorno de Django.

6.3.2. Models

Hasta el momento solo se ha declarado la BBDD pero es necesario crear las tablas, por lo que se generan tantas clases como sean necesario en el fichero **models.py**¹.

Para que estos cambios sean visibles ejecutamos '**python migrate.py makemigrations**' para migrar los cambios y '**python migrate.py migration**' para hacerlos efectivos.

Para comprobar que la creación se ha realizado correctamente consultamos la BBBDD a través del **prompt** de MySQL, figura 6.3.

```
mysql> use AppBBDD;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
Database changed
mysql> show tables;
+-----+
| Tables_in_AppBBDD |
+-----+
| WebMultimedia_cancion
| WebMultimedia_cancion_buyUser
| WebMultimedia_cantante
| WebMultimedia_cantante_listDiscos
| WebMultimedia_cantante_listImagenes
| WebMultimedia_cantante_listVideos
| WebMultimedia_cartshop
| WebMultimedia_compra
| WebMultimedia_contacta
| WebMultimedia_disco
| WebMultimedia_disco_listCanciones
| WebMultimedia_entradas
| WebMultimedia_evento
| WebMultimedia_evento_listEntradas
| WebMultimedia_evento_listVideos
| WebMultimedia_evento_listcantantes
| WebMultimedia_galeria
| WebMultimedia_order
| WebMultimedia_orderitems
| WebMultimedia_orderitems_producto
| WebMultimedia_perfiluser
| WebMultimedia_perfiluser_listCompra
| WebMultimedia_video
| auth_group
| auth_group_permissions
| auth_permission
| auth_user
| auth_user_groups
| auth_user_user_permissions
| django_admin_log
| django_content_type
| django_migrations
| django_session
+-----+
33 rows in set (0.00 sec)
```

Figura 6.3: Creación de tablas de la aplicación.

Una de las ventajas que presenta Django es su capacidad de abstraer la capa de BBDD a través del interfaz admin al que se puede acceder por medio de la url **128.0.0.1:/WebMul-**

¹Apéndice A

`timedia/admin/`, donde se visualizan los modelos incluidos en el fichero `admin.py`, figura 6.4.

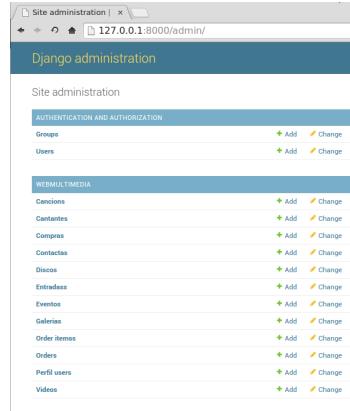


Figura 6.4: Interfaz admin Django

6.3.3. URL's

Tras terminar de configurar las tablas y accesos a la BBDD es necesario definir las distintas URLs a las que los usuarios tendrán acceso. Para ello incluimos los valores de las URLs dentro del fichero `urls.py` en forma de dupla. El primer elemento indica el path que puede contener parámetros que serán utilizados posteriormente y el segundo elemento indica la vista que trata la petición.

```
urlpatterns = [
    url(r'^$', views.PageInit),
    url(r'^Cantantes/(?P<idCantante>[0-9]+)/$', views.PageCantante),
    url(r'^Eventos/(?P<idEvento>[0-9]+)/$', views.PageEvent),
    url(r'^Search/$', views.SearchItem),
    url(r'^WebServRequest/$', views.WSRequest),
    url(r'^WebServInfoSite/$', views.WSRequestSite),
    url(r'^Contacta/$', views.PageContact),
    url(r'^Register/$', views.PageRegister),
    url(r'^Login/$', views.PageLogin),
    url(r'^Perfil/$', views.PagePerfil),
    url(r'^Logout/$', views.PageLogout),
    url(r'^AddCar/(?P<idEvento>[0-9]+)/(?P<idTicket>[0-9]+)/$', views.
        AddCarShop),
    url(r'^UpdateCart/(?P<idEvento>[0-9]+)/(?P<idTicket>[0-9]+)/$', views.
        UpdateCart),
    url(r'^RemoveCart/(?P<idEvento>[0-9]+)/(?P<idTicket>[0-9]+)/$', views.
        deleteItemCar),
    url(r'^DetailCar/$', views.DetailCarShop),
    url(r'^Checkout/$', views.Pagecheckout),
]
```

Listing 6.2: Definicion de las Url's del proyecto.

Por necesidades del proyecto es necesario definir ***URL's estáticas***, aquellas que son independientes del elemento seleccionado por el usuario, ejemplo '`URL(r'register', views.PageRegister)`', y ***url's dinámicas*** aquellas que depende del elemento seleccionado ya que el parámetro forma parte de la url ,ejemplo `xxxxxx`.

6.3.4. Formularios

Para gestionar o validar información enviada por los usuarios es necesario utilizar formularios por lo generamos el fichero `forms.py` que contiene la clase de los formularios que utilizamos a lo largo de la aplicación.

Register

La clase `RegisterForm(forms.Form)` define los campos utilizados en el registro de los usuarios.

```
class RegisterForm(forms.Form):
    typeSexo = (
        ('M', 'Mujer'),
        ('H', 'Hombre'),
    )
    nick=nombre=forms.CharField()
    correo=forms.EmailField(required=True)
    nombre=forms.CharField()
    apellido=forms.CharField()
    password=forms.CharField(widget=forms.PasswordInput())
    password2=forms.CharField(widget=forms.PasswordInput())
    telefono=forms.IntegerField()
    direccion=forms.CharField()
    sexo=forms.ChoiceField(widget=forms.RadioSelect, choices=typeSexo)
    edad=forms.IntegerField()
    pais=forms.CharField()
    provincia=forms.CharField()
```

Listing 6.3: Campos del formulario de Registro.

Login

La clase `LoginForm(forms.Form)` define los campos utilizados al realicen Login por los usuarios .

```
class LoginForm(forms.Form):
    nick=forms.CharField()
    password=forms.CharField(widget=forms.PasswordInput())
```

Listing 6.4: Campos del formulario de Login.

Contacta

La clase `ContactaForm(forms.form)` define los campos que se utiliza cuando un usuario necesite enviar un mensaje a la aplicación pidiendo información.

```
class ContactaForm(forms.Form):
    typeArea = (
        ('Compra', 'Compra'),
        ('Tecnica', 'Tecnica'),
        ('Eventos', 'Eventos'),
        ('Otros', 'Otros'),
    )
    nombre=forms.CharField()
    correo=forms.EmailField(required=True,label='Correo electronico')
    numeroTlf=forms.IntegerField(label='Numero telefono')
    area=forms.ChoiceField(widget=forms.RadioSelect, choices=typeArea)
    motivo=forms.CharField(widget=forms.Textarea)
```

Listing 6.5: Campos del formulario de Contacto.

SelecItem

La clase *itemsCountsForm(forms.Form)* permite al usuario seleccionar el numero de productos que quiere adquirir.

```
class itemsCountsForm(forms.Form):
    PRODUCT_QUANTITY_CHOICES = [(i, str(i)) for i in range(1, 21)]
    quantity =
        forms.TypedChoiceField(label='Num. Entradas',
            choices=PRODUCT_QUANTITY_CHOICES, coerce=int)
    update = forms.BooleanField(required=False, initial=False, widget=forms.
        HiddenInput)
```

Listing 6.6: Campos del formulario de numero de productos.

OrdenCompra

La clase *OrdenForm(forms.ModelForm)* define un formulario a partir de un modelo, es decir, el formulario esta formado por campos del modelo en este caso **OrdenModel**.

```
class OrdenForm(forms.ModelForm):
    class Meta:
        model = Order
        fields = '__all__'
```

Listing 6.7: Campos del formulario de Ordenes.

6.3.5. Vistas

Pasamos al ultimo componente que se encarga de interactuar con la BBDD para cada una de las peticiones que recibe el servidor y de esta forma guardar y obtener la información que se solicita, a continuación explicamos en detalle cada una las vistas implementadas.

Página principal

La función *PageInit(request)* devuelve la información de la pagina principal de la Web. Realiza una búsqueda dentro de la tabla de Eventos y Artistas filtrando el contenido por la fecha de inserción ya que nos interesa solo los elementos que se han introducido en los últimos 15 días y enviamos la información al fichero **esqueletoWeb.html**.

```
def PageInit(request):
    dateTimerHoy = datetime.datetime.now()
    dateTimerPass = datetime.datetime.now()
    date=dateTimerPass.date();
    numDias = datetime.timedelta(days=15);
    datePass=dateTimerHoy-numDias
    list_NewVideos=Video.objects.filter(fechaModif__range=[datePass,
        dateTimerHoy])
    list_NewEvento=Evento.objects.filter(fechaModif__range=[datePass,
        dateTimerHoy])
    list_NewCantante=Cantante.objects.filter(fechaModif__range=[datePass,
        dateTimerHoy])
    context = {
        'eventos':list_NewEvento,
        'cantantes':list_NewCantante
    }
    return render(request, 'esqueletoWeb.html', context)
```

Listing 6.8: Vista pagina principal.

Eventos

La función *pageEvent(request,idEvento)* recibe como parámetro el identificador del evento seleccionado con el que se realiza un filtrando en la tabla Eventos para obtener la información solicitada y enviarla al fichero **Evento.html**.

```
def PageEvent (request,idEvento):
    eventSelec = Evento.objects.get(id=idEvento)
    formCantidad = itemsCountsForm()
    context = {'evento':eventSelec,'form':formCantidad}
    return render(request, 'Evento.html', context)
```

Listing 6.9: Vista de la pagina de Eventos.

Artista

La función *PageCantante(request,idArtista)* recibe como parámetro el identificador del cantante seleccionado con el que se realiza un filtrando en la tabla Artistas para obtener la información solicitada y enviarla al fichero **Artista.html**.

```
def PageCantante (request,idCantante):
    cantanteSelec = Cantante.objects.get(id=idCantante)
    context = {'cantanteSelec':cantanteSelec}
    return render(request, 'Artista.html', context)
```

Listing 6.10: Vista pagina de Artista.

Contacta

La función '*PageContact(request)*' se encarga de tratar las peticiones de información que el usuario realiza. Para ello, la función evalúa primero el método de la petición ya que si es **GET** genera una instancia de *ContactaForm()* mientras que si es **POST** se obtiene la información del cuerpo del mensaje y genera con ella una instancia *ContactaForm()* por medio de la cual valida que los campos sean correctos a través del método *is_valid()*.

Si el método es valido se crea un nuevo elemento en la tabla **Contacta** con la información anterior en caso contrario se informa del error.

```
def PageContact(request):
    if request.method == 'POST':
        formContacta = ContactaForm(request.POST)
        if formContacta.is_valid():
            usuario = formContacta.cleaned_data['nombre']
            email = formContacta.cleaned_data['correo']
            tlf = formContacta.cleaned_data['numeroTlf']
            area = formContacta.cleaned_data['area']
            motivo = formContacta.cleaned_data['motivo']
            newContact = Contacta(nombre=usuario, email=email, telefono=tlf, motivo=
                area, texto=motivo);
            newContact.save()
            return HttpResponseRedirect('/WebMutimedia/')
        else:
            contacta = ContactaForm()
            context = {
                'form':contacta
            }
            return render(request, 'contacta.html', context)
```

Listing 6.11: Vista Petición del contacta.

Buscador

La función *SearchItem(request)* realiza una búsqueda dentro de las tablas de Eventos y Cantantes con la información que el usuario envía en el cuerpo de la petición y envía el resultado al fichero **desplegable.html**

```
def SearchItem(request):
    text= request.POST['textRequest']
    infoCantante=Cantante.objects.filter(nombre__startswith=text)
    infoEvento=Evento.objects.filter(nombre__startswith=text)
    contexto={'ListEvento':infoEvento,
              'ListCantante':infoCantante
    }
    return render(request,'desplegable.html',contexto)
```

Listing 6.12: Vista Buscador.

