UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Uso de Imagen y Vídeo en aplicaciones Web con HTML5

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Adrián Gimeno Balaguer
Tutor: Manuel Agustí Melchor

Curso 2017/2018

A toda mi familia,
por el apoyo emocional recibido en los momentos clave
durante el desarrollo de este trabajo final de grado.

Agradecimientos

Quisiera agradecer a mi tutor *Manuel Agustí Melchor* por toda la ayuda ofrecida y la orientación adecuada para lograr confeccionar este trabajo final de grado. Sin él no habría sido posible.

Resumen

Este trabajo constituye un enfoque práctico a HTML5 y las WebAPI modernas, que progresivamente ofrecen nuevas oportunidades de forma estandarizada. Así, se presta especial atención a sus mecanismos de entrada y salida de imagen, vídeo y animaciones. Todo ello se combina con la integración de bibliotecas externas de código abierto y el empleo de códigos QR por parte de los usuarios para construir un prototipo moderno de autenticación e interactividad en tiempo real haciendo las veces de Realidad Aumentada, y totalmente abierto a futuras mejoras con alta viabilidad en el escenario tecnológico actual. Por último, la aplicación se plantea que sea diseñada de manera compatible para ser ejecutada tanto en local como a través de un servidor web desde el primer momento, dotándola de mayor flexibilidad.

Palabras clave: HTML5, autenticación, código QR, interacción, cámara

Resum

Este treball constituïx un enfocament pràctic a HTML5 i les WebAPI modernes, que progressivament oferixen noves oportunitats de forma estandarditzada. Així, es presta especial atenció als seus mecanismes d'entrada i eixida d'imatge, vídeo i animacions. Tot això es combina amb la integració de biblioteques externes de codi obert i l'ús de codis QR per part dels usuaris per a construir un prototip modern d'autenticació i interactivitat en temps real fent les vegades de Realitat Augmentada, i totalment obert a futures millores amb alta viabilitat en l'escenari tecnològic actual. Finalment, l'aplicació es planteja que siga dissenyada de manera compatible per a ser executada tant en local com a través d'un servidor web des del primer moment, dotant-la de més flexibilitat.

Paraules clau: HTML5, autenticació, codi QR, interacció, cámara

Abstract

This work is a practical approach to HTML5 and modern WebAPI's, which progressively offer new opportunities in a standardized way. Thus, special attention is paid to its mechanisms of input and output of image, video and animations. All this is combined with the integration of external open source libraries and the use of QR codes by users to build a modern prototype of real-time authentication and interactivity acting as Augmented Reality, and fully open to future improvements with high viability in the current technological scenario. Finally, the application states to be designed in a compatible way to be executed both locally and through a web server from the first moment, giving it greater flexibility.

Key words: HTML5, authentication, QR code, interaction, camera

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

1.1 Motivación

En los últimos años se han producido notables avances en cuanto a tecnologías Web se refiere. El HyperText Markup Language (HTML) es la principal tecnología —un lenguaje de marcado— que permite definir una página Web. Esta tecnología, surgida a partir de las bases sentadas a mediados de los años 80 en el Standarized General Markup Language (SGML), hizo su primera aparición en el año 1991 [1].

HTML entonces contaba con apenas doce etiquetas disponibles para elaborar los documentos Web, y algunas partes eran privativas de empresas. En el año 1995, el Internet Engineering Task Force (IETF) adoptó la tecnología para comenzar a estandarizarla, pero no fue hasta el siguiente año cuando el World Wide Web Consortium (W3C) lanzó la versión 3.2 de HTML, la cual eliminaba los diversos elementos privativos introducidos previamente.

Posteriormente, en el año 1998, surgió la versión 4.0 de HTML. Esta versión añadía mecanismos para crear hojas de estilo, que permitían definir todo el diseño de los documentos web de forma separada a su contenido (resultando ser más acorde con la idea de separación de códigos de marcado y de estilo en un documento). También añadía cierta tecnología de *scripting*, soporte para incrustar objetos embebidos, y mejoraba elementos visuales ya existentes tales como tablas, formularios y la accesibilidad.

Sin embargo, HTML continuaba ofreciendo únicamente la posibilidad de realizar páginas Web estáticas en el sentido de proporcionar texto y poco más. Según un desarrollador del navegador Opera, Bruce Lawson, en la Web solo "se podía leer texto y hacer clic en enlaces".

Entonces, algunas compañías crearon tecnologías para añadir más funcionalidad a la Web que la estándar y permitir a los desarrolladores construir aplicaciones Web ricas. Algunos ejemplos son Adobe Flash, Apple QuickTime o Microsoft Silverlight. Con las anteriores se empezó a hablar de reproducción multimedia en la Web, sin embargo, de forma no estándar y tratándose de *software* privativo: *plugins* ejecutados como objetos embebidos en HTML.

No obstante, la actual revisión de HTML, que permanece en estado de recomendación, da paso a su versión 5 y pretende establecer nuevas funcionalidades estándar que realicen el mismo trabajo que los *plugins* privativos y otras novedades independientes. Uno de los objetivos del grupo W3C, es permitir que este lenguaje impulse el desarrollo aplicaciones Web que se ejecuten completamente en el navegador y, por tanto, en la máquina del cliente en lugar de depender de un servidor Web. En este sentido, algunas novedades principales de HTML5, en adelante HTML5, se enumeran a continuación:

Canvas. Se trata de una nueva tecnología y elemento del lenguaje HTML5 que permite renderizar gráficos y animaciones. De esta forma las aplicaciones que hagan uso de gráficos se ejecutarán más rápido, evitando el cuello de botella de los servidores Web y el ancho de banda.

Vídeo. Esta tecnología, con su correspondiente elemento del lenguaje HTML5, constituye un nuevo estándar para reproducir vídeo sirviéndose de códecs abiertos tales como Ogg Theora o H.264. Además, permite procesar la imagen obtenida de otras fuentes, por ejemplo, periféricos de entrada como una cámara.

Servicios de localización. Posibilitan la interacción con la tecnología GPS y sus datos, a través de una correspondiente Interfaz de Programación de Aplicación o API.

Almacenamiento local. HTML5 proporciona una base de datos relacional local, lo cual facilita ahora crear aplicaciones que trabajen con datos *offline*.

Web Workers. Son objetos que permiten ejecutar tareas deseadas en segundo plano sin la posibilidad de ser interrumpidas por otros *scripts* o acciones del usuario. Esto acelera dichas tareas.

Sintaxis y semántica mejorada. Consiste principalmente en la división en dos modos de redactar documentos, uno incorporado en esta versión que añade restricciones en la definición de etiquetas y acelera la interpretación del navegador (XML), y otro siendo el HTML convencional, que requiere algo menos de esfuerzo por parte del programador que la sintaxis XML.

Este proyecto está motivado principalmente por el avance de la tecnología web especialmente en los navegadores en los últimos años, con especial interés en los aspectos relacionados con las actuales posibilidades de acceso a la imagen y vídeo desde una cámara en una aplicación web.

1.2 Objetivos

Teniendo en cuenta los antecedentes que han llevado a la versión 5 de HTML y su multitud de nuevas capacidades, el proyecto actual busca mostrar dichas capacidades. En concreto, las relativas a sus funciones de vídeo y creación de gráficos y animaciones, que ya no dependen de *plugins* a diferencia de versiones previas de HTML. Más concretamente, se demuestran las capacidades y sus partes del desarrollo asociadas siguientes:

Vídeo. Se hace uso de esta tecnología de manera distinta a la reproducción multimedia. En su lugar, procesando los datos de un periférico de entrada: la cámara. Por tanto, la secuencia de imágenes capturadas en directo por la misma. La adquisición e inicialización previas de la cámara se realizan con una API específica, como se expone más adelante.

Canvas. Esta novedosa tecnología de renderización de gráficos se emplea principalmente en un minijuego de ejemplo. Además, se toman como entrada los datos capturados de la cámara (presentes en un elemento de vídeo), frente a la cual el usuario ha de interactuar de cierta manera para influir en la renderización de gráficos y animaciones desarrollados. El usuario debe mostrar a la cámara un código QR que permite una doble funcionalidad de forma flexible: autenticarse en un portal de ejemplo y controlar el avatar del minijuego mencionado, en la medida de lo posible en tiempo real. Por tanto, se puede hablar de una aplicación concreta de Realidad Aumentada.

Con lo anterior se propone un mecanismo de acceso más moderno al área identificada de las páginas webs que los tradicionales, y una experiencia de usuario a su vez más entretenida, en este caso en un minijuego, pero aplicable a otras áreas. Por otra parte, la aplicación está enfocada a ordenadores de escritorio, pero se ha probado también en dispositivos móviles. Todo esto se muestra a lo largo del proyecto y posteriormente se analizan los resultados obtenidos, para determinar si la aplicación es factible para procesar datos en tiempo real o no.

1.3 Requisitos

El desarrollo a realizar toma como partida una aplicación desarrollada previamente en una asignatura, realizada (en parte), por el autor de este trabajo. Esta se trata de una implementación del clásico juego del Pong [2], realizada sobre HTML. En el documento actual se aborda la interacción con la cámara, en parte, añadiendo funcionalidad al minijuego.

La primera versión desarrollada de este minijuego contaba con un mecanismo de seguimiento y control de la posición del avatar por detección de color a través de la cámara. La funcionalidad de la primera versión del minijuego no aparece documentada en esta memoria, puesto que quedaron debidamente expuestas en la asignatura y actualmente se desea experimentar nuevas posibilidades de interacción gráfica y con entrada del usuario en la Web. El lector puede acceder a la versión inicial de la aplicación en [3].

Teniendo en cuenta lo anterior, para el proyecto actual se espera cumplir los siguientes requisitos funcionales:

1. Proveer autenticación en la web mediante un código QR generado tras el primer registro con entrada por teclado del usuario, y la cámara. Con esto se pretende hacer un proceso de autenticación más eficiente frente al realizado con teclado, por dos motivos. Primero, el usuario puede completar el proceso de identificación antes si dispone del código de antemano. Segundo, el código ayudaría a identificarse sin necesidad de recordar contraseñas y en aquellos casos en los que el usuario la olvidase.

2. Segurizar las credenciales codificadas en el código QR anterior. El formato de los datos útiles debe ser exactamente el **nombre de usuario**, seguido de un **espacio** y la **contraseña**, este último campo de forma **transformada** para evitar ser relevada como texto plano. La contraseña transformada debe ser lo que se conoce como *hash* o *message digest* [4], el cual solo usará un conjunto reducido de caracteres en el aplicativo actual.

3. Permitir el control del avatar en una variante sencilla del clásico juego del Pong, implementado en la misma web, mediante el código QR anterior y una cámara web como interfaces.

4. Añadir muestreo en vivo desactivable -en la pantalla del inicio de sesión- del vídeo capturado por la cámara.

5. Separar la funcionalidad propia al autenticado de la propia del minijuego de ejemplo (dos capas). De esta forma, se pretende dar paso a facilitar futuras integraciones de la capa de autenticación en otras aplicaciones.

Y los siguientes requisitos no funcionales:

1. Proveer un movimiento fluido del minijuego.

2. Proveer compatibilidad de la página Web tanto en entornos locales cuando se acceda directamente desde el Explorador de Archivos, como en un servidor Web a través del protocolo HTTP, pudiendo encontrarse alojado en un servidor remoto o no.

3. Proporcionar compatibilidad en los navegadores Web principales, empleando exclusivamente API o tecnologías genéricas y estandarizadas en la medida de lo posible para solventar o reducir la problemática.

1.4 Estructura del documento

Para el desarrollo del proyecto actual se ha dividido todo el documento en diversos capítulos. La estructura seguida es la siguiente:

En el capítulo 1, se ha resumido la historia que lleva a HTML y sus aspectos que motivan el desarrollo de la aplicación a realizar. También, se han identificado en su sección 1.3 los requisitos específicos de la aplicación que se plantea en este proyecto, así como los casos de uso que constituyen las acciones disponibles por parte de los usuarios de la aplicación.

En el capítulo 2, se expone el mecanismo escogido que permite demostrar las capacidades de Realidad Aumentada, concretamente los códigos QR y, en menor medida, los códigos de barras tradicionales, sujetos a ser empleados por parte del usuario junto a la cámara como interfaces.

En el capítulo 3, se trata el diseño de la aplicación planteada. Por una parte, se muestra y comenta una serie de prototipos realizados tanto en los inicios del proyecto como de forma iterativa a lo largo del desarrollo, que reflejan con cierta fidelidad las distintas vistas que forman la aplicación planteada. Por otra parte, se presenta un diagrama de capas de la aplicación, que está separada en las mismas por motivos de diseño como se explican en el mismo capítulo.

En el capítulo 4, se aborda la implementación de la aplicación desarrollada. Atendiendo a sus distintas secciones, inicialmente se listan las distintas herramientas y tecnologías utilizadas. Posteriormente, se detalla en profundidad cada una de ellas y su relación con la aplicación. Al final, se expone un algoritmo de forma genérica que define el proceso gráfico de animaciones de mayor peso en la aplicación.

En el capítulo 5, se dan a conocer una serie de pruebas realizadas de especial importancia, algunas centradas en los tiempos de respuesta dado el requisito implícito de la ejecución en tiempo real inherente al tipo de aplicación desarrollada y su nivel de interactividad requerido.

En el capítulo 6, se indaga en aspectos considerados durante y tras el desarrollo para formalizar y mejorar en el futuro la aplicación construida a partir de ideas más allá del alcance contemplado en este proyecto.

CAPÍTULO 2
Estado del arte

En este capítulo se presentan, por un lado, una serie de aplicaciones externas y bibliotecas de soporte existentes que giran en torno al ámbito de este trabajo y, de forma más general, a la Realidad Aumentada. Por otro lado, se introducen los códigos QR y de barras tradicionales haciendo énfasis en sus características principales que muestran cómo han servido de utilidad para integrarse con el trabajo actual.

2.1 Realidad Aumentada en la Web

Cada vez más son las aplicaciones que se están desarrollando y exploran la interactividad con el usuario a través de periféricos, tales como la cámara. Hacen uso de la detección de objetos para mostrarlos de diversas maneras sobre la pantalla, resultando en animaciones que posibilitan directamente crear aplicaciones complejas como videojuegos. Esto constituye un novedoso campo de investigación conocido como Realidad Aumentada [5]. Así, en esta sección se exponen casos de uso de la Realidad Aumentada directamente en la web, puesto que es el entorno objeto de este trabajo.

Normalmente, el proceso de Realidad Aumentada, en adelante R.A., consta de dos etapas [5]. En la primera etapa, se ha de obtener la imagen de la realidad a través de un periférico de entrada (cámara), y detectar y procesar marcas u objetos en la misma que sirven de referencia a la aplicación. En la segunda etapa, se realizan transformaciones gráficas a partir de los datos de la primera etapa que permiten recrear escenas tridimensionales o 3D.

Para continuar, existe la posibilidad de usar la captura de imagen y vídeo como fuente de entrada en aplicaciones web. Para acceder a las cámaras existe un gran grupo de API pertenecientes al WebRTC [6] (Web Real Time Communications), siendo la más importante `getUserMedia` [7] para solicitar el acceso a la cámara y otros tipos de periféricos. Esta API distingue perfectamente entre posibles fuentes de información distintas en el mismo dispositivo, por ejemplo, el micrófono y el propio sensor óptico en una cámara. Ahora bien, también se acercan nuevas propuestas para manejar el vídeo de la cámara de forma complementaria a `getUserMedia` [7], como Media Capture and Streams [8].

Por otro lado, el elemento Canvas de HTML5 permite crear animaciones y realizar un gran número de transformaciones gráficas tradicionales (translación, rotación y escala), así como renderizar desde formas básicas hasta gráficos matemáticos avanzados, pasando por tipografías.

Combinando las API de acceso a la cámara y Canvas, se dispone de lo suficiente para construir aplicaciones de Realidad Aumentad en la Web. Así, existen ejemplos de uso que permiten tomar una foto a la vez que se muestra la captura de la cámara en vivo [9],

tutoriales para obtener acceso a la cámara [10] y manipular su imagen a través de Canvas [11] [12], etc.

La R.A. puede hacerse con y sin marcas [13] [14]. Las marcas son objetos físicos específicos que la aplicación trata de detectar en la imagen, normalmente impresiones en blanco y negro. Las aplicaciones sin uso de marcas aplican algoritmos de detección de características naturales para procesar el entorno, siendo más flexibles que las anteriores; no obstante, también pueden caracterizar el entorno por ejemplo a través de datos obtenidos mediante GPS.

Acorde con la línea de la aplicación web a desarrollar, existe una gran variedad de aplicaciones fáciles de usar [15] relacionadas con la detección y procesado de marcas en la R.A., siempre que se disponga de una cámara lista. Aplicaciones interesantes como [15] no se han considerado para el uso en el trabajo actual debido a la limitación de las marcas a detectar, marcas de información reducida para renderizar modelos 3D.

Finalmente, la R.A. también puede darse en el escritorio o en el móvil. En este trabajo se decanta por realizar una aplicación web para escritorio debido al espacio necesario de libertad de movimiento por parte de los usuarios, y estimaciones en cuanto a las posibles diferencias de rendimiento y el espacio necesario en la pantalla, no obstante, se ha probado en ambos dispositivos. Además, hace uso de los códigos QR y códigos de barras tradicionales como marcas de R.A.

