UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Desarrollo e integración de aplicaciones en Facebook

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Andreu Climent Montell

Tutor: Manuel Agustí Melchor

2015-2016

“*Todos hacemos elecciones, pero al final,*

nuestras elecciones nos hacen a nosotros”

Andrew Ryan - Bioshock

Resumen

Hoy en día la mayoría de las personas utiliza la red social *Facebook* diariamente, por lo que se ha convertido en un medio de promoción de aplicaciones muy efectivo. Integrar nuestra aplicación en *Facebook* nos ayudará a posicionarla en los buscadores de la red dando a conocerla más rápidamente que si sólo utilizásemos técnicas de posicionamiento en los buscadores de la red (SEO).

En el presente proyecto hemos desarrollado una aplicación web con el objetivo de integrarla en *Facebook* y realizar un estudio de cómo influye esta red social a la hora de dar a conocer una aplicación al público, tomando la nuestra como caso experimental. Con esto, podemos deducir que el proyecto está dividido en dos partes claramente diferenciadas: el desarrollo de la aplicación y el estudio de su evolución.

A lo largo de esta memoria se describirán los diferentes elementos necesarios para llevar a cabo esta tarea: las diferentes tecnologías web que hemos utilizado (HTML5, CSS3, Createjs, JavaScript…), el proceso de adquisición de un dominio web, la instalación del servidor y del certificado SSL y, por último, cómo integrar la aplicación en *Facebook*.

Una vez tenemos la aplicación en marcha, hemos realizado un seguimiento de su evolución en la plataforma social mediante las herramientas que esta nos proporciona y, con los datos obtenidos, hemos hecho un estudio comparativo con otras aplicaciones existentes presentado nuestras conclusiones.

Palabras clave: Facebook, tecnologías web, aplicación, juego, integración, redes sociales.

Resum

Hui en dia la majoria de les persones utilitzen la xarxa social *Facebook* diàriament, per el que s'ha convertit en un mitjà de promoció d'aplicacions molt efectiu. Integrar la nostra aplicació al *Facebook* ens ajudarà a posicionar-la en els buscadors de la xarxa donant a conèixer-la més ràpidament que si únicament utilitzarem tècniques de posicionament als buscadors de la xarxa (SEO).

Al present projecte hem desenvolupat una aplicació web amb l'objectiu d'integrar-la al *Facebook* i realitzar un estudi de com influeix aquesta xarxa social a l'hora de donar a conèixer una aplicació al públic, agafant la nostra com a cas experimental. Amb tot això, podem deduir que el projecte està dividit en dos parts clarament diferenciades: el desenvolupament de l'aplicació i l'estudi de la seua evolució.

Al llarg d'aquesta memòria es descriuran els diferents elements necessaris per a portar a terme aquesta tasca: les diferents tecnologies web que hem emprat (HTML5, CSS3, Createjs, JavaScript…), el procés d'adquisició d'un domini web, d'instal·lació del servidor i del certificat SSL i, per últim, com integrar l'aplicació al *Facebook*.

Una vegada hem posat en marxa l'aplicació, hem realitzat un seguiment de la seua evolució a la plataforma social mitjançant les eines que aquesta ens proporciona i, amb les dades obtingudes, hem realitzat un estudi comparatiu amb altres aplicacions existents presentant les nostres conclusions.

Paraules clau: Facebook, tecnologies web, aplicació, joc, integració, xarxes socials.

Abstract

Today most people use the social network *Facebook* daily, so it has become a highly effective channel to promoting applications. Integrating our application on Facebook will help us to locate it in the web search engines giving faster results than if we only use search engines optimization techniques (SEO).

In the present project we have developed a web application in order to integrate it into Facebook and make a study on how this social network influence on advertise an application to the public, taking ours as an experimental case. With this, we can deduce that the project is divided into two distinct parts: the development of the application and the study of its evolution.

Throughout this paper it will be described the different needed elements to do this task: the different web technologies we have used (HTML5, CSS3, CreateJS, JavaScript ...), the process of acquiring a web domain, server and SSL certificate installation and, finally, how to integrate it in the Facebook application.

Once we have the application running, we have been monitoring its evolution in the social platform using the tools that it provides to us. With the data obtained, we made a comparative study with other existing applications and presented our findings.

Keywords: Facebook, web technologies, application, game, integration, social networks.

Índice

Índice de Figuras

1. Introducción y justificación

Hoy en día, con el auge de las nuevas tecnologías de la información y comunicación, la interacción de las personas en las redes sociales se ha disparado y con esto, la demanda de más contenidos. Según el estudio que expone Schou (Schou, 2015), el IAB (*Interactive Advertising Bureau*) recoge en sus datos que hay 14 millones de usuarios de las redes sociales solo en España, donde el 96% de los encuestados menciona a *Facebook*, dándole el título de la más utilizada por excelencia. Si además tenemos en cuenta que *Facebook* es una red social que va creciendo en un 15% anualmente, según el estudio que expone Zephoria Digital Marketing (Zephoria, 2016), no es mala idea presentar una aplicación en este medio, pues va a ser visto por mucha gente.

Aprovechando este aumento, ya hace unos años que las grandes compañías de desarrollo tecnológico utilizan este medio de comunicación de masas como una lanzadera para sus aplicaciones. Integrar una aplicación web en una red social nos permitirá, entre otros beneficios, reducir el tiempo que necesita un usuario a la hora de registrarse, los costes de desarrollo y promoción de dicha aplicación, atraer más fácilmente a nuevos usuarios e interactuar de forma más próxima con estos para conocer lo que quieren, necesitan y demandan (Facchin, 2015), (Hernández, 2013).

Aunque el reducir el tiempo de registro en la aplicación es una gran ventaja, el principal beneficio, y motivo de elección de *Facebook* como promotora de la aplicación, es el que se obtiene con la utilización de la API de la red social para implementar métodos por los que el propio usuario será el que publicite y la dé a conocer entre las personas de su entorno. La misma red social exhibe las facilidades y formas de promocionar un proyecto a través de sus servicios (*Facebook, 2016*). Así pues, con solo pulsar un par de botones, se puede disfrutar de la aplicación sin tener que preocuparse de ingresar los datos personales y se pueden compartir los logros obtenidos sin dificultades añadidas. Esta publicidad gratuita aumenta la base de usuarios muy rápidamente, además de permitirle al desarrollador conocer qué tipo de aplicación está más demandada en ese momento.

Según mi experiencia en la empresa Trading People S.L., una *StartUp* del sector tecnológico y financiero, el lanzamiento de un juego en *Facebook* propició que la compañía obtuviera un incremento de más de 3.000 miembros registrados en sus bases de datos en tan sólo un mes, tal y como se puede observar en la figura 1. Con este hecho en mente, nació la idea de hacer el Trabajo de Final de Grado centrado en realizar un estudio del impacto que puede tener *Facebook* en el lanzamiento y evolución de una aplicación web.

Figura 1: *Evolución de una aplicación del 15 de Agosto al 15 de Septiembre del 2015.*

2. Objetivos y Especificaciones

Cuando surgió la idea de integrar una aplicación en la red social *Facebook* como proyecto de final de grado, lo primero que hicimos fue plantearse por qué esta plataforma, qué esperamos obtener y qué nos aportará el proyecto. Al encontrar la respuesta a esas preguntas nos hemos marcado una serie de objetivos y especificaciones que pretendemos cumplir al desarrollar y llevar a cabo la idea.

2.1 Objetivos del proyecto

A continuación presentaremos los objetivos, tanto generales como específicos, que marcarán nuestra hoja de ruta a través del trabajo de final de grado con el fin de obtener los mejores resultados en nuestra investigación.

2.1.1 Generales

Dado que pretendemos desarrollar una aplicación con distintas tecnologías web e integrarla en la red social *Facebook*, los objetivos generales que nos hemos planteado son:

- Afianzar los conocimientos sobre lenguajes y metodologías de desarrollo de aplicaciones web.
- Aprender a manejar las distintas herramientas que proporciona *Facebook* para desarrolladores de aplicaciones.
- Mejorar nuestras habilidades de gestión de sistemas informáticos tales como la instalación de servidores web, la instalación de certificados de seguridad SSL y la creación de un dominio propio en la red.

2.1.2 Específicos

En cuanto a los objetivos específicos que pretendemos alcanzar para considerar que el trabajo y el desarrollo de la aplicación son un éxito son:

- Hacer funcionar un servidor web *Apache* con un certificado de seguridad SSL.
- Crear un dominio web propio de la aplicación.
- Implementar completamente la aplicación con tecnologías web y que sea funcional.
- Adaptar e integrar la aplicación en *Facebook* para que aparezca en la plataforma de juegos de la red social y así distribuirla al mayor número de personas posibles.
- Realizar un estudio para valorar la influencia de integrar en *Facebook* nuestra aplicación.

2.2 Especificaciones de la aplicación

Tal y como nos describe Letelier (Letelier, 2003), la primera fase del proyecto es definir las especificaciones de nuestro producto, en nuestro caso, vamos a desarrollar un mini juego para *Facebook* sencillo con el que cualquier persona, sin importar su edad, se pueda divertir y que, además, pueda compartir sus logros con sus amigos de la red.

Dada su sencillez, debemos diseñar una interfaz simple e intuitiva, con pocas funcionalidades extra dando lugar a que el usuario se aburra fácilmente y nos cueste mucho más que comparta sus logros en la red.

Al tratarse nuestro producto de una aplicación web integrada en *Facebook*, vamos a utilizar los lenguajes de programación más utilizados, HTML, CSS y JavaScript, para implementarla. Estos lenguajes se caracterizan por interpretarse y ejecutarse en el navegador del cliente quitando carga de trabajo al servidor permitiéndonos utilizar una máquina más económica, con menores prestaciones, a la hora de entregar la aplicación a varios clientes simultáneamente. Aunque no todo son ventajas puesto que el que sea el cliente el que ejecuta toda la aplicación provoca que no podamos almacenar ningún tipo de información en el servidor y puede darse el caso de que la máquina del usuario no sea capaz de interpretar parte del código dejando al usuario sin poder jugar.

En cuanto a su funcionalidad, tenemos claro que el mini juego será del mismo estilo que el mini juego del dinosaurio de *Google* (véase la figura cinco) por lo que, constará únicamente de un hombre corriendo y saltando para evitar colisionar contra unas elevaciones en el terreno y así obtener una puntuación. Este tipo de mini juego es relativamente sencillo de programar ya que lo más complejo que nos vamos a encontrar va a ser el algoritmo que calcule la física y las colisiones con el terreno. Por el contrario como la mecánica es tan sencilla, nos costará más plantear un reto a los usuarios de la aplicación creando a posibilidad de que no captemos el suficiente interés en los usuarios potenciales.

Figura 2: *Mini juego del dinosaurio de Google Chrome.*

Como hemos dicho, el usuario conseguirá una puntuación al avanzar en el juego posibilitándonos el realizar una clasificación entre él y sus amigos y además compartirla en la red social. Para poder hacerla el usuario debe darnos permiso para acceder a su lista de amigos de *Facebook* y poder invitarlos a jugar, hecho que será posible gracias a la API Graph de la red (*Facebook*, 2016). Si el usuario no nos permite acceder a la lista de sus amigos perderemos la oportunidad de generar una clasificación, además de no poder invitar a sus amigos limitando el alcance de la difusión de la aplicación.

3. Desarrollo e integración de la aplicación

En este apartado de la memoria vamos a hacer una breve explicación de las distintas tecnologías web que hemos utilizado durante el desarrollo e integración de la aplicación en *Facebook*, además del propio proceso que hemos seguido para implementarla e incorporarla en la red social.

3.1 Tecnologías web

Cuando decimos que estamos navegando en la red o buscando alguna cosa en Internet lo que estamos haciendo realmente es interconectando nuestro terminal con otro dispositivo, ya sea un ordenador personal, un teléfono móvil, un servidor, etc., con el fin de intercambiar información. Los mecanismos que hacen posible este intercambio son las tecnologías web. Tal y como nos describen en el sitio reference.com (Reference*, 2004) y en Red de Desarrolladores de Mozilla (MDN, 2015), existen diferentes tipos de tecnologías web y nosotros procederemos a explicar las que hemos utilizado.

3.1.1 HTML y CSS

El estándar HTML, siglas del inglés *Hyper Text Markup Language*, tal y como definen en un artículo de la página Acerca de HTML (Acerca de HTML, 2016), es el lenguaje de marcado con el que podemos construir una página web. Éste define una estructura básica y un código para la definición del contenido de una página web como texto, imágenes, vídeos o audio entre otros. Existen diferentes versiones del estándar y, en nuestro caso, utilizaremos la sexta versión, HTML5.

En el lenguaje HTML, cuando añadimos un elemento externo a la página que estamos desarrollando, por ejemplo una imagen, esta no se incrusta directamente en el código, si no que se hace una referencia a la ubicación de la imagen mediante texto. De este modo, la página web solo contendrá texto y recaerá en el navegador que estemos utilizando el interpretar el código y unir todos los elementos para mostrarnos la imagen en la página. En el caso de todo el contenido de la página web que nos muestra el navegador al interpretar el código HTML, aparecerá sin ningún formato ni diseño.

Para que nosotros podamos darle la apariencia que deseemos a la página web utilizaremos las hojas de estilo o CSS, siglas del inglés *cascading style sheets*. Las CSS, como podemos leer en Introducción a CSS (Eguiluz, 2016), son un lenguaje de programación creado para controlar el aspecto o presentación de las páginas web definidas con un lenguaje estructurado como el HTML. Con las CSS conseguimos separar los contenidos de la web de su presentación de la mejor forma, siendo imprescindible para crear páginas web complejas.

En nuestro caso de estudio, además de CSS básico, hemos hecho uso de la librería Bootstrap, que combina el uso de CSS y JavaScript. Gracias a esta, conseguimos estructurar y diseñar la apariencia del mini juego más rápidamente. Además, con las clases predefinidas en la librería, hemos hecho que la apariencia de nuestra aplicación se adapte a cualquier tamaño de pantalla haciendo más fácil su portabilidad entre dispositivos.

3.1.2 JavaScript y jQuery

En la red de desarrolladores de Mozilla (Mozilla Developer Network, 2016) definen el JavaScript como un lenguaje de programación ligero e interpretado, dinámico y orientado a objetos. Principalmente se utiliza para implementar páginas web dinámicas aunque también se usa en entornos del lado del servidor como *node.js* o *Apache CouchDB*. El lenguaje JavaScript es la pieza principal de nuestra aplicación ya que, sin él, no podríamos generar todas las animaciones y efectos que proporcionan la experiencia deseada en nuestra aplicación.

Aunque utilizamos el lenguaje JavaScript en nuestra aplicación, no utilizaremos todas las funciones que vienen definidas de forma nativa si no que, para nuestra comodidad, utilizaremos un conjunto de librerías con las que podremos seleccionar, manipular y trabajar más fácilmente, una de estas librerías es jQuery (jQuery, 2016).

La librería jQuery está formada por una serie de funciones y métodos de JavaScript con las que podremos desarrollar aplicaciones enriquecidas del lado del cliente compatibles con todos los navegadores.

3.1.3 CreateJS y EaselJS

CreateJS, de la web de nombre homónimo (Skinner, 2016), es una interfaz de edición web de software libre que nos provee de las librerías modulares y herramientas que nos proporcionan todo lo necesario para poder crear contenido interactivo mediante JavaScript y HTML5.

Dentro de la interfaz de Create.js encontramos el módulo de la librería JavaScript EaselJS, extraído del sitio comentado anteriormente (Skinner, 2016). Esta nos provee de las funciones, métodos y clases necesarias para interactuar con el elemento *canvas* del lenguaje HTML5, haciendo más fácil el trabajar con elementos gráficos en dos dimensiones.

En el desarrollo de nuestra aplicación utilizaremos la librería EaselJS para crear dentro del *canvas* la animación con la que mostraremos en el mini juego a un pequeño hombre corriendo y saltando a través de un terreno cercano al océano, tal como se observa en la figura 3.

Figura 3: *Animación del mini juego Lost Runner*.

3.1.4 Facebook API Graph

Uno de los objetivos de nuestro trabajo es integrar una aplicación en *Facebook* y así poder introducir o consultar información de los usuarios en la red social. Para poder realizar este intercambio, *Facebook* nos proporciona a los desarrolladores una interfaz de programación de aplicaciones, abreviada como API proveniente del inglés *Application Programing Interface*, la API Graph versión 2.6 (Facebook developers, 2016).

La API Graph de *Facebook* está basada en HTML de nivel bajo, la cual se puede utilizar para consultar los datos del perfil de usuario, publicar puntuaciones obtenidas en su muro, invitar a jugar a sus amigos y muchas otras tareas que se pueden requerir en la aplicación. El nombre de la API vendría de la idea de "gráfica social", la forma de representación de la información en *Facebook*. Dicha representación tendría tres elementos:

- **Nodos**: los "objetos" que conforman la red, como por ejemplo un usuario, y que tienen un identificador único.
- **Perímetros**: son las conexiones entre los nodos, como la publicación de la puntuación del mini juego en el muro.
- **Campos**: son la información que obtenemos de los nodos, la foto del perfil, edad, email, etc.

La API Graph está basada en el protocolo HTTP, por lo que es compatible con cualquier lenguaje que tenga una biblioteca HTTP, como cURL o urllib, lo que significa que puedes utilizarla directamente en tu navegador. Por ejemplo, podemos solicitar a *Facebook* la foto del perfil del usuario para colocarla en nuestra aplicación, siendo esto el equivalente a hacer una petición GET a la API Graph como en la figura 4.

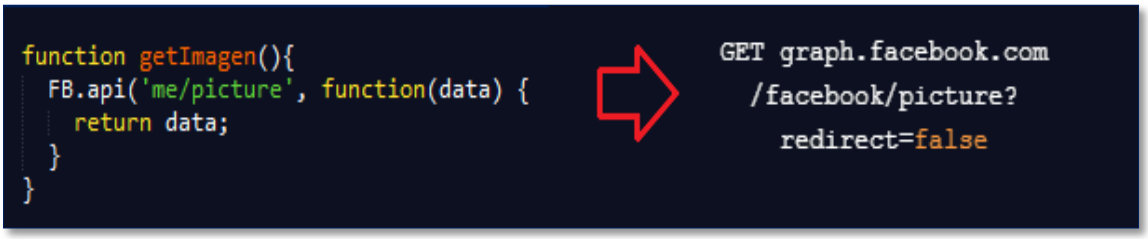

Figura 4: *Ejemplo de petición de la foto de perfil a Facebook.*

En nuestro caso de estudio, haremos varias peticiones a *Facebook* tanto para enviar como para solicitar información. Por ejemplo solicitaremos la información relativa al usuario (nombre, foto, lista de amigos y correo electrónico), para poder iniciar sesión en la aplicación y mostrar el mensaje de bienvenida, el avatar y la clasificación con sus amigos, y enviaremos una historia al muro de *Facebook* del usuario cuando este comparta la puntuación que acaba de obtener en una partida.

3.1.5 Dominio de internet

A grandes rasgos, un nombre de dominio o dominio es el nombre único y exclusivo que se le asigna a una página web en Internet. El dominio, por tanto, será el nombre con el que los usuarios van a buscar nuestra aplicación en la red.

A nivel técnico, un dominio es algo más complejo, ya que lo que tenemos realmente tras un nombre de dominio es una dirección que apuntará a un sistema de nombres de dominio o servidor DNS, siglas del inglés *Domain Name System*, que se encargará de redirigirnos a la dirección IP del ordenador en el que están alojados los archivos que conforman nuestra página web. Generalmente, los nombres de dominio constan de dos partes: nombre y extensión del dominio.

- El **nombre**: es la parte con la que realmente los usuarios identifican nuestra página web, por lo que debe ser fácil de recordar. En nuestro caso, hemos elegido utilizar el nombre del mini juego Lost Runner unido a las siglas de Trabajo de Fin de Grado, es decir, *lostrunnertfg*.
- La **extensión de dominio**: es la parte final, y no menos importante, ya que podemos tener varias para un mismo dominio. Con la extensión indicamos la naturaleza del sitio web al que apuntamos siendo las siguientes las más conocidas:
  - *.com* (uso comercial)
  - *.net* (servicios de internet)
  - *.edu* (instituciones académicas)
  - *.org* (asociaciones sin ánimo de lucro)
  - *.info* (páginas web de información)
  - Extensiones de localización geográfica: *.es* (España) o *.tk* (Tokelau)

3.1.6 Certificado SSL

Cuando creamos una página web o una aplicación deseamos que esta sea segura y confiable para el usuario y que así pueda ingresar sus datos personales sin miedo a que vulneren su integridad. Instalando un certificado SSL en nuestro servidor logramos brindar la seguridad que demanda el usuario de nuestra aplicación o página web.