Las próximas dos vistas realizan peticiones al WebServices de Google Maps por lo que es necesario instalar el modulo resp de Python por medio de '*pip install resp*'. En cada petición se utiliza el método *response.get(url,data)* que recibe como parámetro la url donde se encuentra localizado el servicio web y los datos de consulta.

Geolocalización

La función **WSRequest(request)** se encarga de realizar la petición al WS sobre la geolocalización de un determinado lugar que se recupera del cuerpo de la petición con el que compone la url de la petición.

```
def WSRequest(request):
    nameSite= request.POST['site']
    r = requests.get('https://maps.googleapis.com/maps/api/geocode/json?
                      address=' +nameSite+ '&key=AIzaSyBliq3S6sQ0pJSt1xWJDiMtPuM1sn9xzaM')
    return HttpResponse(r.text)
```

Listing 6.13: Vista de Geolocalización.

La respuesta del WebServices la obtiene el servidor por medio del método **resp.text** cuyo formato se ve en la figura 6.5.

```
{<Response [200]>
"results": [
  {
    "address_components": [
      {
        "long_name": "Sant Adrià de Besòs",
        "short_name": "Sant Adrià de Besòs",
        "types": [ "locality", "political" ]
      },
      {
        "long_name": "Barcelona",
        "short_name": "Barcelona",
        "types": [ "administrative_area_level_2", "political" ]
      },
      {
        "long_name": "Catalunya",
        "short_name": "CT",
        "types": [ "administrative_area_level_1", "political" ]
      },
      {
        "long_name": "Spain",
        "short_name": "ES",
        "types": [ "country", "political" ]
      }
    ],
    "formatted_address": "Sant Adrià de Besòs, Barcelona, Spain",
    "geometry": {
      "bounds": {
        "northeast": {
          "lat": 41.4352038,
          "lng": 2.2371873
        },
        "southwest": {
          "lat": 41.4125921,
          "lng": 2.2061164
        }
      },
      "location": {
        "lat": 41.42668,
        "lng": 2.2245813
      },
      "location_type": "APPROXIMATE",
      "viewport": {
        "northeast": {
          "lat": 41.4352038,
          "lng": 2.2371873
        },
        "southwest": {
          "lat": 41.4125921,
          "lng": 2.2061164
        }
      }
    },
    "place_id": "ChIJw3gOxaqpBIRAA6kIeD6AAQ",
    "types": [ "locality", "political" ]
  }
],
"status": "OK"
}
```

Figura 6.5: Respuesta WebServices Geolocalizacion lugar.

Localización de Servicios

La función **WSRequestSite(request)** se encarga de realizar otra petición al WS sobre la localización de servicios alrededor de un punto. Para realizar esta consulta obtiene del cuerpo del mensaje las coordenadas y el servicio para componer la url de la petición.

```
def WSRequestSite(request):
```

```

infoSite= request.POST['infoSite']
r = requests.get('https://maps.googleapis.com/maps/api/place/
    nearbysearch/json?' + infoSite + '&key=
    AIzaSyBliq3S6sQ0pJsT1xWJDiMtPuM1sn9xzaM')
return HttpResponse(r.text)

```

Listing 6.14: Vista búsqueda de servicios.

El servidor obtiene la respuesta por medio del método `resp.text` cuyo formato se ve en la figura 6.6 y finalmente se envía al navegador.

```

.....
{
  "geometry": {
    "location": {
      "lat": 41.42668,
      "lng": 2.2245813
    },
    "viewport": {
      "northeast": {
        "lat": 41.4352038,
        "lng": 2.2271873
      },
      "southwest": {
        "lat": 41.4125921,
        "lng": 2.2061164
      }
    }
  },
  "icon": "https://maps.gstatic.com/mapfiles/place_api/icons/geocode-71.png",
  "name": "Sant Adrià de Besòs",
  "photos": [
    {
      "height": 1982,
      "html_attributions": [
        "\ud83c\udfa Samuel Garcia Fernandez\ud83c\udfa"
      ],
      "photo_reference": "CmRaQ0dubAAAAYzGd0AfufXEmhKWD10rewIGB151w7Ttq8Y7vXXAbg6uLy5fne8T0ATSLVs5X11_dBfF1N_FmHgVkdnevEcIChHuC1Ms_JEclstUEdxa01apgSyBP1USSCCgdJfyosnqk8w7Bz31f6RVxXBeinhJQd
      "width": 2643
    }
  ],
  "place_id": "ChIJw3gOKaq8p8IRAAkTe06AQ",
  "reference": "CmRaQ0dubAAAAYzGd0AfufXEmhKWD10rewIGB151w7Ttq8Y7vXXAbg6uLy5fne8T0ATSLVs5X11_dBfF1N_FmHgVkdnevEcIChHuC1Ms_JEclstUEdxa01apgSyBP1USSCCgdJfyosnqk8w7Bz31f6RVxXBeinhJQd
  "type": ["locality", "political"],
  "vicinity": "Sant Adrià de Besòs"
}

```

Figura 6.6: Respuesta WebServices Servicios Adicionales

Otro punto a tener en cuenta es la gestión de los usuarios de la aplicación ,por lo que se definen las siguientes vistas.

Register

La función `PageRegister(request)` gestiona el registro de los usuarios dentro de la BBDD de la aplicación.La función evalúa el método de la petición ya que si es `GET` crea una instancia del formulario `registerForm()` mientras que si es `POST` se obtiene los datos del cuerpo de la petición para crear la instancia del formulario `registerForm()` y validar la información por medio el método `is_valid()` para poder generar un nuevo elemento en la tabla User y UserProfile.

```

def PageRegister(request):
    if request.method == 'POST':
        formRegister = RegisterForm(request.POST)
        if formRegister.is_valid():
            #datos que guardamos en tabla User
            usuario = formRegister.cleaned_data['nick']
            email = formRegister.cleaned_data['correo']
            nombre = formRegister.cleaned_data['nombre']
            apellido = formRegister.cleaned_data['apellido']
            password = formRegister.cleaned_data['password2']
            password2 = formRegister.cleaned_data['password2']
            #guardamos el usuario
            newUser = User.objects.create_user(username=usuario,email=email,
                password=password,first_name=nombre,last_name=apellido)

```

```

newUser.save()
#datos que se guardaran en el perfil del User
edad = formRegister.cleaned_data['edad']
tlf = formRegister.cleaned_data['telefono']
direccion = formRegister.cleaned_data['direccion']
sexo = formRegister.cleaned_data['sexo']
pais = formRegister.cleaned_data['pais']
provincia = formRegister.cleaned_data['provincia']
perfilNewUser = PerfilUser(usuario=newUser, telefono=tlf,
    direccion=direccion, sexo=sexo, pais=pais, provincia=provincia, edad=
    edad)
perfilNewUser.save()
return HttpResponseRedirect('/WebMutimedia/')
else:
register = RegisterForm()
context = { 'form':register }
return render(request,'register.html',context)

```

Listing 6.15: Vista de registro de usuarios.

Login

La función *PageLogin(request)* validar que el username y contraseña que el usuario ha introducido son correctas y existe dentro de la BBDD. La información se obtiene del cuerpo de la petición para crear una instancia del formulario *loginForm()* y validar con el método '*autenticathe()*' que el usuario existe en el sistema. Si la autenticación es correcta logeamos al cliente con método *login()* en caso contrario informamos que este proceso no ha funcionado correctamente.

```

def PageLogin(request):
if request.method == 'POST':
user = request.POST['usuario']
passw = request.POST['pass']
data = {'nick': user, 'password': passw}
Formlogin = LoginForm(data)
if Formlogin.is_valid():
user = Formlogin.cleaned_data['nick']
password = Formlogin.cleaned_data['password']
usuario = authenticate(username=user, password=password)
if usuario is None:
mensaje = False
return HttpResponseRedirect(mensaje)
else:
mensaje = True
login(request, usuario)
return HttpResponseRedirect(mensaje)

```

Listing 6.16: Vista de login usuarios.

Perfil Usuario

La función *PageProfile(request)* busca el perfil del usuario por medio del **username** que se encuentra como parámetro de la variable **request**.

Con el nombre realizamos una búsqueda dentro de la tabla User para obtener el id del cliente y de esta forma obtener el perfil en la tabla ProfileUser y enviar esta información al fichero **profile.html**.

```
def AddCarShop(request,idEvento,idTicket):
    numEntradas = request.POST['quantity']
    v_Evento = Evento.objects.get(id=idEvento)
    ticket = Entradas.objects.get(id=idTicket)
    precio = ticket.precio
    tipoTicket = ticket.tipoEntrada
    Total = int(numEntradas)*precio
    cart = CartShop(request)
    cart.add(v_Evento.nombre,str(v_Evento.imgCartel),
             idTicket,tipoTicket,numEntradas,Total)
    return HttpResponseRedirect('/WebMutimedia/Eventos/' +idEvento+ '/')
```

Listing 6.17: Peticiones de añadir productos.

Logout

La función *appLogout(request)* se encarga de cerrar la sesión actual del usuario a través del método *logout(request)* y redirecciona a la pagina principal de la aplicación.

```
def appLogout (request):
    logout(request)
    return HttpResponseRedirect('/WebMutimedia/')
```

Listing 6.18: Vista Logout usuario.

Para finalizar esta sección creamos una serie de vistas que encargan de gestionar los productos que los usuarios desean adquirir durante su estancia en la aplicación.

Añadir

La función *AddCarShop(request,idEvento,idTicket)* recibe como parámetro el id del Evento y el de la entrada para realizar una búsqueda dentro la tabla Entradas y Eventos ademas de obtener el numero de entradas del cuerpo de la petición.

Con la información obtenida de la entrada y el numero de entradas calculamos el precio total para guardarlo posteriormente. Para finalizar el proceso realiza una instancia del objeto *CartShop()* y guardamos el producto a través del método *addCart(name,linkImg,typeTicket,unidades,precio)* al que se le pasan los siguientes parámetros :

- **Name:** nombre del evento.
- **LinkImg:** imagen del evento.
- **TypeTicket:** tipo de entrada.
- **Unidades:** numero de entradas.
- **Precio:** valor de la compra.

```
def AddCarShop(request,idEvento,idTicket):
    numEntradas = request.POST['quantity']
    v_Evento = Evento.objects.get(id=idEvento)
    ticket = Entradas.objects.get(id=idTicket)
    precio = ticket.precio
    tipoTicket = ticket.tipoEntrada
    Total = int(numEntradas)*precio
    cart = CartShop(request)
    cart.add(v_Evento.nombre,str(v_Evento.imgCartel),
             idTicket,tipoTicket,numEntradas,Total)
    return HttpResponseRedirect('/WebMutimedia/Eventos/' + idEvento + '/')
```

Listing 6.19: Vista añadir productos carrito.

Detalle

La función *DetailCarShop(request)* se encarga de mostrar el contenido que el usuario tiene en su carrito. Para ello realizamos una instancia del objeto '*CartShop(request)*' y del formulario *itemsCountForm()* que se utiliza para que el usuario pueda actualizar el numero de entradas que tiene en su carrito y enviarlo al fichero **CartDetail.html**.

```
def DetailCarShop(request):
    objCart = CartShop(request)
    formCantidad = itemsCountsForm(initial={'update': True})
    context = {'objCart':objCart,'formCantidad':formCantidad}
    return render(request,'CartDetail.html',context)
```

Listing 6.20: Vista detalle contenido carrito.