2.2 Introducción a los códigos QR

Un código QR es un código de barras de dos dimensiones que puede ser leído por dispositivos tales como móviles u ordenadores a través de una cámara. Constituyen un mecanismo útil para vincular el mundo físico con el electrónico, otorgando el potencial de intercambiar información de forma eficiente. Existen cuatro casos de uso básicos aceptados comúnmente en cuanto a los códigos QR [16], como muestra la tabla 2.1.

Caso de uso	Descripción
1. Enlace con una URL	Permite redirigir al usuario a una página web cuya URL se halla contenida en el código.
2. Texto plano	Almacena información textual, normalmente breve.
3. Número de teléfono	Contiene un número telefónico de interés. El dispositivo, una vez reconocido el número, puede facilitar al usuario la posibilidad de realizar una llamada al mismo.
4. Mensaje de texto preestablecido	De características similares a las del caso 3, se trata del contenido de un SMS (y opcionalmente un número de teléfono de destino). Útil, por ejemplo, para facilitar la subscripción a servicios digitales.

Tabla 2.1: Tabla informativa acerca de los casos de uso típicos relativos a los códigos QR.

2.2.1. Características relevantes

1. Capacidad de codificación de los datos [17]. Mientras que los códigos de barras tradicionales son capaces de almacenar aproximadamente 20 dígitos, los códigos QR son

Figura 2.1: Ejemplo de código QR. Se destacan los distintos *patrones funcionales* que emplea. Fuente: `http://qrcode.com`.

capaces de manejar un número de caracteres más elevado. Así, pueden contener hasta 7.089 dígitos.

2. Tamaño de impresión [17]. Puesto que QR transporta información en sentido horizontal y vertical, es capaz de codificar la misma información contenida en un código de barras tradicional en aproximadamente la décima parte del espacio total de éste.

3. Ángulos de lectura [17]. Los códigos QR son capaces de ser leídos estando posicionados en cualquier ángulo (desde 0º hasta 360º) por cualquier software que respete el estándar. Esto es posible gracias a los *patrones funcionales* de posicionamiento impresos en las esquinas del símbolo (normalmente tres, en la primera versión de la patente). Los patrones anteriores permiten una lectura del código QR precisa y con tiempos de respuesta breves, incluso en presencia de interferencias.

4. Resistencia a suciedad y daños [17]. QR tiene la capacidad de corrección de errores. Esto es, la información contenida se puede restaurar incluso si el símbolo (la impresión) está parcialmente sucio o dañado. Así, se pueden recuperar hasta un máximo de 30 % de los *codewords* totales en el símbolo. Un *codeword* es la palabra o unidad básica de 8 bits de datos en el símbolo.

2.2.2. Tipos de códigos QR

QR alberga un conjunto de tipos con características únicas cada uno. El factor distintivo principal entre éstos es el número de módulos. Un módulo en este contexto es la unidad mínima de información (un bit). Representa un 1 binario solo si se muestra como una casilla negra, y un 0 binario solo si lo hace en forma de casilla blanca. En esta memoria se establece a voluntad del autor el término *patrón funcional*. Un *patrón funcional* consiste en un conjunto de módulos que desempeña una determinada función en el código QR y, por tanto, en la lectura de este. En este sentido, el *codeword* es el *patrón funcional* principal, puesto que constituye un carácter de la información contenida en el código QR. Los *patrones funcionales* existentes se pueden observar en la figura 2.1. Los códigos QR se pueden clasificar según los distintos modelos que se han ido lanzando en los últimos años, a su vez con múltiples versiones cada uno:

1. Modelos 1 y 2 de los códigos QR [17]. El primer modelo que se diseñó engloba distintas versiones con un número de módulos cada, constando la de mayor tamaño de 73x73 módulos, capaz de almacenar así hasta 1.167 números. Tras este modelo, se lanzó el segundo modelo como mejora del primero, aumentando su capacidad máxima a 177x177 módulos. Este modelo es el más empleado actualmente.

2. Código Micro QR [17]. Este modelo soporta un número reducido de módulos (17x17), es capaz de almacenar hasta 35 números y mide el menor tamaño de todas las variantes. Requiere de un solo patrón de orientación, por lo que ahorra espacio.

3. Código iQR [17]. Este código tiene la particularidad de poder emplear bien casillas cuadradas o rectangulares como impresión de los módulos. Soporta hasta 422x422 módulos en su versión más grande, pudiendo codificar así hasta 40.000 números.

4. Código SQRC [17]. Es una modalidad del código QR igual en estructura a los modelos 1 y 2, pero añade una restricción de lectura mediante encriptación de los datos para ser leído por escáneres específicos a los que se haya inyectado la clave apropiada. Por tanto, es adecuada para manejar información interna en empresas.

5. Código Frame QR [17]. Se trata de una modalidad del código QR que permite imprimir imágenes en el mismo, para dotarle de apariencia publicitaria, corporativa o similar.

2.3 Códigos de barras tradicionales

En esta sección se propone dar una breve introducción a los códigos de barras empleados tradicionalmente y establecer una breve comparativa entre los principales tipos existentes, puesto que la aplicación desarrollada hace uso de estos códigos, aunque en menor medida que los códigos QR, con énfasis en la comparación de resultados de tiempos de respuesta.

Los códigos de barras tradicionales son ampliamente utilizados en logística, comercios, etc. Se caracterizan por estar formados por dos tipos de barras (blancas y negras) dispuestas unidimensionalmente. Concretamente, un código se puede clasificar según las siguientes características: conjunto de caracteres que codifica, densidad de los datos, uso de un *checksum* y longitud máxima.

En la figura 2.2, se puede apreciar una serie de códigos de barras tradicionales y sus características distintivas. De entre los tipos mostrados, se descartan algunos para el uso en la aplicación. Dado que se desea codificar en los códigos las credenciales segurizadas de los usuarios (nombre y contraseña transformada), es necesario considerar solo aquellos que ofrezcan principalmente un conjunto de caracteres suficiente. Esto es, letras y números. Concretamente y, como se expone en el capítulo 4, solo se necesita disponer del conjunto $0 - 9$ y $a - f$. Esto conduce a preferir tipos de códigos de barras como el Code 128 o el Code 39. La figura 2.3 muestra una impresión de un Code 128.

Código	Tipo de datos que codifica	Densidad	Checksum	Longitud
Code 128 A	A-Z, 0-9, caracteres especiales, códigos de control ASCII	Alta	Si	Variable
Code 128 B	A-Z, a-z, 0-9, caracteres especiales	Alta	Si	Variable
Code 128 C	0-9	Alta	Si	Variable, numero par de caracteres
Code 39	A-Z, 0-9, -.$/+%	Media	Opcional	Variable
Code 93	ASCII Completo	Alta	Doble	Variable
EAN 13	0-9	Media	Si	Fija, 13 dígitos
EAN 8	0-9	Media	Si	Fija, 8 dígitos
UPC-A	0-9	Media	Si	Fija, 12 dígitos
UPC-E	0-9	Media	Si	Fija, 8 dígitos
Codabar	0-9 -.:$/+	Alta	No	Variable
Bookland	0-9	Media	Si	Fija, 13 + 5

Figura 2.2: Tabla que compara las características de los principales códigos de barras tradicionales. Fuente: `http://http://el-codigo-de-barras.awardspace.com/resumen/`.

Figura 2.3: Ejemplo de código de barras tipo Code 128.

CAPÍTULO 3
Diseño

La aplicación desarrollada constituye una página web estática sin servidor. En este sentido, el sistema almacena a los usuarios registrados de forma temporal en los datos de sesión del navegador, por tanto, situándose en el *frontend* o lado del cliente. Dado que el objeto principal de la aplicación consiste en experimentar las posibilidades de interactividad gráficas en la Web, el nivel de persistencia escogido se considera suficiente.

Los casos de uso planificados realizables por el usuario se enumeran a continuación:

1. Registrarse empleando la interfaz de entrada tradicional (teclado).
2. Autenticarse si ya se encontraba registrado, empleando también el teclado.
3. Autenticarse mediante un código QR cuya información son sus credenciales seguri-zadas, mostrándolo a la cámara.
4. Controlar la posición del avatar del minijuego del Pong mediante el mismo código QR anterior y la cámara como interfaz.

3.1 Prototipado

A continuación, se exponen una serie de prototipos o maquetas de baja fidelidad relativos a todas las vistas que forman la aplicación, que muestran con claridad su información contenida.

En primer lugar, el usuario se encuentra nada más al acceder a la aplicación con una ventana tipo *modal* o superpuesta en la cual puede introducir con teclado sus datos para darse de alta o identificarse (autenticarse) de nuevo en el sistema. Además, un texto le informa de que es posible identificarse usando un código de autenticación a mostrar a la cámara, en caso de que dispusiera del mismo. A su vez, aparece un mensaje que le informa de la posibilidad de subir una fotografía para validarla como código de autenticación, y un botón de subida correspondiente. Esto es útil cuando el usuario se encuentra en un dispositivo móvil, puesto que con toda seguridad le resulte más cómodo autenticarse mediante imagen almacenada en el propio dispositivo que mostrando a la cámara la imagen desde otro medio.

En la misma vista, se presenta al usuario un contenedor desplegable mediante botón para mostrar en el mismo una vista previa de la captura de la cámara. El usuario puede ocultar el contenedor, actualizándose automáticamente el texto “Ver captura en vivo” a “Ocultar captura en vivo”, cuando el contenedor está desplegado. Ver figuras 3.1 y 3.2.

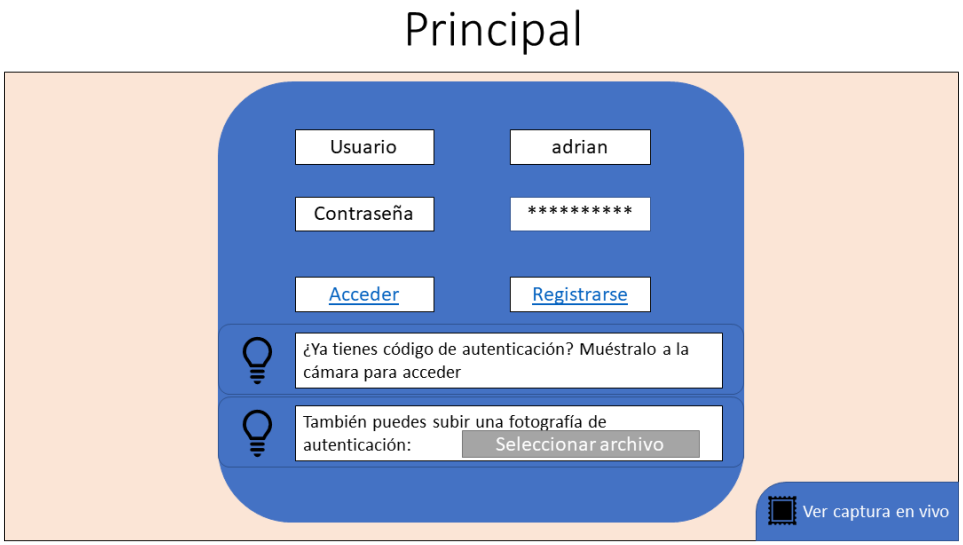

Figura 3.1: Diseño de la vista de autenticación, la cual es la primera con la que se encuentra el usuario al acceder a la aplicación.

En la misma ventana de autenticación, se contempla que aparezcan varios mensajes de error cuando el usuario introduce credenciales inválidas, habiendo tres casos. El primero ocurre cuando el usuario pulsa el botón "Acceder" habiendo introducido un nombre de usuario no existente en el sistema. El segundo ocurre cuando el usuario pulsa el mismo botón habiendo introducido un nombre de usuario existente en el sistema, pero una contraseña incorrecta asociada al mismo. El tercero ocurre cuando el usuario pulsa el botón "Registrarse" con un nombre de usuario ya registrado, independientemente del valor del campo "Contraseña" en ese momento. En cualquier caso, si hay algún campo vacío al pulsar alguno de los botones de acceso, el propio diseño de la validación no pretende mostrar ningún error y en su lugar se delega la retroalimentación en el comportamiento por defecto del navegador. Referirse a la figura 3.3 para visualizar uno de los mensajes de error.

Una vez el usuario se identifica correctamente en el sistema, la ventana de autenticación desaparece y automáticamente se muestra una nueva ventana y un contenedor superpuesto el cual contiene una imagen de un código QR con el cual podrá autenticarse alternativamente, como se ha descrito en los casos de uso al principio de este capítulo. También se muestra un mensaje informativo sobre el propósito de la imagen anterior, así como una recomendación de fotografiarla o imprimirla. Por último, este contenedor contiene un botón "Entendido. Ir a la Web." para así permitir al usuario avanzar al contenido principal de la página cuando esté listo. Ver figura 3.4.

Como última vista, se encuentra la del minijuego del Pong, que se puede dividir en dos sub-vistas. La primera es el menú principal, un *modal* que se activa una vez el usuario completa la ventana de generación de su código QR. Este *modal* contiene dos listas desplegables. La primera sirve para elegir el modo de detección con el cual controlar a los jugadores. Por el momento solo se presenta la opción asociada al mecanismo de interacción mediante el código QR de autenticación de los usuarios. La segunda es para tipos específicos al modo de detección; cuando el usuario ha elegido en la primera lista la opción "Detección por código QR autenticado", entonces la segunda lista permite al usuario escoger el número de jugadores: "Un jugador", "Dos jugadores (local)" o "Cuatro jugadores (local)".

Figura 3.2: Diseño de la vista de autenticación tras expandir el panel de captura en vivo.

Una vez el usuario configura las opciones requeridas del menú principal, puede pulsar el botón "Empezar" para comenzar la partida del minijuego (ver figura 3.5). Así, se le presenta la segunda sub-vista que consiste en el propio escenario del minijuego del Pong, cuyos componentes se describen mejor a través de la maqueta de la figura 3.6. En esta maqueta se asume que el usuario ha elegido la modalidad específica "Un jugador".

Es importante destacar que, pese a no quedar reflejado en la misma maqueta, los modos "Dos jugadores (local)" y "Cuatro jugadores (local)" ofrecen una versión multijugador en la misma máquina del usuario, donde el bloque del oponente controlado por el sistema se sustituye por otro con las mismas características que el del jugador primario (avatar), y se añaden dos nuevos avatares en la versión de cuatro jugadores.

Estos avatares son controlables mediante la cámara sobre sus correspondientes áreas (mitad izquierda o derecha del campo de juego, según proceda), a partir de cálculos de escalado tradicionales en función del área de captura de la cámara. Por último, la implementación de la versión multijugador ha planteado algunos retos destacables que se presentan en la sección 4.9.

3.2 Diagrama de capas

En esta sección, se procede a explicar cómo se ha organizado la aplicación de forma general. La aplicación se ha dividido en varias capas. Es importante conocer al menos cómo están estructuradas las capas en esta aplicación y el orden de sus activaciones; posteriormente, en el capítulo 4 se habla sobre cómo se comunican dichas capas, lo cúal es necesario para que funcione la aplicación en su conjunto.

Las capas empleadas contienen, al menos, código del contenido más el diseño de la interfaz gráfica (mediante HTML y CSS3, respectivamente), o código asociado a la lógica de la aplicación (a través de *scripts* en JavaScript). Las diversas capas de la aplicación se ilustran en la figura 3.7. A continuación, se procede a describirlas:

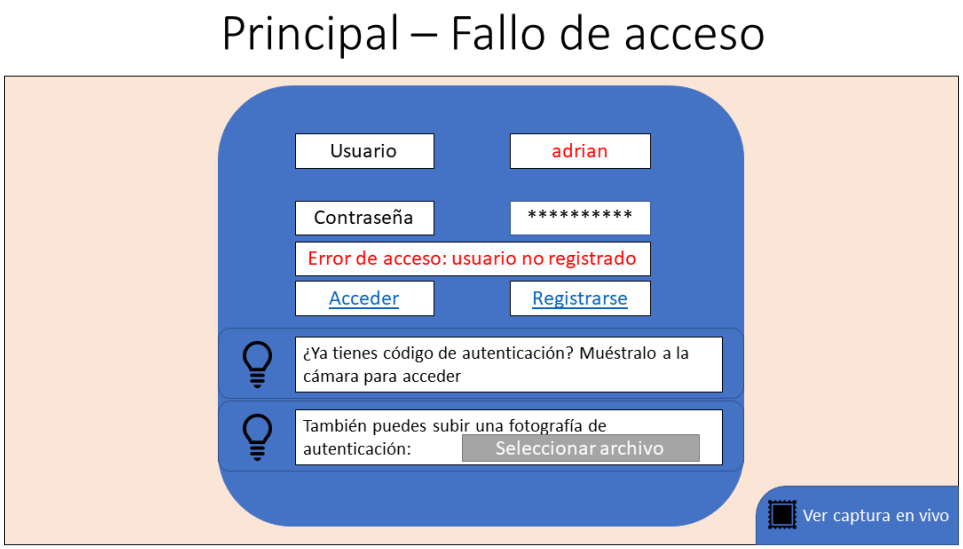

Figura 3.3: Diseño de la ventana de autenticación tras introducir credenciales inválidas. En este caso, se muestra un error debido a que el usuario no está registrado.