Las siglas SSL vienen del inglés *Secure Socket Layer*, el cual es un protocolo de seguridad que permite mediante un cifrado que los datos introducidos por el usuario viajen de manera íntegra y segura desde que los introduce hasta que llegan al servidor donde se aloja la aplicación.

Un certificado SSL nos garantiza que los datos que transmitimos lleguen al servidor correcto. Esto es debido a que un servidor que corre SSL crea una vía de comunicación con un cifrado único para las sesiones privadas a través de la red y, por tanto, utilizará un par de claves, pública y privada, en la comunicación entre el cliente y el servidor para cifrar con la clave pública la información al salir del cliente y descifrarla al llegar al servidor con la clave privada.

3.1.7 Servidor web

Comúnmente nos referimos cómo servidor al dispositivo que gestiona la información del lado del servidor aunque, en realidad, es el programa que gestiona las conexiones, bidireccionales o unidireccionales y, de modo síncrono o asíncrono con el cliente, genera una respuesta en cualquier lenguaje o aplicación en el lado del cliente.

Un servidor web es el programa que responde a una petición de un cliente enviando los datos en forma de páginas web. Las comunicaciones, entre cliente y servidor se realizan a través del protocolo HTTP (véase la figura cuatro). La tarea del servidor web únicamente es la de responder a las peticiones HTTP que los clientes le envíen por lo que él solo enviará los datos sin interpretarlos, es decir, tras una petición, enviará los archivos de código, imágenes, etc., que el cliente precise para interpretar el código y mostrar la página web o aplicación.

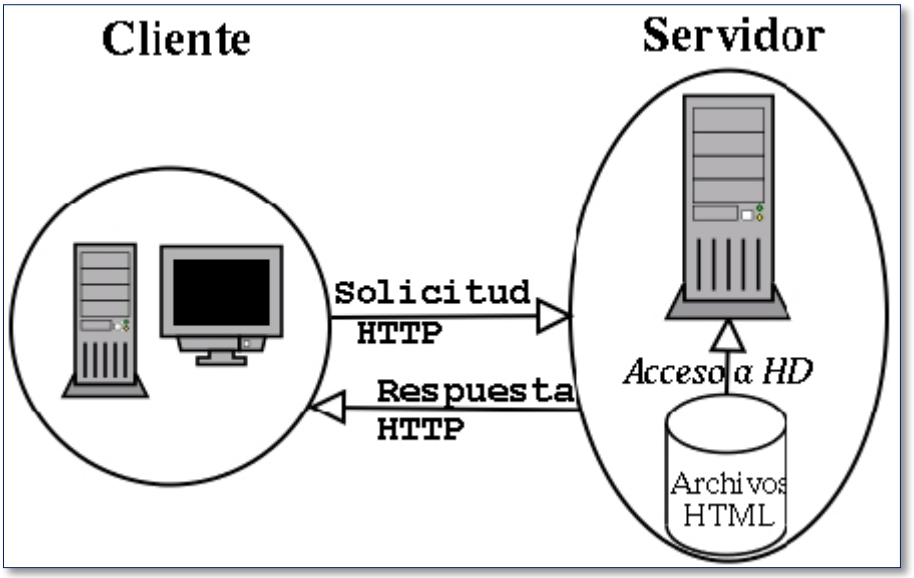

Figura 5: *Arquitectura del funcionamiento de un servidor web.*
Fuente: López (2001).

En la red podemos encontrar distintos tipos de servidores web, entre ellos, entre los más conocidos y utilizados se pueden citar: *Apache, Microsoft ISS, Sun Java System Web Server, Ngnix* y *Lighttp.* En nuestro caso vamos a utilizar un servidor *Apache 2.0.*

Un servidor web *Apache* se caracteriza por ser potente y flexible y puede funcionar en una amplia variedad de plataformas y entornos gracias a su diseño modular. Los servidores *Apache* son gratuitos y de código abierto.

3.2 Programación de la aplicación

En el momento en que decidimos que implementaríamos una aplicación web para utilizarla como objeto de estudio, empezó nuestro camino a través del proceso de desarrollo de software. En este apartado de la memoria vamos a explicar cómo hemos seguido dicho proceso.

3.2.1 Diseño y estructura del mini juego

Antes de empezar a programar cualquier aplicación es necesario conocer y planificar cómo será su interfaz y su estructura interna de archivos y carpetas. En el momento en que decidimos crear una aplicación sencilla, decidimos hacer un desarrollo centrado en el usuario, y, con ello en mente, empezamos a diseñar cómo sería la interfaz que vería el usuario.

El primer paso del diseño de la interfaz del usuario ha sido realizar unos bocetos (véanse las figuras A1 y A2 del anexo) con lo que nos guiarnos a la hora de implementar el programa. Tras el diseño preliminar hemos pensado que la interfaz del usuario constará de tres pantallas distintas:

- **Inicio**: esta pantalla será el índice de nuestra aplicación. En ella podremos ver el logotipo de nuestra aplicación y tres botones que nos darán el acceso al juego, al tutorial y a la clasificación.
- **Clasificación**: aquí encontraremos la clasificación que nos proporciona *Facebook* según las máximas puntuaciones enviadas por nosotros y nuestros amigos. Además, también incluiremos dos botones, el de invitar a nuestros amigos y el de regresar a la pantalla de inicio.
- **Juego**: vamos a dedicar una pantalla íntegra para el juego, así será mucho más fácil para el usuario jugar sin que tenga ninguna distracción. Para que el usuario pueda volver al inicio, reiniciar el juego o publicar en *Facebook* la puntuación obtenida, sacaremos un cuadro de diálogo con los tres botones necesarios para llevar a término dichas acciones.

Una vez hemos definido la interfaz del usuario tenemos que organizar los archivos y carpetas de imágenes, código y librerías. En nuestro caso, hemos optado por la estructura tradicional en el diseño web:

Incluiremos las carpetas **css**, **js** e **images** que contendrán:

- **css**: contendrá la hoja de estilo *estilo.css* además de la librería de diseño *bootstrap*.
- **js**: en esta carpeta almacenaremos todos los archivos de JavaScript, así como el directorio *libs* que contendrá las librerías easeljs y jQuery. Los JavaScript que incluiremos serán:
  - *App.js:* aquí irá el código principal para correr el mini juego.
  - *Cargador.js:* este archivo llevará el código necesario para simular una clase *cargador* que cargará las imágenes del juego.
  - *Facebook.js:* localizaremos en este archivo todas las interacciones con la API Graph de *Facebook.*
  - *Utils.js:* este archivo únicamente tendrá algunas funciones de utilidad para el desarrollo y funcionamiento de la aplicación.
- **images**: en este directorio almacenaremos todas las imágenes que formarán parte del mini juego.

Además de las carpetas tendremos los cuatro archivos de código HTML que formarán la aplicación:

- ***index.html****:* este archivo será la raíz de la aplicación por lo que contendrá el código necesario para mostrarnos la página de inicio.
- ***app.html****:* al separar el mini juego en otra pantalla, hemos decidido colocar aquí todo el código necesario para cargarlo y mostrarlo.
- ***error.html*** *y* ***policy.html****:* son dos archivos requeridos por *Facebook* para el funcionamiento de la aplicación. Como sus nombres indican uno saldrá ante cualquier error de carga del juego y el otro muestra las políticas de privacidad de la aplicación, en nuestro caso ambos son archivos muy sencillos sin apenas texto.

Podemos observar en la figura 6 como quedarán todos los archivos y directorios estructurados en nuestro proyecto.

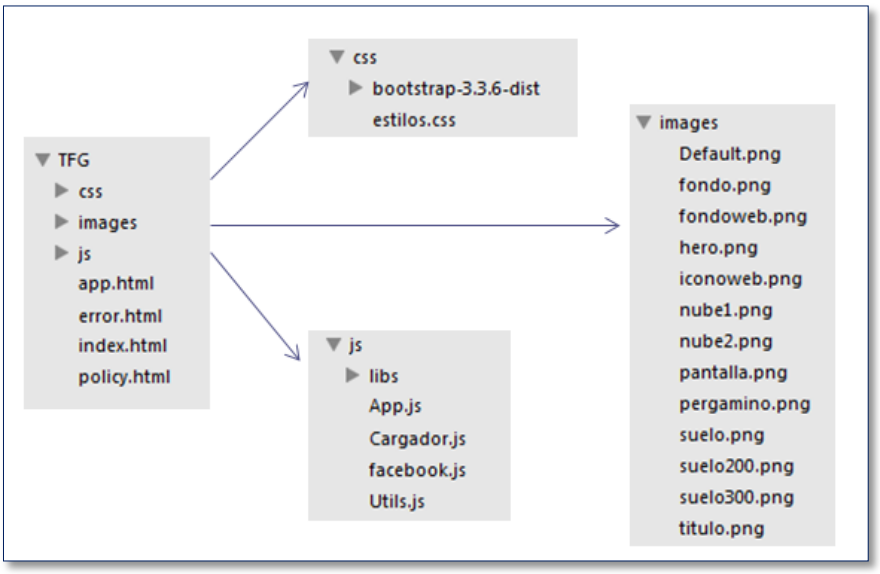

Figura 6: *Diagrama de la estructura interna de la aplicación.*

3.2.2 Cuerpo de la aplicación: HTML

En el punto anterior hemos descrito la estructura de archivos que va a seguir nuestra aplicación, de los que cuatro serán archivos HTML. Entre dichos archivos destacaremos dos, *app.html* e *index.html*, ya que ambos conformarán el cuerpo de la pantalla de juego y de la de inicio de nuestra aplicación.

Aunque los archivos *app.html* e *index.html* son los dos más importantes, no debemos olvidarnos de los archivos *policy.html* y *error.html*. Estos dos archivos los incluimos en la estructura de nuestro mini juego porque *Facebook* nos los requiere en la configuración de las aplicaciones *Canvas* en la red social.

Cada uno de los ficheros HTML de nuestra aplicación utiliza la estructura básica del estándar HTML5, tal y como podemos ver en la figura 7, compuesta por dos partes: El <*head*>, donde se añade toda la información que el navegador necesita pero que no se visualiza, y el <*body*>, que agrupa todo el contenido de la página y que el navegador sí que muestra.

```
<!doctype html>
<html>

  <head>
    <meta charset="utf-8"/>
    <title>Título de la web</title>
  </head>

  <body>
    Contenido de la web
  </body>

</html>
```

Figura 7: *Estructura básica de un documento HTML5.*

En los archivos *policy.html* y *error.html* no hemos agregado mucho contenido a la estructura básica ya que, únicamente nos interesa mostrar un mensaje informando al usuario de que la aplicación es una prueba o que ha ocurrido un error al ejecutarse la aplicación. Podemos ver el código completo de ambos archivos en las figuras A3 y A4 del anexo respectivamente.

Los documentos *index.html* y *app.html* son más complejos que los dos anteriores. Empezando por la cabecera, la etiqueta <*head*>, podemos encontrarnos con una etiqueta <*meta*>, con la que informamos del autor del proyecto al navegador, o las etiquetas <*link*> y <*script*> desde las que cargamos la hoja de estilo, las librerías y los archivos JavaScript en los que tenemos nuestras funciones .

La cabecera de los dos ficheros, aunque muy parecida, se distingue en el número de archivos JavaScript y librerías que cargamos, tal y como podemos apreciar en las figuras 8 y 9. En el *index.html*, al no necesitar arrancar el juego, no necesitaremos las librerías *easeljs*, ni tampoco las clases y funciones del juego *app.js* y *cargador.js*.

```
<head>

    <title>Lost Runner</title>

    <!-- Metas -->
    <meta charset="utf-8">
    <meta name="author" content="Andreu Climent Montell">

    <!-- Estilos -->
    <link rel="stylesheet" type="text/css" href="/css/estilos.css">
    <link rel="stylesheet" type="text/css" href="/css/bootstrap-3.3.6-dist/css/bootstrap.min.css">


    <!-- Librerias -->
    <script type="text/javascript" src="/js/libs/jquery-2.1.4.min.js"></script>
    <script type="text/javascript" src="/js/Utils.js"></script>
    <script type="text/javascript" src="/js/facebook.js"></script>


</head>
```

Figura 8: *Cabecera del documento index.html.*

```
    <!-- Librerias -->
    <script type="text/javascript" src="/js/libs/easeljs-0.8.2.min.js"></script>
    <script type="text/javascript" src="/js/libs/jquery-2.1.4.min.js"></script>

    <!-- Scripts -->
    <script type="text/javascript" src="/js/Utils.js"></script>
    <script type="text/javascript" src="/js/facebook.js"></script>
    <script type="text/javascript" src="/js/Cargador.js"></script>
    <script type="text/javascript" src="/js/App.js"></script><!-- Cargar el ultimo -->
```

Figura 9: *Llamadas a ficheros extra en la cabecera del documento app.html.*

Como hemos podido comprobar las cabeceras de ambos archivos son muy parecidas pero en lo referente a su contenido, son completamente diferentes. De los dos archivos el que menos código HTML contendrá en su etiqueta *<body>,* será el fichero *app.js.*

Si observamos detenidamente el código de la etiqueta *<body>* del archivo *app.js* en la figura 10, podremos diferenciar cuatro bloques (* *) distintos:

- ***joc:*** este bloque es en el que crearemos mediante JavaScript la animación del mini juego de correr como el del dinosaurio de *Google*.
- ***puntuacion***: tal y como el propio nombre indica, en este bloque iremos actualizando la puntuación que obtenga el usuario a medida que avanza en el juego.
- ***imguser***: en este bloque colocaremos la imagen del perfil del usuario que obtendremos mediante una llamada a la API Graph de *Facebook.*
- ***fin***: cuando finalice la partida mostraremos al usuario este bloque. En este diálogo le proporcionamos al usuario la puntuación total que ha obtenido y tres botones con las opciones de volver a jugar, compartir e ir al inicio (véase la figura A5 del anexo).

```
<body>
    <div class="container">
        <div id="joc"></div>
        <div id="puntuacion">0</div>

        <div id="imguser" style="display:none;">
            <img src="" onerror="$(this).attr('src','/images/Default.png')">
        </div>

        <div id="fin" class="col-sm-8 col-sm-push-2" style="display:none">
            <div class="center-block">
                <h1> Game Over </h1>
                <p>¡Enhorabuena! Tu puntuación es</p>
                <h2 id="punts"></h2>

                <div id="botones">
                    <button id="reset" class="btna">Jugar</button>
                    <button id="compartir" class="btna">Compartir en Facebook</button>
                    <button id="inicio" class="btna">Volver al inicio</button>
                </div>
            </div>
        </div>

    </div>
</body>
```

Figura 10: *Contenido del archivo app.html.*

El contenido del archivo *index.html* (véase la figura A6 del anexo) está dividido en varios bloques al igual que ocurre con el *app.html*. Estos bloques no estarán todos visibles para el usuario a la vez, mostraremos uno u otro dependiendo de la sección a la que se dirija el usuario, creando así la ilusión de que está navegando por distintos archivos. Por ejemplo cuando el usuario entra a la aplicación solo verá el bloque *btnMuro* de la figura 11.

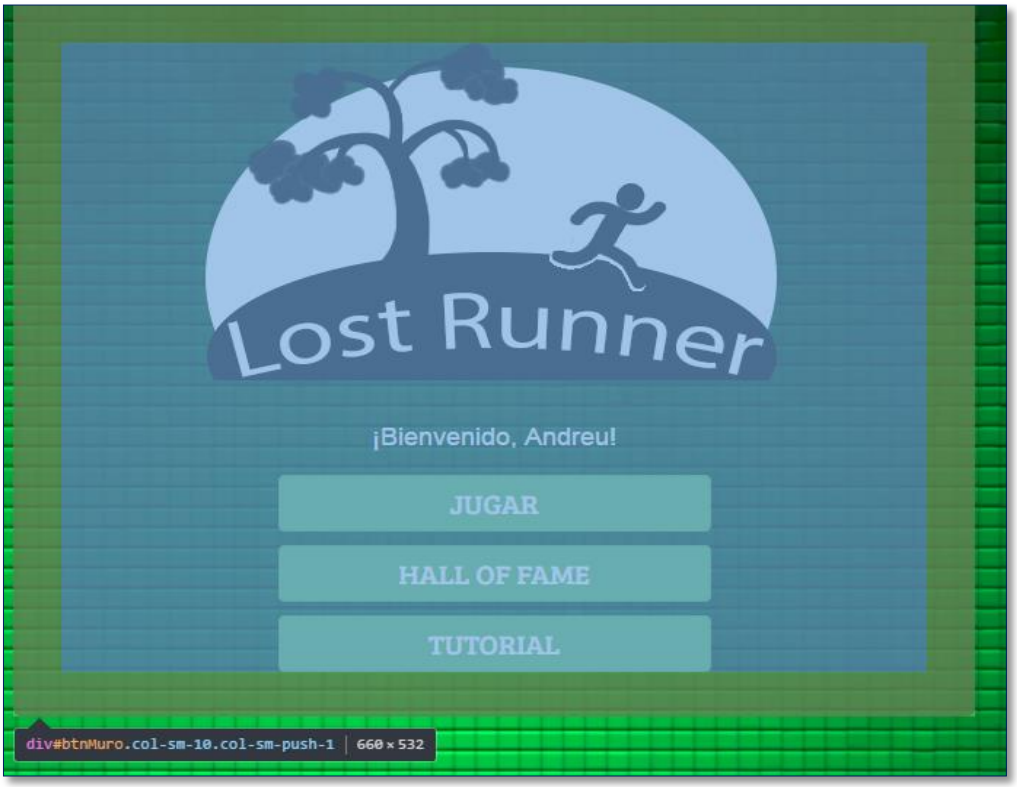

Figura 11: *Bloque btnMuro del fichero index.html.*

Encontraremos cuatro bloques distintos en la etiqueta <*body*> del documento *index.html*. A continuación vamos a explicarlos uno a uno:

- ***btnMuro***: este bloque está visible desde que iniciamos la aplicación. En el vemos el logotipo de la aplicación y los tres botones con los que podemos ir a jugar, ver nuestra clasificación o visitar el tutorial de la aplicación.
- ***rank***: en este bloque tenemos implementada la lista que, con los datos que obtengamos de *Facebook*, mostrará la clasificación con nuestros amigos. Además de la clasificación tendremos dos botones, uno de ellos hará visible el bloque *invitaciones* (véase la figura A7 del anexo).
- ***invitaciones***: este bloque contiene el formulario con la lista de todos los amigos del usuario. Desde aquí es donde el usuario seleccionará todos los amigos a los que quiera invitar a jugar (véase la figura A8 del anexo).
- ***tuto***: al pulsar sobre el botón del tutorial este bloque se hará visible y se esconderán el resto de bloques. En esta sección veremos un pequeño tutorial de la aplicación (véase la figura A9 del anexo).

3.2.3 Dando vida al mini juego: JavaScript

Después de haber construido el cuerpo de la aplicación procederos a darle vida al mini juego. Para llevar a cabo esta tarea utilizaremos de la interfaz de desarrollo *CreateJS* las librerías *EaselJS* y código JavaScript.

Siguiendo con la estructura que definíamos en el punto 3.2.2 se han creado los archivos *Utils.js, App.js y Cargador.js*. En este punto de la memoria vamos a describir cada uno de los archivos remarcando las funciones más importantes de cada uno de ellos.

El primer archivo del que vamos a hablar es *Utils.js*. En este documento se han implementado todas las funciones con las que realizamos alguna acción que afecte a la interacción del usuario con la aplicación como por ejemplo el pulsar un botón.

Una de las acciones más importantes que registramos en el fichero *Utils.js* es detectar el evento derivado de haber pulsado una tecla, botón o la pantalla del dispositivo desde el que entramos al mini juego. Podemos ver el código a continuación en la figura 12.

```
document.addEventListener('touchstart', function(e) {
  OnPulsar();
}, false);

document.onkeydown = OnPulsar;
document.onmousedown = OnPulsar;
```

Figura 12: *Captura del evento de pulsar sobre la pantalla, el teclado o el ratón.*

Al observar la figura anterior podemos ver como en cualquiera de las tres acciones que se pueden registrar el resultado será llamar a la función *OnPulsar*. Esta función lanza una llamada a uno de los métodos que se han implementado en el archivo *App.js* que provoca que el hombre de la animación del mini juego salte hacia arriba como se puede ver en las figura 13.

```
function OnPulsar(e)
{
  if(typeof(app) != "undefined"){
    app.saltar();
  }

}
```

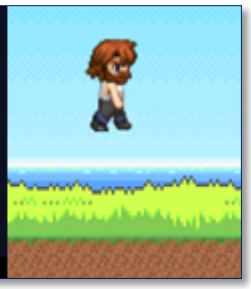

Figura 13: *Código y animación de la llamada a la función saltar al pulsar.*

En JavaScript no existen las clases tal y como las conocemos en el lenguaje *java*, pero podemos simularlas utilizando una función autoejecutable que inicie la ejecución de la función con el nombre de la clase. Podemos ver en la figura 14 un ejemplo de la estructura que seguimos para crear la clase *Cargador* en JavaScript.

```
(function(scope){
    function Cargador(){
        this.initialize();
    }

    var cargadas = Cargador.prototype;
    var totales = Cargador.prototype;

    Cargador.prototype.initialize = function(){
        cargadas = 0;
        totales = 0;
    }


    scope.Cargador = Cargador;

}(window));
```

Figura 14: *Código básico para crear la clase Cargador.*

Aprovechando esta forma de implementar el código en JavaScript hemos creado dos clases para implementar nuestro mini juego, la clase *App* y la clase *Cargador*, cada una de ellas codificada en un archivo diferente con el mismo nombre.