Update

La función *UpdateCart(request,idEvento,idTicket)* recibe como parámetro el idEvento y el idTicket además del nuevo numero de entradas del cuerpo de la petición. Con la información buscamos dentro de la tabla Eventos y Entradas. Tras esto vuelve a calcular el precio con el numero de entradas y el precio de la entrada con la información obtenida de la consulta anterior y genera una instancia del objeto '*CartShop(request)*' del cual utilizamos el método *getNameKey(nameEvento,idTicket)* para obtener el nombre de la clave donde se encuentra el antiguo valor del objeto y así utilizar el método *update(cantidad,precio,key)* para finalizar la actualización del contenido y redireccionar a '/WebMutimedia/DetailCar'.

```
def UpdateCart(request,idEvento,idTicket):
    newNumEntradas = request.POST['quantity']
    ticket = Entradas.objects.get(id=idTicket)
    v_Evento = Evento.objects.get(id=idEvento)
    Total = int(newNumEntradas)*ticket.precio
    #SIGUIENTE PASAMOS A ENCONTRAR LA CLAVE
    instCart = CartShop(request)
    nameKey = instCart.getNameKey(v_Evento.nombre,idTicket)
    instCart.update(nameKey,newNumEntradas,Total)
    return HttpResponseRedirect('/WebMutimedia/DetailCar/')
```

Listing 6.21: Vista actualizar contenido del carrito.

Remove

La función `deleteItemCar(request,idEvento,idTicket)` recibe como parámetro el idEvento y el idTicket. En este caso se realiza una instancia del objeto '`CartShop(request)`' del cual utilizamos el método `getNameKey(nameEvento,idTicket)` para obtener el nombre de la clave asociada al objeto y así poder eliminarlo por medio del método '`remove(key)`' y redirigir a `/WebMutimedia/DetailCar/`.

```
def deleteItemCar(request,idEvento,idTicket):
    instCart = CartShop(request)
    v_Cart = instCart.cart
    v_Evento = Evento.objects.get(id=idEvento)
    nameKey = instCart.getNameKey(v_Evento.nombre,idTicket)
    instCart.remove(nameKey)
    return HttpResponseRedirect('/WebMutimedia/DetailCar/')
```

Listing 6.22: Vista eliminar producto del carrito.

6.4. Front-End

Esta capa de la aplicación se encarga de generar ficheros html por medio del lenguaje de plantillas de Django y de esta forma cargar la información que envía el servidor. Definimos el fichero '`esqueletoWeb.html`'² como el fichero raíz de la aplicación ya que contiene los elementos comunes entre las plantillas con el fin de no repetir código y hacer uso de la herencia entre plantillas, ademas se define los siguientes bloques dentro del fichero que permiten sustituir o introducir nueva información.

- % block JS %: bloque para incrustar scripts.
- % block css %: bloque para incrustar ficheros de estilos.
- % block body %: bloque para incluir el cuerpo de la pagina correspondiente.

Ahora pasamos a detallar el contenido de cada una de las plantillas que utiliza la aplicación y su funcionalidad de la misma forma de un ejemplo de la apariencia que tiene una vez las ha cargado el navegador.

6.4.1. Barra de Navegación

Es el elemento principal ya que en el se encuentran los distintos accesos a las ventas de la aplicación. Muestra un desplegable de Cantantes a través de la variable `listCantantes` y de Eventos a través de la variable `listFestivales`. Ademas de esto incluye los siguientes enlaces:

1. Contacta
2. Register

²Apéndice 3

3. Login
4. Perfil

Por ultimo, tiene un enlace al detalle del carrito de la compra en que se muestra el numero de elementos por medio la variable **items** y el total de la compra por medio de la variable **precio**,imagen 6.7.



Figura 6.7: Barra de navegación.

Buscador

Permite a los usuarios buscar cantantes o eventos introduciendo la cadena de caracteres en el buscador obteniendo como resultado un desplegable con las coincidencias encontradas, figura 6.8.



Figura 6.8: Buscador de la aplicación.

Funcionalidad: Para establecer la conexión con el servidor y enviar la cadena de caracteres creamos la función **sendInfo()** que recupera la información introducida por el usuario y valida su longitud ya que tiene que ser mayor a dos caracteres para realizar una búsqueda correcta.

Tras la validación realizamos una llamada Ajax a la url '*/WebMutimedia/Search/*' por medio de la función '**AjaxRequest(data)**'.

```

function sendInfo() {
    texto = $('#textSearch').val();
    console.log(texto);
    if(texto.length > 2) {
        AjaxRequest(texto);
    } else{
        $("table#tbSearch").remove();
    }
}

function AjaxRequest(data) {
    console.log(data);
    $.ajax({
        type: "POST",
        url: '/WebMutimedia/Search/',
        data: {
    
```

```

    'textRequest' : data,
    'csrfmiddlewaretoken': $("input[name=csrfmiddlewaretoken]").val(),
},
success: resulSearch,
dataType: 'html',
});
}
}

```

Listing 6.23: Petición Ajax buscador aplicación.

La función **resulSearch(data, textStatus, jqXHR)** definida en la petición se encarga de recibir la respuesta e insertarla en la pagina.

```

function resulSearch(data, textStatus, jqXHR) {
    console.log(data);
    $('#listSearch').html(data);
}

```

Listing 6.24: Respuesta Ajax buscador aplicación.

Contacta

Al acceder a este enlace se redirecciona a la url '*/WebMutimedia/Contacta*' que muestra la información de la variable **form** en el fichero '**Contacta.html**'³, figura 6.9.

The image shows a contact form titled 'Contacta'. It contains four input fields: 'Nombre', 'Email', and 'Teléfono', each with a corresponding label above it. Below these is a section labeled 'Área:' with four radio buttons labeled 'Compra', 'Técnica', 'Eventos', and 'Otros'. At the bottom of the form is a large text area and a blue 'Enviar' button.

Figura 6.9: Pagina Contacta.

Register

Al acceder a este enlace se redirecciona a la url '*/WebMutimedia/Register*' en que se muestra la información de la variable **form** en el fichero '**Register.html**'⁴, figura 6.10.

³Apéndice 3: Contacta

⁴Apéndice 3: Register

Figura 6.10: Pagina Registro.

Login

Al pulsar sobre el enlace generamos un formulario que tiene como campos el nombre de usuario y password para enviar la información a la url '/WebMultimedia/Login' y validar el contenido.

```
<form id="post_form" method="POST">
{ % csrf_token %}
<div class="form-group">
<p id="login_error" style="position: center"></p>
<label for="username"><span class="glyphicon glyphicon-user"></span>
    Username</label>
<input type="text" class="form-control" id="username" placeholder="Enter Username">
</div>
<div class="form-group">
<label for="psw"><span class="glyphicon glyphicon-eye-open"></span>
    Password</label>
<input type="password" class="form-control" id="psw" placeholder="Enter password">
</div>
<button type="submit" class="btn btn-success btn-block"><span class="glyphicon glyphicon-off"></span>
    Login</button>
</form>
```

Listing 6.25: Petición eliminar producto del carrito.

Una vez el navegador termina de cargar el fichero y al acceder al enlace se obtiene como resultado la figura 6.11.

Figura 6.11: Formulario Login.

Perfil

Al acceder a este enlace se redirecciona a la url '*/WebMultimedia/Perfil*' en que se muestra la información de la variable **ProfileUser** en el fichero '*perfil.html*'⁵ que consta de la información del usuario durante el proceso de registro y las compras que ha realizado en la aplicación .

Figura 6.12: Pagina Perfil Usuario.

Logout

Al acceder a este enlace se redirecciona a la url '*/WebMultimedia/Logout*' para finalizar el sesión del usuario.

Resumen Compra

Es necesario disponer de una zona en la que se muestre el numero y el precio total de los productos que el cliente ha añadido a el carrito este punto se desarrolla en detalle mas

⁵Apéndice 3: Perfil

adelante.

6.4.2. Home

La pagina principal de la aplicación consta de un carrusel de imágenes que se cargan de forma estática y de una sección de novedades de cantantes y eventos añadidos en los últimos 15 días por medio de las variables '**ICantantes**' y '**IEventos**' respectivamente ⁶, figura 6.13.

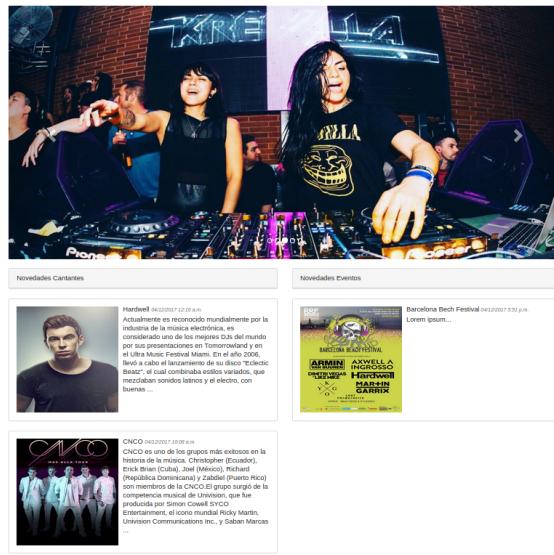


Figura 6.13: Pagina Principal.

6.4.3. Cantantes

El usuario selecciona un cantante del desplegable provocando la redirección a la url '[/WebMultimedia/Cantante/*idCantante*](#)', donde **idCantante** corresponde al identificador del cantante dentro de la tabla **Cantante**.

La información se vuela en el fichero '**Cantante.html**' ⁷ que consta de cuatro secciones, que se explican a continuación.

Información

La información correspondiente a la tabla **Cantante** se vuela en esta sección a través de la variable **cantanteSelec** que contiene la información enviada por el servidor y se distribuye dentro del panel, figura 6.14.

⁶Apéndice 3:Home

⁷Apéndice 3:Cantante

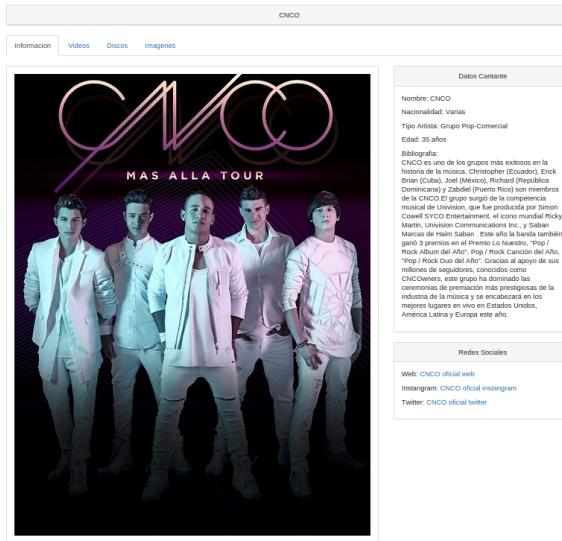


Figura 6.14: Pagina Cantante panel Información.

Vídeos

La información correspondiente a la tabla Vídeo en relación al cantante seleccionado se vuela en esta sección a través de la variable **cantanteSelec.listVideos** que se distribuye dentro del panel creando una lista y la correspondiente zona de reproducción, figura 6.15.

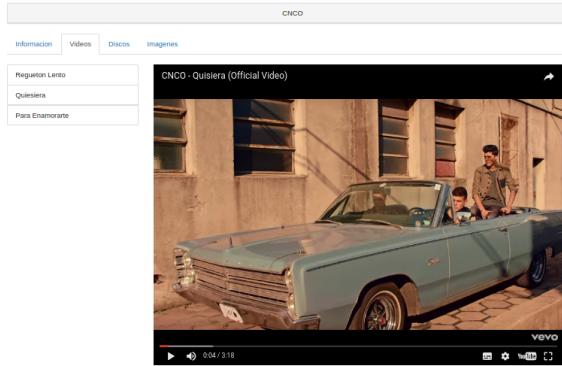


Figura 6.15: Pagina Cantante panel Información.