Capa general. La capa general es la primera capa que se activa al acceder de manera natural a la aplicación, en concreto, al archivo *index.html*. Esta capa es la única que controla y se comunica con todas las demás. No contiene interfaz gráfica, puesto que su objetivo es ser capaz de activar las demás capas cuando se requiera, que sí contienen vistas y pueden ser independientes entre sí.

Capa de autenticación. A diferencia de la anterior, la capa de autenticación es la primera capa invocada automáticamente por el controlador general. Proporciona una vista que consta del formulario de registro e inicio de sesión junto a un pequeño contenedor desplegable que muestra la captura de la cámara en vivo. Es la primera vista que observa el usuario al acceder a la aplicación.

Capa de generación del código QR del usuario. Esta capa se encarga de mostrar una ventana superpuesta activada por el controlador de autenticación inmediatamente después de que el usuario inicie sesión. En esta se renderiza una imagen del código QR asociado a la cuenta del usuario, para poder ser fotografiada y empleada alternativamente en posteriores inicios de sesión. Pese a ofrecer una interfaz y funcionalidad sencillas que guardan relación con la capa de autenticación, se ha decidido separarla de la misma puesto que no siempre podría ser deseable mostrar su vista en cada inicio de sesión, y se podría mover en un futuro a alguna sección más apropiada que sea accesible a petición del usuario en cualquier momento.

Minijuego del Pong. En último lugar, la capa del minijuego del Pong, parte de unas bases establecidas en una primera versión realizada previamente al proyecto actual. En dicha versión se implementó un sistema de simulación de los objetos con una detección de colisiones y movimientos básicos. El bloque del jugador se controlaba mediante la cámara y realizando un seguimiento por color con uso de cierta biblioteca. El proyecto actual añade funcionalidad al minijuego, de forma que el avatar del jugador ahora se puede controlar interactivamente a través de la cámara mostrando la impresión o fotografía digital del código QR asociado a la cuenta del usuario, realizando parte del trabajo gráfico —objeto principal de este proyecto— mediante la tecnología Canvas.

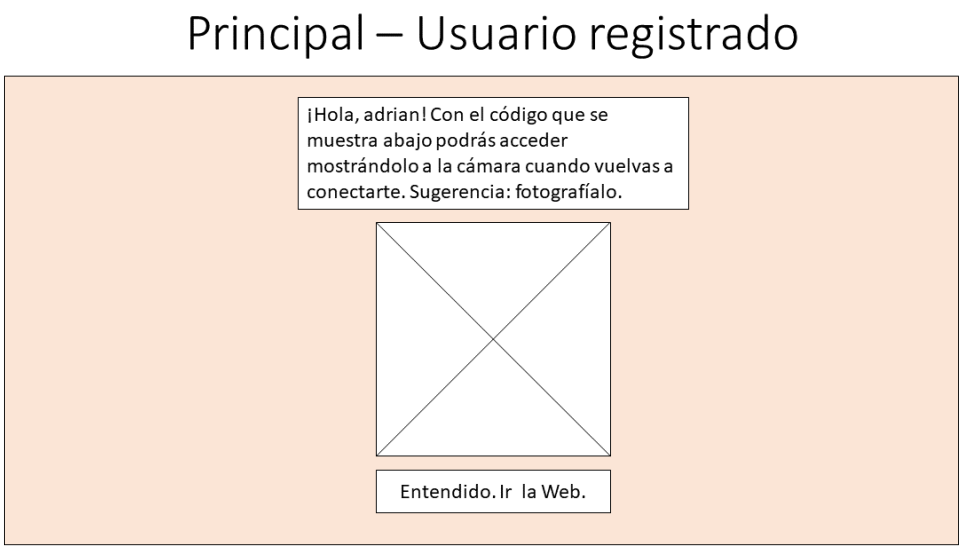

Figura 3.4: Diseño de la ventana de bienvenida, a mostrar inmediatamente después de que un usuario se identifique. Contiene la imagen del código QR generado asociado a su cuenta.

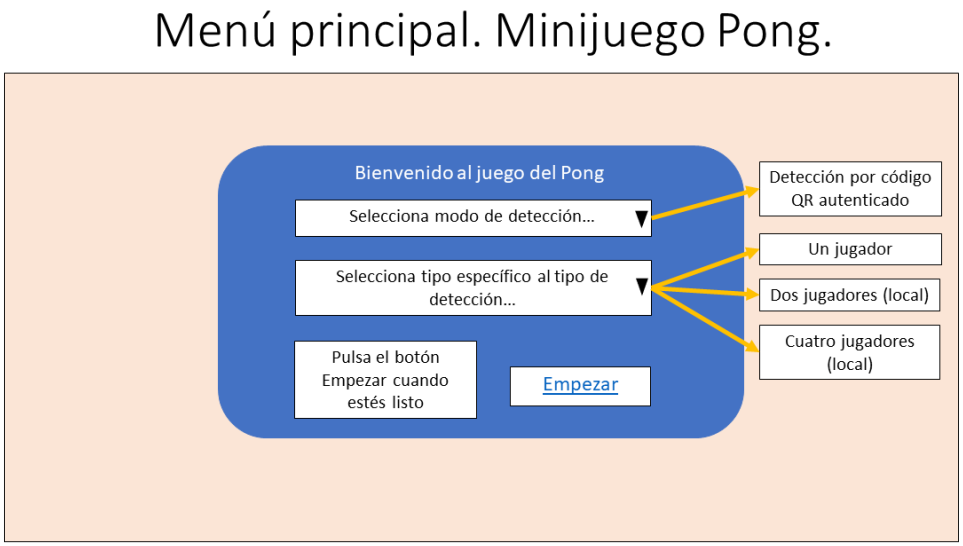

Figura 3.5: Diseño del menú principal del minijuego del Pong. Se muestra la opción contenida en el primer desplegable.

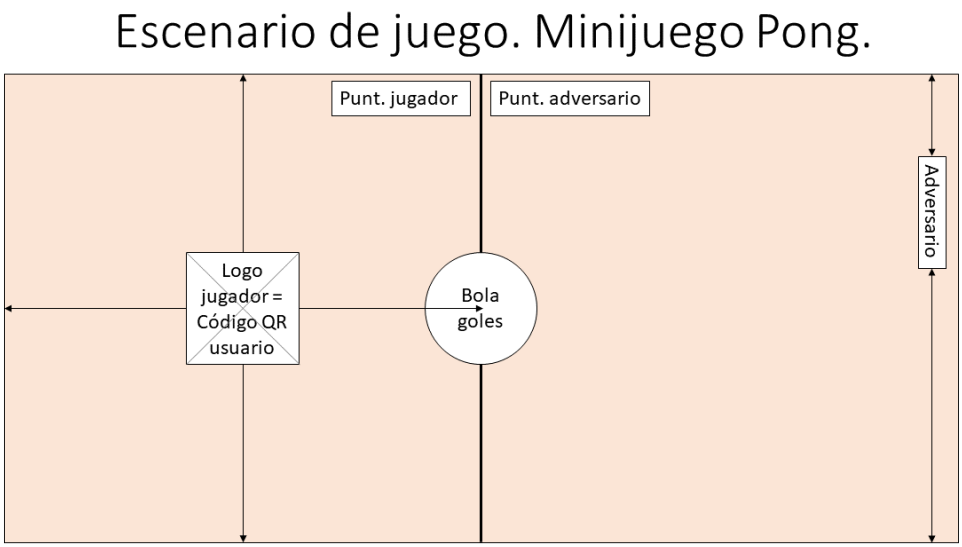

Figura 3.6: Diseño del escenario del minijuego del Pong, con algunas anotaciones a modo de esquema.

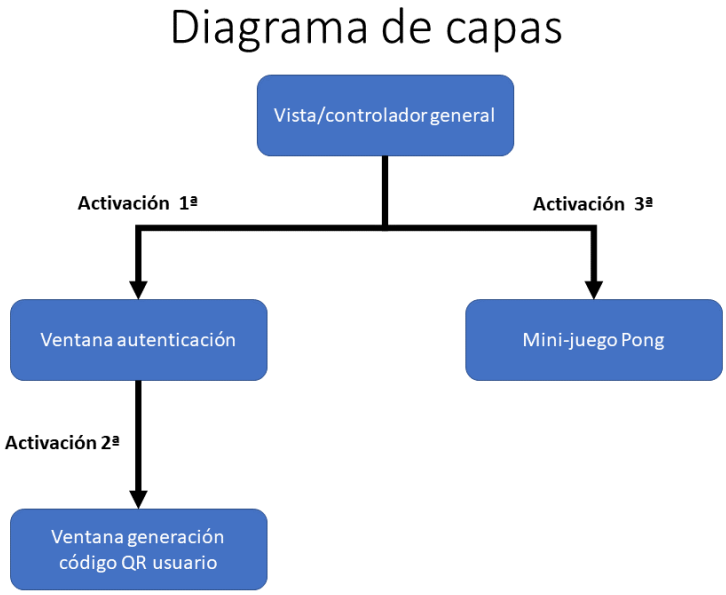

Figura 3.7: Diagrama que muestra las diversas capas que forman la aplicación Web, y su estructura.

CAPÍTULO 4
Implementación

Para el desarrollo de la aplicación Web se ha hecho uso de las siguientes tecnologías principales:

1. Lenguaje Web basado en marcas HTML5.
2. Lenguaje Web de hojas de estilo en cascada, CSS3.
3. Lenguaje Web de *scripting* JavaScript.
4. biblioteca *jQuery* de facilidades para el desarrollo Web (API en JavaScript).
5. biblioteca Bootstrap para el prototipado gráfico en la Web (API en JavaScript).
6. biblioteca JOB (Javascript Only Barcode Reader) para el escaneo de códigos de barras tradicionales mediante la cámara.
7. biblioteca *jsqrcode* para el escaneo de códigos QR mediante la cámara.
8. API de Google Charts para la generación de códigos QR.
9. Biblioteca *jshashes* empleada para la seguridad de las contraseñas (API en JavaScript).

A lo largo de este capítulo se describen estas tecnologías, no sin antes exponer primero el flujo de la aplicación.

4.1 Flujo de aplicación

En este sección, se expone cómo se ha implementado el flujo de la aplicación. Se recomienda haber comprendido el diagrama de capas de la aplicación, explicado en el capítulo 3, puesto que se menciona repetidamente en esta sección.

Como se ha explicado en el capítulo 3, la aplicación está dividida en cuatro capas. Esto ha propiciado el uso de un mecanismo para comunicar dichas capas entre sí. Así, la aplicación utiliza un mecanismo de paso de mensajes o eventos entre las diversas capas, por lo cual es importante conocer los eventos existentes y sus características relevantes: remitente, destinatario y la información que transportan, así como el propósito de la misma.

Los eventos de la aplicación se ilustran en la figura 4.1, y se describen a continuación de forma enumerada:

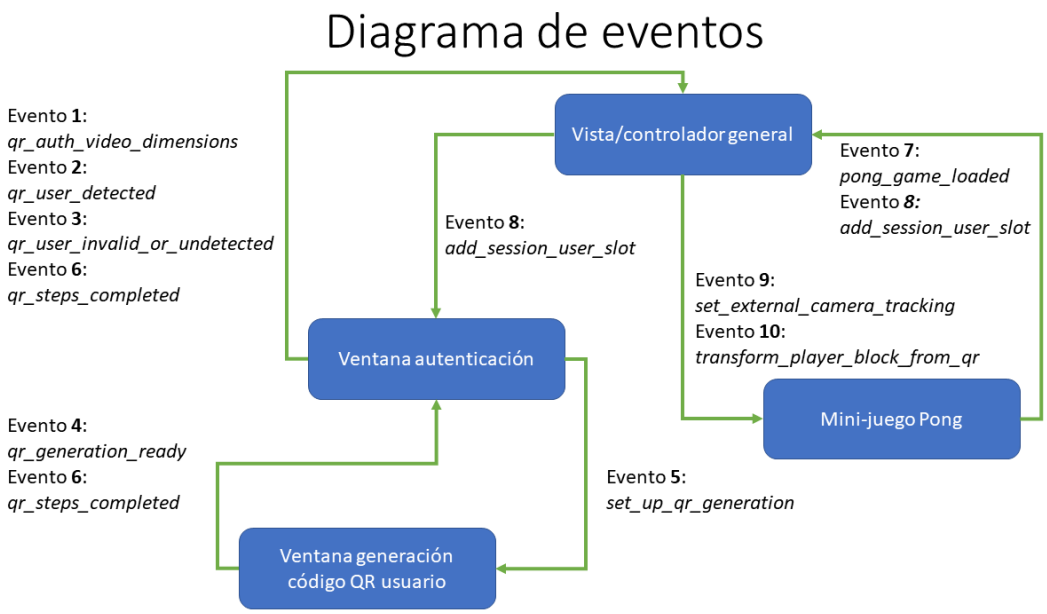

Figura 4.1: Diagrama que ilustra los distintos eventos enviados entre las distintas capas durante el uso completo de la aplicación Web.

Evento 1: **qr_auth_video_dimensions**. Este evento se genera en dos ocasiones. La primera sucede cuando, una vez cargado el árbol DOM (Document Object Model) de la capa de autenticación por completo, el usuario concede el permiso solicitado de la cámara por esta misma capa. La segunda sucede de forma desencadenada al evento `add_session_user_slot` (detallado más abajo).

Por otra parte, este evento transporta el **ancho**, el **alto** y la **relación de aspecto** del elemento de vídeo que reproduce la captura de alguna cámara accedida. Estos datos sirven al controlador principal para retransmitirlos más tarde en el evento `transform_player_block_from_qr`. Los permisos de la cámara a utilizar -tanto de concesión como de denegación-, se pueden tomar de la configuración guardada previamente en el navegador para esta página. La activación de este evento hace uso concretamente de la API `getUserMedia` [7] para acceder a la cámara; esta API se expone en la sección 4.4.

Evento 2: **qr_user_detected**. Este evento lo envía el controlador de autenticación al controlador general cada instante de tiempo que el de autenticación detecta el código QR asociado a la cuenta de un usuario válido a través de la cámara. Antes de continuar describiéndolo, se ha de realizar un inciso: el controlador de autenticación es capaz de permitir y manejar varios usuarios en la misma sesión.

Dicho esto, para que valide a un usuario escaneado y dispare este evento, se deben cumplir los requisitos a continuación. Primero, el código debe contener las credenciales correctas de un usuario registrado. Segundo, el usuario escaneado bien ha de estar ya presente en la sesión activa o bien existir una ranura libre dentro de un número máximo de usuarios permitidos en la sesión. Este número máximo es incrementable tras la activación del evento `add_session_user_slot`, como se detalla más abajo.

Así, este evento transporta el **nombre del usuario** y las coordenadas de los **tres patrones de posicionamiento** del código QR detectado. El controlador general emplea a posteriori las coordenadas anteriores para reenviarlas al minijuego del Pong (evento `transform_player_block_from_qr`), aplicándoles justo antes una breve transformación a modo de efecto espejo. Este evento se lanza desde de una función llamada en

cada *frame* o cuadro de renderizado de forma optimizada por el navegador, gracias al moderno API `requestAnimationFrame` [18]. Esta API se detalla en la sección 4.7.

Evento 3: **qr_user_invalid_or_undetected**. Este evento se genera cada instante de tiempo que el controlador de autenticación no detecta ningún código QR asociado a un usuario válido a través de la cámara. Es decir, cumple las condiciones opuestas al evento anterior (`qr_user_detected`) para ser disparado. No transporta dato alguno, y actualmente no provoca ninguna acción. Sin embargo, existe al resultar útil para incorporar tareas especialmente gráficas en el futuro.

Evento 4: **qr_generation_ready**. Este evento lo envía el controlador de generación del código QR del usuario cuando se carga el árbol DOM de su vista asociada al controlador de autenticación, lo cual sucede cuando un usuario inicia sesión, ocultándose también la vista de autenticación. No contiene ningún dato. Sirve para garantizar que dicho árbol DOM está cargado y permitir al controlador de autenticación "responderle" con el siguiente evento, `set_up_qr_generation`, con la seguridad de que lo recibirá.

Evento 5: **set_up_qr_generation**. Este evento lo envía el controlador de la capa de autenticación al de generación del código QR del usuario, y transporta las credenciales serializadas de la cuenta del usuario para que la vista de la segunda capa genere la imagen del código QR del usuario, haciendo uso de la API de Google Charts.

Evento 6: **qr_steps_completed**. Este evento es enviado por el controlador de generación del código QR del usuario cuando este hace clic en el botón "Entendido. Ir a la Web." perteneciente a su vista asociada. Inicialmente lo recibe la capa de autenticación y esta lo propaga al controlador general. Cuando el controlador general lo recibe, entonces oculta la ventana de generación anterior y carga el HTML del minijuego del Pong.

Evento 7: **pong_game_loaded**. Este evento lo envía el controlador del minijuego del Pong inmediatamente después de cargarse su árbol DOM al controlador general. De forma similar al evento `qr_generation_ready`, garantiza al controlador general que el árbol DOM del minijuego está cargado y el controlador del Pong podrá recibir sus eventos. No transporta ningún dato.

Evento 8: **add_session_user_slot**. Se trata de un elemento clave en la posibilitación de la modalidad multijugador en el minijuego del Pong. En este sentido, este evento se genera en la capa del minijuego cuando el usuario comienza la partida del minijuego bien con la modalidad específica "Dos jugadores (local)"o "Cuatro jugadores (local)" y se envía al controlador general, que a su vez lo reenvía al controlador de autenticación.

