

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE

TELECOMUNICACIÓN

GRADO EN INGENIERÍA EN TECNOLOGÍAS DE LAS TELECOMUNICACIONES

**TRABAJO FIN DE GRADO**

VIRUS RAMPAGE: UNA APLICACIÓN HÍBRIDA DESARROLLADA CON APACHE CORDOVA DEL JUEGO DE MESA VIRUS

Autor: **Daniel Rodríguez Cabello**

Tutor: **José Centeno González**

Curso académico 2017/ 2018

# Resumen

Índice

[1 Introducción 4](#_Toc503374319)

[1.1 Contexto y motivación del proyecto 5](#_Toc503374320)

[1.2 Objetivos del proyecto y de la aplicación web 7](#_Toc503374321)

[1.3 Estructura del documento 8](#_Toc503374322)

[2 Estudio de las distintas tecnologías 8](#_Toc503374323)

[3 Análisis de requisitos 8](#_Toc503374324)

[3.1 Requisitos funcionales 8](#_Toc503374325)

[3.2 Requisitos no funcionales 10](#_Toc503374326)

[4 Descripción de la solución 11](#_Toc503374327)

[4.1 Diseño 11](#_Toc503374328)

[4.1.1 Metodología 11](#_Toc503374329)

[4.1.3 Diseño de la base de datos 15](#_Toc503374330)

[4.1.4 Imagen 18](#_Toc503374331)

[4.2 Implementación 18](#_Toc503374332)

[4.3 Pruebas 20](#_Toc503374333)

[5 Conclusiones 20](#_Toc503374334)

[6 Bibliografía 20](#_Toc503374335)

# 1 Introducción

Actualmente estamos viviendo en una época de gran auge tecnológico. Desde la aparición de internet y el fácil acceso de éste por el público general, se ha producido un aumento de las tecnologías web.

Desde el lenguaje de marcado HTML y las hojas de estilo CSS, pasando por Frameworks como Jquery y Bootstrap, hasta tecnología más moderna como angular.js; podemos constatar que el navegador web es todavía, y será por mucho tiempo aun, un referente hacia donde encaminar gran parte de este desarrollo tecnológico.

En añadido, la aparición de nuevos dispositivos portátiles como teléfonos inteligentes, tabletas y otros similares, ha encaminado el desarrollo web hacia un diseño adaptativo debido a los diferentes tamaños de pantalla. Pero la forma de trabajar con el navegador es muy parecida en todas las plataformas.

Sin embargo, esto no ocurre cuando queremos desarrollar software nativo para cada familia de dispositivos. Android con Java, iOS con Objetive-C, Windows con C# entre otros, Mozilla, BlackBerry…etc. Es necesario un entorno de desarrollo y un lenguaje diferente dependiendo de en qué mercado queramos trabajar y, por supuesto, esto hace que sea muy difícil desarrollar de forma nativa para todas las plataformas.

¿Qué está ocurriendo? Normalmente el desarrollo se enfoca a únicamente a los dos grandes: Android de Google, e iOS de Apple. Y muchas veces esto ni siquiera es necesario, porque, aunque en Estados Unidos, iOS es capaz de seguir muy de cerca a Android, con un 40% de venta de dispositivos, frente al 52% que ostentan los de Google; cuando salimos de las fronteras de EEUU, globalmente la cosa cambia: Android se lleva el 70% de las ventas frente al 21% de iOS. Toda esta estadística hace que a veces únicamente se desarrolle para Android. Y es que cuando es necesario desarrollar para varios sistemas operativos, prácticamente tengamos que realizar un nuevo proyecto desde cero debido a las notables diferencias que hay en el diseño de unos y otros.

Es en este punto donde es interesante recordar el hecho de que los navegadores sí que se habían apegado a los estándares, entre otros las especificaciones del lenguaje HTML, CSS, XML y DOM. Sin embargo, aunque las aplicaciones web creadas para funcionar mediante el navegador y con tecnología web, parecían funcionar prácticamente en la totalidad de dispositivos, distaban mucho de tener la misma funcionalidad de una aplicación nativa. Las aplicaciones web no son capaces de acceder a tantas funcionalidades del hardware del dispositivo ni a librerías del sistema.

Y he aquí la gran pregunta, ¿existe la posibilidad de aprovechar las bondades de las aplicaciones nativas y de las aplicaciones web? La respuesta es sí, y se llaman aplicaciones híbridas.

Las aplicaciones híbridas utilizan una capa de abstracción sobre la plataforma en cuestión, que nos permite encapsular una aplicación web en una aplicación nativa. Pudiendo de esta forma reutilizar un mismo código para diferentes plataformas y dispositivos.