Funcionalidad: Para permitir a los usuarios seleccionar un vídeo de la lista creamos la función '*loadVideo(file, tipo)*' y lo vinculamos a cada uno de los elementos de la lista.

La función evalúa el tipo de archivo del que se trata ya que puede ser **<iframe>** o **<video>** y así reproducirlo a través del elemento adecuado.

```
function loadVideo(file, tipo) {
  if(tipo == 'iframe'){
    $('iframe').attr("src",file);
```

```

$( "video" ).hide();
$( "iframe" ).show();
} else{
  var path =' /media/' +file;
  $( 'video' ).attr("src",path );
  $( "iframe" ).hide();
  $( "video" ).show();
}
}

```

Listing 6.26: Función carga de videos.

Discos

La información correspondiente a la tabla Discos y Canciones se vuelca en esta sección a través de la variable **cantanteSelec.listDiscos** y **album.listCanciones** que se distribuye dentro en el panel, figura 6.16.

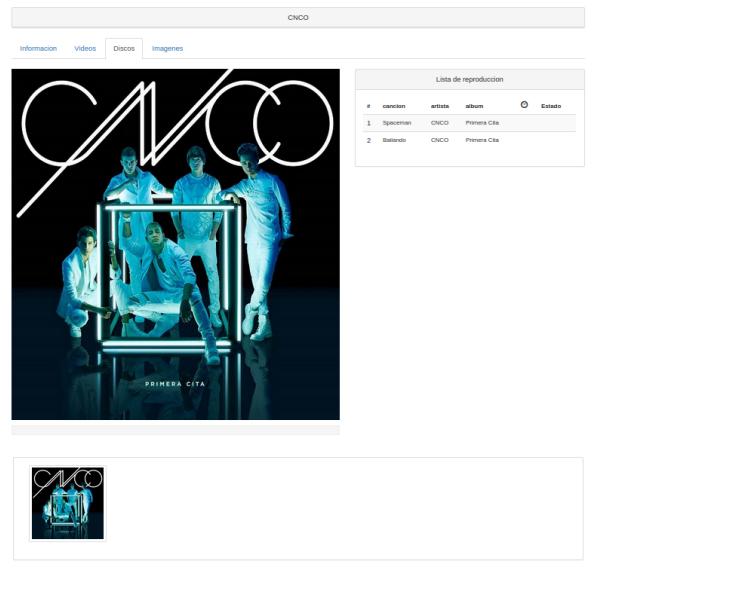


Figura 6.16: Pagina Cantante panel Discos.

Funcionalidad: Los usuarios pueden seleccionar un disco que desean escuchar pinchando sobre él provocando la llamada a la función **'loadList(this,imgLink)'**. La función se encarga de cargar la imagen correspondiente al disco seleccionado y carga las canciones en la lista de reproducción clonando el contenido de la etiqueta **<table>** que se encuentra oculto cuando se carga la pagina.

```

function loadList(elemento,imagen) {
  $('#ImgRepro').attr('src',' /media/' +imagen);
  var idElemento = $(elemento).attr("id");
  var divListSong = $('#'+idElemento).siblings('div');
  var idDivListSong = $(divListSong).attr("id");
  $("table#x").remove();
}

```

```
$(' #' +idDivListSong).children('table').clone().appendTo($(' #x'));
```

Listing 6.27: Función carga lista de reproducción.

Una vez se ha cargado la lista de canciones definimos la función **loadSong(name,idElement,urlSong)** que permite reproducir la canción seleccionada. La función genera el path de la fuente que sirve el servicio con el parámetro **name** y pasa a comprobar si existe una reproducción en curso por medio de la etiqueta **<audio>** en caso afirmativo la pausa antes de producir el nuevo elemento.

Para iniciar la reproducción utiliza el método **onloadeddata** que se ejecuta cuando la etiqueta **<audio>** tiene toda la información del fichero y así obtiene su duración para llamar a la función **convertSeg_Min(duration)** y actualizar este valor en la lista de reproducción.

```
function loadSong(name,idElement,urlSong) {
    var idTimer = 'timer_'+idElement;
    var state_Actual = 'state_'+idElement;
    var path = '/media/'+urlSong;
    var audio = document.getElementById('repro');
    if(!audio.paused) {
        audio.pause();
        StopDrawProgres();
    }
    if (state_Actual != stado_Old && stado_Old != '') {
        document.getElementById(stado_Old).innerHTML = '';
    }
    audio.src = path;
    audio.onloadeddata=function() {
        var valor = audio.duration;
        var str_Time = convertSeg_Min(valor);
        document.getElementById(state_Actual).innerHTML = 'Reproduciendo';
        document.getElementById(idTimer).innerHTML = str_Time;
        porcentaje = convert(valor);
        drawProgress(porcentaje);
        audio.play();
        stado_Old = state_Actual;
    }
}
```

Listing 6.28: Función reproducir canción.

Imágenes

La información correspondiente a la tabla Imágenes en relación al cantante seleccionado se vuela en esta sección a través de la variable **cantanteSelec.listImagenes** que se distribuye dentro del panel creando una galería de imágenes, figura 6.17.

6.4.4. Eventos

El usuario selecciona un evento del desplegable provocando una redirección a la url **'/WebMultimedia/Eventos/idvento/'** donde **idvento** corresponde al identificador del cantante dentro de la tabla **Evento**.

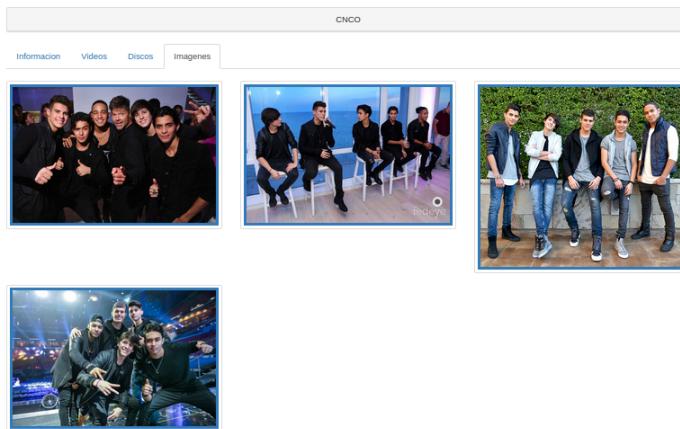


Figura 6.17: Pagina Cantante pestaña imágenes.

La información se renderiza en el fichero **Evento.html**⁸ que consta de cuatro secciones que explicamos a continuación..

1. Información

La información correspondiente la tabla Evento se vuelca en esta sección a través de la variable **evento** que contiene la información enviada por el servidor que se distribuye dentro del panel, figura 6.18.

The screenshot shows a web page for the 'Barcelona Beach Festival'. The main content area displays a large poster for the festival featuring various artists like Armin van Buuren, Axwell & Ingrosso, Hardwell, and Martin Garrix. To the right of the poster are two panels: 'Información' and 'AfterMovie'. The 'Información' panel contains event details: 'Tipo Evento: festival', 'País: España', 'Ciudad: Barcelona', 'Dirección: San Adrián del Besós', and 'Fecha: April 12, 2017'. The 'AfterMovie' panel shows a video player with the title 'Ultra Korea 2016 (Official 4K Aftermovie)'.

Figura 6.18: Pagina Evento panel Información.

⁸Apéndice 3:Eventos

2. Ubicación

En este panel se muestra la ubicación y servicios alrededor del dicha ubicación en un mapa de **Google Maps**. La información se distribuye en una sección donde se realizara la instancia del Mapa tras la renderización por medio del navegador y otra con servicios adicionales, figura 6.19.

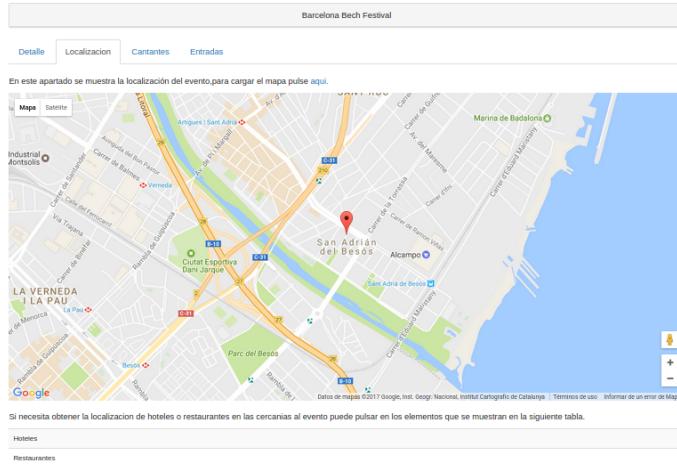


Figura 6.19: Pagina Evento panel ubicación.

Funcionalidad: Cuando la pagina se carga por completo creamos una llamada Ajax a la url `/WebMutimedia/WebServRequest/`. En el cuerpo de la petición enviamos el nombre del lugar del que deseamos obtener las coordenadas geográficas y definimos `searchSucess(data, textStatus, jqXHR)` que se encarga de recuperar la respuesta del servidor.

```
$ .ajax({
  type: "POST",
  url: "/WebMutimedia/WebServRequest/",
  data: {
    'site' : $('#direct').text(),
    'csrfmiddlewaretoken' : $("input[name=csrfmiddlewaretoken]").val(),
  },
  success: searchSucess,
  dataType: 'html',
});
```

Listing 6.29: Petición Ajax Ubicación Lugar.

La función `searchSucess(data, textStatus, jqXHR)` parsea a formato JSON la información que contiene la variable `data` de donde obtiene la información de las coordenadas accediendo a `geometry.location` tras esto se llama a la función `WarchMap()` que es la encargada de crear el mapa con esta información.

```
function searchSucess(data, textStatus, jqXHR) {
  var infoRequest = JSON.parse(data);
  coordenadas = infoRequest.results[0].geometry.location;
  WarchMap();
```

```
}
```

Listing 6.30: Respuesta Ajax Ubicación Lugar.

Para utilizar los mapas interactivos de google maps es necesario descargarse el **script JS** que han creado para esto. Tras esto en la variable **mapProp** se definen las características que queremos tenga el mapa y generamos la instancia del mapa por medio ***new google.maps.Map()*** al que se le pasa el lugar del documento donde se tiene que crear y las propiedades anteriores.

Hasta este momento solo se visualiza el mapa para mostrar con mas claridad el punto exacto donde se celebra el evento creamos un **marker** al que le pasa la posición, el tipo de animación, el mapa donde se tiene que crear y un cartel informativo.

```
function WarchMap() {
    var lat = parseFloat(coordenadas.lat);
    var long = parseFloat(coordenadas.lng);
    var mapProp = {
        center: new google.maps.LatLng(lat, long),
        zoom: 15,
        mapTypeId: google.maps.MapTypeId.ROADMAP,
    };
    map = new google.maps.Map(document.getElementById("Maps"), mapProp);

    var marker = new google.maps.Marker({
        position: {lat:parseFloat(coordenadas.lat), lng:parseFloat(coordenadas.lng)},
        draggable: true,
        animation: google.maps.Animation.BOUNCE,
        map: map,
        title: 'Concierto'
    });
}
```

Listing 6.31: Creación Mapa con la Ubicación.

3. Cantantes

La información correspondiente a la tabla Cantantes en relación al evento seleccionado se vuelca en esta sección a través de la variable **evento.listcantantes** que se distribuye dentro del panel, figura 6.20.

4. Entradas

La información correspondiente a la tabla Entradas en relación al evento seleccionado se vuelca en esta sección a través de la variable **evento.listEntrada** que se distribuye dentro del panel, figura 6.21.

Funcionalidad: Los usuarios añaden contenido al carrito de la compra por medio del botón **Add Car** del formulario que redirecciona la acción a la url **/WebMultimedia/AddCar/evento.id/ticket.id**.