En concreto, se envía una vez por cada jugador a instanciar según el número seleccionado en el desplegable, descontando una unidad que se asume por la presencia implícita permanente del jugador primario. Una vez el controlador de autenticación recibe cada evento, **incrementa el número máximo de usuarios** en la sesión activa en una unidad, y trata de acceder e inicializar otra cámara más, lo cual queda justificado en la sección 4.9. Por lo tanto, el evento provoca que se permita la simultaneidad de ambos jugadores en el control de la partida. Por último, no contiene ningún dato.

Evento 9: **set_external_camera_tracking**. Este evento lo envía el controlador general al del minijuego del Pong, con el objetivo de ordenarle tan pronto sea posible que el minijuego no inicialice la cámara y por tanto no realice otros seguimientos a través de la misma, programados también en su propia capa para el control del avatar. De lo contrario, la solicitud de acceso de la cámara daría lugar a conflicto.

Evento 10: **transform_player_block_from_qr**. Este evento lo envía el controlador general al del minijuego del Pong. Siendo el evento que más datos transporta, tiene el objetivo de posicionar al avatar del jugador según la posición de su código QR con respecto a la captura de la cámara. Para ello se realiza un escalado proporcional entre el área de captura y el área de movimiento en el escenario del minijuego. Esta y otras transformaciones gráficas realizadas tras la recepción de este evento se exponen en la sección 4.8.

Cabe destacar que uno de los datos que transporta es el **índice del usuario** detectado con respecto al *array* que agrupa a los usuarios en la sesión activa, lo que permite directamente al minijuego del Pong determinar cuál es el avatar que debe reposicionar y tener en cuenta el desplazamiento (en el eje horizontal) de la mitad de campo asignada a dicho avatar en los cálculos gráficos. Esta es una de las tareas donde precisamente interviene, en parte, el API del Canvas de HTML5, dedicado en la sección 4.5. Por último, el evento contiene datos recibidos en ambos eventos anteriores `qr_auth_video_dimensions` y `qr_user_detected`.

4.2 Bibliotecas *jQuery*, Bootstrap y *jsqrcode*. API de Google Charts.

En esta sección se describen algunas bibliotecas fundamentales en el desarrollo de la aplicación. Puesto que se dedican pocos párrafos a cada una, se ha decidido tratarlas de forma unificada en una misma sección.

La implementación de la aplicación web habría resultado más lenta de no ser por el uso de ciertas bibliotecas que facilitan la manipulación del árbol DOM y la construcción de elementos gráficos requeridos.

En primer lugar, se ha incorporado la biblioteca *jQuery* [19]. Esta biblioteca consta de un elevado conjunto de funciones en JavaScript que facilitan la modificación del árbol DOM. No está orientada a crear vistas, si no que se trata de una herramienta de propósito general. Es una biblioteca muy empleada por los programadores Web. La versión que finalmente usa la aplicación web de esta biblioteca es la **v4.1.1**.

Sin lugar a duda, la función más empleada de esta librería es `jQuery` o `$`. Esta función recibe como parámetro una cadena que se interpreta como un selector de elementos propia de CSS3. Gracias a ella, se agiliza la tarea de modificar los elementos del árbol DOM en el documento HTML, puesto que hay funciones equivalentes en JavaScript puro, pero son más largas. Este es solo un ejemplo de la agilidad para la programación de esta biblioteca, que facilita un elevado número de tareas en la manipulación de dichos elementos (animaciones, captura de eventos, encadenamiento de funciones sobre el mismo elemento, etc.). En la figura 4.2 se muestra un ejemplo del uso de esta función en la aplicación web que aprovecha el mecanismo de encadenamiento de funciones, acortando en gran medida el código necesario con relación a JavaScript puro.

```
1 $('#camera_videos_wrapper').children().not('#camera1_video')
2   .each((index, video) => {
3     // Código no relacionado oculto, por claridad
4   });
```

Figura 4.2: Extracto de código de la aplicación web que, mediante *jQuery*, selecciona elementos de vídeo para realizar cierta tarea sobre cada uno de ellos en bucle (oculta por simplicidad). Se aprecia el uso del encadenamiento de funciones posibilitado por esta biblioteca.

En segundo lugar, la biblioteca Bootstrap [20] es un *framework* que proporciona una amplia gama de elementos gráficos populares y profesionales a partir de nuevas definiciones en las hojas de estilo en cascada o CSS3, y nuevas clases para los elementos del árbol DOM en HTML. Es la biblioteca más popular para prototipado gráfico en la Web. La versión de esta biblioteca presente en la aplicación es la **v3.3.1**. En la figura 4.3 se muestra parte de la declaración de la ventana superpuesta o *modal* sobre sintaxis HTML, usando clases Bootstrap, dentro de la cual se genera la imagen del código QR del usuario.

```
<div id="qr-gen-modal" class="modal fade" tabindex="-1" role="dialog"
  data-backdrop="static" data-keyboard="false">
<div class="modal-dialog modal-dialog-centered" role="document">
  <div class="modal-content"></div>
  <div class="modal-body"></div>
  <div class="modal-footer"></div>
</div>
```

Figura 4.3: Extracto de código del árbol DOM en HTML que muestra la declaración parcial, haciendo uso de clases Bootstrap, de la ventana superpuesta o *modal* donde se genera la imagen del código QR del usuario.

En tercer lugar, se procede a describir la biblioteca *jsqrcode* [21]. Esta es una biblioteca de código abierto que permite decodificar códigos QR. Es a su vez una adaptación al lenguaje JavaScript de la biblioteca nativa ZXing escrita C+-, donde la adaptación se encarga de trasladar dicha funcionalidad directamente a la Web de forma local. Tiene dos posibles fuentes de obtener la imagen:

1. A partir de un objeto de Adobe Flash, con extensión `.swf`, situable como objeto embebido en HTML.

2. A partir de un elemento Canvas, elemento incorporado HTML5 tal como se indica en el capítulo 1. La API Canvas se expone en la sección 4.5.

En la aplicación web desarrollada se hace uso del segundo modo, el Canvas, puesto que forma parte de las tecnologías gráficas de HTML5 estándar y de licencia abierta, más el interés del proyecto actual.

En cuarto y último lugar, en esta biblioteca es fácil modificar aspectos como los valores devueltos en la respuesta de cada llamada necesaria para decodificar códigos QR, a saber, `qrcode.decode()`, lo cual se realiza en esta aplicación y se explica en la sección 4.8. Por este motivo y por su rapidez en el tiempo de respuesta del proceso de decodificación, como se muestra en la sección 5.3, se ha decidido hacer uso de esta biblioteca. Existen más bibliotecas para este menester en la Web; muchas de ellas, para ser procesadas en un servidor web, pero también se encuentran similares para ejecutarse en local [22].

Por último, se introduce ahora brevemente el API de Google Charts [23]. Este se trata de un API en línea de la mano de Google de tipo RESTful. REST es un mecanismo que hace uso de los métodos HTTP (GET, HEAD, POST, PUT) para solicitar la descarga o edición de recursos en línea. Google Charts ofrece la confección de multitud de tipos de gráficas. De esta forma, la biblioteca Google Charts se ha empleado para generar códigos QR, cuya configuración se establece mediante solicitudes GET con los parámetros deseados tales como los datos y codificación deseados en la propia URL con cierto formato debidamente documentado [23].

4.3 Biblioteca *jshashes*

Tal como está pensado el diseño de la aplicación, no se requiere emplear un mecanismo para encriptar las contraseñas a almacenar en los códigos QR, puesto que no está destinada a ser una arquitectura *backend* persistente (los usuarios se almacenan mientras dure la sesión del navegador). Sin embargo, conviene añadir un mínimo de seguridad en las contraseñas debido a que, por un lado, se hace uso de una API remota para generar los códigos QR (Google Charts), y ésta recibirá los datos para generarlo. Y por otro lado, debido a que es esencial impedir que posibles intrusos escaneen dicho código y averigüen su contraseña si se codificase en texto plano.

Por tanto, se ha decidido hacer uso de una biblioteca que codifique las contraseñas. En concreto, se ha empleado la biblioteca *jshashes* escrita en JavaScript y hecho uso del algoritmo `MD5` [24] soportado en esta biblioteca para generar un *hash* de la contraseña de cada usuario recién dado de alta, y codificarlo en el código QR generado. En relación con otros algoritmos criptográficos de *hashing*, se ha elegido este puesto que se ha obtenido una buena relación de seguridad y tiempo de detección en la aplicación debido a su longitud idónea de caracteres producidos, sin ser el algoritmo más seguro. En la figura 4.4, se muestra la llamada al API de `MD5` de esta biblioteca usada en el momento de la segurización de las contraseñas de los usuarios.

```
1 User.prototype.setHashFromPlainPass = function (pass) {
2   this.passHash = new Hashes.MD5().hex(pass);
3 }
```

Figura 4.4: Ejemplo de invocación al API `MD5` de *jshashes*, empleada en la aplicación. La función `setHashFromPlainPass` es un breve método envolvente asociado a la clase `User`, que recibe como parámetro la contraseña introducida por el usuario en plano, y seguidamente se le pasa a la función `hex()` del `MD5`, que obtiene el *hash* de la contraseña exclusivamente formada por caracteres hexadecimales.

Por otra parte, un factor a favor destacable a la hora de usar esta biblioteca se trata de que sus funciones de transformaciones permiten obtener a partir de una cadena con caracteres y codificación arbitrarias, otra cadena (el *hash*) exclusivamente formada por caracteres hexadecimales ($0-9$ y $a-f$). Esto garantiza la compatibilidad total de datos arbitrarios para ser codificados en los códigos QR, dado que, además, las longitudes de los *hashes* generados no exceden el tamaño de datos máximo de la mayoría de versiones de códigos QR. Por descontado, la ventaja expuesta está presente en otras bibliotecas y algoritmos a emplear.

4.4 API getUserMedia

La historia del acceso a los periféricos de entrada multimedia desde la Web ha llevado recientemente a la creación de estándares implantados con mayor o menor soporte en los navegadores Web principales.

Actualmente, la forma de acceder a la cámara y el micrófono se realiza mediante la API conocida por el nombre de su rutina principal, `getUserMedia` [7], concretamente a través de JavaScript. Esta API es sin duda la protagonista en la aplicación desarrollada, puesto que es el requisito fundamental a todos los procesamientos de imagen realizados en la aplicación. Por tanto, requiere una especial mención.

A través de esta API se pueden configurar multitud de atributos deseados o requeridos de los dispositivos a solicitar, e incluso especificar el identificador del dispositivo

a capturar. Así, en cuanto a las cámaras, se puede especificar atributos como la relación de aspecto, dimensiones, cuadros por segundo o selección de cámara concreta (frontal o trasera) en dispositivos móviles. En la figura 4.5 se muestra un extracto del uso de esta API en la aplicación.

```
1  // Wrapped getUserMedia
2  // Input: cameraIdString - Optional device id to request a specific camera
3  // $video - Optional (jQuery) wrapper of Video element to attach stream to
4  function wrappedGUM(cameraIdString, $video) {
5    // Giving preference to newest API.
6    // Correct prefix needs to be used to prevent context error (tested).
7    let prefix = navigator.mediaDevices || navigator;
8
9    // NOTE: It'd be a syntax error making a new variable out of a property.
10   // Simply re-assign the existing property to the compatible API:
11   prefix.getUserMedia = prefix.getUserMedia || prefix.mozGetUserMedia
12     || prefix.webkitGetUserMedia || prefix.msGetUserMedia;
13
14   // Init video, in the hope that a natural camera resolution will be used
15   let retVal = prefix.getUserMedia({
16     video: { deviceId: cameraIdString }, audio: false
17   }, stream => onCameraSuccessEx(stream, $video), onCameraError);
18
19   // Código menos relacionado oculto, por claridad
20
21   if (typeof(retVal) === 'object') {
22     // Assumme returned value is a Promise from the newest getUserMedia API
23     retVal.then(stream => onCameraSuccessEx(stream, $video)).catch(
           onCameraError);
24   }
25 }
```

Figura 4.5: Extracto del código que muestra la llamada al API `getUserMedia` [7] en la aplicación para solicitar acceso a las cámaras. La función `wrappedGUM` es una función envoltorio reutilizada en la aplicación web que recibe como parámetro la identificación de la cámara a solicitar (`cameraIdString`) y el objeto *jQuery* del elemento de vídeo sobre el cual asignar la captura en vivo de la cámara en caso de éxito. En las líneas 5 a 12 se prepara una independización del API `getUserMedia` entre navegadores, y seguidamente se invoca. En la línea 21 se condiciona el éxito o fallo para invocar a una función u otra.

4.5 API Canvas

Canvas constituye una incorporación moderna a los navegadores Web populares. Se trata de un espacio que se puede emplear para gráficos y animación [25] a través de un conjunto estandarizado de funciones, que se asentaron con la llegada del HTML5. Estas funciones se ejecutan mediante el lenguaje de *scripting* JavaScript. Cada instancia Canvas se ha de definir inicialmente mediante el elemento `<canvas>` y las dimensiones de su área o espacio gráfico disponible, de la forma expuesta en la figura 4.6. En la misma figura se muestra la definición completa del avatar del jugador primario perteneciente la parte del minijuego del Pong de la aplicación web.

La unidad del ancho y el alto especificados es siempre en píxeles. Además, se debe tener en cuenta que las otras propiedades `width` y `height` de mismo nombre que las de la figura 4.6, pero pertenecientes a la hoja de estilos CSS3 y también aplicables al elemento `<canvas>` en cuestión, no guardan relación con las dimensiones del espacio disponible para las funciones gráficas. Estas propiedades simplemente determinarán el espacio estático del elemento en cuestión.

```
1 <canvas id="player_block1" class="elemento_absoluto default_quarter1" width
    ="150" height="150"></canvas>
```

Figura 4.6: Declaración del Canvas sobre el DOM en HTML5 asociado al bloque-avatar del jugador principal en el minijuego del Pong.

Canvas permite trabajar en modo 2D o 3D. Para ello, hay que obtener el contexto deseado previo al trabajo gráfico, llamando a la función `getContext` con el parámetro `'2d'` o `'webgl'` si se desea emplear el modo gráfico en dos dimensiones o en tres dimensiones, respectivamente (ver figura 4.7). El valor `'webgl'` hace referencia a WebGL, una especificación para gráficos en tres dimensiones con API sobre JavaScript creada en 2011 por el grupo Khronos [25], que fue incorporada posteriormente a Canvas.

```
1 let context = $canvas[0].getContext('2d');
```

Figura 4.7: Selección de contexto gráfico (2D) de un elemento Canvas mediante *jQuery* en la aplicación Web.

Una vez se ha seleccionado uno de los dos modos gráficos permitidos (2D/3D), ya no es posible cambiar de modo sobre el mismo elemento `<canvas>`. De intentarlo, el valor nulo (`null`) sería devuelto en la nueva llamada a `getContext` con el valor distinto del parámetro del contexto. A partir de las preparaciones explicadas y, asignada una variable al objeto devuelto con la llamada a getContext (tal como muestra la figura 4.7), se puede comenzar a hacer uso de sus funciones gráficas.

4.6 Comunicación mediante *iframes*

Dado que la aplicación Web consta de diversos archivos HTML interconectados, como se ha resumido en la sección 3.2 ha sido necesario hacer uso de un mecanismo existente para la inclusión y otro para la comunicación entre los distintos archivos. En este apartado se indaga en cómo se ha implementado esta necesidad y se alude al contexto evolutivo que ha permitido estandarizar estas funcionalidades en la World Wide Web.

Durante muchos años, los navegadores Web podían contener una página (a nivel de usuario) como máximo en cada ventana, es decir, no existían las pestañas [26]. Previo al lanzamiento de HTML5, los archivos HTML podían cargar otras páginas de cualquier origen dentro de la misma mediante el elemento *frame*, formando ventanas embebidas. Sin embargo, dichas páginas solo podían comunicarse entre sí en el caso de pertenecer al mismo origen, por motivos razonables de seguridad. Esta limitación se superó con la llegada de HTML5, el cual, además de simplificar el proceso a través de su nuevo elemento *iframe* del árbol DOM, permite comunicar ventanas de cualquier origen entre sí. Funciona mediante un mecanismo bidireccional de paso de mensajes o eventos: la ventana remitente ha de invocar el método `postMessage` sobre la ventana destino, indicando como primer parámetro el mensaje deseado y como segundo, el origen que se presupone existente en la ventana receptora. De esta forma, la ventana recipiente es capaz y debería comprobar que el origen es fiable. Este método, expuesto en JavaScript, se puede apreciar de forma más concisa en la figura 4.8.

Otro mecanismo de incrustación de páginas o ventanas en HTML5 es la función y paradigma conocido como XMLHTTPRequest (o XHR, para mayor brevedad). Se trata de una petición asíncrona para cargar cierto contenido de forma aislada mediante el protocolo HTTP dentro de otra página. La técnica también se conoce como AJAX (Asynchronous

```
targetWindow.postMessage(message, targetOrigin, [transfer]);
```

Figura 4.8: Prototipo o declaración de la función `postMessage`, perteneciente al API en JavaScript de las ventanas en HTML5.