Al estar embebidas en un navegador de una aplicación nativa, las aplicaciones híbridas tienen el mismo acceso a las APIs de cada sistema operativo y a los recursos de cada dispositivo (GPS, cámara, acelerómetro) como si de una aplicación nativa se tratase.



En este paradigma de aplicación híbrida, surgen muchos Frameworks claves: Phonegap, Apache Cordova (que no es sino la versión de código abierto de Phonegap), Ionic y varios más. Todos ellos, aunque de diferente modo, son capaces de convertir una aplicación web, en una aplicación híbrida.

Esta tecnología, la cual abordaremos en mayor detalle más adelante, es el fundamento de este proyecto.

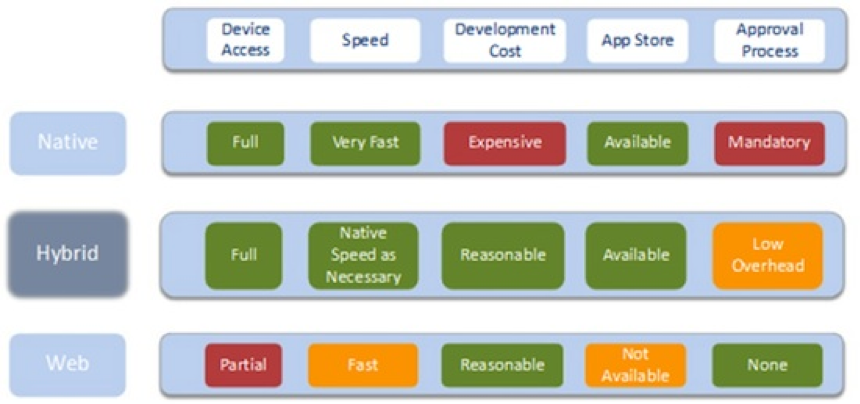
## 1.1 Contexto y motivación del proyecto

El mercado del desarrollo de aplicaciones ha experimentado un auge en los últimos años. Se ha multiplicado exponencialmente el número de aplicaciones disponibles para los usuarios y prácticamente se hace necesaria una aplicación dedicada para cada problema.

Además, estamos viviendo el inicio del internet de las cosas. La industria ya está prácticamente informatizada. Ganaderos, redes de transporte, medicina y un largo etcétera. Pequeñas y grandes industrias utilizan los dispositivos móviles para tener siempre encima información actualizada de sus negocios. Y es debido a esto que se está formando un gran mercado entorno a dichas aplicaciones, ya que en la mayoría de casos no existe software abierto al público y aun en el caso de existir, no suele ser muy útil pues se requiere que esté personalizado al caso práctico en cuestión.

Además, desarrollar software nativo no suele ser una buena opción. Para el cliente es necesario desarrollar rápidamente, con bajo coste y multiplataforma, y para la empresa desarrolladora, en añadido, ahorrando código, que se traduce en más ahorro de tiempo y dinero.

Bien es cierto que las aplicaciones nativas ofrecen un rendimiento mayor, sobre todo si la aplicación utiliza mucha funcionalidad nativa o requiere de muchos gráficos (esto es debido al renderizado de las imágenes) como sería el caso de aplicaciones dedicadas, tales como juegos, o aplicaciones con un gran volumen de usuarios tras las cuales suele haber grandes empresas.



Sin embargo, esto no es lo que ocurre en muchos casos, donde únicamente se requiere una interfaz en la que poder volcar datos obtenidos de un servidor. Para esto, las aplicaciones híbridas son la solución.

Y así se refleja en el mercado laboral, donde cada vez más empresas reclaman conocimientos en estas tecnologías, tendencia que continuará creciendo en los próximos años.

## 1.2 Objetivos del proyecto y de la aplicación web

En este proyecto se propone la creación de una aplicación híbrida mediante el uso de tecnologías web. La aplicación elegida, en este caso, será un juego online basado en el popular juego de cartas Virus.

La creación de la aplicación es un objetivo en sí mismo. La incursión en el mercado de aplicaciones móviles es algo que siempre me he planteado como un buen escenario de negocio, donde aplicaciones muy sencillas, reciben un increíble volumen de beneficios. Aunque el estudio de los diversos sistemas de obtención de ingresos de las diferentes aplicaciones (aplicaciones de pago, anuncios, pagos internos...), es algo que queda fuera del ámbito de este proyecto, la creación del proyecto en sí mismo es un buen primer paso para plantearme en un futuro el desarrollo de una aplicación con ánimo de lucro.

En consecuencia, uno de los objetivos principales del desarrollo de la aplicación en este proyecto, es lograr aunar los conocimientos adquiridos durante la carrera con suficientes conocimientos nuevos, para en el futuro tener todas las herramientas para poder crear nuevas aplicaciones.