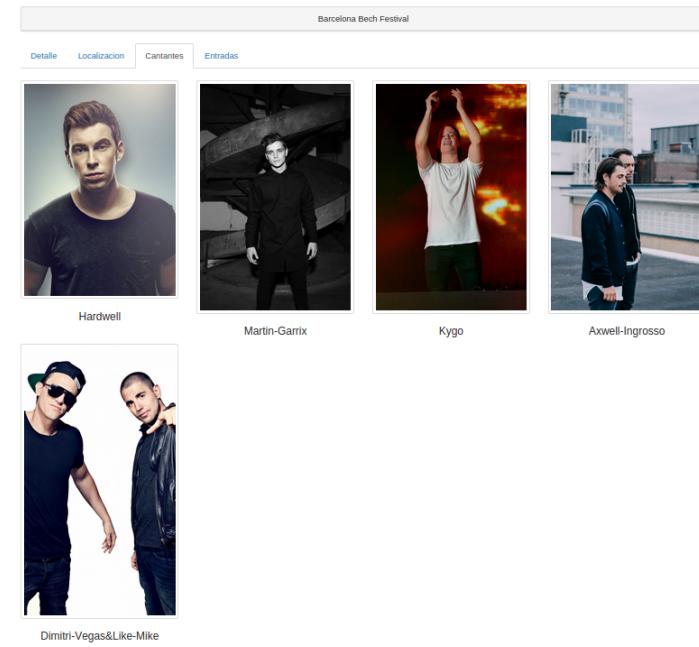


Figura 6.20: Pagina Evento panel Cantantes.

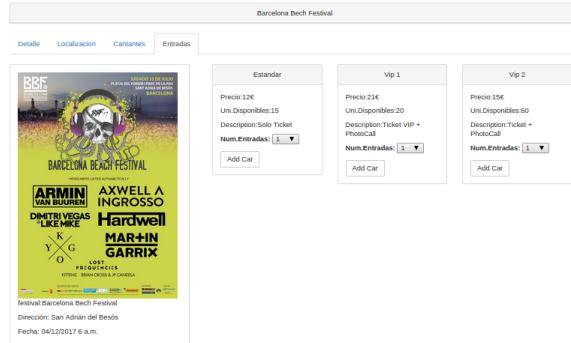


Figura 6.21: Pagina Evento pestaña Entradas.

6.4.5. Carrito de la Compra

Es necesario permitir a los usuarios visualizar el detalle de su compra por lo que al pulsar sobre el enlace de la barra de navegación accedemos a la url '/WebMultimedia/DetailCart/' mostrando cada uno de los elementos en forma de tabla a través de la variable `objCart.cart` en el fichero `CarDetail.html`⁹, imagen 6.22.

Funcionalidad: Dentro del detalle de la compra es necesario proveer de varias acciones que explicamos a continuación para que el usuario pueda realizar modificaciones del contenido de su carrito.

⁹Anexo 3:Detalle CarShop

Detalle compra					
Imagen	Producto	Cantidad	Update	Eliminar	Valor Total
	Barcelona Beach Festival Vip 1	1	<input type="button" value="Update"/>	<input type="button" value="Remove"/>	21€
	Barcelona Beach Festival Extender	2	<input type="button" value="Update"/>	<input type="button" value="Remove"/>	24€
Total:					45€
<input type="button" value="Checkout"/>					

Figura 6.22: Pagina Detalle Carrito.

- Actualizar:** Al seleccionar un nuevo numero de entradas y pulsar '**Update**' se redirecciona la petición a la url '*WebMultimedi/UpdateCart/idEvento/idTicket/*' que se encarga llevar acabo la acción.
- Eliminar:** Al pulsar '**Remove**' se redirecciona la petición a la url '*WebMultimedia/RemoveCart/idEvento/idTicket/*'
- Checkout:** Permite pasar a tramitar la orden de compra de los producto redireccinando la acción a la url '*WebMultimedia/Checkout/*'.

Orden de Compra

Esta ventana es visible cuando el usuario desea terminar la compra del contenido de su carrito ya que muestra un resumen de los elementos añadidos a través de la variable **objCart.cart** y el formulario con los campos de la orden a llenar en el fichero **Orden.html**¹⁰, figura 6.23.

Orden de compra	Su orden
Nombre <input type="text"/>	Producto Barcelona Beach Festival
Apellido <input type="text"/>	Vip 1 x 1
Email <input type="text"/>	Precio 21€
Teléfono <input type="text"/>	Barcelona Beach Festival
Dirección <input type="text"/>	Vip 1 x 1
Código Postal <input type="text"/>	24€
País <input type="text"/>	Total 45€
<input type="button" value="Enviar"/>	

Figura 6.23: Pagina Orden Compra.

6.5. Pruebas

¹⁰Apéndice 3:Orden

Capítulo 7

VideoConferencia usando WebRTC

7.1. Enunciado

Las aplicaciones multimedia en tiempo real permiten conectar a distintos usuarios a través de la red e intercambiando información de audio,vídeo u otro tipo. Un ejemplo de este tipo de aplicación es Skype que ha tenido mucho éxito ya que permite a sus usuarios establecer videollamadas e intercambiar información una vez se haya instalado el software.



Figura 7.1: Skype interfaz videollamada.

Por lo tanto en esta ultima practica se pide desarrollar una aplicación web similar a Skype que permita a los usuarios conectarse entre si a través de la url de la aplicación.

Requisitos Los usuario dentro de la aplicación podrán seleccionar los elementos multimedia(audio y vídeo) que desean compartir, ademas de seleccionar o crear una sala donde se establecerá la videoconferencia.

los participantes de la videoconferencia en la sala dispondrán de un chat y de la posibilidad de intercambiar ficheros por medio de WebRTC ya que la comunicación sera Peer-to-Peer en este punto.

Para llevar acabo lo descrito anteriormente sera necesario crear un servidor de señalización para establecer la fase de comunicación inicial de WebRTC.

Tecnologías Sera necesario emplear WebRTC, API File, JavaScript para el lado cliente mientras que para el servidor de señalización utilizamos NodeJS. Ademas se utilizará WebSockets como mecanismo de comunicación con el servidor y Boostrapps para trabajar la apariencia de la aplicación.

7.2. Diseño

7.3. Desarrollo Servidor Señalización

El primer paso para crear el servidor es importar la librería **node-static,http y socket.io**. La librería **http** permite crear un servidor a través del método **createServer** en el puerto 8181 en nuestro caso mientras que la librería **node-static** permite enviar el documento a los usuarios que se conecten.

Finalmente, la librería **socket.io** se utiliza para establecer comunicación entre el cliente y servidor, figura 7.2.

```
var static = require('node-static');
var http = require('http');
var file = new(static.Server)();
var app = http.createServer(function (req, res) {
    file.serve(req, res);
}).listen(8181);
var io = require('socket.io').listen(app);
```

Listing 7.1: Creación Servidor de Señalización.

```
C:\Users\waltercito\Desktop\tfgPractica4\Chat_multi_usuario>node server.js
Servidor escuchando en el puerto:8181
  info  - socket.io started
```

Figura 7.2: Ejecución Servidor de Señalización APP.

El siguiente paso es permitir al servidor gestionar adecuadamente cada conexión que se establece a través de socket.io. Por ello, a través del método 'connected' mantendremos conectado.

Primero se define el evento **InfoRoom** que gestiona las peticiones sobre las salas disponibles. A esta petición el servidor contesta con un mensaje **ReplayInfoRoom** con la lista de las salas en la variable **listRoom**, figura 7.3.

```
socket.on('infoRoom', function(name) {
  socket.emit('ReplayInfoRoom', listRooom);
});
```

Listing 7.2: Request/Replay lista de salas existentes.

```
Node.js command prompt - node server.js
C:\Users\waltercito\Desktop\tfgPractica4\Chat_multi_usuario>node server.js
Servidor escuchando en el puerto:8181
  info  - socket.io started
  debug - served static content /socket.io.js
  debug - client authorized
  info  - handshake authorized X_PYhV3DxQ-4IVXLfF7_
  debug - setting request GET /socket.io/1/websocket/X_PYhV3DxQ-4IVXLfF7_
  debug - set heartbeat interval for client X_PYhV3DxQ-4IVXLfF7_
  debug - client authorized for
  debug - websocket writing 1::[{"name": "ReplayInfoRoom", "args": [{"name": "streaming", "state": ""}]}]
Se ha establecido una conexión con un browser.
Recibimos un mensaje InfoRoom.
Enviamos un mensaje ReplayInfoRoom:
  debug - websocket writing 5:::{ "name": "ReplayInfoRoom", "args": [{"name": "streaming", "state": ""}]} }
```

Figura 7.3: Request/Replay lista de salas del Servidor.

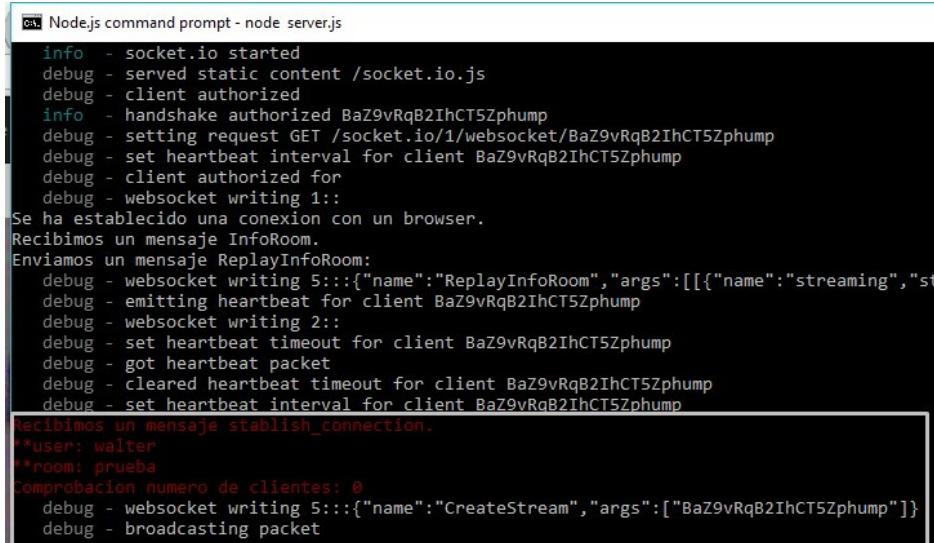
Tras el envío de la lista de salas disponibles el servidor recibirá un mensaje ..Definimos el evento **Stablish_connect** que recibe el nombre y sala a la que el usuario desea conectarse. Con esta información comprueba si el nombre de sala existe a través de la función **getRoom(nameRoom)** en caso de no existir se guardara en la lista con la función **setRoom()**. Ademas, comprueba que la sala no este completa para enviar un mensaje **CreateStream** con el identificador de conexión y un mensaje **NewJoined** al resto de usuario de la sala para que conozcan la existencia del nuevo miembro, figura 7.4.

```
socket.on('stablish_connection', function(name, room) {
  if(!getRoom(room)) {
    setRoom(room, '');
  };
  var numClients = io.sockets.clients(room).length;
  if(numClients < 3) {
    socket.username = name;
    socket.room = room;
    socket.join(room);
    socket.emit('CreateStream', socket.id);
    socket.broadcast.to(room).emit('New_Joined', socket.id);
  }else{
    socket.emit('RejectStream', socket.id);
  }
});
```

Listing 7.3: Request/Replay del establecimiento de conexión.