JavaScript And XML). Permite cargar y embeber, por tanto, otras páginas HTML5. A diferencia de los *iframes*, cuando se carga el nuevo contenido con éxito, pertenece al mismo árbol DOM que la página inicial, y por tanto se puede controlar todo mediante JavaScript. Aquí, la carga de una ventana embebida solo se permite cuando ambas pertenecen al mismo origen, y sobre ciertos protocolos. La figura 4.9 muestra lo que ocurre cuando el *script* de una página que se ha abierto desde el Explorador de archivos intenta cargar otra página HTML5, sobre el navegador Google Chrome.

Figura 4.9: Error de depuración obtenido en Google Chrome al invocar un XHR sobre cierta página HTML del archivo *test.html*.

Por tanto, la aplicación ha hecho uso en todo momento de los *iframes* para comunicar los distintos archivos entre sí (sobre todo teniendo en cuenta que se requería separar la capa de autenticación de la del minijuego), y concretamente, enviándose entre sí los mensajes descritos en la sección 4.1.

4.7 Otras consideraciones

En esta sección se exponen tareas desarrolladas de menor peso en relación con los objetivos perseguidos de la aplicación, pero interesantes.

Función requestAnimationFrame. Se trata de una función incorporada en las WebAPI modernas sobre JavaScript. Mediante esta función, se le indica al navegador que se desea registrar otra función propia en la aplicación, a modo de *callback*, para ser llamada en cada fotograma de renderizado desde el hilo principal [18]. De esta forma, no hay que emplear llamadas tradicionales como `setInterval`.

Por otra parte, es un mecanismo que, según la estandarización, permite optimizar (omitir) el trabajo de renderizado del navegador cuando, por ejemplo, el usuario se encuentra visualizando completamente otra pestaña otra ventana. Se trata de una función útil y actualmente muy utilizada en videojuegos en la web [27] y otros procesos gráficos en tiempo real.

Así, en la aplicación web desarrollada se ha empleado en dos tareas. Primero, para el proceso de copia periódico sobre el elemento Canvas y decodificación de códigos QR sobre el mismo. Y segundo, para la simulación también periódica de los objetos del minijuego del Pong, que incluye manejo de colisiones, velocidad, dirección, transformaciones gráficas, etc.

Respuesta ante la desconexión de las cámaras. Existen diversos casos por los cuales una cámara inicializada previamente en la aplicación web podría dejar de emitir su flujo de captura a la máquina. Entre estos se encuentran la desconexión manual voluntaria o descuidada, la desconexión ante un fallo de su hardware o de su controlador o *driver* y un fallo provocado por el sistema operativo.

Ahora bien, se ha contemplado actuar ante cada desconexión, de la siguiente manera. Deteniendo el proceso implementado que copia la imagen de la cámara desde el elemento `<video />` asociado a cada cámara hacia un Canvas y la posterior decodificación de los códigos de barras desde el mismo. De esta forma se pretende optimizar el rendimiento de la aplicación.

Por tanto, se ha investigado la posibilidad de detectar la desconexión a través de atributos o funciones de las WebAPI. Se ha investigado y averiguado que es posible, a través de la API concreta `captureStream()`. Así, se ha aplicado esta forma. Sin embargo, dado que esta API no tiene soporte en todos los navegadores principales (a fecha), se ha implementado una detección alternativa heurística por temporización de los instantes de captura.

Este mecanismo alternativo espera a que transcurran **5** segundos sin recibir un cuadro de la cámara desde cada activación del *callback* del evento `timeupdate` del elemento `<video />`, para activar la respuesta. Ambos mecanismos se muestran en la figura 4.10.

4.8 Transformaciones gráficas empleadas

Tras haber abordado las premisas en cuanto a tecnologías empleadas en las secciones anteriores de este capítulo, se puede dar a paso a explicar el algoritmo seguido para realizar las transformaciones del avatar en el minijuego del Pong, el cual incluye uso de las funciones de la API Canvas en la aplicación.

En la aplicación desarrollada, el proceso comienza después de que la biblioteca *jsqrcode* detecta un código QR a través de la cámara en cualquier fotograma, devuelve el resultado a la capa de autenticación, esta valida al usuario según los datos decodificados y envía el evento `qr_user_detected` al controlador general.

La biblioteca *jsqrcode* ha sido modificada para devolver, además del contenido decodificado del código QR (por defecto), las coordenadas de sus puntos de posicionamiento (sus centros), las cuales son parte del resultado del algoritmo de escaneo de códigos QR interno de esta biblioteca y se han requerido para realizar las transformaciones gráficas.

En la figura 4.12 se muestran renderizados unos puntos vía Canvas que representan los patrones de posicionamiento del código QR de un usuario en un instante de tiempo, sobre el visor de la cámara en la vista de autenticación. A continuación, se describe el proceso de transformaciones gráficas que se ha desarrollado, de forma genérica:

Paso 1. Se parte de una entrada inicial consistente en un código QR y las coordenadas de sus tres puntos de posicionamiento (sus centros) que pertenecen al sistema de coordenadas de un área de captura, más las dimensiones de esta área y su relación de aspecto. El área de captura, en el aplicativo actual, es un visor de la captura de la cámara mostrada en pantalla. Es importante destacar que en este momento dicho área debería conservar las mismas proporciones que la realidad, es decir, se haya inicializado a una relación de aspecto ideal y, por tanto, el código QR capturado se visualice de forma totalmente cuadrada para garantizar la fidelidad de la imagen.

Paso 2. Se contempla ahora una nueva entrada: las dimensiones de un área gráfica de destino en la cual se dibujará un nuevo código QR equivalente al obtenido en el paso **1**. Para ello, se copian los puntos de posicionamiento del área de captura en el área de destino. Seguidamente, empleando la relación de aspecto del área de captura adquirida en el paso **1**, y las dimensiones de la nueva área, se realiza un escalado en el eje

de abscisas y ordenadas de los nuevos puntos de forma que resulten proporcionales al área de destino.

$$\overrightarrow{A_{destino}} = (\overrightarrow{A_{captura,\mathbf{x}}} * ancho_{destino}/ancho_{captura}, \overrightarrow{A_{captura,\mathbf{y}}} * alto_{destino}/alto_{captura})$$

$$\overrightarrow{B_{destino}} = (\overrightarrow{B_{captura,\mathbf{x}}} * ancho_{destino}/ancho_{captura}, \overrightarrow{B_{captura,\mathbf{y}}} * alto_{destino}/alto_{captura})$$

$$\overrightarrow{C}_{destino} = (\overrightarrow{C_{captura,\mathbf{x}}} * ancho_{destino}/ancho_{captura}, \overrightarrow{C_{captura,\mathbf{y}}} * alto_{destino}/alto_{captura})$$

Paso 3. Dada la configuración obtenida en el paso **2** sobre el área de destino, se calcula el centro de sus puntos de posicionamiento a partir del vector con origen el punto inferior izquierdo (A) y extremo el punto superior derecho (C), como el extremo de su vector mitad.

$$\overrightarrow{centro_{destino}} = (\overrightarrow{B_{destino}} + \overrightarrow{C_{destino}})/2$$

Paso 4. Ahora se procede a calcular las dimensiones de una caja de inclusión que envolverá al código QR de forma mínima. Esta caja se emplea en el minijuego del Pong de la aplicación como objeto asociado al código QR (avatar del jugador) para detectar las colisiones con la bola móvil, de forma que se simplifiquen al no tener en cuenta la rotación del código QR contenido. Para ello, se escala el vector con origen el punto inferior izquierdo (A) y extremo el superior izquierdo (B), y el vector con origen el punto superior izquierdo (B) y extremo el superior derecho (C), por una constante $k = 29/26$ empírica que produce vectores de módulo o longitud igual a su lado paralelo en el código QR (ver figura 4.15). Seguidamente se suman, por un lado, las coordenadas horizontales y por otro, las verticales de sendos vectores resultantes para obtener el ancho y el alto de la caja de inclusión mencionada respectivamente.

$$\overrightarrow{AB_{destino'}} = \overrightarrow{AB_{destino}} * (29/26)$$

$$\overrightarrow{BC_{destino'}} = \overrightarrow{BC_{destino}} * (29/26)$$

$$ancho_{caja} = \overrightarrow{AB_{destino',\mathbf{x}}} + \overrightarrow{BC_{destino',\mathbf{x}}}$$

$$alto_{caja} = \overrightarrow{AB_{destino',\mathbf{y}}} + \overrightarrow{BC_{destino',\mathbf{y}}}$$

Paso 5. Se debe calcular un nuevo vector a partir de uno de los calculados en el paso **4**, $\overrightarrow{BC_{destino'}}$. Se escoge una dimensión arbitraria, por ejemplo, la horizontal, y, empleando la relación de aspecto del área de captura y la medida de esta dimensión en el área de destino, se multiplica en esta dimensión sobre el vector de forma que los puntos de posicionamiento resultantes respeten la misma relación de aspecto que el área de captura. Esto significa que el vector resultante pertenece a un supuesto código QR cuya forma es cuadrada siempre que el del paso **1** también lo fuese y, gracias a esto, se puede calcular su ángulo de rotación y su longitud (ver figura 4.16), que se usarán para dibujar en el siguiente paso.

$$ratio = ancho_{destino} / relacion\_aspecto_{captura} = ancho_{destino} * alto_{captura} / ancho_{captura}$$

$$\overrightarrow{BC_{destino''}} = (\overrightarrow{BC_{destino'},\mathbf{x}} * ratio, \quad \overrightarrow{BC_{destino'},\mathbf{y}})$$

$$\alpha = \text{arc tg}(\overrightarrow{BC_{destino''},\mathbf{y}} / \overrightarrow{BC_{destino''},\mathbf{x}})$$

$$\|\overrightarrow{BC_{destino''}}\| = \overrightarrow{BC_{destino''},\mathbf{x}} / \cos\alpha$$

Paso 6. Se dibuja una imagen cuadrada de un código QR con la misma información que el código referido en los pasos anteriores, cuya longitud del lado sea el módulo $\|\overrightarrow{BC_{destino''}}\|$ obtenido en el paso **5**, y con centro el obtenido en el paso **3**. Inmediatamente después, se traslada el origen del sistema de coordenadas al centro de la imagen para escalar la imagen por el factor inverso al obtenido en el paso **5**, es decir, $1/ratio$, en la dimensión escogida en el mismo paso, y se aplica el ángulo de rotación α también calculado en ese paso. La imagen resultante es totalmente proporcional al área de destino y tiene el ángulo de rotación adecuado, independientemente de que constituya una forma cuadrada o no. En el aplicativo actual, lo anterior se ha realizado con los métodos `scale` y `rotate` de la API Canvas. El algoritmo concluye.

4.9 Modo multijugador

Tal como se ha introducido en el capítulo 3, en la aplicación desarrollada se ha implementado una modalidad multijugador en la capa del minijuego del Pong. En concreto, se trata de un modo para dos jugadores en la misma máquina. Esta modalidad ha planteado nuevos retos a la hora de conseguir la jugabilidad con dos usuarios, habiéndose identificado varias limitaciones a priori:

1. El natural aumento de la distancia necesaria para interactuar sobre una misma cámara ambos jugadores, en relación con la capacidad de respuesta de la biblioteca *jsqrcode* bajo esta situación.
2. De forma simétrica al factor anterior, la adecuación del espacio disponible para ambos jugadores suponiendo que la distancia respecto a la cámara no variase respecto al modo de un jugador.
3. La propia capacidad de la biblioteca *jsqrcode* para escanear y decodificar con éxito más de un código QR en la misma imagen o *frame* exacto de la captura en vivo.

Teniendo en cuenta los puntos anteriores, se ha decidido comprobar el factor más limitante, esto es, la capacidad de la biblioteca *jsqrcode* para decodificar múltiples códigos QR en la misma imagen. Mediante una sencilla prueba, consistente en realizar una subida al formulario de autenticación de una fotografía que contiene dos códigos QR perfectos asociados a cuentas de usuario registradas en la página, se ha comprobado que la biblioteca no ofrece esta capacidad. En la figura 4.18 se muestra la imagen probada.

Por tanto, se ha determinado la necesidad de elaborar una solución alternativa. Esta alternativa consiste en comprobar la presencia y solicitar el acceso a una cámara extra en el momento en el que se inicia la partida con el modo multijugador, por cada solicitud de reserva una nueva ranura de usuario en la sesión, a través del evento `add_session_user_slot`. Por supuesto, no es la técnica más eficaz puesto que el usuario quizás no disponga de otra

cámara conectada. Aun así, cuando esta técnica tiene éxito la aplicación permite que cada usuario interactúe con su código QR y una cámara distinta. Esto reduce los problemas de espacio y distancia expuestos como parte de los factores limitantes identificados.

Para llevar a cabo esta idea se ha investigado la posibilidad de solicitar acceso a otros dispositivos (en este caso a una segunda cámara) haciendo uso de las WebAPI en HTML5. Tras investigarlo, se ha concluido que es perfectamente posible, haciendo uso de la API concreta `navigator.mediaDevices` e invocando su método `enumerateDevices()`. De hecho, esta función devuelve un objeto `Promise` que permite obtener un listado de todos los periféricos conectados a la máquina y atributos. Tras obtener el listado con éxito, se ha realizado sobre el mismo una búsqueda de otra cámara distinta a la original, por tanto, habiendo sido necesario almacenar el identificador de la primera cámara para compararlo. En la figura 4.19 se muestra el código empleado para buscar una segunda cámara y solicitar acceso a la misma. Esta API de HTML5, perteneciente a WebRTC, se ha consultado en [6].

4.10 Subida y descarga del código QR del usuario

Como se ha contemplado en el capítulo 3, el usuario tiene la posibilidad de subir al formulario de autenticación un fichero de imagen que contenga el código QR asociado a su cuenta. De esta forma el controlador de autenticación puede validarlo e incorporar al usuario en la sesión activa, ofreciéndole la siguiente ventana según el flujo de la aplicación web. Se recuerda que también se permite decodificar códigos de barras tradicionales para autenticar, a fin de realizar comparativas entre cada tipo de código.

Para esto existe un elemento de HTML, `input`, que permite realizar las subidas. Esta gestión se complementa con funcionalidad en el controlador. En la figura 4.20 se muestra a modo de ejemplo la configuración del elemento *input* correspondiente empleado en la aplicación.

Por otra parte, la versión 5 de HTML ofrece la posibilidad de descargar multitud de contenido presente en las páginas web como ficheros con el formato que proceda en el dispositivo. Así, las imágenes contenidas en los elementos `<img />` clásicos del árbol DOM se pueden recuperar. Entre otras cosas, también es posible construir ficheros de imagen a partir de un elemento Canvas, lo cual permite recuperar desde estos elementos el dibujo que contengan en cualquier instante deseado si se realizan animaciones sobre ellos.

Esta capacidad de HTML ha permitido implementar la funcionalidad en la aplicación web para visualizar y descargar la imagen del código QR asociado a la cuenta del usuario. El diseño gráfico relativo a esta funcionalidad ha quedado contemplado en el capítulo 3. Para ello, ha sido necesario contar con un elemento `<img />` y un elemento de hipervínculo `<a />` definidos en el árbol del árbol DOM de la vista de generación del código QR del usuario, tal como se refleja en la figura 4.21.

Dado que, por defecto, los hiperenlaces redirigen al usuario al recurso de la URL que contienen, se ha configurado debidamente el hiperenlace empleado para descargar el recurso en su lugar, lo cual se realiza añadiéndole el atributo `download` a sus líneas de HTML y como valor opcional el nombre del archivo a descargar: "Mi código QR" (la extensión se añade automáticamente y se detalla en los párrafos posteriores).

Sin embargo, se ha lidiado con una limitación. La preparación para la descarga a priori sería tan sencilla como asignar la URL de la imagen al elemento de hipervínculo presente (a su atributo `href`). Se recuerda que las imágenes de los códigos QR de los usuarios se generan como respuesta a una petición HTTP RESTful al API de Google Charts. Pero,

concretamente con la URL de esta petición, no se ha conseguido descargar la imagen, en su lugar avisando por la salida estándar del mensaje mostrado en la figura 4.22 y redirigiendo al usuario a esta URL. Esta situación se puede salvaguardar con una serie de pasos explicados a continuación. La implementación conjunta de los mismos se muestra en la figura 4.23.

En primer lugar, se ha de añadir el atributo `crossorigin` al elemento `<img />` del árbol DOM que contiene la imagen cargada desde la URL en cuestión, y fijar su valor a la cadena '`anonymous`'. De esta forma, la política conocida como Intercambio de Recursos de Origen Cruzado (CORS) [28] concederá permiso para descargar la imagen desde un elemento Canvas auxiliar donde se ha de copiar la imagen anterior. De otra forma, la misma denegaría tal descarga por motivos de seguridad, al pertenecer a un dominio externo a la aplicación web.

En segundo lugar, se procede a modificar el controlador de la capa de generación del código QR del usuario (con JavaScript). Así, se crea un elemento Canvas auxiliar y se copia en este la imagen contenida en el elemento `<img />`, mediante el método `drawImage` de la API Canvas. Seguidamente, se invoca otra función de esta API, `toDataURL`, la cual construye una pseudo-URL formada por ciertos metadatos (tales como una extensión del archivo, por defecto PNG), seguidos de la información exacta de su dibujo embebido o el código QR del usuario en este caso. La extensión anterior, mantenida en este caso como PNG, será la utilizada en el fichero de la imagen finalmente descargada.