La clase *Cargador* de la aplicación (véase la figura A10 del anexo) tendrá el propósito de crear una etiqueta imagen en HTML a partir de la ruta de esta en el servidor. El proceso que seguimos es el siguiente:

1. Inicializamos la clase con las variables cargadas y totales a cero.
2. Realizamos una llamada a la función *loadImagenes* con un vector con las rutas de todas las imágenes que queremos cargar.
3. Desde *LoadImagenes* ejecutamos la función *cargaImagen* que creara la imagen y ejecutará la función *imagenCargada*.
4. Al ejecutar la función *imagenCargada* lo que hacemos es actualizar el contador de imágenes cargadas. Cuando este llegue a tener el mismo valor que el número de imágenes devolverá un *this.onComplete*, indicador de que se han cargado todas las imágenes.

Por si sola la clase *Cargador* no tiene ninguna utilidad ya que debemos pasarle una lista con las imágenes que queremos cargar desde otra función, en nuestro caso se hace desde la clase *App*.

La clase *App* es la encargada de crear el *canvas* y la animación que da vida a nuestro mini juego *Lost Runner*. Al iniciar la clase al cargar el archivo *App.js* se ejecutará la función *initialize* de la figura 15. En esta función crearemos una etiqueta *canvas* que colocaremos en el bloque *joc* del archivo *app.html*, además de crear un objeto del tipo *Cargador* en el que cargaremos todas las imágenes que utilizaremos en el mini juego.

```
// funcion inicio
App.prototype.initialize = function () {
    var self = this;

    //creamos el objeto canvas
    canvas = document.createElement('canvas');
    canvas.width = 800;
    canvas.height = 600;
    var joc = document.getElementById('joc');
    joc.appendChild(canvas);

    ancho = canvas.width;
    alto = canvas.height;

    //inicializamos el stage
    stage = new createjs.Stage(canvas);

    //carga de imagenes
    cargador = new Cargador();
    cargador.onComplete = function () {
        self.imgCargadas();
    };
    cargador.loadImagenes([rhero, rnube1, rnube2, rsuelo,rsuelo2,rsuelo3, rfondo]);
```

Figura 15: *Código de la función de inicialización de la clase App.*

Crear una etiqueta *canvas* no es lo único que debemos hacer para poder crear la animación del mini juego, necesitamos de un escenario en el que representarla. Haciendo uso de la API de las librerías *EaselJS* (EaselJS, 2016), creamos dicho escenario mediante la línea de código **stage = new createjs.Stage(canvas)**, colocándolo dentro de la etiqueta *canvas.*

Una vez tenemos el escenario en el que representar nuestro mini juego debemos colocar todos los actores y el decorado. Al terminar de cargar todas las imágenes del objeto *Cargador* ejecutaremos la función *imgCargadas* de la figura A10 del anexo. Está función creará un mapa de bits de cada imagen y lo añadirá al escenario de dentro hacia afuera por lo que, debemos respetar el orden en que las creamos poniendo primero el fondo y finalmente al héroe.

Será importante cargar varias imágenes de fondo y terreno ya que, cuando se inicie la animación irán desplazándose de derecha a izquierda, tal y como vemos en la figura 16, por lo que al mostrarse el final de la imagen dentro del canvas, si no tenemos el siguiente fondo o terreno justo detrás, se mostrará un trozo en blanco que romperá la continuidad de la animación.

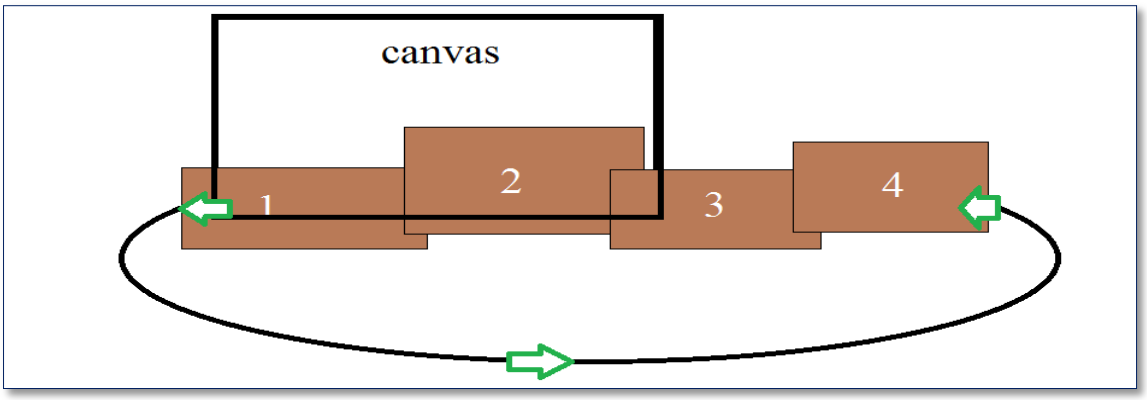

Figura 16: *Funcionamiento de la animación.*

Una vez creado el escenario, colocado los actores y el decorado tenemos que iniciar la animación, para ello se va a utilizar el temporizador o *ticker* de la librería *EaselJS*. La función *ticker* del escenario se encarga de realizar una serie de tareas cada vez que ocurre un *tick*. Podemos establecer la velocidad en la que ocurrirá cada *tick* en *fps*, siglas del inglés *frames per second*. En la figura 17 podemos ver como iniciamos el *ticker*.

```
createjs.Ticker.setFPS(20);
createjs.Ticker.on("tick", function (e) {
```

Figura 17: *Función ticker de la librería EaselSJ.*

A continuación vamos a describir las partes de código más importantes dentro de la función *ticker* (se puede consultar el código completo en la figura A11 del anexo):

Una de las partes más importantes de nuestro mini juego es el cálculo de las colisiones del héroe con el suelo. Esta acción la realizamos cada vez que movemos el suelo en cada *tick* llamando a la función *detectaColision* de la figura 18. La función devuelve un punto que muestra la localización del suelo más próximo al héroe. Para ello utiliza el método *localToLocal* de la API de *EaselJ S* que devuelve las coordenadas de la distancia entre dos objetos de dentro de un *Stage*.

```
function detectaColision(limit,hero,target){

  var pc = hero.localToLocal(0,limit,target);


  if(hero.y < pc.y && pc.x > -10 ){
    return {"x": pc.x ,"y":pc.y, "t":target};
  }

  return false;

}
```

Figura 18: *Función detectaColision del App.js.*

Después de calcular el punto de colisión llamamos a la función *calculaSuelo* de la figura A12 del anexo. En esta función tenemos toda la lógica que controla la posición del héroe con respecto a los suelos. Si nuestro héroe esta colisionando con algún suelo terminaremos la partida del juego deteniendo la animación. En el caso de que no exista colisión calculáremos la altura del suelo para posicionar al héroe encima de este.

Hablando de la posición de los suelos, cada vez que un suelo atraviesa el *canvas* tendremos que recolocarlo al final de la fila de suelos y además variar su altura ya que, nos interesa ir alternando la altura de cada suelo para crear desniveles. Podemos ver cómo realizamos esto en el bucle de la figura A13 del anexo.

Finalmente cabe destacar la función *saltar*, con la que se ha animado el salto del héroe cada vez que se llama a la función *OnPulsar*. Está función aumenta la velocidad de desplazamiento del héroe con respecto al eje y teniendo en cuenta el que al caer no supere la altura límite. Se permitirá que el héroe salte dos veces seguidas doblando la altura que alcance. Para controlar que no se salte más de una vez utilizaremos el semáforo *saltox2* que estará a *true* cuando se permita saltar dos veces. Podemos ver el código de la función en la figura 19.

```
App.prototype.saltar = function () {
    if ( ensuelo ) {  // si esta en el suelo puede saltar
        hero.velocity.y = -10;
        ensuelo = false;
        saltox2 = true;

    } else if ( saltox2 ) { // si esta en el aire puede hacer un segundo salto
        hero.velocity.y = -10;
        saltox2 = false;
    }
};
```

Figura 19: *Código de la función saltar.*

3.3 Elegir y preparar un servidor web

Una vez terminada la aplicación tenemos que subirla a un servidor web y realizar algunos ajustes antes de poder integrarla en las aplicaciones de *Facebook*. A continuación, vamos a explicar en este punto de la memoria todo el proceso que hemos seguido para llevar a término dicha tarea.

3.3.1 Amazon AWS: Preparando una instancia

El primer reto con el que nos enfrentamos, antes de poder instalar y configurar un servidor web, es el de encontrar un buen proveedor de infraestructuras de tecnologías de la información en donde poder alojar nuestra aplicación. Al tratarse nuestra aplicación de un mini juego que estará integrado en la red social *Facebook*, vamos a necesitar que nuestro servidor este activo continuamente, sin interrupciones y que sea capaz de tener muchas conexiones concurrentes. Por ello, nos hemos decantado por los servicios que ofrecen en *Amazon Web Services*[1], en adelante AWS.

Al utilizar AWS obtenemos una plataforma de infraestructura escalable, de confianza y de bajo coste ya que trabajamos en la nube, pudiendo reservar una pequeña porción de esta mediante una instancia. Nos ofrece distintas herramientas y servicios para poder alojar nuestra aplicación en la nube, desde la instancia donde instalaremos el servidor hasta la creación de un dominio o una dirección IP estática. Además, podemos obtener parte de estos servicios utilizando la capa gratuita que nos ofrece la compañía para realizar nuestro proyecto (*Amazon*, 2016).

En nuestro caso, únicamente vamos a utilizar el servicio *Amazon Elastic Compute Cloud* o EC2 que nos permite hospedar un servidor virtual en una instancia que contendrá una máquina virtual con un sistema operativo y unas aplicaciones ya preinstaladas. A continuación, procederemos a explicar los pasos necesarios para poner en marcha una instancia *t2.micro* con el sistema operativo Linux y un servidor web *Apache2*:

1. El primer paso será crear la instancia. Para ello, accedemos con nuestra cuenta a la consola de Amazon EC2 y pulsamos el botón *Launch Instance*, tal y como podemos ver en la figura 20.

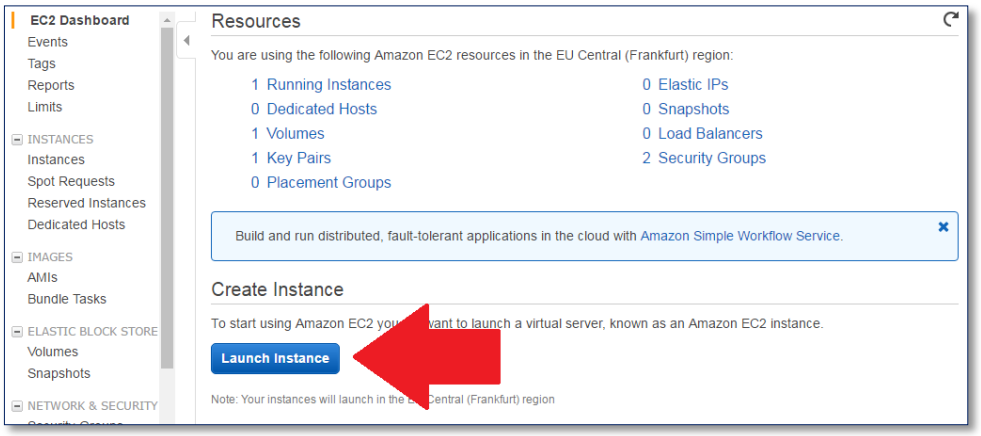

Figura 20: *Consola de Amazon EC2.*

[1] https://aws.amazon.com

2. Tras pulsar el botón, habremos accedido al asistente de configuración de la instancia. Aquí podremos elegir el sistema operativo (SO), el cual vendrá instalado en la imagen de la máquina virtual y el tipo de instancia. Hemos elegido en nuestro caso una *t2.micro* con Ubuntu Server 14.04 LTS (HVM), SSD Volume Type. Después de elegir el SO y el tipo de instancia pulsaremos el botón *Review and Launch*, tal y como podemos observar en la figura 21, aunque esta acción dejará con la configuración por defecto la instancia en los pasos siguientes.

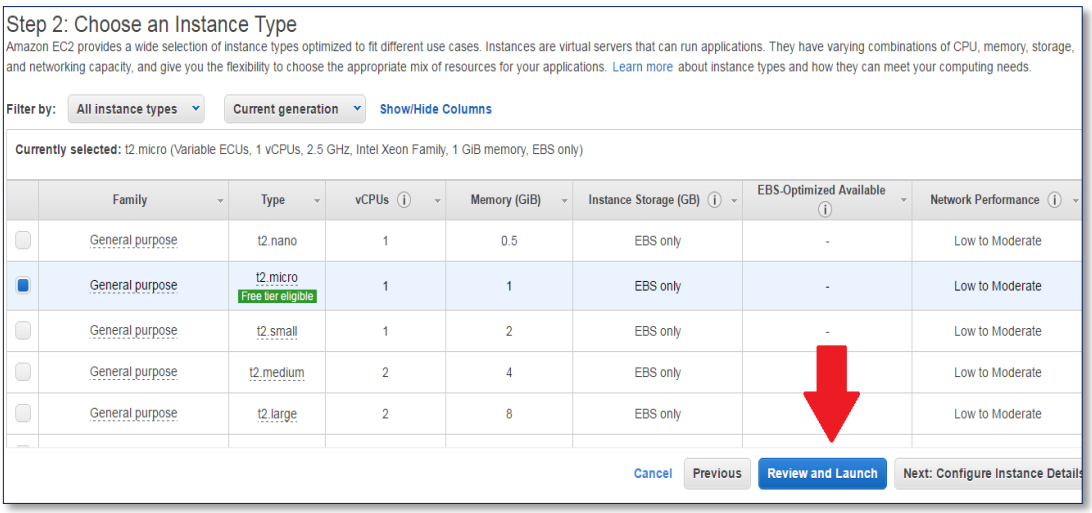

Figura 21: *Asistente de configuración de la instancia del AWS.*

3. Después de pulsar en *Review and Launch*, el asistente nos trasladará directamente al último paso, donde tendremos que generar un nuevo par de claves con las que nos conectaremos mediante *SSH*, del inglés *Secure Shell* o intérprete de órdenes seguro, a nuestra instancia. Se recomienda que se guarden las claves en el directorio *.ssh* de *root* si se trabaja en Linux o en *C:\Users\miusuario\.ssh* si se trabaja en Windows. Podemos ver en la figura 22 el cuadro de diálogo que nos aparecerá para crear el par de claves. Una vez creadas las llaves podemos darle al botón *Launch Instances* para poner en marcha la instancia.

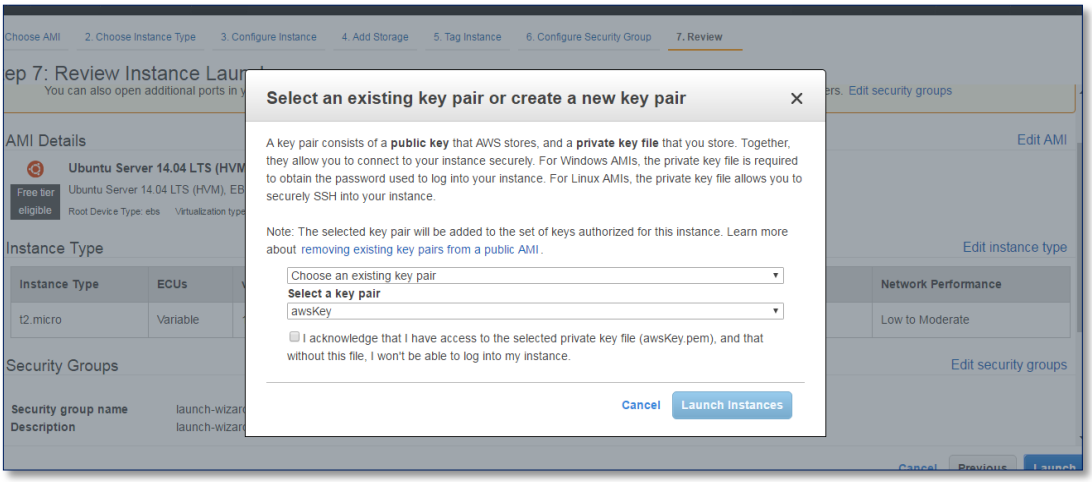

Figura 22: *Formulario creación de par de llaves en AWS.*

3.3.2 Configuración e instalación del servidor web Apache2

En el apartado anterior hemos explicado cómo poner en marchar la instancia, de modo que a continuación procederemos a acceder a esta utilizando las claves generadas e instalaremos y configuraremos el servidor web *Apache2*.

Para acceder al servidor necesitaremos un programa con el que poder establecer la conexión, siendo el nuestro el *MobaXterm*[2], la dirección IP pública de la instancia y el nombre de usuario. Para obtener la dirección IP de la instancia debemos ir al apartado *Instances* de nuestra consola del EC2 en AWS y copiar la *Public IP* como vemos en la figura 23. Los posibles nombres de usuario que tendrán las instancias de Linux son *ec2-user*, *root*, *ubuntu* y *fedora*. Dado que nosotros hemos optado por una instancia con el SO Ubuntu, debemos utilizar el usuario *ubuntu*.

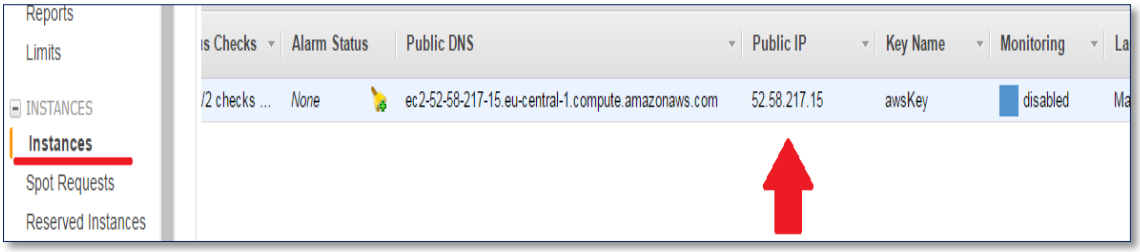

Figura 23: *Dirección IP pública de una instancia en AWS.*

Una vez estemos conectados a la instancia en la que instalaremos nuestro servidor web, lo primero que haremos será actualizar el SO. El actualizar el sistema será tan sencillo como escribir el comando ***sudo apt-get update*** y, una vez termine, reiniciar la máquina. Cuando volvamos a conectarnos, instalaremos el servidor web *Apache2* con el comando ***sudo apt-get install apache2***. Si todo ha ido bien, al introducir la IP pública y acceder desde un navegador, deberíamos ver la página por defecto de nuestro servidor web de la forma en que se muestra en la figura 24.

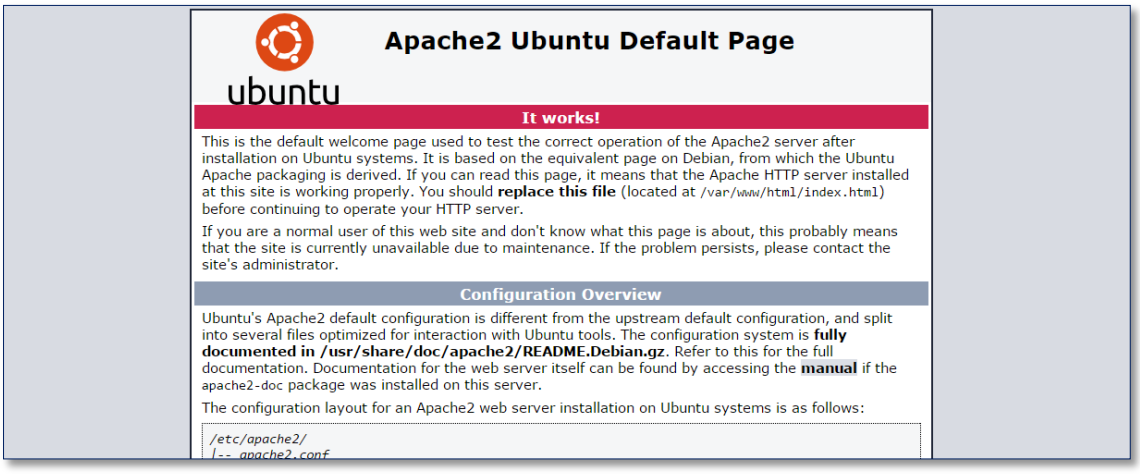

Figura 24: *Página web por defecto del servidor web Apache2.*

[2] Puedes obtener el programa gratuitamente en http://mobaxterm.mobatek.net/

En estos momentos ya tenemos en funcionamiento el servidor web, ahora lo que queremos es que muestre nuestra aplicación para ello, debemos crear la estructura de directorios necesaria y configurar el archivo del *virtual host* del *Apache2*.