Tras haber enviado el mensaje **New_Joined** definimos el evento **message** para gestionar los mensajes de señalización entre navegadores ya que el servidor es transparente a este proceso.

```
socket.on('message', function(message, room) {
  io.sockets.socket(message.id_dest).emit('message', message);
```



```

  info  - socket.io started
  debug - served static content /socket.io.js
  debug - client authorized
  info  - handshake authorized BaZ9vRqB2IhCT5Zphump
  debug - setting request GET /socket.io/1/websocket/BaZ9vRqB2IhCT5Zphump
  debug - set heartbeat interval for client BaZ9vRqB2IhCT5Zphump
  debug - client authorized for
  debug - websocket writing 1:::
Se ha establecido una conexión con un browser.
Recibimos un mensaje InfoRoom.
Enviamos un mensaje ReplayInfoRoom:
  debug - websocket writing 5:::{name:"ReplayInfoRoom","args":[]}
  debug - emitting heartbeat for client BaZ9vRqB2IhCT5Zphump
  debug - websocket writing 2:::
  debug - set heartbeat timeout for client BaZ9vRqB2IhCT5Zphump
  debug - got heartbeat packet
  debug - cleared heartbeat timeout for client BaZ9vRqB2IhCT5Zphump
  debug - set heartbeat interval for client BaZ9vRqB2IhCT5Zphump
Recibimos un mensaje establish_connection.
"user: walter"
"room: prueba"
Comprobación número de clientes: 0
  debug - websocket writing 5:::{name:"CreateStream","args":["BaZ9vRqB2IhCT5Zphump"]}
  debug - broadcasting packet

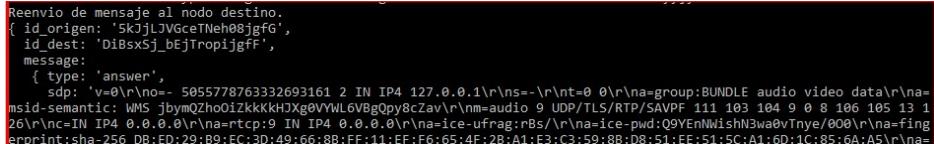
```

Figura 7.4: Request/Replay conexión al Servidor.

});

Listing 7.4: Mensajes de señalizacion.

La figura 7.5 encamina la oferta, la figura 7.6 encamina la oferta y la figura 7.7 encamina los IceCandidate. El servidor deja de participar en el intercambio de información entre los

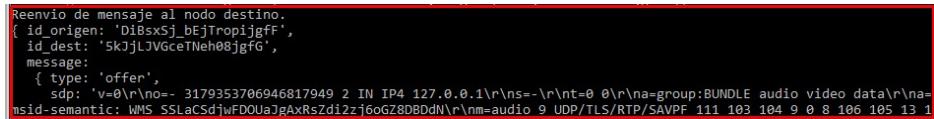


```

Reenvío de mensaje al nodo destino:
{ id_origen: '5kjLjVGceTNeh08jgfG',
  id_dest: 'DibxsXj_bejTropijgff',
  message:
    { type: 'answer',
      sdp: 'v=0\r\no=- 5055778763332693161 2 IN IP4 127.0.0.1\r\ns=-\r\nnt=0 0\r\nna=group:BUNDLE audio video data\r\nna=
msid-semantic: WMS jbymQZh0i0ZkkkH3xgVVWL6/Bgopy8cZav\r\nm=audio 9 UDP/TLS/RTP/SAVPF 111 103 104 9 0 8 106 105 13 1
26\r\nnc=IN IP4 0.0.0.0\r\nna=rtp;9 IN IP4 0.0.0.0\r\nna=ice-ufrag:nBz\r\nna=ice-pwd:9YEnWishN3wa0vTnye/008\r\nna=fing
erprint:sha-256 DB:ED:29:B9:EC:3D:49:66:8B:FF:11:EF:F6:65:4F:2B:AE:E3:C3:59:8B:D8:51:EE:51:5C:A1:6D:1C:85:6A:A5\r\nna='

```

Figura 7.5: Mensaje tipo Answer.



```

Reenvío de mensaje al nodo destino:
{ id_origen: 'DibxsXj_bejTropijgff',
  id_dest: '5kjLjVGceTNeh08jgfG',
  message:
    { type: 'offer',
      sdp: 'v=0\r\no=- 3179353706946817949 2 IN IP4 127.0.0.1\r\ns=-\r\nnt=0 0\r\nna=group:BUNDLE audio video data\r\nna=
msid-semantic: WMS SSLaCSdiwFDUlaJgAxRsZdi2z16oGZ8DBDdN\r\nm=audio 9 UDP/TLS/RTP/SAVPF 111 103 104 9 0 8 106 105 13 1

```

Figura 7.6: Mensaje tipo Offer.

peers cuando el proceso de señalización termina ya que desde ese momento la comunicación es Peer-to-Peer.

```
Renvío de mensaje al nodo destino.
{ id_origen: 'DiBsxsj_bEjTropijgff',
  id_dest: '5kJLJVGceTNeh08jgfG',
  message:
    { type: 'iceCandidate',
      label: 1,
      id: 'video',
      candidate: 'candidate:288073040 1 udp 2122255103 2001::5ef5:79fd:1868:1ede:d1e4:e4e4 61342 typ host generation 0
ufrag CKHe network-id 1 network-cost 50' }
    debug - websocket writing 5:::{ "name": "message", "args": [ { "id_origen": "DiBsxsj_bEjTropijgff", "id_dest": "5kJLJVGceTNeh08jgfG", "message": { "type": "iceCandidate", "label": 1, "id": "video", "candidate": "candidate:288073040 1 udp 2122255103 2001::5ef5:79fd:1868:1ede:d1e4:e4e4 61342 typ host generation 0 ufrag CKHe network-id 1 network-cost 50" } } ] }
```

Figura 7.7: Mensaje tipo Icecandidate.

7.4. Desarrollo Cliente Peer-to-Peer

Los usuarios se conectan al servidor a través de la url **http://localhost:8181** en un navegador que recibe el fichero **index.html**¹.

Se establece la conexión con el servidor a través de Websockets y se pide al usuario que introduzca el nombre que utilizará en la aplicación.

```
var socket = io.connect("http://localhost:8181");
```

Listing 7.5: Instancia WebSockets en el cliente.

Mensajes de sala

Cuando el usuario accede a la aplicación envía un mensaje **infoRoom** a través de WebSockets al servidor para obtener la lista de salas disponibles, figura 7.8.

```
socket.emit('infoRoom');
```

Listing 7.6: Petición salas disponibles.

Se establece el evento **ReplayInfoRoom** para recibir la lista de salas disponibles. Esta información se añade a la página, figura 7.8.

```
socket.on('ReplayInfoRoom', function(listRoom) {
  for(var i = 0; i < listRoom.length; i++) {
    var room = listRoom[i];
    $('#listRoom').append('<li><a id=' + room.name + '>' + room.name + '</a></li>');
    $('#' + room.name).click(function() {
      nameRoom = $(this).text();
      attachmentElements();
    });
  }
});
```

Listing 7.7: Creación desplegable de salas.

¹Apéndice 4: Index

Establecimiento de conexión

El usuario dispone en la barra de navegación de la posibilidad de seleccionar los elementos que desea compartir, figura 7.8.

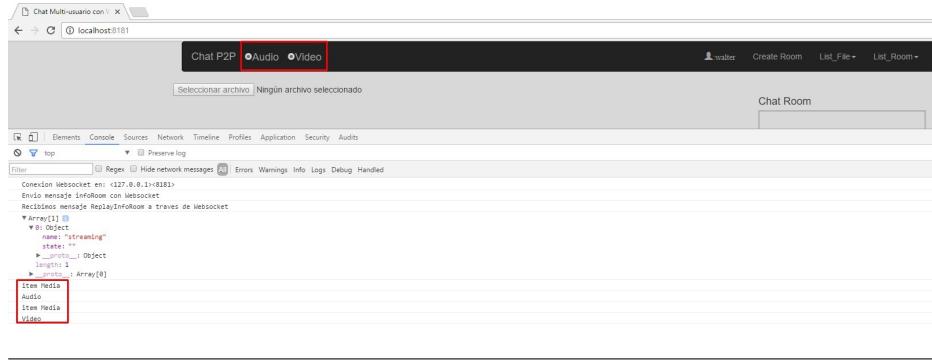


Figura 7.8: Petición/Recepción salas.

Una vez los elementos multimedia a compartir han sido seleccionados el usuario puede unirse a una de las salas existentes o crear una nueva sala.

Con cualquiera de estas opciones se envía un mensaje **stableish_connection** con el nombre del usuario y el de la sala a la que desea conectarse, figura 7.9.

```
socket.emit('stableish_connection', name, nameRoom);
```

Listing 7.8: Envío mensaje inicio conexión.

Para tratar la respuesta definimos el evento **CreateStream** con el id de la conexión, figura ??.

```
socket.on('CreateStream', function(id) {
  my_id = id;
});
```

Listing 7.9: Recepción respuesta inicio conexión.

Mientras que los demás usuarios recibirán el mensaje 'New_Joined' con el identificador asociado al usuario que ha solicitado la conexión.

```
socket.on('New_Joined', function(id) {
  id(newUser = id;
  create_connection(id(newUser));
});
```

Listing 7.10: Incluir elementos multimedia remotos I.

A partir de este mensaje empieza el proceso de señalización. Este proceso se divide en tres etapas:Offer,Answer y Icecandidate.

Offert

La función **create_connection** utiliza el identificador del nuevo cliente e inicia el proceso de oferta. Primero definimos la configuración del protocolo ICE mediante la variable

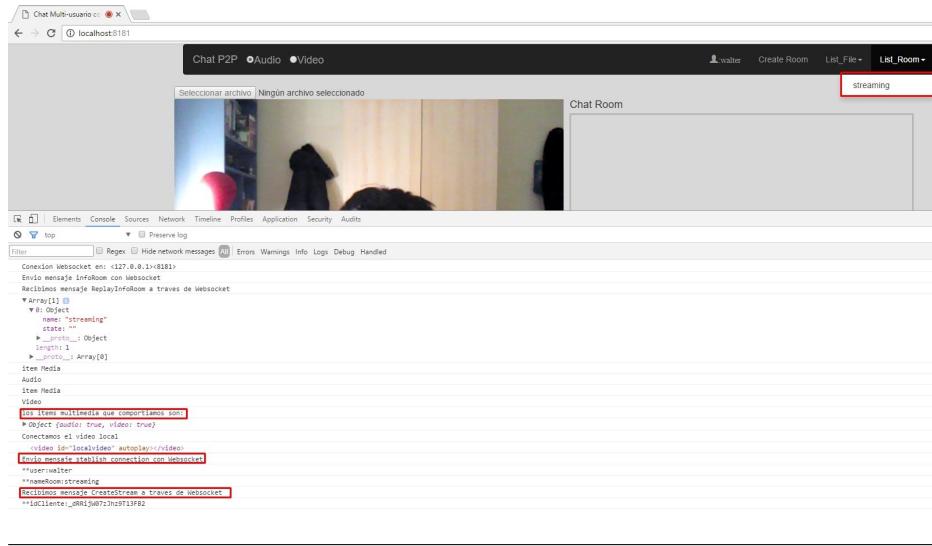


Figura 7.9: Petición/Respuesta establecimiento Conexión.

pc_config y generamos instancia del objeto **RTCPeerConnection()** que se almacena en la variable **pc**.

```
var pc_config = {'iceServers': [{}]};
var pc = new RTCPeerConnection(pc_config, {});
var num_user = 'user_' + list_user.length;
new_remote(num_user);
```

Listing 7.11: Instancia RTCPeerConnection.

Tras esto, es necesario generar una nueva etiqueta de vídeo para tratar el vídeo remoto una vez establecida la conexión.

```
function new_remote(num_user) {
  $('#list_remote').append('<video id=' + num_user + '></video>');
}
```

Listing 7.12: Creación tag vídeo remoto.