En tercer y último lugar, se asigna la pseudo-URL obtenida en el paso anterior al elemento de hiperenlace presente en el árbol DOM de la capa de generación del código QR del usuario, concretamente a su atributo `href`, de nuevo mediante JavaScript en el controlador de la misma capa. En este punto el hipervínculo está listo para descargar la imagen con éxito cuando el usuario pulse sobre el botón designado "Descargar código" para este propósito.

```
1  // Copy captured image to canvas and scan QR from it
2  // (jsqrcode library requires it)
3  function captureToCanvas_ScanQR() {
4    // Código no relacionado oculto, por claridad
5
6    // Is newer API available and initialized? Also, did stream end?
7    if (
8      video.cachedCaptureStreamObject != null
9      && video.cachedCaptureStreamObject.readyState == 'ended'
10   ) {
11     // Stream disconnected.
12     // Log error, filter out Video and mark to stop rAF.
13     stopVideoStream($video);
14     return false;
15   }
16
17   // Código no relacionado oculto, por claridad
18
19   if (now >= video.nextCheckMsTime) {
20     // Video stopped responding: camera may disconnected.
21     // Log error, filter out Video and mark to stop rAF.
22     stopVideoStream($video);
23     return false;
24   }
25
26   // Código no relacionado oculto, por claridad
27 }
28
29 $('#camera1_video')[0].onplay = function() {
30   // Código no relacionado oculto, por claridad
31
32   try {
33     // BUG: We catch the reference of captureStream() here instead of
34     // within rAF, because calling it there stops some cameras's ticks!
35     this.captureStream = this.captureStream || this.mozCaptureStream;
36     this.cachedCaptureStreamObject = this.captureStream();
37   } catch (error) {
38
39   }
40
41   // Código no relacionado oculto, por claridad
42 };
43
44 $('#camera1_video')[0].ontimeupdate = function() {
45   delayCanvasCopyTimeout(this); // NOTE: this would be undefined in lambda
46 };
```

Figura 4.10: Código que muestra ambas medidas de detección de desconexión de las cámaras. De las líneas 7 a 15 y 35 a 36, se aprecia el uso de la WebAPI `captureStream()` que toma la prioridad frente a la detección heurística por temporización, presente en las líneas 19 a 23 (incluida respuesta ante la desconexión de la cámara en curso).

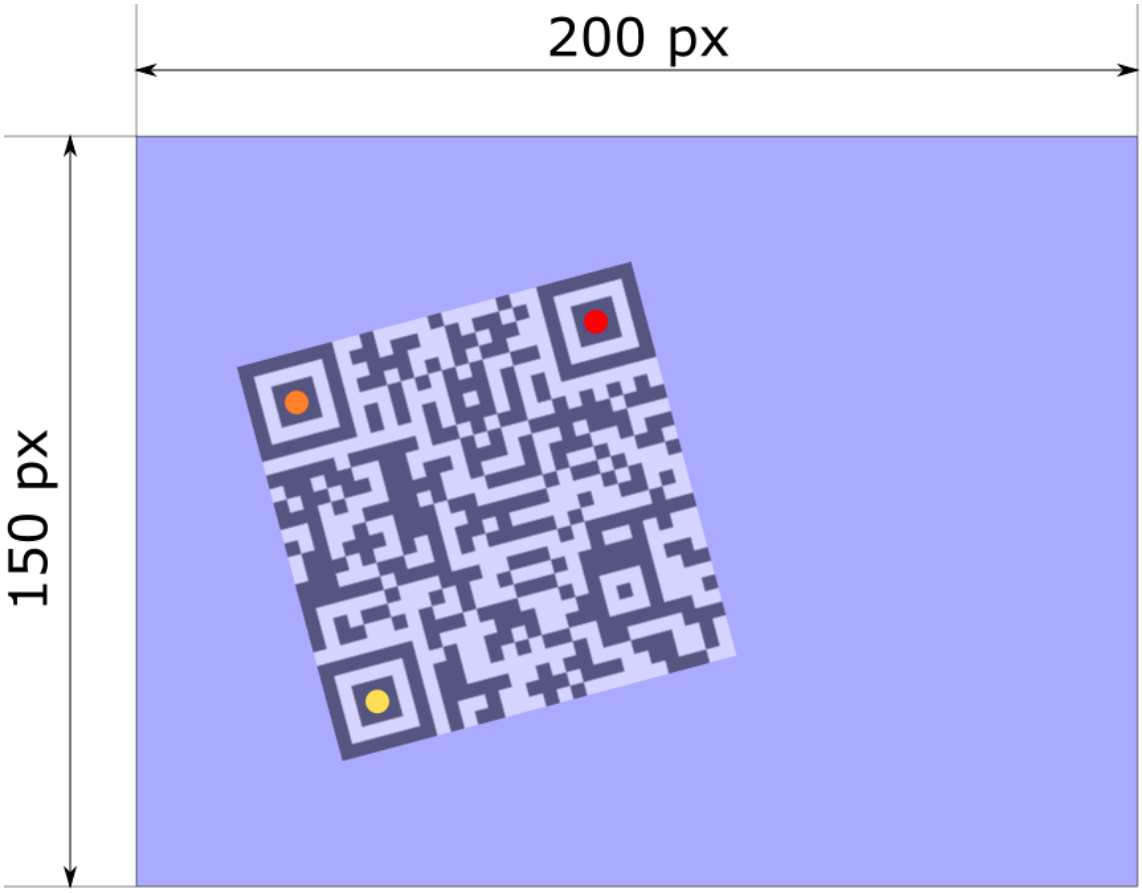

Figura 4.11: Ejemplo gráfico a modo de esquema del paso inicial del algoritmo de transformaciones por código QR. Se ha inicializado un área de captura con 200 píxeles de ancho por 150 píxeles de alto. En ella, los puntos de posicionamiento de un código QR se remarcan en amarillo, naranja y rojo.

Figura 4.12: Pantallazo de la aplicación que muestra el visor la captura de la cámara en vivo y un código QR de autenticación en un instante de tiempo. Se ha activado el modo de depuración para que la aplicación dibuje los puntos de posicionamiento detectados.

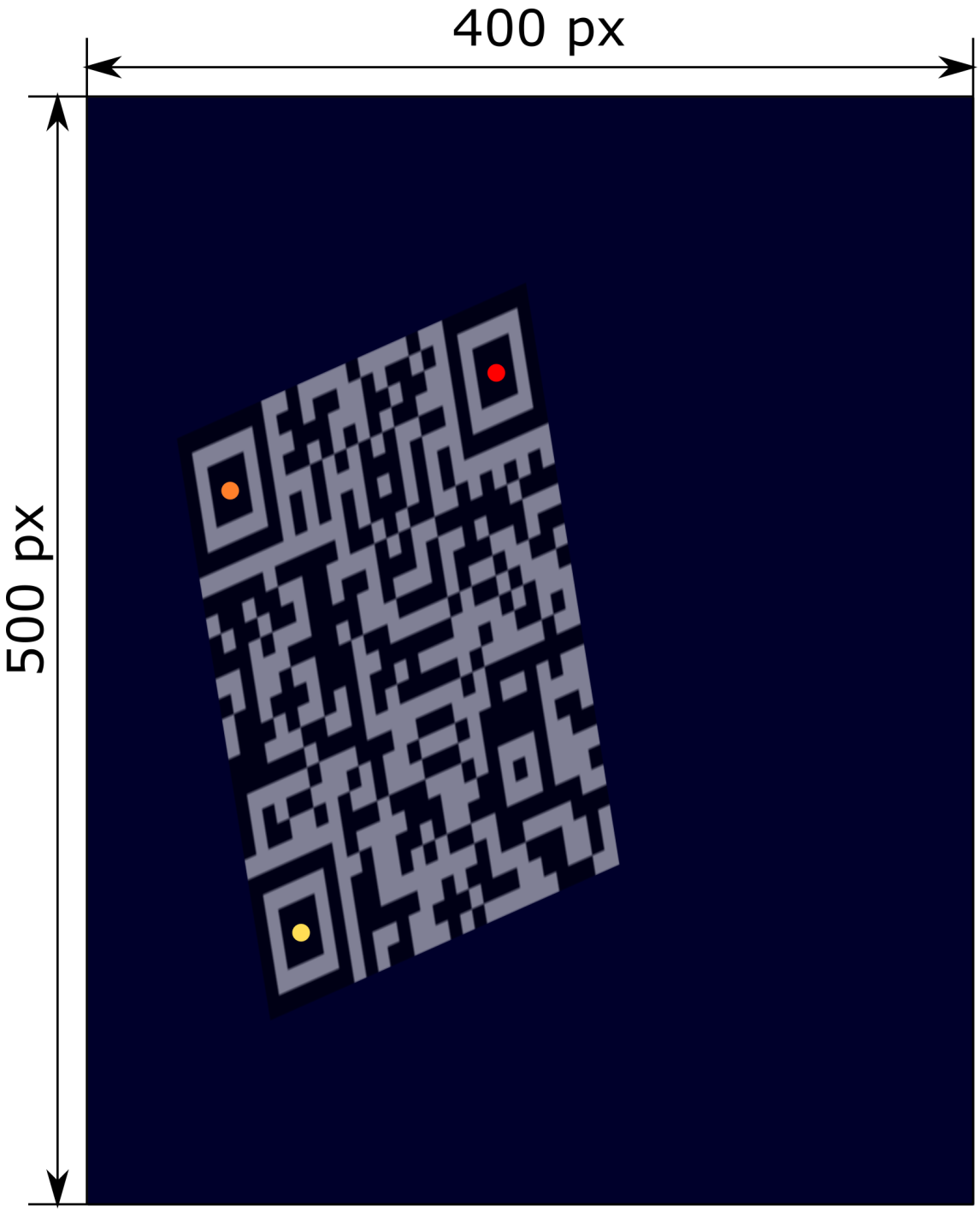

Figura 4.13: Ejemplo gráfico a modo de esquema del segundo paso del algoritmo de transformaciones por código QR. Se muestra un área de destino de 400 píxeles de ancho por 500 píxeles de alto, y los puntos de posicionamiento que han sido escalados a la misma área.

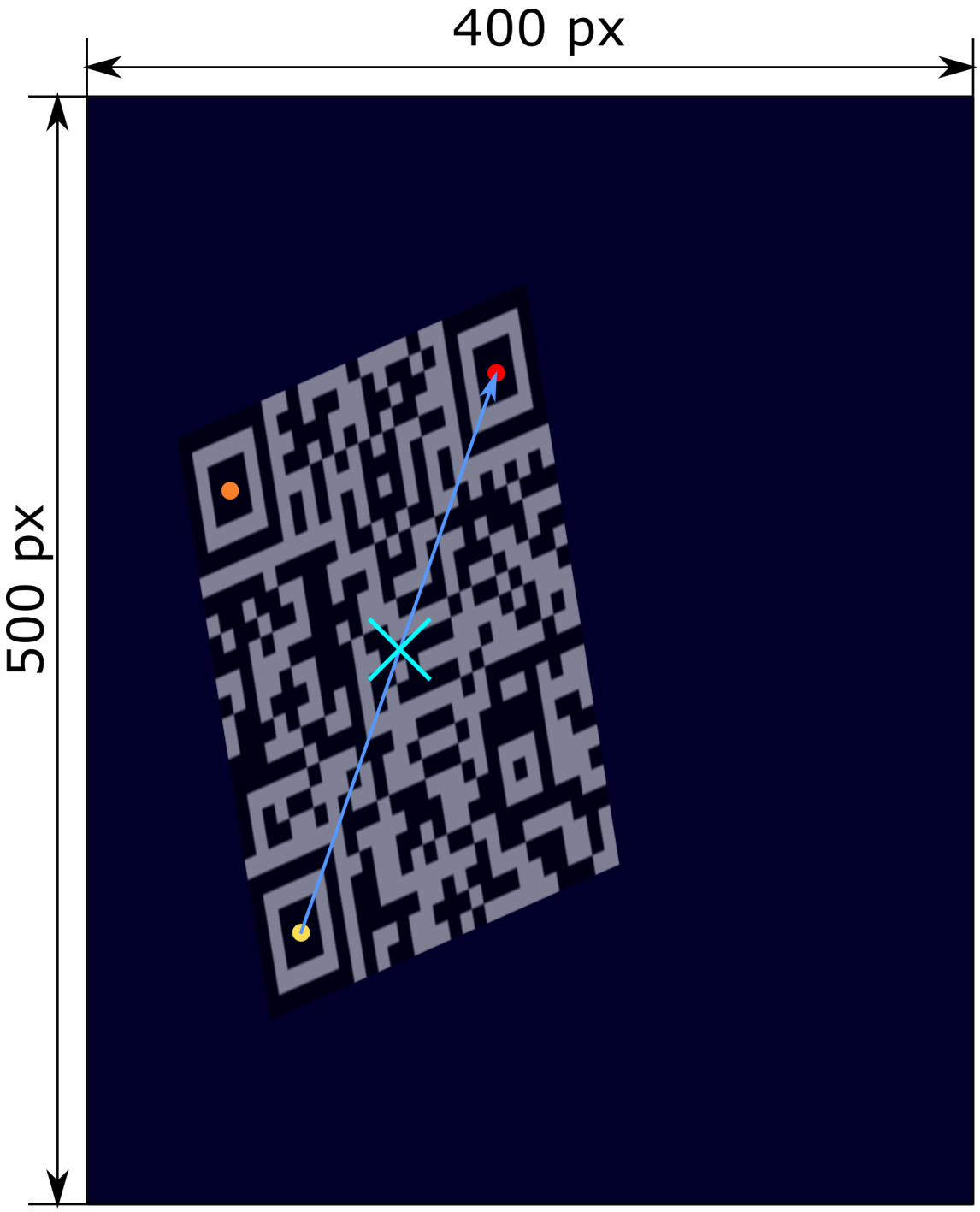

Figura 4.14: Ejemplo gráfico a modo de esquema del tercer paso del algoritmo de transformaciones por código QR. A partir de los puntos de posicionamiento inferior izquierdo y superior derecho se ha calculado el centro del código QR provisional, destacado en la ilustración.

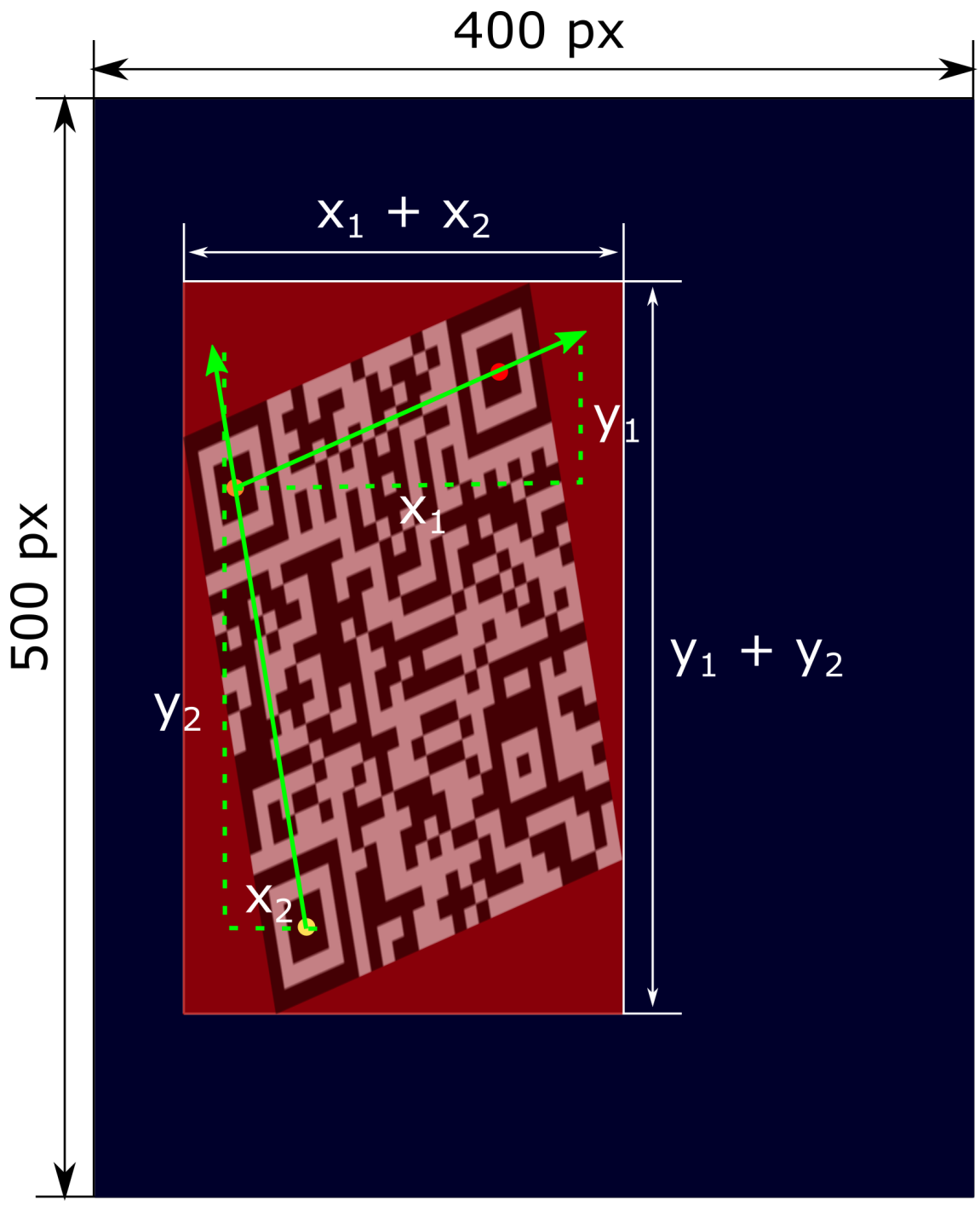

Figura 4.15: Ejemplo gráfico a modo de esquema del cuarto paso del algoritmo de transformaciones por código QR. Se remarcan en verde los dos vectores escalados cuya longitud es igual a su lado paralelo, así como la proyección de sus componentes, y en blanco las medidas genéricas que permiten obtener el ancho y el alto de la caja de inclusión.