El directorio raíz donde el servidor *Apache2* busca el contenido de la web llevará el nombre del dominio en el que vamos a registrar nuestra aplicación. Este estará situado en la ruta ***/var/www*** y además contendrá el directorio ***public_html,*** que será el directorio público en el que almacenaremos todo el contenido de la aplicación. Para crear toda la estructura de directorios introduciremos el comando ***sudo mkdir –p /var/www/lostrunnertfg.tk/public_html*** tal y como se ve en la siguiente figura.

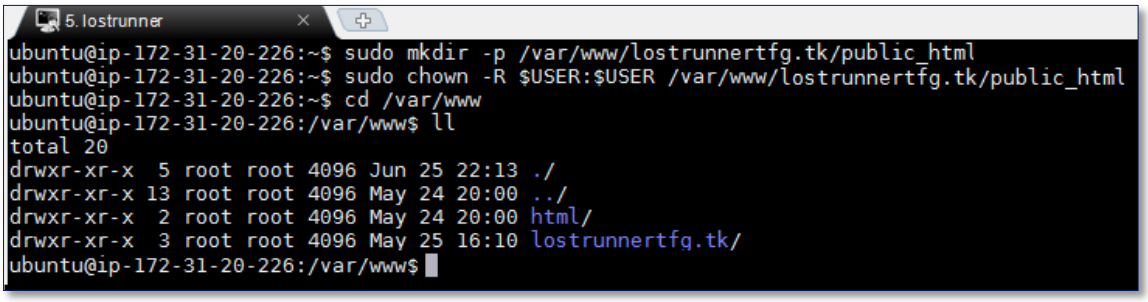

Figura 25: *Comandos creación estructura de directorios en el servidor*.

La segunda orden que hemos introducido sirve para cambiar el propietario del directorio, tomado $USER el valor del nombre de dicho usuario, y se utilizaría en caso de que estuviésemos trabajando con un usuario distinto de *root*. Si hemos cambiado el propietario de la carpeta también deberemos asegurarnos que *Apache2* tiene permisos de lectura mediante la orden ***sudo chmod -R 755 /var/www***.

Ahora que tenemos toda la estructura de directorios montada solo nos queda crear y configurar el archivo *virtual host* de la aplicación para que cada vez que accedamos al dominio el servidor nos muestre el contenido. Para realizar esta tarea será necesario que realicemos los siguientes pasos:

1. Dado que *Apache2* tiene un archivo *virtual host* por defecto, lo copiaremos y trabajaremos sobre él. Para ello introduciremos el comando ***sudo cp /etc/apache2/sites-availible/000-default.conf /etc/apache2/sites-availible/lostrunnertfg.tk.conf.***

2. El siguiente paso es configurar el archivo host que hemos copiado. Abrimos el archivo con un editor de textos, por ejemplo el *nano*, siendo el usuario root y personalizamos los datos para que sean acordes a nuestro dominio web tal y como se ve en la figura 26.

```
<VirtualHost *:80>
        ServerAdmin anclimon88@gmail.com
        ServerName lostrunnertfg.tk
        ServerAlias www.lostrunnertfg.tk
        DocumentRoot /var/www/lostrunnertfg.tk/public_html
        ErrorLog ${APACHE_LOG_DIR}/error.log
        CustomLog ${APACHE_LOG_DIR}/access.log combined

</VirtualHost>
```

Figura 26: *Archivo configuración del virtual host para el dominio lostrunnertfg.tk.*

3. Finalmente, para que los cambios y configuraciones se hagan efectivos, únicamente nos resta habilitar el archivo del *virtual host* en el servidor. Para hacerlo introduciremos el comando ***sudo a2ensite lostrunnertfg.tk.conf*** y reiniciaremos el servidor *Apache2* con ***sudo service apache2 restart***.

Llegados al final de la configuración, subiremos los archivos de la aplicación a la carpeta **public_html** mediante el protocolo FTP, del inglés *file transfer protocol*. Desde este momento ya podremos acceder a la aplicación introduciendo el nombre de dominio de la aplicación en nuestro navegador, por lo que debemos crearlo y vamos a explicar cómo en el siguiente apartado.

3.3.3 Creando un dominio propio para la aplicación

A la hora de crear nuestro propio nombre de dominio debemos tener en cuenta, tal y como explicábamos en el punto 3.1.5 de la memoria, que es el nombre con el que los usuarios van a buscar nuestra aplicación en la red, por lo que es una parte muy importante a la hora de publicitarla.

En esta sección vamos a explicar todo el proceso que hemos seguido para obtener un nombre de dominio. Muchos proveedores de infraestructuras TIC nos ofrecen el espacio necesario para tener nuestro propio servidor web, además del nombre de dominio que elijamos. Otros, por el contrario, ofrecen las dos cosas por separado, pudiendo comprar un nombre de dominio y tantos servidores web como queramos. En nuestro caso, AWS ofrece el crear tu propio nombre de dominio de forma separada.

Debemos tener claro que el nombre de dominio es un identificador único en la red, por lo que para mantenerlo hay que pagar una cuota al proveedor. Nosotros decidimos buscar alternativas al servicio que nos ofrece AWS puesto que existen algunos proveedores en los que, durante un tiempo determinado, puedes obtener un dominio de forma gratuita.

Tras varias consultas, encontramos el sitio web Dot TK (Dot TK, 2016). En este, se puede obtener un dominio de segundo nivel, dominios que identifican el país, tal y como nos lo explican en la Oficina Española de Patentes y Marcas (OEPM, 2016), siendo este gratuito durante un año.

El proveedor de dominios Dot TK nos proporcionará el nombre de dominio que deseemos en base a su disponibilidad. A cambio, únicamente pide nuestro registro en el sitio, ya que, creando nuestro dominio *.tk*, estamos dando a conocer las islas Tokelau, siendo este el objetivo del proveedor, tal y como podemos leer en el texto de la figura 27.

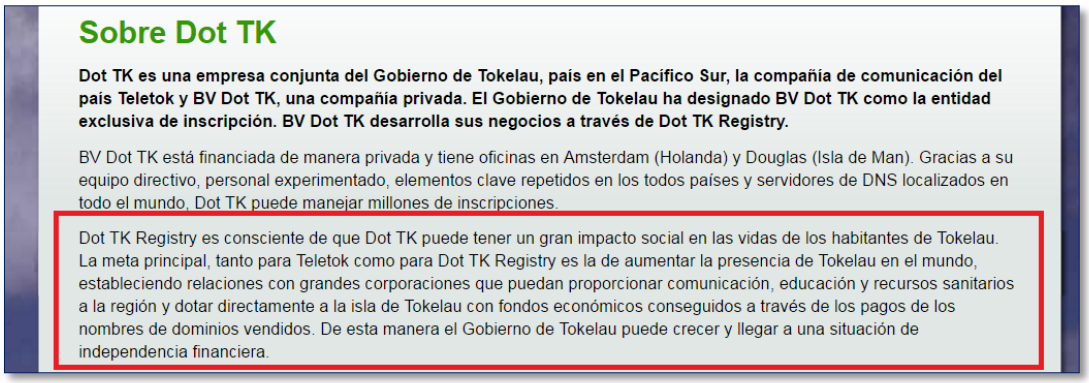

Figura 27: *Objetivo del proveedor Dot TK Registry.*

A continuación, realizaremos el registro de nuestro dominio. Teniendo en cuenta que vamos a crear el dominio para nuestra aplicación *Lost Runner* y que estamos realizando un TFG, hemos decidido que nuestro dominio será *lostrunnertfg.tk*. Seguidamente describiremos los pasos que hemos seguido para crear el dominio:

1. Introduciremos el nombre que deseemos en el formulario y pulsaremos *OK*.

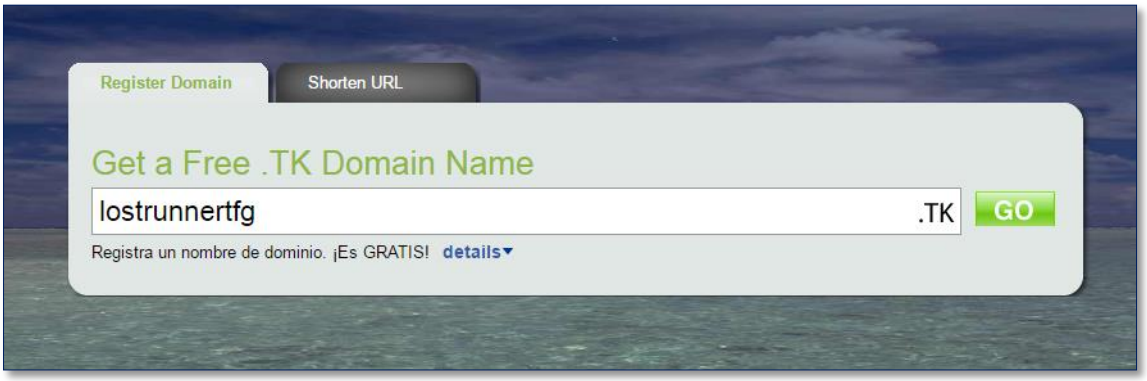

Figura 28: *Primer paso de la creación de un dominio TK.*

2. Como disponemos ya de un servidor en una instancia en AWS, dirigiremos el tráfico del dominio a la dirección IP pública de la instancia y seleccionaremos el tiempo que queremos que esté activo el dominio.

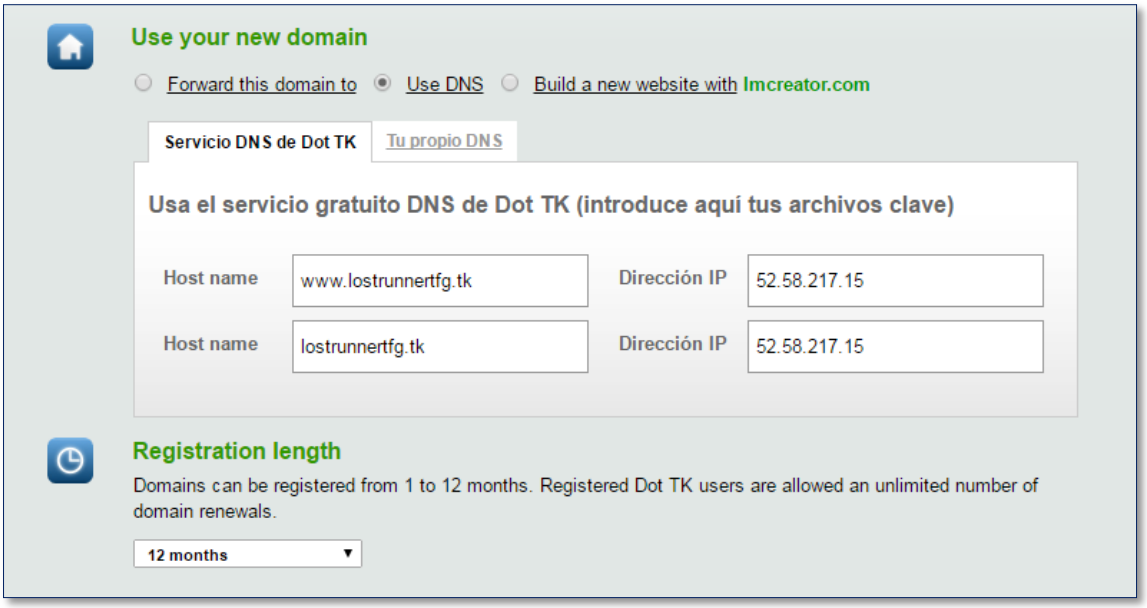

Figura 29: *Segundo paso de la creación de un dominio TK.*

3. Por último, solo nos queda completar el registro en el sitio de Dot TK para tener activo nuestro dominio.

Si hemos seguido correctamente los pasos deberíamos poder acceder a nuestra aplicación a través de la dirección *lostrunnertfg.tk*. Si queremos realizar algún cambio o configuración podemos acceder con nuestra cuenta de Dot TK a Freenom (Freenom, 2016).

3.3.4 Certificado SSL: registro e instalación

Uno de los requisitos de *Facebook* para poder integrar una aplicación con la plataforma es que esta tenga un certificado de seguridad instalado en su servidor para blindar las conexiones entre la aplicación y la red social. En esta sección de la memoria vamos a hablar de cómo obtener un certificado e instalarlo en el servidor.

Al igual que ocurre con los dominios, elegir un buen proveedor de certificados de seguridad SSL y que se adapte a nuestro presupuesto es una tarea de investigación. En nuestro caso, como nuestra aplicación la tendremos accesible al público durante un corto periodo de tiempo, hemos optado por utilizar el certificado gratuito con una duración de noventa días que nos ofrece la autoridad certificadora *Comodo*[3], una empresa del sector tecnológico especializada en mantener la confianza y seguridad en la red (Comodo, 2016).

Seguidamente procederemos a explicar todo el proceso que hemos seguido para obtener nuestro certificado de seguridad:

1. Accedemos al servicio que nos ofrece *Comodo* a través de la dirección https://www.instantssl.com/free-ssl-certificate.html y pulsamos sobre el botón *Free Download*.

2. Una vez hemos empezado el proceso para descargarnos el certificado gratuito, lo primero que nos piden es el CSR o *Certificate Signing Request*, un bloque de texto cifrado generado desde nuestro servidor. Para generar nuestro CSR debemos asegurarnos primero que tenemos installado *OpenSSL* en nuestro servidor mediante el comando **sudo apt-get install openssl**. El siguiente paso será generar la clave con el comando **openssl req –nodes –newkey rsa:2048 –keyout miservidor.key –out server.csr**. El comando anterior iniciará un asistente de configuración que deberemos rellenar tal y como podemos observar en la figura 30.

```
ubuntu@ip-172-31-20-226:~$ openssl req -nodes -newkey rsa:2048 -keyout m
key -out server.csr
Generating a 2048 bit RSA private key
.....................+++
.....................................+++
writing new private key to 'miserver.key'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:ES
State or Province Name (full name) [Some-State]:Valencia
Locality Name (eg, city) []:Valencia
Organization Name (eg, company) [Internet Widgits Pty Ltd]:miempresa SA
Organizational Unit Name (eg, section) []:IT
Common Name (e.g. server FQDN or YOUR name) []:lostrunnertfg.tk
Email Address []:anclimon88@gmail.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
```

Figura 30: *Ejemplo del proceso de generación del texto CSR.*

[3] https://www.comodo.com/e-commerce/ssl-certificates/free-ssl-certificate.php

3. Tras generar nuestro archivo *server.csr*, deberemos abrirlo con un editor de texto y copiarlo en el formulario de la web de Comodo y elegir el tipo de servidor que tenemos instalado, de la misma forma en que vemos en la figura 31.

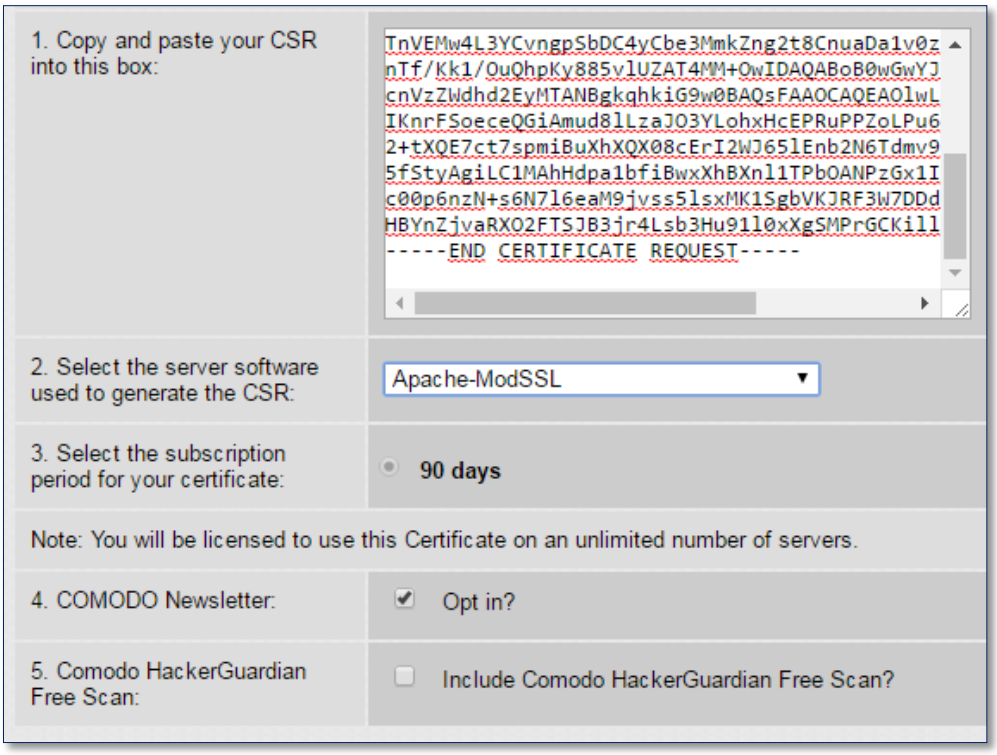

Figura 31: *Formulario de creación de un certificado SSL en Comodo.*

4. A continuación, tendremos que demostrar que tenemos el control de nuestro dominio web para poder instalar el certificado. El asistente nos brinda distintas opciones pero, como en nuestro caso no disponemos de e-mail, realizaremos la comprobación mediante las claves MD5 y SHA-1, por lo que las copiaremos para utilizarlas posteriormente.

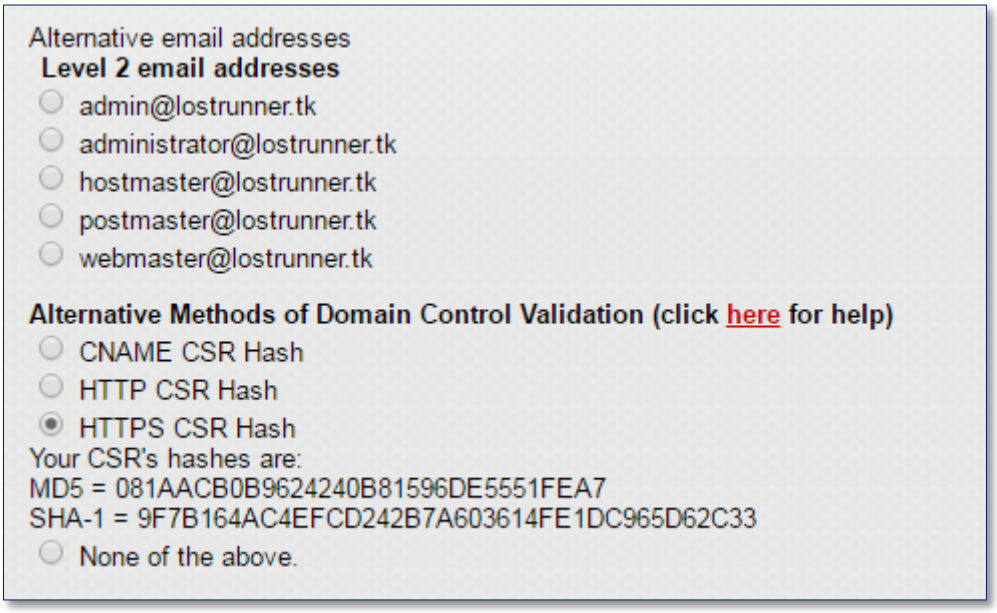

Figura 32: *Formulario elección del método de control del dominio.*

5. En el paso siguiente del asistente se requieren una serie de datos personales, los cuales rellenaremos y aceptaremos los términos y condiciones de uso.

6. Para finalizar el proceso completamente tendremos que demostrar que controlamos el dominio. Al haber elegido demostrarlo mediante el par de claves MD5 y SHA-1, deberemos dirigirnos al panel de control de nuestro dominio en Freenom (Freenom, 2016). Una vez hallamos accedido deberemos ir a *Domains* > *my Domains* y pulsar en *Manage Domain.*

7. Una vez estemos en este panel iremos a la sección *Manage Freenom DNS* y añadiremos un nuevo registro igual que en la figura 33, donde la primera parte será la clave MD5 y la parte posterior la SHA-1.