A continuación, por medio de la variable **pc** accedemos al método **addStream()** al que le pasamos el flujo de vídeo local y al evento **onaddstream** que se encargara de presentar el flujo de vídeo remoto, figura 7.11.

```
/* video local */
pc.addStream(streaming);
/* video remoto*/
pc.onaddstream = function(event) {
  var video = document.querySelector('#' + num_user);
  video.mozSrcObject = event.stream;
  video.play();
};
```

Listing 7.13: Vinculamos vídeo local/remoto a RTCPeerConnection.

El siguiente paso es definir el canal de comunicación de datos a través del método **pc.createDataChannel** al que se le pasa el nombre del canal y definimos los eventos necesarios para manejar los mensajes, figura 7.10.

```
/* canal de datos */
var sendChannel = pc.createDataChannel("sendDataChannel", {reliable: true
})
/* guardamos el canal */
list_send.push(sendChannel);
/* eventos manejo de datos */
sendChannel.onopen = ChannelOpen;
sendChannel.onclose = ChannelClose;
sendChannel.onmessage = ChannelReceive;
```

Listing 7.14: Instancia de DataChannel.

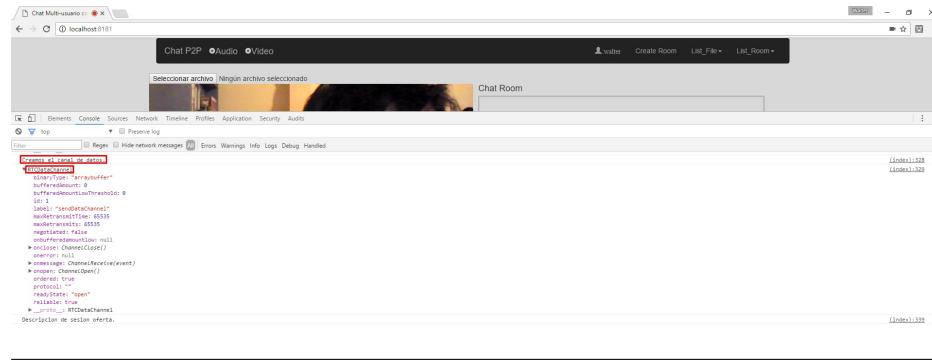


Figura 7.10: Creación canal de datos cliente.

Finalmente, definimos el método **pc.createOffer()** en el que guardamos la descripción de sesión local con el método **pc.setLocalDescription(sessionDescription)** y enviamos un mensaje **message** donde el cuerpo del mensaje es la oferta generada, figura 7.11.

```
pc.createOffer(function(sessionDescription) {
  //guardamos esto en nuestra session
  pc.setLocalDescription(sessionDescription);
  //enviamos nuestra descripcion al nuevo usuario
  var message = create_msg(my_id,id_newUser,sessionDescription);
  socket.emit('message',message);
},function(err){console.log(err);},{});
```

Listing 7.15: Creación de la oferta.

Answer

El cliente que origina la oferta recibe un mensaje de tipo **message** en el que evaluamos el subtipo de mensaje ya que a través de este tipo de mensaje recibimos los distintos tipos de mensaje de señalización.

Al principio generamos una instancia de **RTCPeerconnection()** de la misma forma que se realizó en la oferta. A continuación, con la información recibida se genera un nuevo objeto

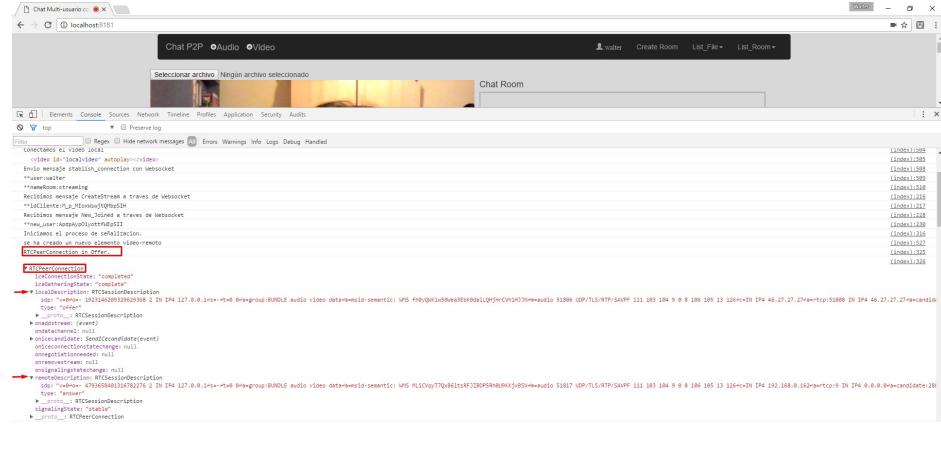


Figura 7.11: Creación oferta cliente.

RTCSessionDescription que se guarda como descripción de sesión remota a través del método **setRemoteDescription()**.

```
pc.setRemoteDescription(new RTCSessionDescription(message.message));
```

Listing 7.16: Guardar descripción de sesión remota.

El siguiente paso es establecer el canal de comunicación que en este caso tenemos que utilizar el evento **ondatachannel** ya que solo se puede crear un canal de comunicación entre dos nodos. De igual forma que en la oferta creamos las funciones para los eventos del canal, figura 7.12.

```
pc.ondatachannel = function(event) {
  list_send.push(event.channel);
  var receiveChannel = event.channel;
  /* evento de recepcion */
  receiveChannel.onmessage = ChannelReceive;
  receiveChannel.onopen = ChannelOpen;
  receiveChannel.onclose = ChannelClose;
}
```

Listing 7.17: Creación de canal de recepción de datos.

Finalmente, creamos la respuesta a la oferta a través del método **createAnswer()** dentro del cual guardaremos la información de nuestra propia sesión a través del método **setLocalDescription** y enviamos un mensaje de tipo **message** con la información de la sesión local para que el nodo remoto guarde esta información.

```
pc.createAnswer(function(sessionDescription) {
  pc.setLocalDescription(sessionDescription);
  var msg = create_msg(my_id,message.id_origen,sessionDescription);
  socket.emit('message',msg);
},function(err){console.log(err);},{});
```

Listing 7.18: Creación de la respuesta.

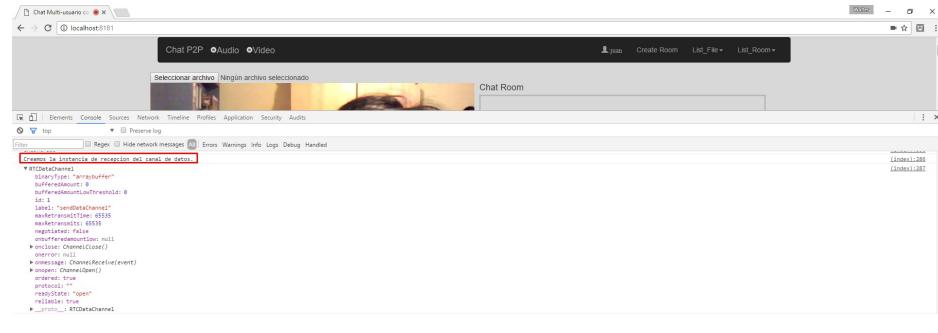


Figura 7.12: Creación canal de recepción datos.

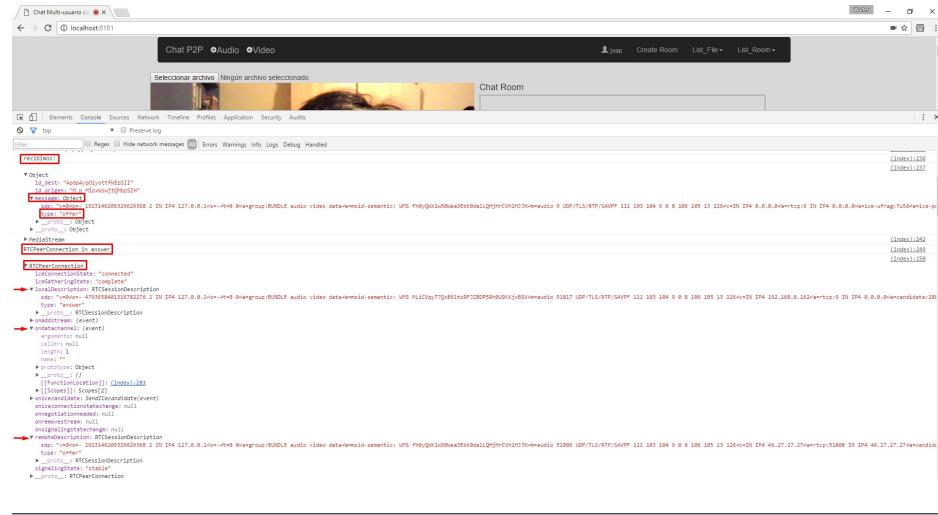


Figura 7.13: Creación de la respuesta.

Icecandidate

Tanto el usuario local y como el remoto tras la generación del objeto 'RTCPeerconnection()' necesitan obtener información de la **Ip:Puerto** disponibles para la conexión por medio del protocolo ICE.

Cuando se encuentra un candidato se ejecuta el evento **onicecandidate** que compone el mensaje con la información red y lo envía a través de un mensaje de tipo **message**.

```
pc.onicecandidate = SendICEcandidate;
.....
function SendICEcandidate(event) {
  if(event.candidate) {
    var ice = {type: 'iceCandidate',
      label: event.candidate.sdpMLineIndex,
      id: event.candidate.sdpMid,
      candidate: event.candidate.candidate
    };
    var msg ={id_origen:my_id,id_dest:id_newUser,message:ice}
    socket.emit('message',msg);
  }
}
```

```
}
```

Listing 7.19: Envío candidatos.

Los usuarios que reciben la información anterior generan un objeto **RTCIceCandidate()** y se llama a la función **addIcecandidate** que se encarga de buscar dentro de la lista de conexiones la correspondiente y así guardar el objeto creado a través del método **addIceCandidate**.

```
var candidate = new RTCIceCandidate({sdpMLineIndex:message.message.label
,
candidate:message.message.candidate
});
addIceCandid(message.id_origen,candidate);
.....
function addIceCandid(id,message) {
for(var i=0;i<list_user.length;i++) {
var user = list_user[i];
if(user.id == id) {
user.peer.addIceCandidate(message);
}
}
}
```

Listing 7.20: Recepción de candidatos.

Una vez finalizado el proceso de señalización la comunicación pasa a ser Peer-to-Peer entre los clientes de una sala, figura 7.14.

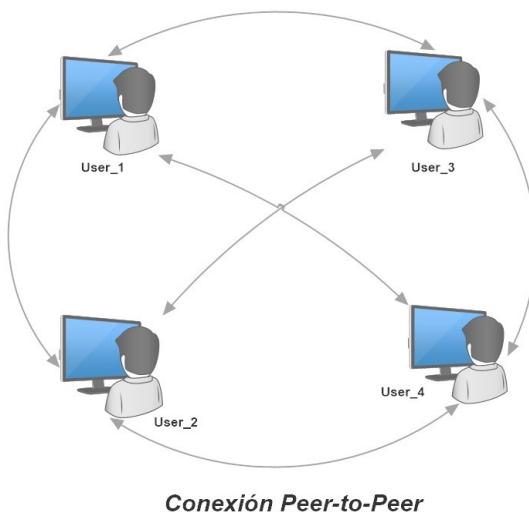


Figura 7.14: Conexión Peer-to-Peer final.