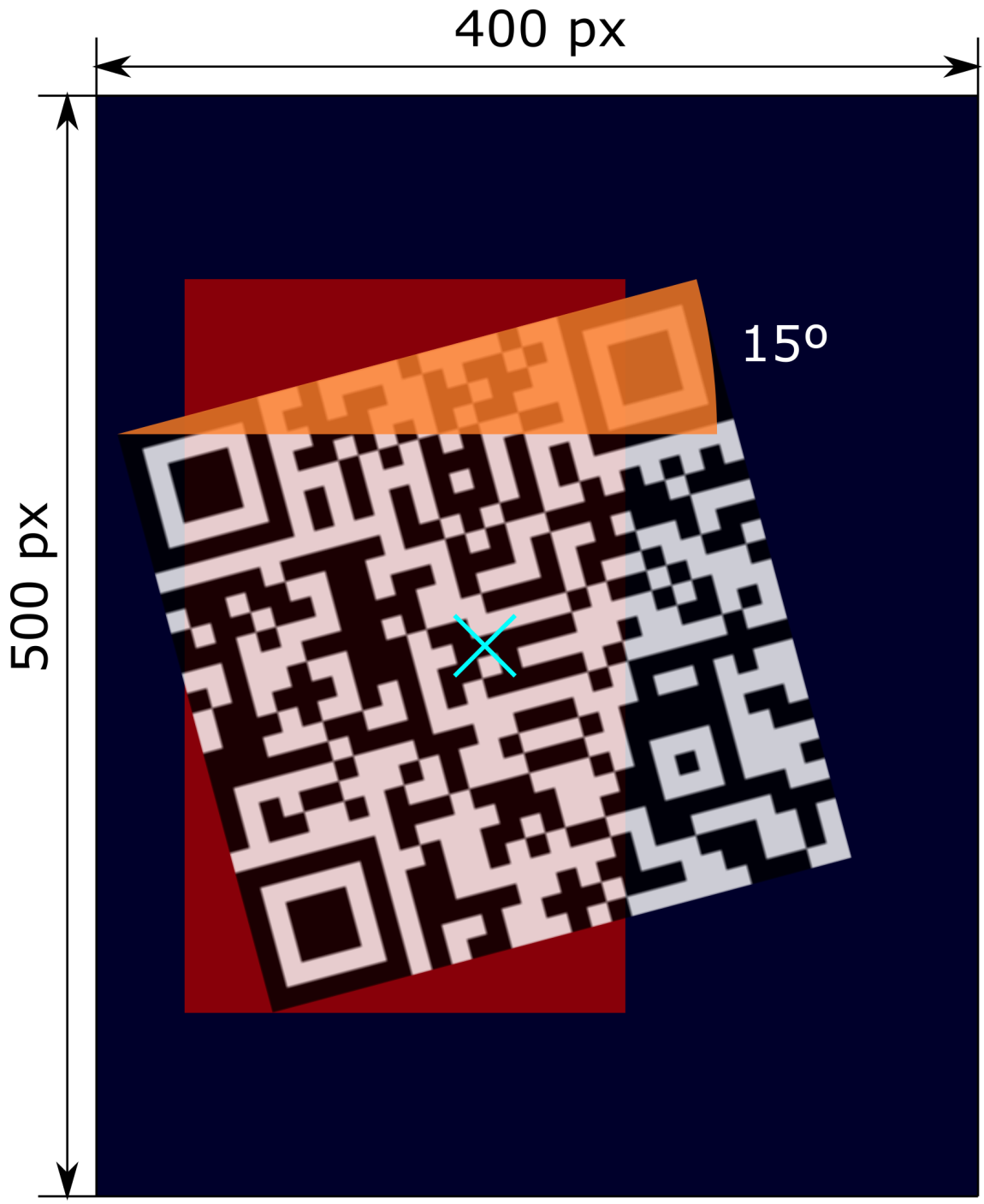

Figura 4.16: Ejemplo gráfico a modo de esquema del quinto paso del algoritmo de transformaciones por código QR. En naranja, el ángulo de rotación asociado al vector $\overrightarrow{AB_{destino''}}$.

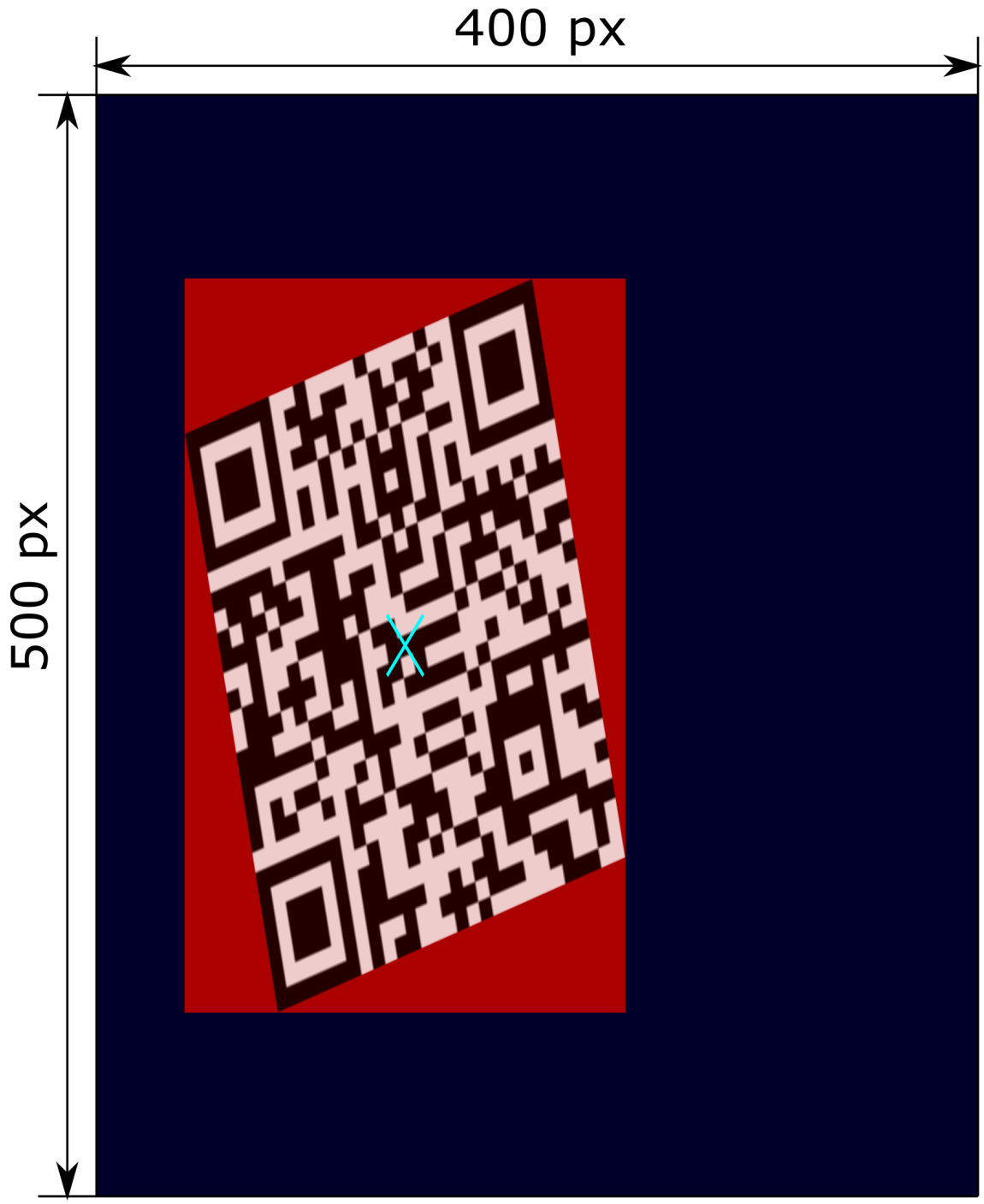

Figura 4.17: Ejemplo gráfico a modo de esquema del sexto y último paso del algoritmo de transformaciones por código QR. Se ha escalado una nueva imagen con centro la coordenada en azul por el factor $1/ratio$ en la dimensión horizontal y se le ha aplicado el ángulo de rotación α obtenido en el paso **5**, ambas operaciones realizadas sobre el punto central como pivote.

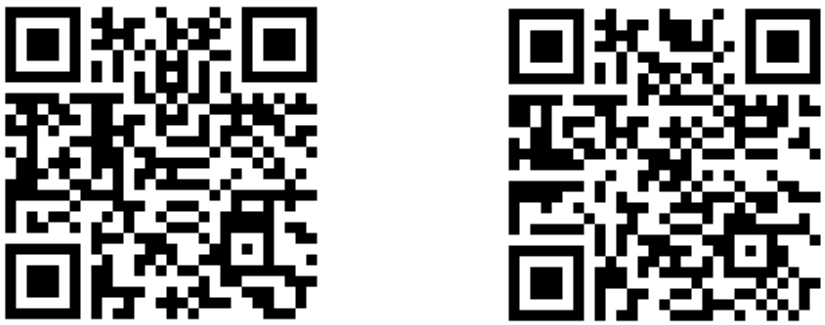

Figura 4.18: Ejemplo de la presencia de dos códigos QR reconocibles, pero no completamente detectables por la biblioteca *jsqrcode*.

```
navigator.mediaDevices.enumerateDevices()
  .then(function(devices) {
    // NOTE: Processing a MediaDeviceInfo object
    for (let device of devices) {
      if (device.kind != 'videoinput') {
        continue;
      }

      if (g_RequestedCamsIdString.indexOf(device.deviceId) != -1) {
        continue;
      }

      let $video = $('#camera_videos_wrapper').children()
        .eq(g_RequestedCamsIdString.length);
      wrappedGUM(device.deviceId, $video);
      return;
    }
  });
});
```

Figura 4.19: Extracto del código de la aplicación que realiza una búsqueda de la primera cámara que no haya sido solicitada previamente y, si la encuentra, solicita acceso a la misma.

```
<input id="image_upload_input" class="btn btn-secondary" type="file" accept
    ="image/*" />
```

Figura 4.20: Ejemplo del elemento en HTML que permite subir archivos empleado en la aplicación. En este caso se admiten archivos con formatos empleados para imágenes.

```
<div class="modal-body">
  <!-- HTML5 standard requires this crossorigin value to allow
  downloading external image that's copied to Canvas later -->
  <img id="qr_gen_auth_code" crossorigin="anonymous"
  alt=''Imagen del codigo QR generado para autenticarse'' />
</div>
<div class="modal-footer">
  <div class="container d-sm-flex justify-content-between">
    <a id="save_qr_image_btn" class="btn btn-warning"
    download="Mi código QR">Guardar c&oacute;digo</a>
  </div>
</div>
```

Figura 4.21: Extracto del código de la aplicación que contiene los elementos permanentes en el árbol DOM para lograr la descarga de la imagen del código QR del usuario.

```
Resource interpreted as Document but transferred with MIME type image/png:
    "https://chart.googleapis.com/chart?cht=qr&chs=256x256&chl=adrian%2081
    dc9bdb52d04dc20036dbd8313ed055".
```

Figura 4.22: Aviso que se ha obtenido al tratar de descargar la imagen del código QR asignando previamente la URL de Google Charts al elemento de hipervínculo, sin éxito.

```
function onGenerationQrImageLoaded() {
  // Begin obtaining a data URL of the loaded image to succeed on the
  // same-origin policy restriction on raw external URL downloads
  let $canvas = $('<canvas>');
  $canvas.prop('width', $(this).prop('width')); // CSS equivalent = 0
  $canvas.prop('height', $(this).prop('height')); // CSS equivalent = 0
  let context = $canvas[0].getContext('2d');
  context.drawImage($(this)[0], 0, 0);

  try {
    let qrImageDataUrl = $canvas[0].toDataURL(); // PNG is the default
    $('#save_qr_image_btn').show();
    $('#save_qr_image_btn').prop('href', qrImageDataUrl);
  } catch(error) {
    console.log(error);
  }
}
```

Figura 4.23: Extracto del código que configura la descarga del código QR del usuario para ser exitosa. Se aprecia el uso del elemento Canvas intermedio partícipe en la concesión del permiso requerido por la política CORS. Nota: el puntero `this` apunta al elemento `<img />` con la imagen de partida obtenida como respuesta a la petición HTTP sobre URL de Google Charts.

CAPÍTULO 5
Pruebas realizadas

5.1 Navegadores compatibles

A la hora de probar la aplicación web, existe una amplia variedad en los niveles de soporte del conjunto de las WebAPI empleadas en la misma por parte de los navegadores. No solo esto, sino que las bibliotecas externas preexistentes empleadas en la aplicación, tales como *jsqrcode* o JOB, también son susceptibles de no haber sido compatibilizadas por los desarrolladores con todos los navegadores probados. Se ha de notar que la tarea de maximizar la compatibilización de estas bibliotecas supondría un reto no contemplado en el alcance del proyecto, por descontado.

Así, la API `getUserMedia` [7], por ejemplo, tiene un soporte limitado según el navegador y el contexto en el que se emplee. Ahora bien, en la tabla 5.1 se puede comparar la compatibilidad de la ejecución de la aplicación web en distintos navegadores web principales y contextos evaluados.

Entorno	Navegador			
	Google Chrome 67.0.3396.99 (64 bits)	Mozilla Firefox 61.0 (64-bit)	Microsoft Edge 42.17134.1.0	Opera 54.0.2952.46
Local (file:///)	Completo	Limitado[a]	Nulo[b]	Moderado[c] [d]
Servidor web	Mayormente[d]	Completo	Moderado[d]	Mayormente [d]

[a] No se ha conseguido inicializar la cámara inicial, imprimiéndose el error `NotReadableError: Failed to allocate videosource`, formateado por la aplicación.

[b] La carga de la página web termina fallando, obteniendo ventana en blanco con sugerencia de refresco de la URL.

[c] No se ha conseguido inicializar las cámaras extra solicitadas con las modalidades multijugador presentes en el minijuego del Pong.

[d] Ha fallado la autenticación en el portal mediante la subida de fotografías que contienen el código QR asociado al usuario. Por algún motivo, la biblioteca *jsqrcode* no ha sido capaz de decodificarlas.

Tabla 5.1: Tabla que muestra los grados de compatibilidad sobre distintos navegadores (columnas) y entornos de ejecución (filas).

5.2 Cámaras digitales y tiempos de respuesta

En el contexto de una aplicación desarrollada como la presentada, es normal estar sujetos a restricciones de tiempo real. Pese a que la inicialización de cada cámara es una acción que se produce una vez en el transcurso de la aplicación, es interesante conocer el tiempo que puede llegar a consumir dicha tarea para conocer si hay diferencias claras

entre cámaras con características y complejidad distintas, y orientar a los futuros desarrolladores de páginas web que hagan uso de la cámara y las WebAPI asociadas.

Estas mediciones se han realizado en todos los casos con un uso reducido de recursos del computador, sin detallarse aquí. Los tiempos de inicialización de varias cámaras se han medido delimitados por los siguientes instantes y código asociado:

Tiempo inicial. El primer instante se ha registrado dentro de la función asíncrona de respuesta o *callback* asociado al éxito en la solicitud de acceso a la cámara, realizada mediante la API `getUserMedia` [7], y justo antes de la llamada necesaria para solicitar inicializar la cámara, `play`. Ver figura 5.1.

Tiempo final. El instante final sucede tan pronto como un estado interno del elemento de vídeo en cuestión se marca como que contiene suficientes datos para renderizar la primera imagen. Este estado está presente en la API de los elementos de vídeo en HTML5, concretamente en el atributo `readyState`, y cuyo estado a comprobar para detectar lo antes posible la inicialización completa del vídeo es `HAVE_ENOUGH_DATA`[1]. La comprobación se ha realizado de forma periódica con ayuda de la API `requestAnimationFrame`, dentro de la misma función que copia la captura de la cámara desde elemento de vídeo asociado hasta un Canvas oculto e inicia el escaneo de códigos QR desde el mismo Canvas. Referirse a la figura 5.2.

```
1 $('#camera1_video')[0].onplay = function() {
2   this.streamInitMsTime = Date.now();
3
4   // Código no relacionado oculto, por claridad
5 };
```

Figura 5.1: Extracto del código que guarda el instante de tiempo (en milisegundos) del inicio del proceso de inicialización de la cámara.