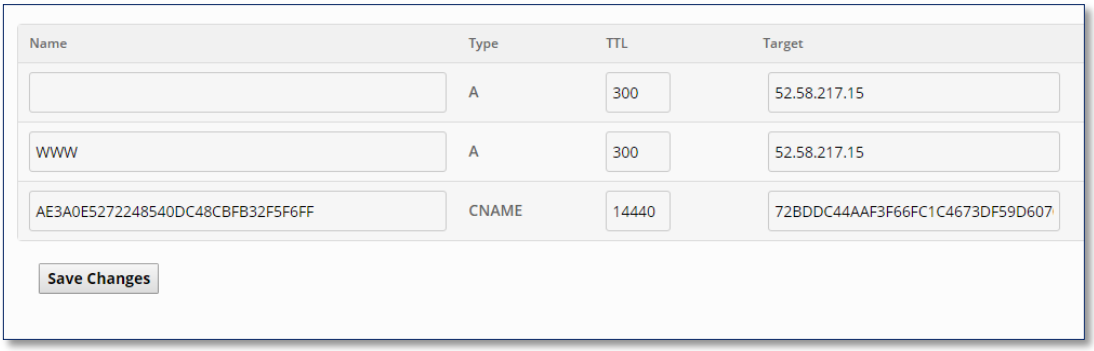

Figura 33: *Sección de configuración del DNS del Dominio .tk.*

8. Para validar nuestro dominio, lo único que resta por hacer es acceder a la dirección formada por la unión de las claves y del dominio de la siguiente forma:
MD5 *lostrunnertfg.tk* **SHA-1**

9. Finalmente, solo nos queda esperar a que la empresa *Comodo* nos envíe al correo que le hemos especificado los archivos que conforman el certificado SSL y descargarlos.

En estos momentos, deberíamos poseer dos archivos, el certificado de seguridad y un archivo con la extensión *.ca-bundle* Subiremos ambos archivos mediante FTP a nuestro servidor al directorio *etc/ssl/*, dentro de una carpeta que crearemos para identificarlos y así poder proceder a instalar el certificado.

El primer paso de la instalación será editar el archivo de configuración del *Apache2* de nuestro dominio para añadir una nueva etiqueta *<Virtual Host>* y permitir las conexiones a nuestro sitio a través de una conexión segura, al igual que mostramos en la figura 34.

```
12  <VirtualHost *:443>
13
14      ServerAdmin anclimon88@gmail.com
15      ServerName lostrunnertfg.tk
16      ServerAlias www.lostrunnertfg.tk
17      DocumentRoot /var/www/lostrunnertfg.tk/public_html
18      ErrorLog ${APACHE_LOG_DIR}/error.log
19      CustomLog ${APACHE_LOG_DIR}/access.log combined
20
21      SSLEngine on
22      SSlCertificateKeyFile /etc/ssl/ssl.key/myserver.key
23      SSLCertificateFile /etc/ssl/ssl.crt/lostrunnertfg_tk.crt
24      SSLCertificateChainFile /etc/ssl/ssl.crt/lostrunnertfg_tk.ca-bundle
25
```

Figura 34: *Configuración del archivo virtual host con SSL.*

Al editar el archivo de configuración debemos asegurarnos de que escribimos correctamente las rutas hacia nuestros archivos:

- **SSLCertificateFile** debe ser el archivo del certificado de *Comodo* para nuestro dominio.
- **SSLCertificateKeyFile** debe ser el archivo de claves generadas al crear la CSR.
- **SSLCertificateChainFile** debe ser el archivo con la extensión *.ca-bundle* proporcionado por la entidad certificadora.

Después, comprobaremos que el archivo de configuración del *Virtual Host* es correcta mediante el comando **apachectl configtest**. En caso afirmativo, procederemos a reiniciar el servidor mediante **sudo service apache2 restart**. Desde este momento, ya podremos conectarnos a nuestro dominio utilizando una conexión segura como se muestra en la figura 35.

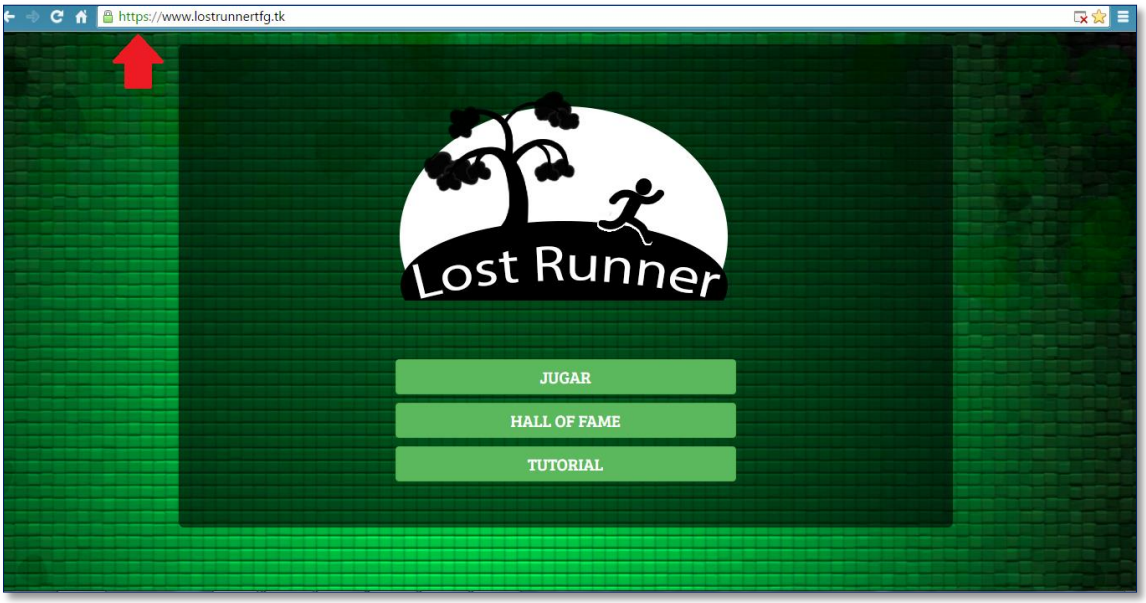

Figura 35: *Acceso a través de https a lostrunnertfg.tk.*

3.4 Integración de la aplicación en Facebook

Prosiguiendo con nuestro proyecto, vamos a describir los pasos que hemos seguido para integrar la aplicación desarrollada en la plataforma de *Facebook*. Hay que tener en cuenta que para poder realizar las tareas que describimos en este punto es necesario disponer de una cuenta de *Facebook*.

3.4.1 Creación de la aplicación en Facebook

Cuando integramos una aplicación en la red social de *Facebook* estamos creando una aplicación en la red, con un identificador único, que apuntará a la dirección web donde la tengamos alojada. Dicho esto, antes de poder integrar nuestra aplicación en la red social deberemos crear una aplicación en *Facebook*.

Para crear la aplicación en la plataforma accederemos a la web de desarrolladores de *Facebook* [4] e iniciaremos sesión. A continuación, en nuestro menú de aplicaciones, seleccionaremos la opción *Añadir una nueva aplicación*, marcada con una flecha en la figura 36.

Figura 36: *Menú "Mis aplicaciones" en la plataforma de desarrolladores de Facebook.*

Al acceder se nos abrirá un diálogo en el que tendremos que elegir qué tipo de aplicación vamos a hacer, en nuestro caso será un *Facebook Canvas* ya que pretendemos vincular nuestro mini juego en la plataforma.

[4] https://developers.facebook.com

Después, al seleccionar la el tipo de aplicación que deseamos utilizar, se iniciara un asistente de creación y configuración de la aplicación. En el primer paso del asistente debemos elegir el nombre de la aplicación, indicarle el correo al que queremos que se dirijan los administradores y la categoría a la que pertenece. En el caso de los juegos, deberemos introducir también la subcategoría en la que clasificaríamos el tipo de juego que hemos implementado. En la figura 37 podemos observar cómo hemos rellenado nosotros los datos.

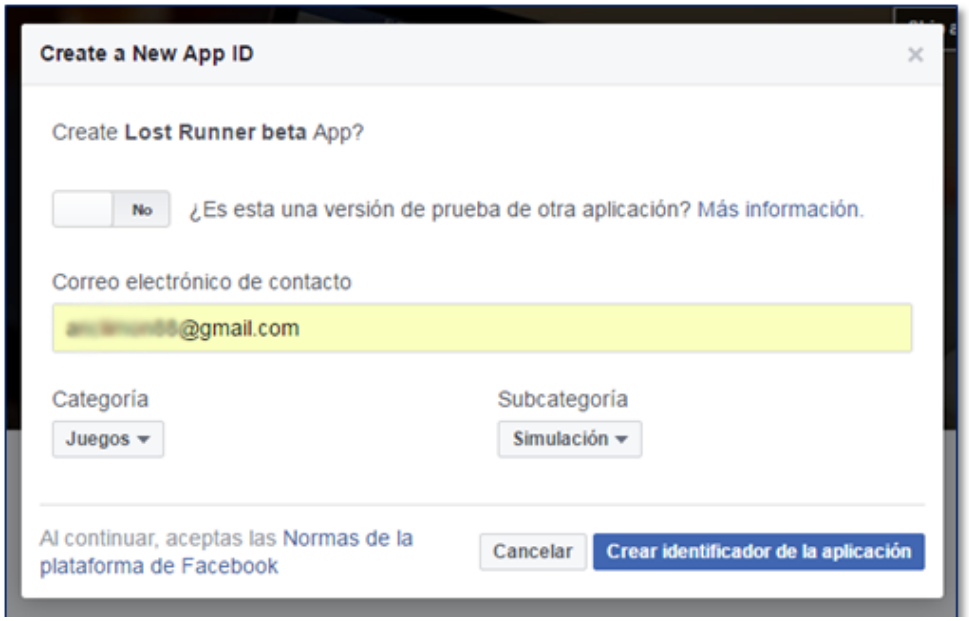

Figura 37: *Primer paso en la creación de una aplicación en Facebook.*

El siguiente paso del asistente será el que permite configurar nuestra aplicación. Como podemos observar en la figura 38, lo primero que nos ofrecen es el código JavaScript básico con el que nuestra aplicación se conectará con la red social de *Facebook*. Podemos copiar este código y pegarlo justo después de la etiqueta *<body>* de nuestro código o añadirlo en un archivo JavaScript a parte que englobe todas las funciones que interactúen con la API de la plataforma.

Figura 38: *Código JavaScript con el que conectar la aplicación con Facebook.*

Justo después del código de la figura 38 encontraremos un pequeño formulario de dos campos en los que configuraremos la dirección de la página de nuestro *canvas* en *Facebook* e introduciremos el dominio de nuestra aplicación como vemos en la figura 39.

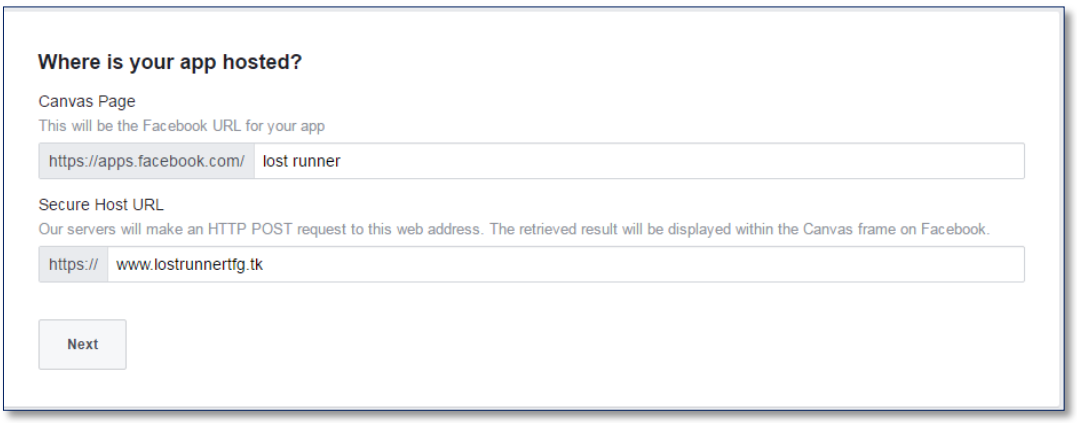

Figura 39: *Formulario de registro de las direcciones de la aplicación.*

Al terminar pulsaremos el botón *Next* y, si todo está correcto, el asistente nos proporcionará otra porción de código JavaScript, que esta vez será la función que permite iniciar sesión en *Facebook* desde nuestra aplicación. Al igual que hemos hecho anteriormente, copiaremos el código para introducirlo en nuestra aplicación.

Si hemos seguido correctamente el asistente, ya tendremos creada la aplicación. En ese punto solo resta comprobar que funcione e ir a nuestro panel de control de la aplicación pulsando sobre *Skip to Developer Dashboard*.

El siguiente paso en la cofiguración de nuestra aplicación será ir al apartado de *Configuración* > *Información básica*, donde rellenaremos todos los campos y guardaremos los cambios realizados. No debemos olvidar mirar la configuración avanzada por si nos interesa cambiar alguna opción.

Finalmente, deberemos hacer pública nuestra aplicación y mandarla a revisión. Aprovecharemos la revisión para pedir a *Facebook* los permisos de *publish_actions* y *user_friend*, los que nos permitiran que el usuario pueda publicar y compartir en su muro la puntuación obtenida en el mini juego e invitar a sus amigos. Cuando la revisión de la aplicación sea positiva tendremos los elementos de la figura 40 aprobados para su uso .

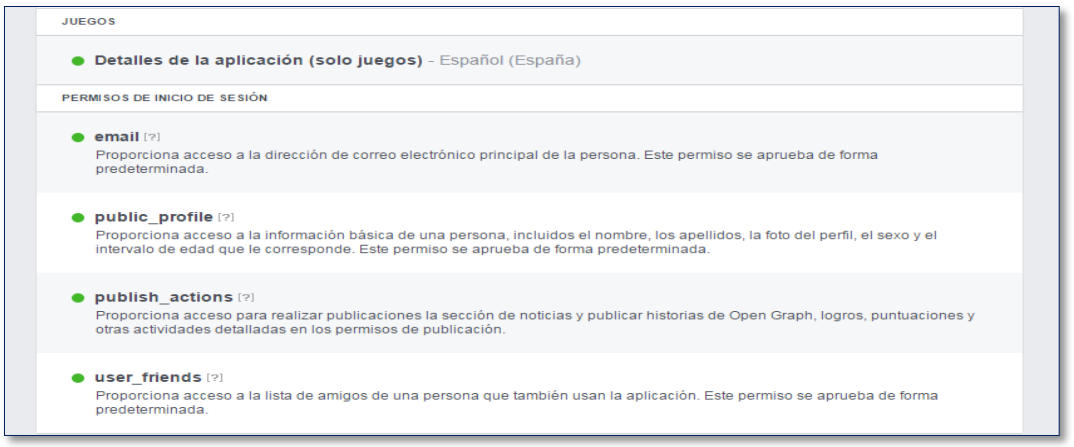

Figura 40: *Elementos aprobados después de la revisión de la aplicación.*

3.4.2 Integrar la funcionalidad de Facebook en la aplicación: API Graph

Una de las principales ventajas de integrar una aplicación en *Facebook* es que la red social nos proporciona una serie de herramientas con las que poder aumentar la usabilidad y visibilidad en la red de nuestra aplicación, la API Graph. En esta parte de la memoria vamos a hablar de qué funciones hemos utilizado y cómo las hemos integrado en nuestra aplicación.

Al diseñar la estructura que tendría nuestra aplicación decidimos que todas las funciones JavaScript estarían agrupadas en un mismo archivo, *Facebook.js*. La primera que debemos utilizar es la que nos proporciona *Facebook* en el asistente de creación de aplicaciones para poder conectar nuestra aplicación con la API. En nuestro caso, la incluiremos dentro de una función que la ejecutará cuando tengamos todo el documento listo en nuestro navegador tal y como vemos en la figura 41.

```
$(document).ready(function(){

  window.fbAsyncInit = function() {
    FB.init({
      appId      : '1187789521262715',
      xfbml      : true,
      version    : 'v2.6',
      status     : true
    });


   compruebaLogin(false);

  };

  (function(d, s, id){
     var js, fjs = d.getElementsByTagName(s)[0];
     if (d.getElementById(id)) {return;}
     js = d.createElement(s); js.id = id;
     js.src = "//connect.facebook.net/en_US/sdk.js";
     fjs.parentNode.insertBefore(js, fjs);
   }(document, 'script', 'facebook-jssdk'));


});//FIN READY
```

Figura 41: *Código de la función para conectar la aplicación con Facebook.*

Si observamos detenidamente el código podremos diferenciar dos métodos: uno que se encarga de conectarnos con *Facebook* y otro que carga en nuestro navegador las librerías necesarias para poder utilizar la API de la red social. Dentro de la función *fbAsyncInit* hemos incluido una llamada a la función *compruebaLogin*, cuyo propósito es comprobar que el usuario haya iniciado o no sesión en *Facebook* en el momento en que se cargue la aplicación.

La importancia de que el usuario inicie sesión en *Facebook* radica en que sin esto no es posible realizar ninguna acción más que incluya una interacción con la red social. Por esta razón, y para asegurarnos que el usuario inicia sesión, hemos incluido la llamada a la función *compruebaLogin* al pulsar el botón *Jugar* de la pantalla de inicio de la aplicación. Podemos observar en la figura 42 como le pasamos una variable booleana (*jugar)* a la función con el fin conocer si la llamada a esta se ha hecho al pulsar el botón *Jugar*.

```
function compruebaLogin(jugar){


    FB.getLoginStatus(function(response) {

      //comprobamos que no este conectado
      if (response.status == 'connected') {
        onLogin(response,jugar);
      } else {

        FB.login(function(response) {
          onLogin(response,jugar);
        }, {scope: 'user_friends, email,publish_actions'});
      }
    });
}
```

Figura 42: *Código de la función compruebaLogin.*

Como podemos ver, estamos utilizando los métodos *getLoginStatus* y *login* de la API Graph que comprueban si el usuario ha iniciado sesión en *Facebook*. En el caso de que no haya iniciado sesión, abriremos el dialogo de inicio de sesión y, además, le pediremos los permisos necesarios para obtener su email, lista de amigos y para poder publicar en su muro de *Facebook*.

Una vez el usuario haya iniciado sesión en *Facebook* ejecutaremos en nuestro código la función *onLogin* que realizará una llamada a la API Graph para obtener algunos datos del usuario como son el correo electrónico, nombre, foto de perfil e identificador de *Facebook*. Además, podremos dar un mensaje de bienvenida, obtener la clasificación con sus amigos, colocar su foto de perfil en el juego o redirigirlo al archivo *app.html*, tal y como vemos en la figura 43.

```
function onLogin(response,jugar) {

  if (response.status == 'connected') {
    FB.api('me?fields=id,email,first_name,name,picture.width(120)', function(data) { // obtenemos datos
      datafb = data;
      var welcomeBlock = document.getElementById('fb-welcome');

      if(welcomeBlock !== null){
        welcomeBlock.innerHTML = '¡Bienvenido, ' + data.first_name + '!'; //ponemos un mensaje de bienvenida
      }

      $("#imguser img").attr('src',data.picture.data.url); // colocamos la imagen del usuario

      getRanking(); //obtenemos el ranking de amigos

      if(jugar){
        window.location.href = "/app.html"; // redirigimos al juego
      }
    });
  }
}
```

Figura 43: *Código de la función onLogin.*

La función *getRanking* de la figura anterior se encarga de hacer una llamada a la API Graph para obtener la clasificación del usuario con sus amigos en relación a la puntuación que hayan obtenido al jugar y generar la clasificación que mostramos en la sección *Hall of Fame* de la aplicación. Podemos ver la función en la figura A14 del anexo.

Hemos visto como generamos una clasificación a partir de la que nos proporciona *Facebook* pero, para poder hacerla, la red social necesita que le enviemos la puntuación máxima obtenida por el usuario. Nosotros hemos implementado la función *publicarPuntos* de la figura 44, la cual se encarga de comprobar, cada vez que el usuario termina una partida, si la puntuación obtenida es mayor que la guardada en las bases de datos de *Facebook* y, en caso afirmativo, la actualiza.

```
function publicarPuntos(puntos) { //envio puntos a app en facebook
    FB.api('/me/scores/', function (res) {
     // console.log(res);
        if(res.data.length == 0 || res.data[0].score < puntos){
          FB.api('/me/scores/', 'post', {score: puntos}, function (res) {/* console.log(res);*/});
        }
    });

}
```

Figura 44: *Código de la función publicarPuntos.*

Tanto el generar la clasificación como el envío de puntos forman parte de los *score* de *Facebook*. Estos son una funcionalidad única de la API Graph de *Facebook* reservada a los juegos por lo que si nuestra aplicación no lo es, no podremos utilizarlos.