Envió de cadena de texto

A través del chat de la sala se puede enviar cadena de caracteres entre los usuarios mediante la función **send_data()**. La función se encarga de obtener los caracteres que el

usuario a tecleado y genera un mensaje que contiene el flag **chat** y el dato obtenido mediante el canal de comunicación, figura 7.15.

```
function send_data(elemento) {
    var msg = $(elemento).val();
    var data = JSON.stringify({info:'chat', data:name+':'+msg});
    for(var i=0;i<list_send.length;i++){
        var user = list_send[i];
        user.send(data);
    }
    $(elemento).val('');
}
```

Listing 7.21: Envío datos del chat.

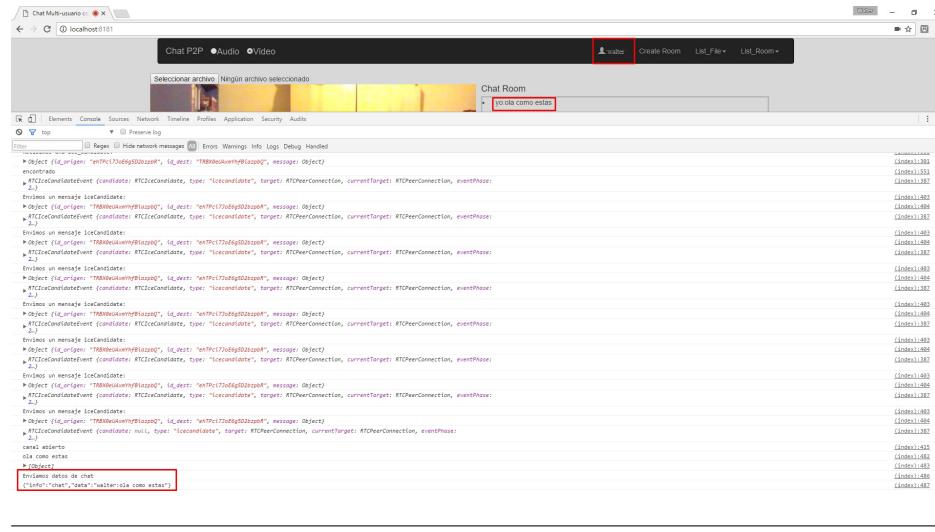


Figura 7.15: Envío de caracteres del chat por RTCDDataChannel.

La recepción por parte de los usuarios se realiza por medio de la función **WriteChat(_data)** que se encarga de presentar dentro del chat la información que se ha recibido.

```
function WriteChat (_data) {
    var line = document.createElement('li');
    var textnode = document.createTextNode(_data.data);
    line.appendChild(textnode);
    $('#texto').append(line);
}
```

Listing 7.22: Recepción datos del fichero.

Envío de ficheros

Los usuarios disponen de un **input** de tipo file con el que carga el fichero que desea compartir. Tras seleccionar el fichero se ejecuta la función **processFiles(file)** para obtener el contenido del fichero a través del objeto **FileReader()**, figura 7.16.

```
function processFiles(file) {
```

```

var files = file[0];
type = files.type;
name_fich = files.name;
var reader = new FileReader();
reader.onload = function (e) {
    var data_file = reader.result;
    data_encrypt = arrayBufferToBase64(data_file);
    send_chucky();
};
reader.readAsArrayBuffer(files);
}

```

Listing 7.23: Lectura del fichero.

El envío de los datos se realiza por medio de la función **send_chucky()** en pequeños fragmentos de longitud fija ya que no sabemos la longitud del archivo y con el fin de no saturar el canal lo enviamos de esta forma.

Cada envió por el canal esta formado por el flag **file** y el fragmento del archivo correspondiente una vez se ha enviado se programa el siguiente envío mediante el evento timer **setTimeout(sendChuncky,time)**.

Al enviar el fragmento final del fichero se añade información adicional del fichero como el nombre y tipo de fichero ya que esta información es necesario para que el usuario receptor pueda reconstruir el fichero, figura 7.16.

```

function send_chucky(){
    var last = false;
    fin = inicio + size_data;
    if(fin < data_encrypt.length){
        var data = JSON.stringify({info:'file',data:data_encrypt.slice(inicio,
            fin)});
        for(var i=0;i<list_send.length;i++){
            var user = list_send[i];
            user.send(data);
        }
        inicio = fin;
        setTimeout(send_chucky, 100);
    }else{
        last = true;
        var more_info ={type:type,name:name_fich};
        var data = JSON.stringify({info:'file',end:last,data:data_encrypt.
            slice(inicio, data_encrypt.length),more:more_info});
        for(var i=0;i<list_send.length;i++){
            var user = list_send[i];
            user.send(data);
        }
        inicio = 0;
    }
}

```

Listing 7.24: Envío de fragmentos del archivo.

El usuario receptor irá acumulando cada uno de los fragmentos que reciba hasta obtener el último fragmento mediante la función **BuildField()**. Al obtener el último fragmento pasamos a generar el documento a través una etiqueta '' donde el atributo href está formado

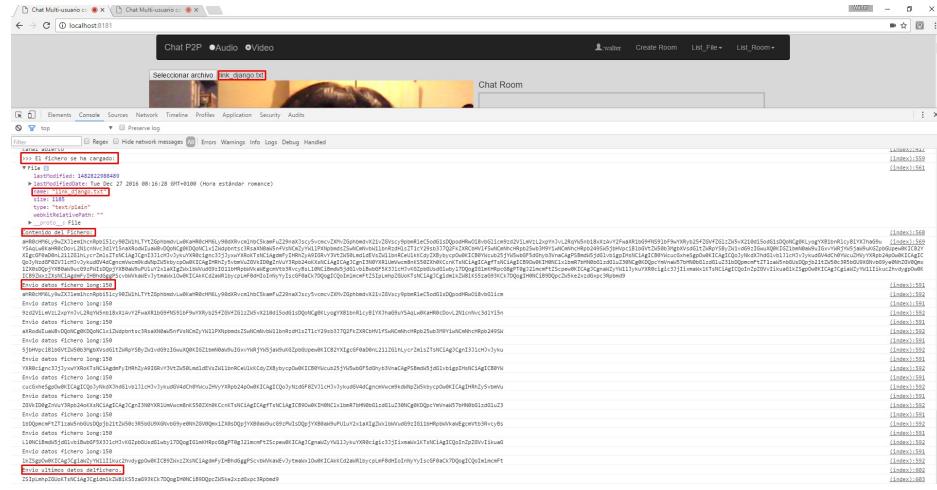


Figura 7.16: Envío información del fichero con RTCDataChannel.

por el tipo de archivo concatenado a los fragmentos del archivos, figura 7.17.

```
function BuildFile(_data) {
blob += _data.data;
if(_data.end != undefined){
if(_data.more.type == 'text/plain'){
var link = document.createElement('a');
link.href = 'data:' + _data.more.type +';base64'+blob;
link.target = '_blank';
link.download = _data.more.name;
var textnode = document.createTextNode(_data.more.name);
link.appendChild(textnode);
$("#listFile").append(link);
}
blob = ',';
}
}
```

Listing 7.25: Recepción y reconstrucción del fichero .

7.5. Pruebas

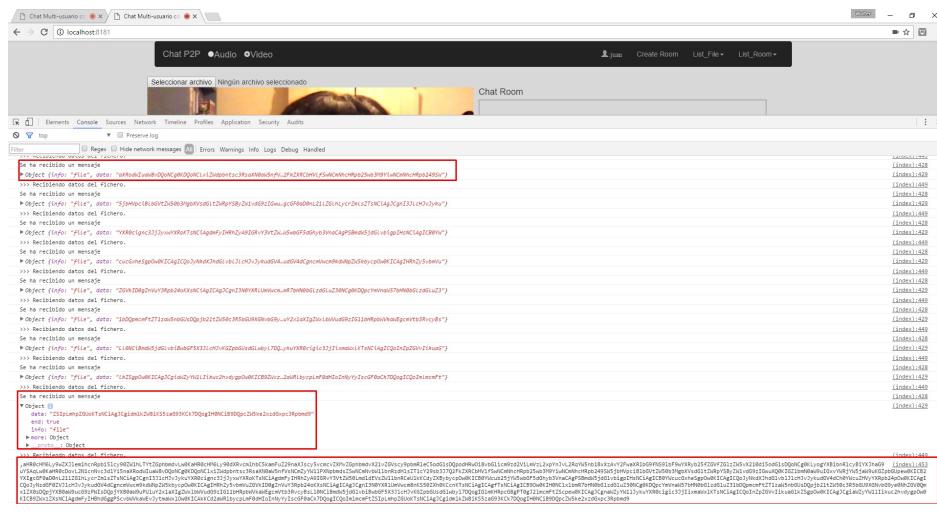


Figura 7.17: Recepción y reconstrucción del fichero.

Capítulo 8

Conclusiones

Concluyo que

Bibliografía

- [1] Curso de CSS3 de W3C http://www.w3schools.com/css/css3_intro.asp
- [2] Página oficial WebGL <http://get.webgl.org/>
- [3] Edgar Barrero. Desarrollo de una aplicación web para sistema domótico. Trabajo Fin de Grado, Grado en Ingeniería de Sistemas Audiovisuales y Multimedia, Universidad Rey Juan Carlos, 2013-2014.
- [4] Mediawiki Edgar Barrero (Surveillance 5.1) <http://jderobot.org/Aerobeat-colab>
- [5] Aitor Martínez.Tecnologías web en plataforma robotica JdeRobot. Trabajo Fin de Grado, Grado en Grado en Ingeniera Telemática, Universidad Rey Juan Carlos, 2015-2016.
- [6] MediaWiki Aitor Martínez (JdeRobotWebClients) [wikihttp://jderobot.org/Aitormf-tfg](http://jderobot.org/Aitormf-tfg)
- [7] Mediawiki Walter Cuenca (Prácticas docentes de desarrollo web.) <http://jderobot.org/Walter-tfg>
- [8] Repositorio de Walter Cuenca (Prácticas docentes de desarrollo web.) <https://github.com/RoboticsURJC-students/2015-TFG-Walter-Cuenca>
- [9] The HTML 5 JavaScript <http://html5index.org/index.html>
- [10] Tutorial Canvas https://developer.mozilla.org/es/docs/Web/Guide/HTML/Canvas_tutorial
- [11] Etiquetas de audio/vídeo HTML5 https://developer.mozilla.org/en-US/docs/Learn/HTML/Multimedia_and_embedding/Video_and_audio_content
- [12] Tutorial API File <https://www.html5rocks.com/en/tutorials/file/dndfiles/>
- [13] Introducción WebSockets <http://html5index.org/index.html>
- [14] Ejemplo WebSockets <http://blog.teamtreehouse.com/an-introduction-to-websockets>
- [15] Tutorial WebRTC https://www.tutorialspoint.com/webrtc/webrtc_tutorial.pdf

- [16] Tutorial JavaScript <http://www.vc.ehu.es/jiwotvim/ISOFT2009-2010/Teoria/BloqueIV/JavaScript.pdf>
- [17] Tutorial Jquery <http://www.cav.jovenclub.cu/comunidad/datos/descargas/jQuery.pdf>
- [18] Web oficial Jquery <http://contribute.jquery.org/documentation/>
- [19] Transparencias Bootstrap UCM <https://www.fdi.ucm.es/profesor/jpavon/web/26-Bootstrap.pdf>
- [20] Web oficial Bootstrap <http://getbootstrap.com/>
- [21] Libro NodeJS Introducción a Nodejs a través de Koans ebook
- [22] Teoría Base de Datos <https://si.ua.es/es/documentos/documentacion/office/access/teoria-de-bases-de-datos.pdf>
- [23] Teoría WebServices http://usershop.redusers.com/media/blfa_files/lpcu104/capitulogratis.pdf
- [24] Teoría tipos de WebServices <http://vis.usal.es/rodrigo/documentos/sisdis/teoria/4-serviciosWeb.pdf>
- [25] Libro Django <http://bibing.us.es/proyectos/abreproy/12051/fichero/libros%252Flibro-django.pdf>