5.3 Pruebas sobre decodificación de los códigos

En esta sección se desarrollan una serie de pruebas realizadas sobre el proceso de autenticación, concretamente mediante el mecanismo de subida de fotografías que contienen un código de autenticación para la aplicación web. Las pruebas se han realizado tanto con códigos de barras tradicionales de una dimensión como con códigos QR. Se recuerda que la propia decodificación de los códigos de barras tradicionales se ha delegado en una biblioteca de código abierto, JOB (JavaScript Only Barcode Scanner), mientras que la decodificación de códigos QR se apoya en otra biblioteca de código abierto, *jsqrcode*. También se recuerda que el contenido de los códigos en cuestión son en todo caso las credenciales segurizadas de un usuario de la aplicación.

En primer lugar, se procede a describir la prueba realizada sobre un Code-39. Se ha generado un código de autenticación con la variante de ASCII completo de este tipo, necesaria para soportar todos los caracteres incluidos en las credenciales. Una vez subida su fotografía correspondiente al formulario de autenticación, se ha experimentado que la

[1] Existe al menos otra posibilidad, menos precisa, de determinar la completitud del proceso de inicialización de la imagen de la cámara. La marca temporal de finalización se podría realizar, tras haber registrado un *callback* del evento `timeupdate` presente en la API del vídeo, justo en su primera activación. Sin embargo, este evento periódico ha sido experimentado que se activa al menos cada 250 ms., incluyendo la primera vez, haciendo así algo más imprecisa la medición.

```
function captureToCanvas_ScanQR() {
  // Código no relacionado oculto, por claridad

  if (video.readyState !== video.HAVE_ENOUGH_DATA) {
    // Image is not ready yet. Allow rAF, but filter out Video.
    shouldRAF = true;
    return false;
  }

  if (!$video[0].isCaptureInitializationHandled) {
    // Initialize an heuristical timeout to catch camera disconnects
    delayCanvasCopyTimeout($video[0]);

    $video[0].isCaptureInitializationHandled = true;
    console.log("Camera image on video element with id: '"
    + $video.prop('id') + "' took " + (now - $video[0].streamInitMsTime)
    + "ms to initialize");
  }

  // Código no relacionado oculto, por claridad
}
```

Figura 5.2: Extracto del código que comprueba la finalización de la inicialización de la cámara e imprime a la salida estándar el tiempo total (en milisegundos) del proceso.

biblioteca JOB no ha sido capaz de decodificar tal código con la longitud de caracteres asociada. La imagen exacta sometida a esta prueba se puede observar en la figura 5.3.

Figura 5.3: Imagen del código tipo Code-39 (ASCII completo) probado en la decodificación vía subida de fotografía de autenticación en la aplicación web.

En segundo lugar, se procede a describir la prueba realizada empleando ahora un código tradicional Code-128. Este se trata de una versión más reciente respecto al Code-39, con un conjunto más amplio de caracteres disponibles, mayor densidad de datos y por tanto generalmente recomendada para reemplazar al Code-39. En este caso la biblioteca JOB lo ha decodificado con éxito; con ayuda de la medición temporal programada en la aplicación web, se muestra por la salida estándar que el proceso ha tardado **1049 ms** (única vez). La imagen exacta sometida a esta prueba se puede observar en la figura 5.5.

En tercer y último lugar, se procede a la prueba realizada sobre un código QR. Los códigos QR se han introducido en el capítulo 2 y por tanto se asume el conocimiento de sus capacidades teóricas de su estándar. Tal como ha devuelto la aplicación por pantalla, el proceso de decodificación ha tardado en este caso **48 ms** (única vez). La imagen concreta sujeta a la prueba se ilustra en la figura 5.4.

Como conclusión a esta serie de pruebas, se puede garantizar con cierta seguridad que los códigos QR superan con creces a los códigos de barras tradicionales de una dimensión en cuanto a la decodificación de los datos, y resultan más adecuados para su uso estando sujetos a restricciones de tiempo real como la que se da en la capa del minijuego Pong de ejemplo en la aplicación, si bien no se requiere tal exigencia en el tiempo de respuesta en el proceso de inicio de sesión automático en la capa de autenticación.

No obstante, es importante destacar una diferencia en las bibliotecas utilizadas. La biblioteca JOB, a diferencia de *jsqrcode*, crea y utiliza instancias de WebWorkers introdu-

Modelo de cámara	Resolución (píxeles)[a]	Tiempos de inicialización (ms)	Tiempo medio de inicialización (ms)
Trust WB 1400-T	355x288 = 102.240	251	332,4
		258	
		286	
		312	
		349	
		393	
		282	
		488	
		387	
		318	
Hercules Dualpix Infinite	2M (solo video)	1691	1836,1
		1785	
		1878	
		1779	
		1881	
		1775	
		1964	
		1778	
		1767	
		2063	

[a]Según especificaciones del proveedor

Tabla 5.2: Tabla que muestra información básica de las cámaras probadas en la aplicación y sus tiempos de inicialización experimentados.

cidas en la versión 5 de HTML para decodificar los datos. Los WebWorkers utilizados por la biblioteca JOB permiten ejecutar el proceso en un hilo independiente, permitiendo al flujo de código invocante no bloquearse durante el proceso. La biblioteca *jsqrcode* no necesita emplear hilos dado su reducido tiempo de respuesta (**48 ms** empíricamente probados en la prueba concreta).

Así, es probable que la biblioteca JOB plantease el uso de WebWorkers sabiendo que el proceso de decodificación tarda cierto tiempo (**1049 ms** empíricamente probados en la prueba concreta). El potencial asíncrono de esta biblioteca implica que, suponiendo que la propia decodificación no estuviera sometida a restricciones de tiempo real, tal como en la aplicación web desarrollada, una biblioteca como JOB podría ser por ende adecuada para aplicaciones de tiempo real con restricciones temporales exclusivamente aplicadas sobre otros elementos u objetos.

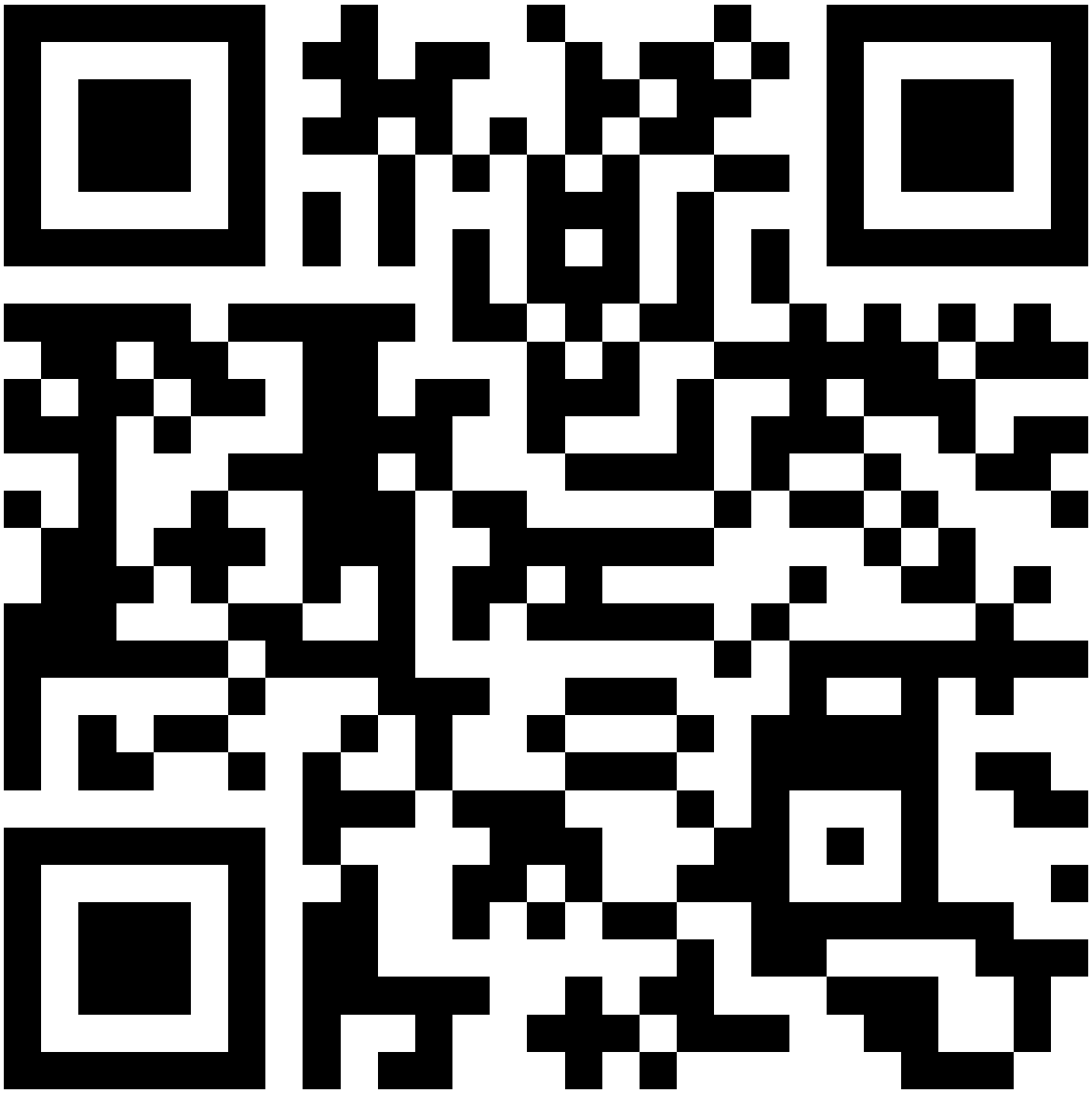

Figura 5.4: Imagen del código QR probado en la decodificación vía subida de fotografía de autenticación en la aplicación web.

Figura 5.5: Imagen del código tipo Code-128 probado en la decodificación vía subida de fotografía de autenticación en la aplicación web.

CAPÍTULO 6
Conclusiones y futuras propuestas

El tipo de aplicación desarrollada da pie a añadir más funcionalidades por el camino del WebRTC y otras WebAPI. También cabe tener en cuenta que en ningún momento del desarrollo se ha hecho uso de funcionalidades *backend* sobre servidor web, por lo que las posibilidades tras considerar el uso de un servidor web aumentan aún más. Así, se han concebido durante el desarrollo y tras el mismo una serie de ideas en esta línea, expuestas a continuación.

En primer lugar, se podría integrar de forma completa ambas capas de autenticación y generación de la imagen del código QR del usuario con algún *software* de código abierto en la línea de portales web, foros o similares que contengan un área identificada. De esta forma, se podría poner en práctica el mecanismo de autenticación flexible donde los usuarios puedan identificarse mediante código QR y cámara en alguna comunidad con suficiente cantidad de usuarios.

En segundo lugar, se ha considerado la idea de ofrecer la posibilidad de una modalidad multijugador en línea en el minijuego del Pong. Esto abre una línea de paradigmas cuanto menos triviales en torno a videojuegos en red, que no se han contemplado en el alcance de este proyecto. Sin embargo, las WebAPI del WebRTC permiten también comunicar el flujo de datos de la cámara entre pares a través de Internet, por tanto, si se determinara que esa funcionalidad es necesaria, entre otras, se estima que este objetivo se podría conseguir.

En tercer lugar, se ha pensado en una característica concreta en cuanto a las cámaras. Se trata de implementar una gestión completa para dar respuesta a la conexión, desconexión, interrupción o fallo de las cámaras solicitadas por la aplicación, con el objeto de facilitar la reincorporación de las cámaras fallidas durante su ejecución o desde que el usuario denegó el permiso, contando con interfaces gráficas adecuadas.

Se recuerda, según lo descrito en la sección 4.7, que la aplicación web actualmente ya cumple con una gestión mínima en el ámbito de este objetivo. Con la ayuda de una WebAPI correspondiente y, cuando no es posible, mediante una heurística temporal (*timeout*), actúa como respuesta a la desconexión de las cámaras para optimizar (anular) las operaciones de copia sobre elementos Canvas que se realizan entre fotogramas. No obstante, se estima que una gestión completa puede conllevar cierta complejidad, pero es asequible.

En cuarto y último lugar, se plantea aumentar la funcionalidad gráfica en el minijuego del Pong en relación con la Realidad Aumentada, campo que está en auge. Solo como ejemplo, se podrían renderizar modelos predefinidos tridimensionales sobre los avatares de los jugadores consistiendo en cubos que muestren la imagen de su código QR

en cada lado, o directamente generar estos modelos en tiempo real a partir del propio código detectado en la cámara y que reflejasen aspectos espaciales tales como la misma perspectiva captada. Existen multitud de API en la web para realizar aplicaciones de Realidad Aumentada [29].

A lo largo de este trabajo se ha cumplido el objetivo general de investigar las posibilidades de interactividad que ofrecen HTML5 y sus API en JavaScript. En concreto, el desarrollo de un mecanismo moderno de autenticación y de control sobre un minijuego de ejemplo, todo ello haciendo uso de una interfaz tan común como la cámara en lugar de los periféricos tradicionales.

Por otra parte, previamente al inicio de este proyecto se contaban con conocimientos técnicos necesarios en cuanto a tecnologías web se refiere: HTML, CSS3 y JavaScript. Durante la carrera no se han abordado las innumerables posibilidades que ofrece el reciente HTML5, puesto que no hay tiempo para profundizar en tecnologías específicas. Sin embargo, gracias a este proyecto se ha descubierto que existe todo un mundo de WebAPIs, algunas en proceso de estandarización, y en concreto las que permiten el manejo de imagen, vídeo y periféricos con un alto grado de control sobre sus características.

Bibliografía

[1] S. J. Vaughan-Nichols. Will HTML 5 Restandardize the Web? *Computer, vol. 43, n.º 4, pp. 13-15, abr. 2010*.

[2] Wikipedia, la enciclopedia libre. Pong. Consultado en `https://es.wikipedia.org/wiki/Pong`.

[3] Gimeno Balaguer, Adrián. Minijuego del Pong (versión inicial). Accesible en `https://juegopong.azurewebsites.net`.

[4] Wikipedia, la enciclopedia libre. Cryptographic hash function. Consultado en `https://en.wikipedia.org/wiki/Cryptographic_hash_function`.

[5] Wikipedia, la enciclopedia libre. Realidad aumentada. Consultado en `https://es.wikipedia.org/wiki/Realidad_aumentada`.

[6] MDN Web Docs. WebRTC API. Consultado en `https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API`.

[7] Documentación web de MDN. MediaDevices.getUserMedia(). Consultado en `https://developer.mozilla.org/es/docs/Web/API/MediaDevices/getUserMedia`.

[8] W3C Working Group. Media Capture and Streams. Consultado en `https://www.w3.org/TR/mediacapture-streams`.

[9] Walsh, David. Camera and Video Control with HTML5 Example. Consultado en `https://davidwalsh.name/demo/camera.php`.

[10] Walsh, David. Camera and Video Control with HTML5. Cosultado en `file:///D:/Documents/Zotero/storage/DE8GGX9Z/browser-camera.html`.

[11] MDN Web Docs. Manipulating video using canvas. Consultado en `https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Manipulating_video_using_canvas`.

[12] Atkins Jnr, Tab. video + canvas = magic. Consultado en `http://html5doctor.com/video-canvas-magic`.

[13] Levski, Yariv. Markerless vs. Marker Based Augmented Reality. Consultado en `https://appreal-vr.com/blog/markerless-vs-marker-based-augmented-reality`.

[14] Kudan. Augmented Reality Fundamentals: Markers. Consultado en `https://www.kudan.eu/kudan-news/augmented-reality-fundamentals-markers`.

[15] Etienne, A. WebAR Playground: AR in a Few Clicks. Consultado en https://medium.com/arjs/webar-playground-ar-in-a-few-clicks-67a08cfb1534.

[16] Ramsden, Andrew. The use of QR codes in Education. *(c) University of Bath*, 2008.

[17] Wave, Denso. What is a QR Code? Consultado en http://qrcode.com.

[18] W3C Working Group. Timing control for script-based animations. Consultado en https://www.w3.org/TR/animation-timing/#requestAnimationFrame.

[19] J. F.- js.foundation. jQuery. Consultado en https://jquery.com.

[20] M. O. contributors Jacob Thornton, and Bootstrap. Bootstrap. Consultado en https://getbootstrap.com.

[21] L. Laszlo. *jsqrcode*. Consultado en https://github.com/LazarSoft/jsqrcode.

[22] C. Schmich. HTML5 QR code scanner using your webcam. Consultado en https://github.com/schmich/instascan.

[23] Google Developers. Portal de Google Charts. Consultado en https://developers.google.com/chart.

[24] Wikipedia, la enciclopedia libre. MD5. Consultado en https://es.wikipedia.org/wiki/MD5.

[25] Taccetti, Antonio. *HTML5 Canvas in Real Time*. ISBN 978-88-260-4404-0, 2017.

[26] Lerner, Reuven M. At the forge: communication in HTML5. *Linux Journal*, 2011.

[27] R. Silveira. *Multiplayer Game Development with HTML5*. Packt Publishing Ltd, 2015.

[28] Documentación web de MDN. Control de acceso HTTP (CORS). Consultado en https://developer.mozilla.org/es/docs/Web/HTTP/Access_control_CORS.

[29] T. M. and G. Garcia. Augmented Reality With HTML5: What Can Mobile Web Browsers Do? Consultado en https://marmelab.com/blog/2017/06/19/augmented-reality-html5.html.