Al terminar una partida del juego, el usuario tiene la posibilidad de compartir en su muro de *Facebook* la puntuación que ha obtenido, siempre y cuando nos haya proporcionado el permiso para ello. Para realizar esta acción es necesario establecer una comunicación con la red social y enviarle los datos para que publique el logro del usuario. Nosotros hemos hecho posible esto con la función *compartir* (véase la figura 45). Así se genera automáticamente una historia de *Facebook* y muestra al usuario la interfaz nativa de la red social para compartirla como vemos en la figura A15 del anexo.

```
function compartir(puntuacion){

  FB.ui({
        method: 'feed',
        link: "https://www.lostrunnertfg.tk",
        description: "He conseguido una puntuación de "+puntuacion+ " en Lost Runner",
        picture: "https://www.lostrunnertfg.tk/images/iconoweb.png",
        name: "¡Observad!"
    }, function (response) {
    });

}
```

Figura 45: *Código de la función compartir.*

Finalmente, solo queda comentar la última interacción de nuestra aplicación con *Facebook*: que el usuario pueda invitar a sus amigos a probar el mini juego. Para que el usuario pueda invitar a sus amigos es necesario que nos dé el permiso *friend_list* al iniciar la aplicación. Las funciones que hacen posible todo el proceso de invitar son:

- ***invitaAmigo***: esta función se encarga de obtener la lista de los amigos del usuario que no están jugando y rellenar la lista del formulario de invitar amigos de la aplicación. Podemos ver la función en la figura A16 del anexo.
- ***enviarInv***: en esta función cogemos los amigos seleccionados del formulario y enviamos una invitación a jugar mediante la función *simpleReq* de la figura 46.

```
function enviarInv() {
    var check = [];
    $('#invForm input:checked').each(function (i, el) {
        check[i] = el.value;
    });
    datafb.invitados = check;
    if (check.length !== 0) {

        simpleReq(check,"¡Ven y diviértete con nosotros en LostRunner");
    }
}
```

Figura 46: *Código de la función enviarInv.*

- ***simpleReq***: es la función más básica de invitaciones de la API. En ella le mandamos un destinatario y un mensaje a *Facebook* y este realiza la petición.

```
function simpleReq(to, msg) {
    var options = {
        method: 'apprequests',
        message: msg
    };
    if (to)
        options.to = to;
    FB.ui(options, function (response) {
        // console.log(response);
    });
}
```

Figura 47: *Código de la función simpleReq.*

3.5 Costes de desarrollo y mantenimiento de la aplicación

En esta parte de la memoria presentamos un presupuesto del coste económico y temporal de realizar el proyecto. En nuestro caso hemos podido realizar todo el trabajo utilizando las versiones de prueba y gratuitas del servidor, del dominio y del certificado SSL. Además, dado que hemos creado nosotros mismos todo el contenido multimedia (imágenes y *sprites)*, solo tendremos que contabilizar el coste temporal del trabajador al que le asignásemos el proyecto.

Aunque a nosotros no nos ha supuesto un coste económico el integrar la aplicación en *Facebook* incluiremos en el presupuesto el coste que tendríamos el mantener el servicio una vez superásemos el tiempo de prueba de cada servicio utilizado.

Servicio	Tiempo sin costes	Costes
Instancia t2.micro AWS	750 horas mensuales durante 1 año	104,83 €/año
Dominio .tk	1 año	8,22 €/año
Certificado SSL Comodo	90 días	76,95 €/año
		190 €/año

Figura 48: *Presupuesto anual de mantenimiento de la aplicación en Facebook.*

En la tabla no está incluido el tiempo de desarrollo invertido, aproximadamente unas 100 horas, por lo que tendríamos que añadir el coste derivado de mantener un programador en el proyecto al presupuesto.

4. Estudio de la evolución de la aplicación

En esta sección de la memoria vamos a realizar un estudio de la evolución de nuestra aplicación durante el primer mes de su lanzamiento en la red social *Facebook.* Todos los datos que utilizamos para realizar el estudio los hemos obtenido de la herramienta de análisis que nos proporciona la misma red, *Facebook analytics for App* (Facebook developers, 2016).

4.1 Herramientas de control: Facebook analytics

Facebook analytics for App es una herramienta de análisis gratuita que nos proporciona la red social *Facebook* para poder controlar con exactitud la evolución de una aplicación tras su lanzamiento.

Con los datos que nos proporciona podemos saber con la edad media, género, país y lengua de los usuarios que la utilizan además, entre otros, de qué dispositivo utilizan para conectarse, el número de accesos a la aplicación y su continuidad.

Gracias a los datos obtenidos sabremos con más precisión el tipo de usuario que utiliza la aplicación, permitiéndonos crear y ajustar campañas publicitarias para captar un nicho más específico de gente y se podrá mejorar la experiencia del usuario actual de la aplicación en base a su comportamiento dentro de la misma, aumentado así el factor de conversión y mejorando el tiempo de vida de esta.

En definitiva, *Facebook analytics for App* es una herramienta que facilita el poder conocer y controlar la audiencia de nuestra aplicación, comprobar la existencia de posibles defectos y aplicar las mejoras necesarias para incrementar los ingresos a través del uso de la aplicación en nuestra empresa.

4.2 Evolución de la aplicación y resultados

A finales del mes de febrero del año 2016 creamos la aplicación *Lost Runner* en la red social *Facebook* para poder integrar la aplicación homónima que habíamos desarrollado. No obstante, no fue hasta el 25 de mayo del mismo año que, tras varios ajustes y pruebas, permitimos su acceso al público.

Seguidamente, vamos a realizar un análisis de la evolución de la aplicación desde que se hizo pública hasta finales del mes de junio del mismo año mediante los datos y estadísticas que hemos obtenido en la herramienta *Facebook analytics*.

Al acceder a *Facebook analytics* desde nuestra aplicación podemos ver en la información general los datos estadísticos y gráficas del acceso de los usuarios de esta. La primera gráfica en la que nos fijaremos será la mostrada en la figura 49 sobre las estadísticas de los usuarios activos.

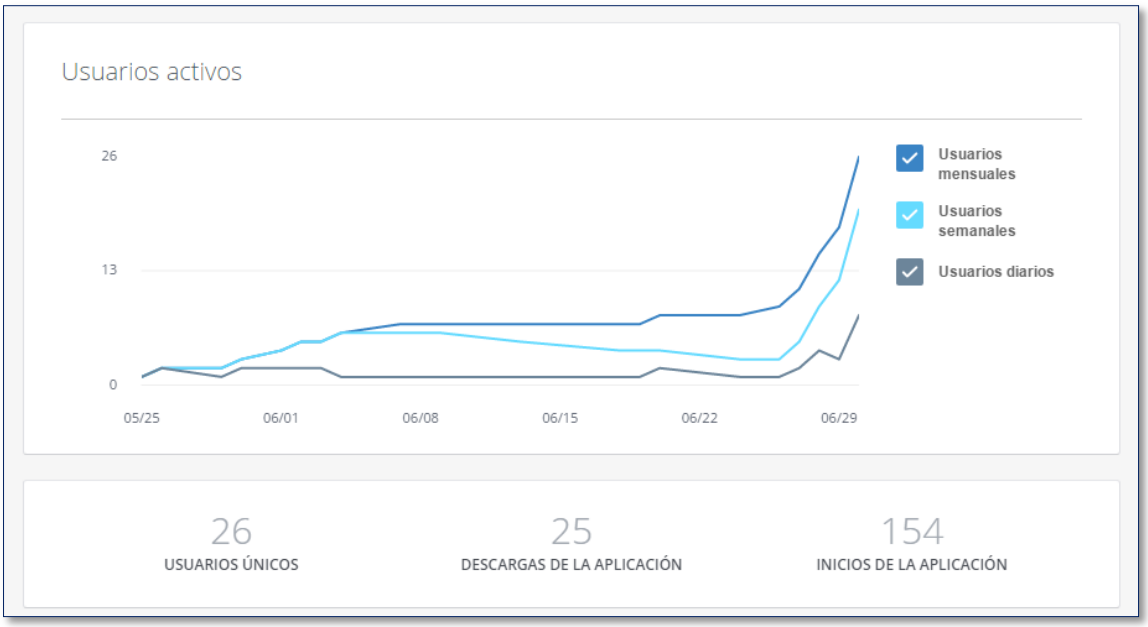

Figura 49: *Gráfico de los usuarios activos en la aplicación Lost Runner.*

A raíz de los datos del gráfico podemos ver que, en el primer mes de hacer pública la aplicación, los accesos a esta no fueron muy numerosos. Este fenómeno es debido a que *Facebook* requiere que una aplicación tenga, al menos, diez usuarios activos el primer mes de publicación para que aparezca en la lista de aplicaciones de su tienda. Nosotros conseguimos obtener trece usuarios gracias a que compartimos nuestra puntuación e invitamos a nuestros amigos a jugar desde la aplicación.

Una vez alcanzado el requisito de la red, el número de usuarios activos se dobla en apenas cuatro días. Al pulsar sobre el gráfico en la herramienta de *Facebook* podremos obtener información más detallada como que los usuarios acceden una media de 5,9 veces a la aplicación, el número de usuarios diarios de usuarios diarios o que acceden desde su ordenador, indicado en la gráfica de la figura 50.

Figura 50: *Gráfico de los usuarios diarios de la aplicación Lost Runner.*

Observando el gráfico de la figura 50 podemos deducir qué días del mes compartimos los resultados de una partida des del juego en nuestro muro, ya que se incrementa ligeramente el número de usuarios únicos activos en los días siguientes, aunque no es necesario especular esta información puesto que podemos consultarla.

En *Facebook analytics* podemos consultar el impacto que tiene una historia[5] en la red social cada vez que se comparte en el muro. Para ello, nos dirigiremos a la sección *Plataforma de Facebook >Historias* en el menú de la herramienta. Como podemos ver en la figura 51, esta nos proporciona las fechas y datos relativos a cada historia publicada.

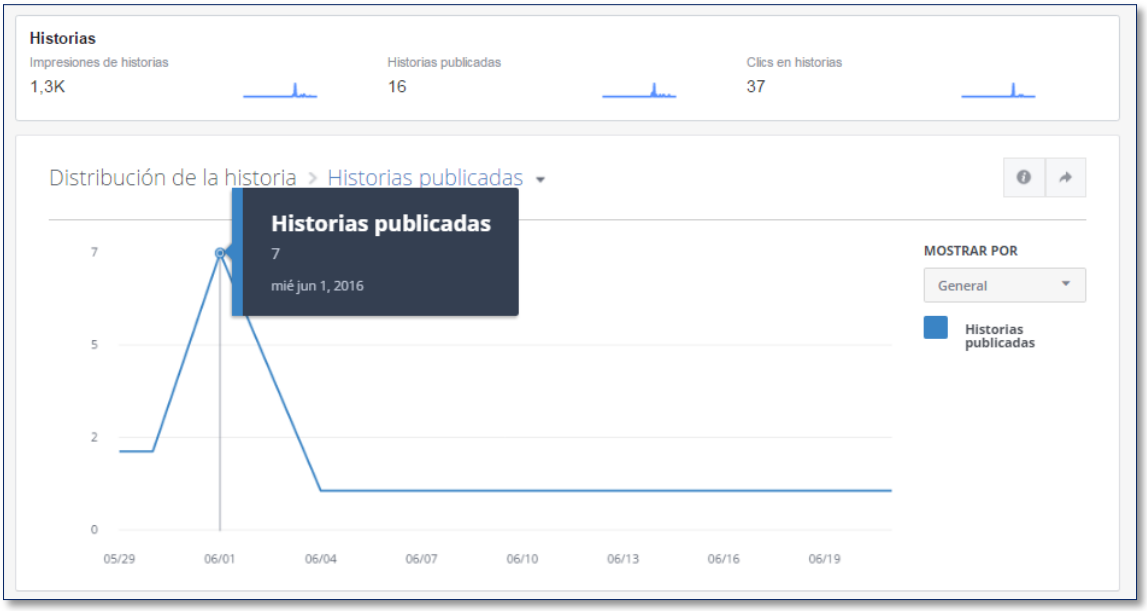

Figura 51: *Gráfico de las historias publicadas desde Lost Runner.*

En la figura anterior podemos ver que se han publicado un total de dieciséis historias desde la aplicación y que estas han sido vistas aproximadamente mil trescientas veces. Para fomentar el ingreso de usuarios, compartimos en nuestro muro una historia una vez al día, con lo que el resto de historias son de otros usuarios.

El día uno de junio fue cuando más historias se compartieron desde la aplicación ya que decidimos compartir nuestra puntuación en un grupo de vecinos de aproximadamente dos mil miembros, dando lugar a que se incrementara el número de veces que se vio la publicación hasta alcanzar las mil trecientas impresiones. En la figura 52 de la página siguiente vemos como sólo ese día más de quinientas personas vieron la publicación.

[5] Una historia es un evento que se comparte en el muro de *Facebook* del usuario.

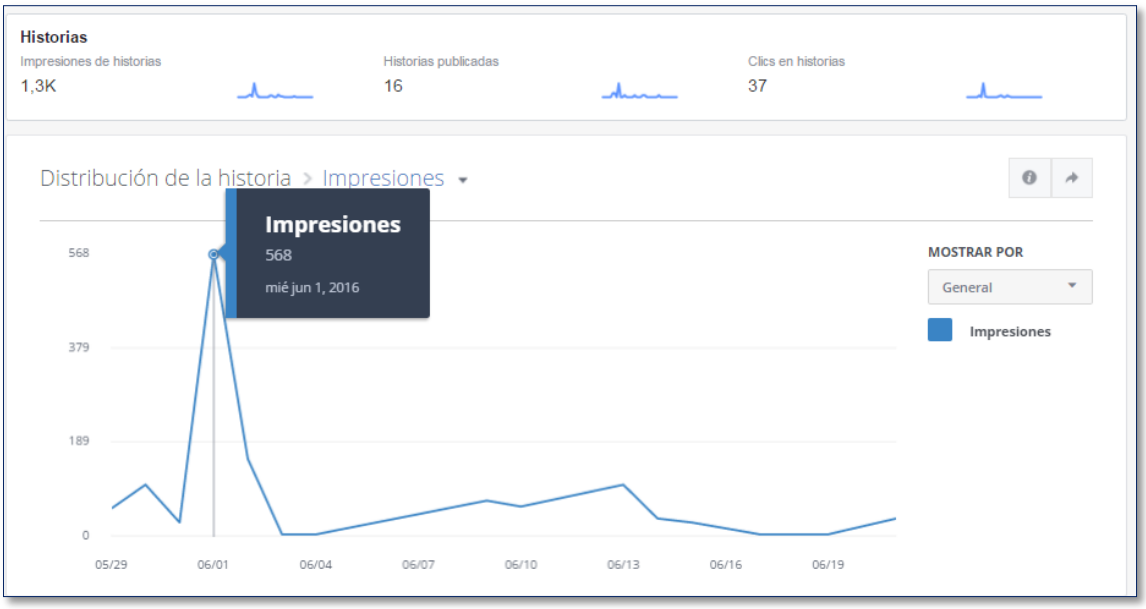

Figura 52: *Gráfico de las impresiones de las historias publicadas desde Lost Runner.*

Siguiendo con el estudio, en la figura 52 podemos ver que, aunque una de las historias fue vista por un elevado número de personas, hemos obtenido únicamente treinta y siete *clicks* en estas, por lo que no han accedido al juego tantas personas como deseábamos.

Si investigamos un poco y cruzamos datos, en el gráfico de la figura 50 se ve como todos los usuarios de nuestra aplicación han accedido al juego desde el ordenador y, según un estudio de la empresa Arnold (Martínez, 2016), la mayoría de usuarios accede a *Facebook* desde el móvil. Este hecho ha afectado al número de accesos debido a las diferencias en la interfaz de la red social: en dispositivos móviles no aparece un botón jugar en la historia, mientras que en un ordenador sí. Claramente, esto ha producido que un público con pocos conocimientos tecnológicos no supiese acceder a la aplicación. Podemos ver este hecho en la figura 53.

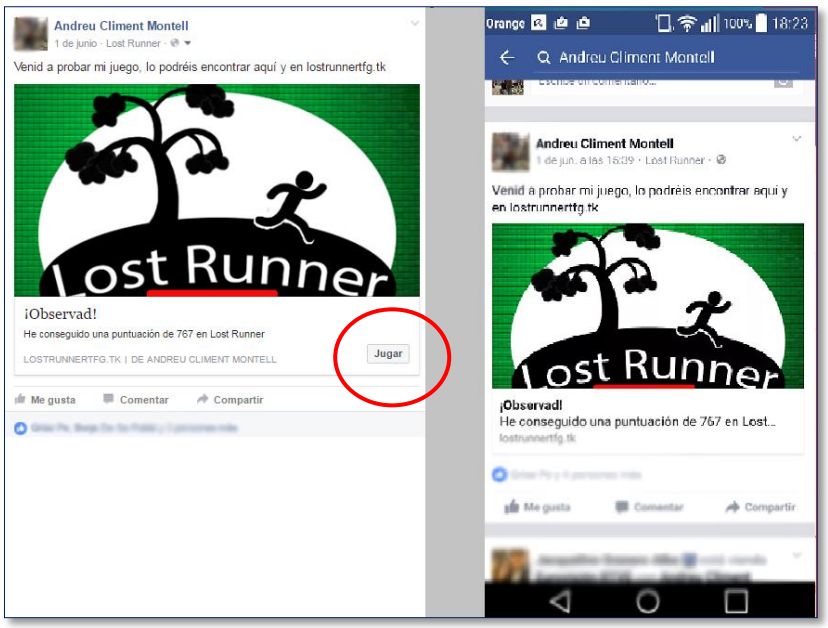

Figura 53: *Comparación de la misma historia en Facebook web y Facebook app.*

Continuando con el análisis, otro de los datos que podemos ver en *Facebook analytics* es la actividad de inicios de sesión en *Facebook* en nuestra aplicación. En la figura 54 podemos observar que el primer día de junio fue cuando tuvimos más usuarios conectados en la aplicación, coincidiendo con la publicación de la historia comentada anteriormente.

Figura 54: *Gráfico de la actividad de inicios de sesión desde Lost Runner.*

Así pues, para concluir nuestro análisis, y teniendo en cuenta que no hemos invertido en publicitar nuestra aplicación, podemos decir que los resultados obtenidos son satisfactorios. Aunque solo hemos conseguido que cuarenta y dos usuarios hayan entrado en la aplicación, hemos conseguido que unas mil trescientas personas sepan que existe y, si observamos los gráficos anteriores, hay una tendencia a que consigamos más usuarios en los próximos meses, confirmando nuestra hipótesis de que *Facebook* es un buen canal para distribuir y dar a conocer nuestra aplicación.

4.3 Comparativa con otras aplicaciones en Facebook

Continuando con el estudio de la evolución de la aplicación en *Facebook*, vamos a comparar los resultados obtenidos el primer mes después del lanzamiento de la aplicación con los mismos resultados de las aplicaciones Bechart y Juego de Bolsa de la empresa Trading People S.L.

Al realizar dicha comparación de datos, debemos tener en cuenta que las dos aplicaciones de la empresa Trading People S.L. han tenido una campaña publicitaria que ha acompañado sus lanzamientos en *Facebook*, dando lugar a una mejora en la adquisición de usuarios en la red.

El primer dato que comparamos son los usuarios activos del primer mes. Con una campaña publicitaria, donde el coste aproximado de adquisición fue de ocho euros por usuario, la aplicación Juego de bolsa obtuvo más de novecientos usuarios únicos (véase el gráfico de la figura 55). En cuanto a la aplicación Bechart, al no tener una campaña publicitaria tan agresiva detrás de ella, obtuvo solamente veintiocho usuarios, tal y como vemos en la figura 56.

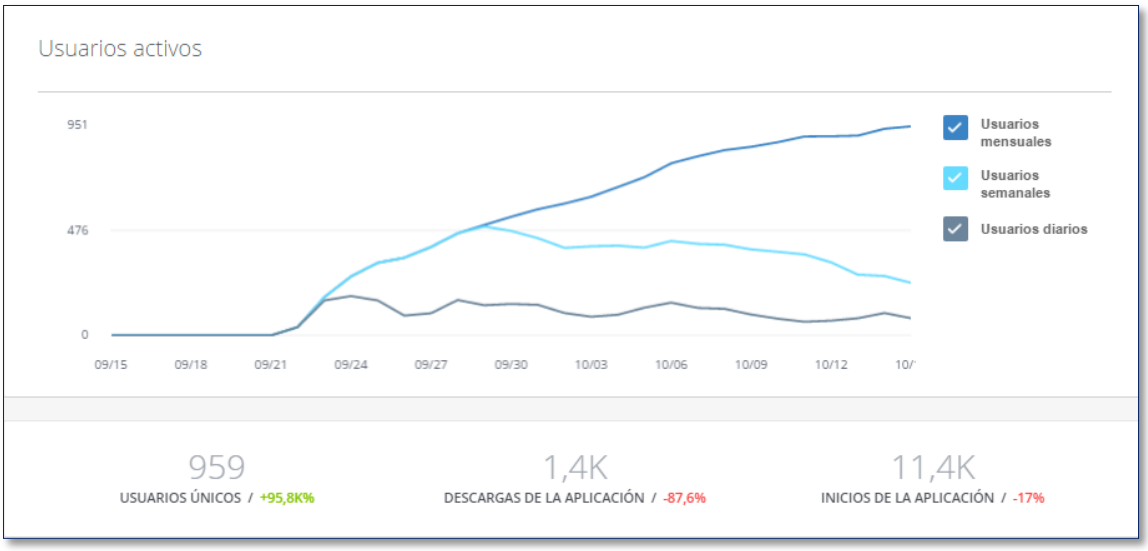

Figura 55: *Gráfico de los usuarios activos de la aplicación Juego de Bolsa.*

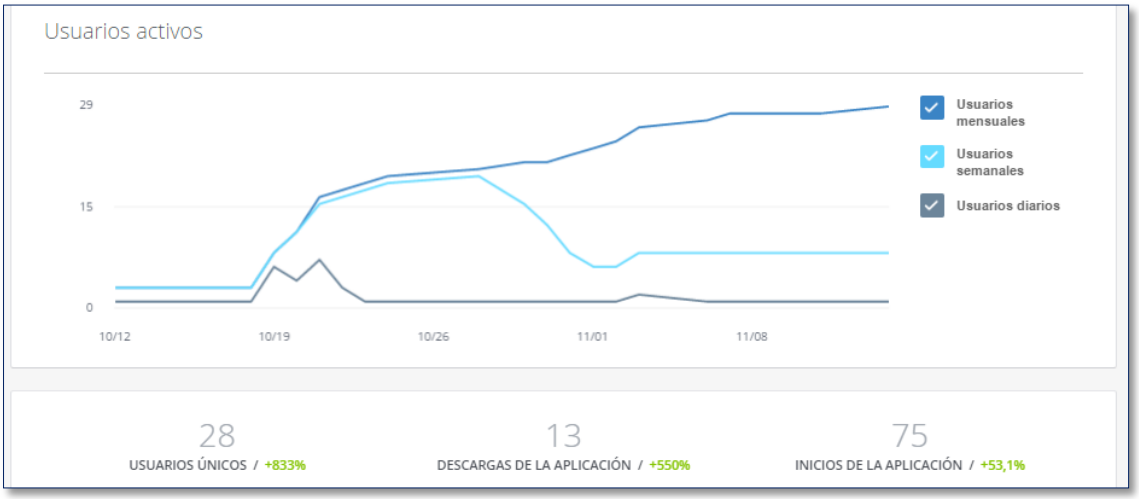

Figura 56: *Gráfico de los usuarios activos de la aplicación Bechart.*

Si comparamos los usuarios obtenidos de Bechart con los de nuestra aplicación (véase la figura 50 del apartado anterior), vemos que, si no contamos con una campaña publicitaria, el número de usuarios es aproximadamente el mismo. Por el contrario, apoyar nuestro lanzamiento con una inversión en publicidad va a suponer un incremento sustancial en el número de usuarios.

A continuación, compararemos los usuarios activos por día, por lo que nos fijaremos en las figuras 57 y 58. Al observar sus divergencias, se observa como la aplicación Juego de Bolsa consiguió un máximo de ciento ochenta usuarios conectados un mismo día, mientras que a la aplicación Bechart tuvo un máximo de siete usuarios.

Figura 57: *Gráfico de los usuarios diarios de la aplicación Juego de Bolsa.*

Figura 58: *Gráfico de los usuarios diarios de la aplicación Bechart.*

Comparándolas con nuestra aplicación, el Juego de Bolsa tiene, obviamente, unos registros mejores que los nuestros al tener publicidad. Con respecto a la aplicación Bechart, *Lost Runner* obtiene un mejor resultado, consiguiendo un máximo de once usuarios activos frente a los siete de la primera.

Observando el análisis en su conjunto podemos decir que nuestra aplicación obtiene buenos resultados si la comparamos con otra aplicación que no ha utilizado ningún tipo de campaña publicitaria para darla a conocer, mientras que si la comparamos con otra aplicación con una campaña respaldándola, obtenemos unos resultados muy desfavorables.

A la vista de esta situación, podemos afirmar que *Facebook* es un buen canal para dar a conocer nuestra aplicación y que si lo respaldamos con una campaña publicitaria, obtendremos fácilmente un millar de usuarios en menos de un mes.

5. Limitaciones y posibles mejoras del trabajo

Durante el transcurso del proyecto ha habido momentos que, tanto por desconocimiento como por falta de experiencia, nos hemos topado con verdaderos retos. No obstante, hemos podido superarlos, aunque eso no significa que nuestra solución sea la mejor.

En el caso de la aplicación que hemos desarrollado para *Facebook*, nos resultó muy complicado implementar una lógica capaz de recrear la física del terreno con la que pudiéramos calcular correctamente las colisiones del corredor.

En cuanto al estudio de la evolución de la aplicación en la red social, nuestro principal problema o limitación ha sido la falta de previsión de tiempo ya que desconocíamos que el juego tardaría aproximadamente un mes en aparecer en las listas de aplicaciones de *Facebook*, por lo que los resultados del estudio se han visto afectados.

Existen varias mejoras que podríamos realizar en la aplicación debido a que es una versión *beta* estable. Algunas de ellas son:

- Mejorar la lógica del cálculo de colisiones y física del juego.
- Reestructurar el diseño de la aplicación para utilizar PHP y así poder guardar el registro del usuario mediante *cookies* y reducir las conexiones a *Facebook.*
- Crear una base de datos con la que poder almacenar la información de los usuarios con fines comerciales o informativos.
- Implementar una mejor forma de reiniciar el juego sin tener que recargar la página.

Así pues, el estudio que hemos realizado se podría mejorar añadiendo una comparación en la que viésemos la evolución de la aplicación utilizando la red social *Facebook* y técnicas SEO SEM.

6. Conclusiones

Tras finalizar el trabajo que aquí se presenta y teniendo en cuenta los objetivos que nos marcamos al principio, consideramos que se han cumplido todos ellos exitosamente. Hemos realizado una investigación y aprendizaje profundos sobre las tecnologías implicadas, además de las ventajas que supone utilizar la red social de *Facebook* como lanzadera de nuestro juego, *Last Runner*.

En cuanto a los objetivos específicos que nos habíamos propuesto, podemos decir que los hemos alcanzado todos de forma satisfactoria. Creemos que hemos adquirido los conocimientos y capacidades necesarias para elaborar un producto atractivo e interesante y poder ofrecerlo al público utilizando la plataforma de juegos de la red social. Gracias a ella, se ha podido analizar de forma exhaustiva la evolución de la aplicación mediante su uso.

Con todo esto, hemos obtenido la experiencia suficiente para crear, en un futuro, unas aplicaciones más complejas y de mayor calidad, respetando las demandas de los usuarios a la vez que obtenemos unas respuestas del comportamiento de estas y del público en tiempo real desde la herramienta de análisis de la red misma social.

Aunque llevar a término el trabajo de final de grado ha resultado ser una tarea compleja y extensa, dado que abarca diferentes campos de la informática, tales como la programación y la gestión de sistemas, además del análisis y documentación de los resultados, ha sido una experiencia gratificante a la vez que formativa. Con ello, no descartaría continuar con el desarrollo del juego hasta obtener beneficios de este y llevarlo a su máximo exponente.

Finalmente, quisiéramos remarcar la importancia de las redes sociales a la hora de darnos a conocer y crearnos un currículo en el ámbito del desarrollo de aplicaciones web. Siendo esto de gran utilidad a la hora de ampliar nuestras salidas profesionales, mientras nos aporta diversas opciones de trabajo autónomo y emprendedor, las cuales son de gran interés en nuestro Grado de Ingeniería Informática.

Bibliografía y referencias bibliográficas

- Acerca de HTML (2016). *¿Qué es y para qué sirve HTML?* Recuperado el 25 de junio del 2016. http://www.acercadehtml.com/manual-html/que-es-html.html

- Amazon Web Services (2016). *Capa gratuita de AWS.* Recuperado el 15 de mayo del 2016. https://aws.amazon.com/es/free/?nc2=h_l2_cc

- Amazon Web Services (2016). *Lanzamiento de una máquina virtual de Linux.* Recuperado el 18 de junio del 2016 https://aws.amazon.com/es/getting-started/tutorials/launch-a-virtual-machine/

- Certsuperior (2016). *¿Qué es un certificado SSL?* Recuperado el 11 de junio del 2016. https://www.certsuperior.com/QueesunCertificadoSSL.aspx

- Comodo (2016). *About us.* Recuperado el 21 de junio del 2016. https://www.comodo.com/about/comodo-company-profile.php

- Comodo (2016). *Certificate Installation: Apache & mod_ssl*. Recuperado el 20 de junio del 2016. *https://support.comodo.com/index.php?/Default/Knowledgebase/Article/View/637/66/*

- Desarrolloweb (2016). *Manual de jQuery*. Recuperado el 7 de junio del 2016. http://www.desarrolloweb.com/manuales/manual-jquery.html

- Dot TK (2016). *Free Domains For All.* Recuperado el 18 de mayo de 2016. http://www.dot.tk/es/index.html

- EaselJS (2016). *EaselJS Module.* Recuperado el 30 de mayo de 2016. http://createjs.com/docs/easeljs/modules/EaselJS.html

- EcuRed (2016). *Servidor Web.* Recuperado el 12 de junio del 2016. http://www.ecured.cu/Servidor_Web

- Eguiluz, J. (2016). *Introducción a CSS. Capítulo 1. Introducción.* Recuperado el 6 de junio del 2016. https://librosweb.es/libro/css/capitulo_1.html

- Eguiluz, J. (2016). *Introducción a JavaScript. Capítulo 1. Introducción.* Recuperado el 6 de junio del 2016. https://librosweb.es/libro/javascript/capitulo_1.html

- Ellingwood, J. (2014). *¿Cómo configurar Virtual Host de Apache en Ubuntu 14.04 LTS?* Recuperado 18 de junio del 2016. https://www.digitalocean.com/community/tutorials/como-configurar-virtual-host-de-apache-en-ubuntu-14-04-lts-es

- Facebook developers (2016). *Documentación.* Recuperado el 8 de junio de 2016 https://developers.facebook.com/docs

- Facebook developers. (2016). *Analytics | Overviews*. Recuperado el 2 de julio del 2016. https://developers.facebook.com/products/analytics

- Facebook developers (2016). *API Graph. Información general sobre la API Graph.* Recuperado el 8 de junio del 2016. https://developers.facebook.com/docs/graph-api/overview

- Facchin, J. (2015). *Lista de ventajas y desventajas de las redes sociales para las empresas*. Recuperado el 8 de septiembre del 2015. http://josefacchin.com/2015/09/08/ventajas-de-las-redes-sociales-para-empresas

- Freenom (2016). *Panel de administración de nuestro dominio.* Recuperado el 25 de junio de 2016. https://my.freenom.com

- GlobalSign (2016). *What is an SSL Certificate?* Recuperado el 11 de junio del 2016. https://www.globalsign.com/en/ssl-information-center/what-is-an-ssl-certificate-2/

- Hernández, J. (2013). *10 beneficios de crear aplicaciones con Facebook.* Recuperado el 10 de octubre del 2015 http://www.marketaria.es/blog/redes-sociales-2/10-beneficios-de-crea-aplicaciones-en-facebook

- jQuery (2016). *API documentation.* Recuperado el 7 de junio del 2016 http://api.jquery.com

- Letelier, P. (2003), *Proyecto Docente e Investigador*. *Proceso de desarrollo del software*. Recuperado el 14 de junio del 2016. www.dsic.upv.es/asignaturas/facultad/lsi/doc/IntroduccionProcesoSW.doc

- López Franco, J.M. (2001). *Integración de tecnologías a través de servidores Web. Punto1.2. Servidores Web.* Recuperado el 12 de junio del 2016. http://trevinca.ei.uvigo.es/~txapi/espanol/proyecto/superior/memoria/node20.html

- Martínez, G. (2015). *Las 10 estadísticas más interesantes de Facebook, Twitter y LinkedIn*. Recuperado el 22 de junio de 2016. http://blog.arnoldmadrid.com/las-10-estadisticas-mas-interesantes-de-facebook-twitter-y-linkedin

- Mozilla Developer Network (2015). *Guía de referencia de tecnologías web.* Recuperado el 6 de junio del 2016. https://developer.mozilla.org/es/docs/Web/Reference

- Mozilla Developer Network (2016). *HTML*. Recuperado el 6 de junio del 2016. https://developer.mozilla.org/es/docs/Web/HTML

- Mozilla Developer Network (2016). *JavaScript*. Recuperado el 6 de junio del 2016. https://developer.mozilla.org/es/docs/Web/JavaScript

- Otto, M., y Thornton, J., Eguiluz, J (traductor) (2016). *Bootstrap 3, el manual oficial.* Recuperado el 10 de mayo del 2016. https://librosweb.es/libro/bootstrap_3

- Reference* (2004). *What is web technology?* Recuperado el 6 de junio del 2016. https://www.reference.com/technology/technology-eebf0fb1e023190a#

- Rubén Andrés (2014). *¿Qué es y para qué sirve el dominio de tu página web?* Recuperado el 11 de junio del 2016. http://computerhoy.com/noticias/internet/que-es-que-sirve-dominio-tu-pagina-web-22007

- Schou, A. (2015). *Estudio sobre los usuarios en las redes sociales 2015 en España – IAB.* Recuperado el 10 de febrero del 2016. http://andreasschou.es/2015/02/estudio-sobre-usuarios-redes-sociales-2015-espana

- Skinner,G (2016). *CreateJS | A suite of JavaScript libraries and tools designed for working with HTML5.* Recuperado el 1 de julio del 2016. http://www.createjs.com

- Wikipedia (2015). *HTML.* Recuperado el 6 de junio del 2016. https://es.wikipedia.org/wiki/HTML

- Wikipedia (2015). *Hoja de estilos en cascada.* Recuperado el 6 de junio del 2016. https://es.wikipedia.org/wiki/Hoja_de_estilos_en_cascada

- Zephoria Digital Marketing (2016). *The Top 20 Valuable Facebook Statistics – Updated May 2016.* Recuperado el 29 de mayo del 2016. https://zephoria.com/top-15-valuable-facebook-statistics

Anexo

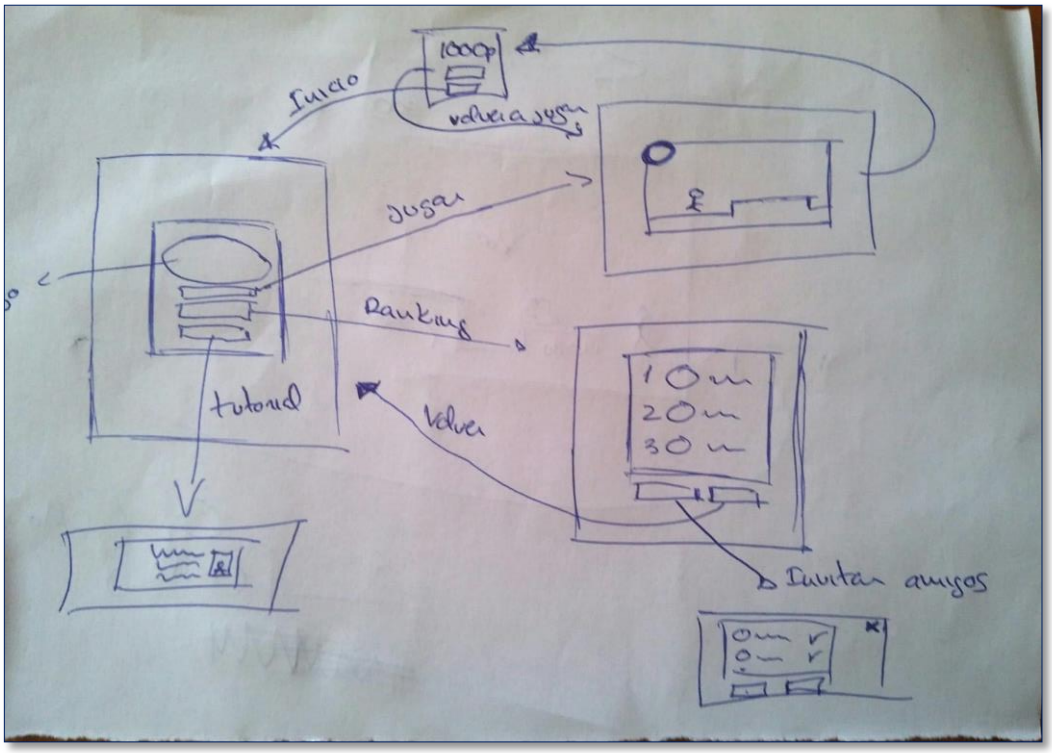

Figura A1: *Boceto del diseño de la aplicación.*

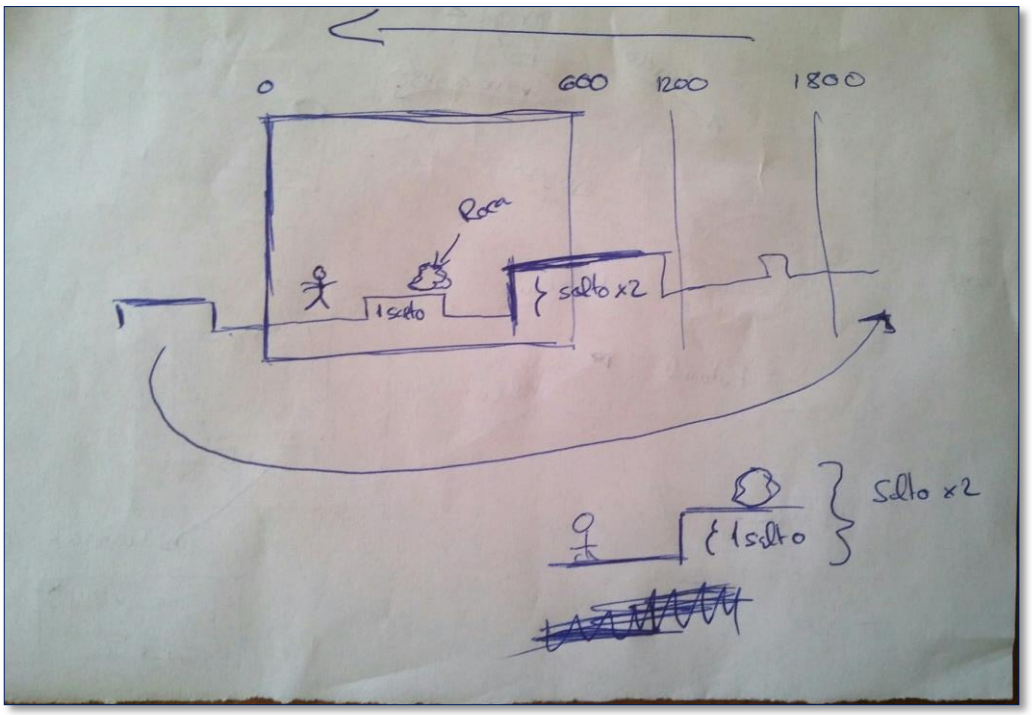

Figura A2: *Boceto del diseño y funcionamiento del mini juego.*

```
<!DOCTYPE html>
<html>
  <head>
     <meta charset="utf-8"/>
     <title>Lost Runner</title>
  </head>
  <body>
    <h1>Pol&iacute;ticas y condiciones de uso</h1>
    <p>
        Dado que esta app se trata de un proyecto de final de carrera no guardamos
        ning&uacute;n dato relativo al usuario que se conecta en ella.
    </p>
    <p>
        Los &uacute;nicos datos que obtenemos son las propias estad&iacute;sticas
        de la app proporcionadas por Facebook.
    </p>
    <br>
    <p>Espero que disfrutes de la app tanto como yo creandola. Atentamente</p>
    <p>ANDREU</p>
  </body>
</html>
```

Figura A3: *Código completo del archivo policy.html*

```
<!DOCTYPE html>
<html>
  <head>
     <meta charset="utf-8"/>
     <title>Lost Runner</title>
  </head>
  <body>
    <p>Lo siento ha ocurrido un error, prueba a conectarte más tarde.</p>
    <p>
        Si el error continua envia un email a anclimon88@gmail.com,
        te responderé lo antes posible.
    </p>
  </body>
</html>
```

Figura A4: *Código completo del archivo error.html*

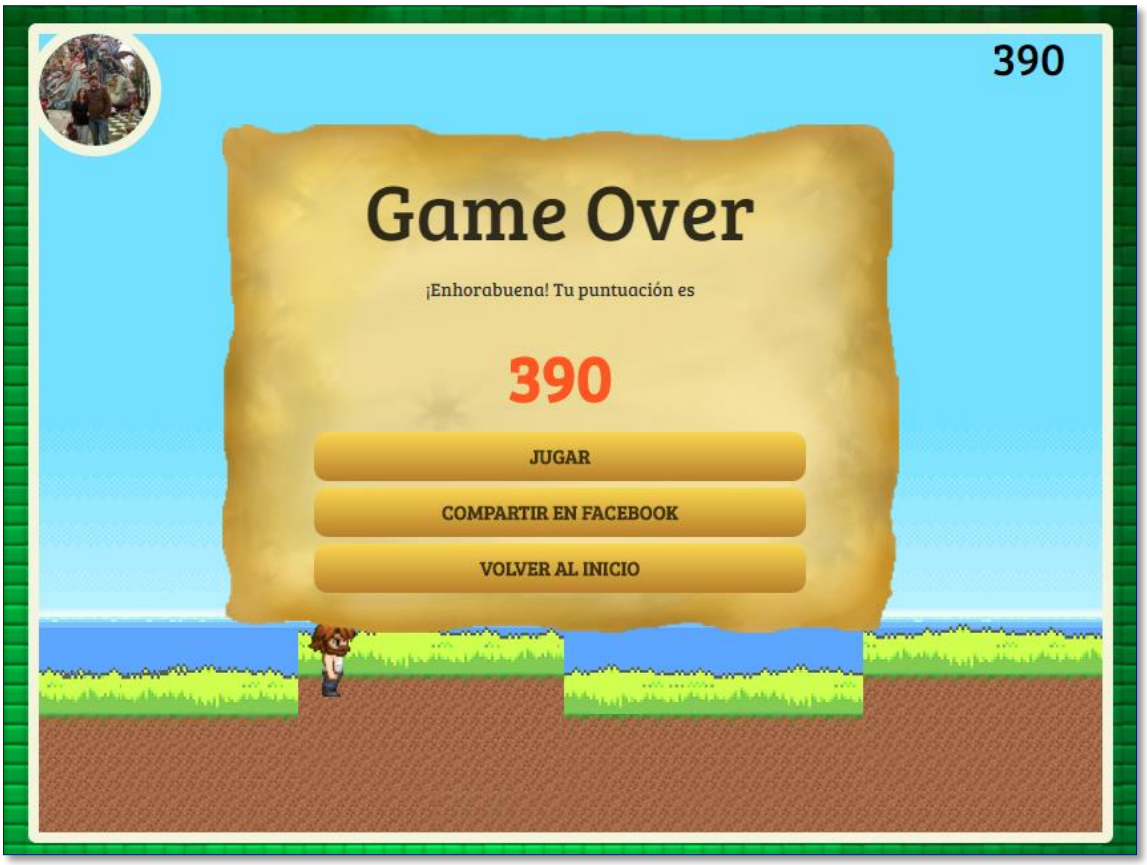

Figura A5: *Ejemplo del cuadro de diálogo de fin de partida.*

```
<body>
<div class="container">
    <div id="contenido" class="col-sm-12">

        <div id="btnMuro" class= "col-sm-10 col-sm-push-1">
            <div id="titulo">
                <img src="/images/titulo.png" alt="Lost Runner" class="center-block">
            </div>
            <div id="fb-welcome"></div>
            <div id="botones" class="center-block">
                <button class=" btn btn-success center-block" onclick="iraJugar()">Jugar</button>
                <button class=" btn btn-success center-block" onclick="verrank()">Hall of Fame</button>
                <button class=" btn btn-success center-block" onclick="vertuto()">Tutorial</button>
            </div>
        </div>

        <div id="tuto" style="display:none" class= "col-sm-10 col-sm-push-1">
            <div class = "center-block">
            <h1> Lost Runner </h1>
            <img src="/images/pantalla.png" alt="pantalla del juego" class="center-block">
            <p>El único propósito de Jake es escapar de la isla en la que está atrapado, para ello, debe salir corriendo.
            Tu objetivo en el juego es evitar que Jake tropiece con los desniveles pulsando una tecla cualquiera de tu teclado.
            </p>
            <h3> ¡Corre, salta y demuéstranos de lo que estás hecho!</h3>
            <button onclick="vertuto()" class=" btn btn-success center-block">Volver</button>
            </div>
        </div>

        <div id="rank" style="display:none" class= "col-sm-12">
            <div id="ramigos">
            </div>
            <br>
            <div class="col-sm-8 col-sm-push-2 col-md-6 col-md-push-3 rbotones">
                <button class="btn btn-info" onclick = "invitar()">Invita a tus amigos</button>
                <button class="btn btn-success" onclick="verrank()">Volver</button>
            </div>
        </div>

        <div id="invitaciones" class="col-sm-8 col-sm-push-2" style="display:none">
            <h2> Invita a tus amigos </h2>
            <input id="checktot" type="checkbox" onclick="checkTodos()"/><span style="color:white;">Marcar todos</span>
            <form id="invForm">
            </form>
            <div class="btns">
                <button class=" btn btn-info" onclick="enviarInv()"> Invitar a mis amigos</button>
                <button class="btn btn-warning" onclick="$('#invitaciones').toggle()"> Cerrar </button>
            </div>
        </div>
```

Figura A6: *Código del contenido del archivo index.html*

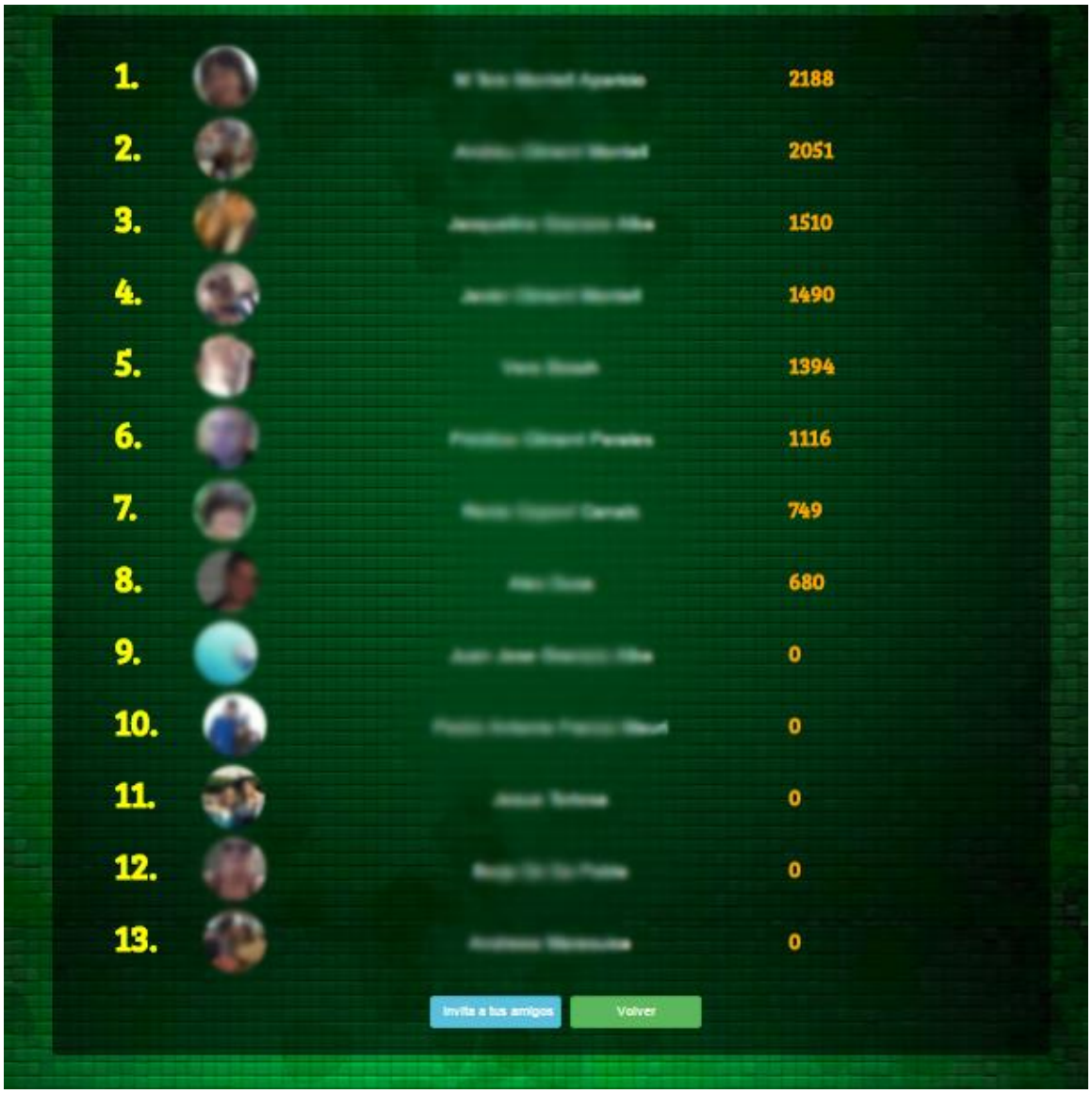

Figura A7: *Clasificación del usuario.*

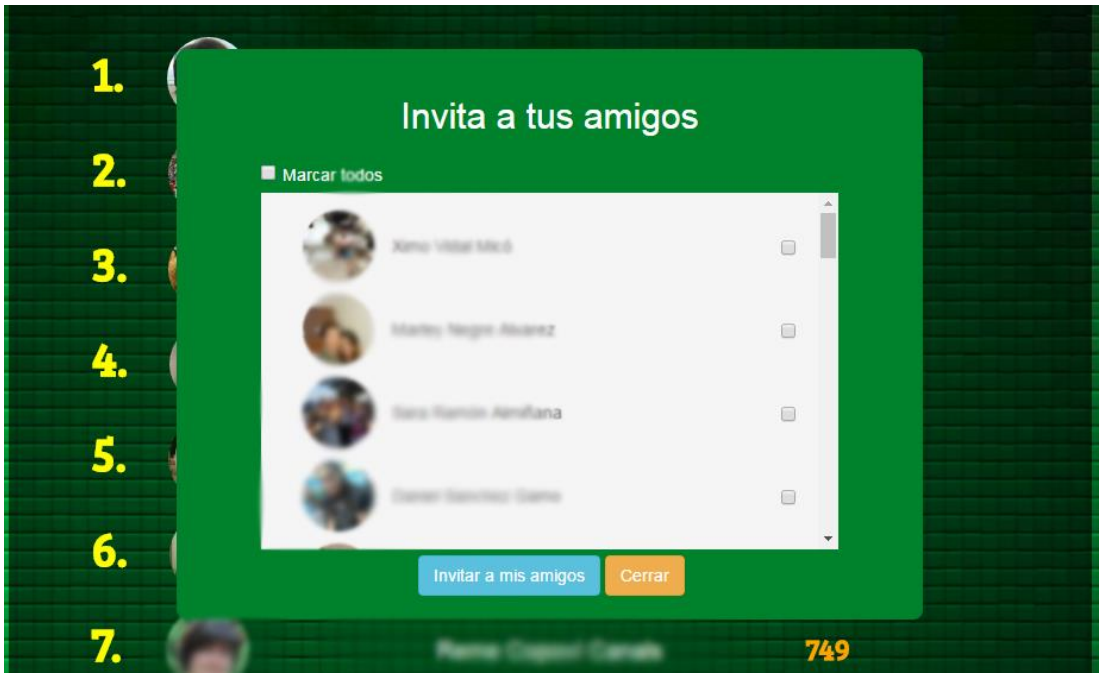

Figura A8: *Formulario de invitar amigos al mini juego.*

Figura A9: *Tutorial de la aplicación.*

```
1
2   (function(scope){
3       function Cargador(){
4           this.initialize();
5       }
6
7       var cargadas = Cargador.prototype;
8       var totales = Cargador.prototype;
9
10      Cargador.prototype.initialize = function(){
11          cargadas = 0;
12          totales = 0;
13      }
14
15      Cargador.prototype.loadImagenes = function(lista){
16          cargadas = 0;
17          totales = lista.length;
18
19          for (var i = 0; i < lista.length; i++) {
20              this.cargaImagen(lista[i]);
21          }
22      }
23
24      Cargador.prototype.cargaImagen = function(url){
25
26          var self = this,
27              image = new Image();
28
29          this[url] = image;
30          image.onload = function(e){
31              self.imagenCargada(e);
32          };
33
34          image.src = image.url = url;
35      }
36
37      Cargador.prototype.imagenCargada = function(e){
38
39          cargadas ++;
40          //console.log("imagen cargada: " + cargadas + " de " + totales);
41
42          if(cargadas === totales){
43              if ( this.onComplete ){
44                  this.onComplete();
45              }else{
46                  console.log("error: onComplete no definido en Cargador");
47              }
48          }
49      }
50
51      scope.Cargador = Cargador;
52
53  }(window));
```

Figura A10: *Código completo de la clase Cargador.*

```
createjs.Ticker.setFPS(20);
createjs.Ticker.on("tick", function (e) {


    hero.y += hero.velocity.y;

    for(i=0; i<colisionables.length; i++){
        var baux = colisionables[i].getBounds();
        var colbound = {"x":colisionables[i].x, "y":colisionables[i].y, "width":baux.width, "height":baux.height };

        if(colbound.x > -1 && colbound.x < 601) {
            colision = detectaColision(limit,hero,colisionables[i]);

            var condicion = colbound.x+colbound.width;
            if(condicion >= hero.x   ){
                col[i] = true;
                //console.log(col,colbound.x);
            }

        }

    }

    self.calculaSuelo(hero,colision);


    if(ensuelo && hero.currentAnimation != "run"){
        hero.gotoAndPlay("run");
    }

    var random = Math.random(), bounds1 = nube1.getBounds(), bounds2 = nube2.getBounds();

    if(stop1 == false && stop2 == false){
        X1 = Math.floor(Math.random()*5)+1,
        X2 = Math.floor(Math.random()*5)+1;
        stop1 = true;
        stop2 = true;
    }

    nube1.x -= X1 + accel;
    nube2.x -= X2 + accel;
    //console.log('nube1: '+nube1.x, 'nube2: '+nube2.x);

    if(nube1.x < - bounds1.width){
       nube1.x = ancho + (ancho/2)*random;
       nube1.y = random > 0.5 ? (random * 300) : (random * 150 + bounds1.height);
       stop1 = false

      // console.log('Alto nube1: '+ nube1.y);
    }
```

```
    if(nube2.x < - bounds2.width ){
       nube2.x = ancho + (ancho/2)*random;
       nube2.y = random > 0.5 ? (random * 150) : (random * 300 + bounds2.height);
       stop2 = false;

       //console.log('Alto nube2: '+ nube2.y);
    }

   for(i=0; i< 6; i++){
       suelo[i].x -= 1 + accel;
       var su = suelo[i].getBounds();
       if(suelo[i].x < - su.width){
           var aux = colisionables[suelo[i]];

           // creando desniveles
           var alt = Math.random() > 0.5 ? alto-su.height-30 : alto-su.height;
           if(i==0 || up == 0){
               alt=alto-su.height-30;
               up = 2;
           }
           if(alt == alto-su.height-30){ up ++ ; }else{ up-- ; }

           var dista = ancho-su.width+ 5*450 + Math.random()*50; // 5 -> numero de suelos

           suelo[i].setTransform(dista, alt);
         // console.log(i+": "+suelo[i].x);

           stage.addChildAt(suelo[i],stage.numChildren); // colocamos penultimo para que tape todo lo de detras
           stage.addChildAt(hero,stage.numChildren); // el heroe siempre tiene que estar el ultimo
           colisionables[aux] = suelo[i];//actualizamos los parametros de colision


       //  console.log( suelo[0].y, suelo[1].y, suelo[2].y);
       //  console.log(colisionables[0].y,colisionables[1].y,colisionables[2].y);
       }

   }


   fondo.x  -= 1 + accel;
   fondo2.x -= 1 + accel;

    if (fondo.x < -1066) fondo.x = 0;
    if (fondo2.x < 0) fondo2.x = 1066; //vamos alternando los fondos

    punts += 0.5;

    if( punts % 100 == 0){
        accel ++;
        createjs.Ticker.setFPS(20 + Math.floor(accel/10));

    } // Incrementamos velocidad cada 100 puntos;



    $("#puntuacion").html(Math.floor(punts));

    stage.update();
});
```

Figura A11: *Código completo del ticker en el fichero App.js*

```
App.prototype.calculaSuelo = function(hero,colision){


    if( hero.y < limit ){
        hero.velocity.y += 1;
        ensuelo = false;

    } else {
        hero.velocity.y = 0;
        ensuelo = true;
        hero.y = hero.y > limit ? limit : hero.y; // nos aseguramos que no atraviesa el limite en el eje y

    }


    if(colision){
       //console.log("c: "+colision.x, "s0: "+suelo[0].x, "s1: "+suelo[1].x);
       if(colision.x > -10 && hero.y > 411){
            $("#punts").html(Math.floor(punts));
            $("#fin").toggle();
            publicarPuntos(Math.floor(punts));
            createjs.Ticker.removeAllEventListeners();

           //console.log("Ha chocado, resetear juego");
        }

        if(colision.x > -10){ // heroe cerca del desnivel
          limit = 410;
          stopLimit = true;
          auxi = colisionables.indexOf(colision.t);
        }else{
           console.log("Ha chocado!!"); // no deberia darse esta opcion
            $("#punts").html(Math.floor(punts));
            $("#fin").toggle();
            createjs.Ticker.removeAllEventListeners();

    }else {
        if(stopLimit){

            cont = detectaColision(440,hero,colisionables[auxi]);
            if( cont.x < (colisionables[auxi].getBounds().width-10) || col[auxi]){
                limit = 410;
                col[auxi] = false;
           }else{
                limit = 440;
           }

        }else{
            stopLimit = false;
            col[auxi] = false;

        }
    }

};
```

Figura A12: *Código de la función calculaSuelo del App.js.*

```
for(i=0; i< 6; i++){
    suelo[i].x -= 1 + accel;
    var su = suelo[i].getBounds();
    if(suelo[i].x < - su.width){
        var aux = colisionables[suelo[i]];

        // creando desniveles
        var alt = Math.random() > 0.5 ? alto-su.height-30 : alto-su.height;
        if(i==0 || up == 0){
            alt=alto-su.height-30;
            up = 2;
        }
        if(alt == alto-su.height-30){ up ++ ; }else{ up-- ; }

        var dista = ancho-su.width+ 5*450 + Math.random()*50; // 5 -> numero de suelos

        suelo[i].setTransform(dista, alt);
      // console.log(i+": "+suelo[i].x);

        stage.addChildAt(suelo[i],stage.numChildren); // colocamos penultimo para que tape todo lo de detras
        stage.addChildAt(hero,stage.numChildren); // el heroe siempre tiene que estar el ultimo
        colisionables[aux] = suelo[i];//actualizamos los parametros de colision
```

Figura A13: *Código con el que calculamos la posición de los suelos.*

```
function getRanking(){
  FB.api('/1187789521262715/scores', {fields: 'score,user.fields(name,picture.width(120).height(120))'}, function (response) {

    var lista = "";
    $("#ramigos").empty();

    for(var i=0; i< response.data.length; i++){
      lista += '<div id="pos'+i+'" class="col-sm-12">';
      lista += '<div class="col-sm-3">';
      lista += '<span class="spos">'+ (i+1) +'.</span><img class="center-block" src="'+response.data[i].user.picture.data.url+'">';
      lista += '</div>';
      lista += '<div class="col-sm-6">';
      lista += '<h3>'+response.data[i].user.name+'</h3>';
      lista += '</div>';
      lista += '<div class="col-sm-3">';
      lista += '<span class="spunt">'+response.data[i].score+'</span>';
      lista += '</div>';
      lista += '</div>';
    }

    if(response.data.length == 1){
      lista += '<li><div class="col-sm-12"><small> Invita a tus amigos a jugar para que formen parte del ranking de la app</small></div></li>';
    }

    $(lista).appendTo($("#ramigos"));

    });
```

Figura A14: *Código de la función getRanking.*

Figura A15: *Ejemplo de compartir la puntuación en la aplicación.*

```
var masamigos="";
var numa = 0;
function invitaAmigos() {
    FB.api('/me/invitable_friends'+masamigos,{fields: 'id,name,first_name,picture.width(60).height(60)'}, function (response)
      var aux=response.paging.next;
      if(aux){
        aux = aux.split('/invitable_friends');
        masamigos = aux[1];
        invForm = $("#invForm");
        for (var i = 0; i < Math.min(response.data.length, 10); i++) {
            var item = document.createElement('div');
            item.id = 'amigo' + numa;
            numa++;
            item.innerHTML = '<img src="'+ response.data[i].picture.data.url+ '"><span class="name">' +response.data[i].name+
             '</span><input type="checkbox" name="amigo" value="'+ response.data[i].id + '" />';
            $(item).appendTo(invForm).attr("class","col-sm-12");

        }

          invitaAmigos();

        $("#invForm input").each(function(){

        })
      }

    });
```

Figura A16: *Código de la función invitaAmigos.*