Es por ello, que se pretende la consecución de unos objetivos de aprendizaje, partiendo de algunos conocimientos previos en tecnologías web para poder incorporar otros nuevos. Cabe destacar los siguientes:

- Estudio y análisis de los distintos frameworks para la creación de aplicaciones híbridas y elección del más conveniente para el desarrollo del presente proyecto.

- Estudio de las diferentes alternativas para la comunicación entre cliente y servidor.

- Búsqueda de plataformas online donde poder alojar un servidor.

- Creación de un servidor, escalable, capaz de recibir y contestar peticiones por parte de los clientes.

- Análisis de bases de datos relacionales y no relacionales, y aprendizaje sobre uso y manejo.

- Aprendizaje de entornos de ejecución multiplataforma, en concreto Node.js.

Todo ello se pretende juntar para posibilitar la creación de una aplicación multiplataforma y multiusuario, que sea capaz de funcionar en tiempo real, y que permita interacción por parte del usuario.

## 1.3 Estructura del documento

La memoria se ha estructurado en varios apartados. A continuación, se enumeran y se describe brevemente el contenido de los mismos.

**- Introducción:** Este apartado es el inicio de todo. Se explica de dónde surge la idea que acaba dando como resultado el desarrollo de esta aplicación. Se introduce el proyecto, las motivaciones y el trasfondo tecnológico del mismo.

**- Estudio de las distintas tecnologías:** Además del desarrollo de la aplicación, este proyecto ha llevado un exhaustivo trabajo de aprendizaje y de recopilación de información sobre algunas tecnologías. Y ya que ha sido relevante en el proyecto en sí, está detallado en esta memoria el contenido más interesante.

**- Análisis de requisitos:** Describimos la funcionalidad de la aplicación. Tanto parámetros de diseño y uso de la aplicación, como otros menos tangibles que tratan de rendimiento, estructura y características técnicas.

**- Descripción de la solución:** Se trata de la parte principal de este proyecto. Se describe la aplicación de forma exhaustiva. Considero importante mencionar de forma independiente los tres subapartados en los que se ha dividido esta parte.

**• Diseño:** Describe la metodología empleada y la forma de trabajar que se ha seguido en el proyecto. Además, se describe el diseño de la base de datos y las colecciones que se han usado. Por último, se habla del diseño referido a la parte estética, que patrones se han seguido y porqué.

**• Implementación:** Toda la funcionalidad de la aplicación está explicada en este subapartado, con referencias a la tecnología empleada y su uso, y una descripción detallada de cada momento relevante que ha habido en el proceso de desarrollo.

**• Pruebas:** Una vez finalizada la aplicación, se han realizado una serie de pruebas para comprobar un correcto funcionamiento. Esto se detalla en este subapartado.

**- Conclusiones:** Una vez finalizado el proyecto, evaluamos si los objetivos que se habían marcado inicialmente se han cumplido, así como qué competencias que se poseían previamente se han mejorado y que nuevas competencias se han adquirido. Por último, hablamos de posibles líneas de trabajo futuro y cuáles son los siguientes pasos de actuación.

# 2 Estudio de las distintas tecnologías

## 2.2.1 Javascript y HTML5

## 2.2.2 Jquery

## 2.2.3 Aplicaciones híbridas

## 2.2.4 Cordova vs Phonegap

Actualmente en el mercado existen tres grandes plataformas móviles que son iOS, Android y Windows Phone, amén de otras menos relevantes como Blackberry, webOS o firefoxOS, que aunque tienen una cuota de mercado prácticamente nula, bueno, existen. Y por supuesto tenemos toda la gama de navegadores, que aunque ya están bastante estandarizados, todavía cuentan con características exclusivas para cada uno de ellos.

Ante tal variedad de maneras de realizar los desarrollos para todas estas plataformas, podemos optar por uno de las siguientes opciones:

- Crear una versión para cada plataforma, usando las herramientas nativas de cada una de ellas.

- Usar algún tipo de framework intermedio, como por ejemplo Xamarín, que sea capaz de funcionar en todas ellas y que nos permita reutilizar la lógica para únicamente desarrollar la interfaz en cada caso.

- Utilizar CSS, Javascrip y HTML y algún tipo de framework multiplataforma como Phonegap o Apache Cordova que nos permita compilar la aplicación para todas las plataformas.

Actualmente para esta última opción contamos con multitud de herramientas, pero dado el caso que nos ocupa nos centraremos en Apache Cordova y en Phonegap.

**¿Cuál es la diferencia entre ambas?** Pues realmente es ninguna, ya que se trata de herramientas prácticamente identicas excepto por los derechos de propiedad de cada una de ellas y algunas características añadidas. Aunque si nos remontamos a la historia de este software hallaremos la respuesta.

En el año 2009 la empresa Nitobi crea el framework de desarrollo móvil multiplataforma llamado PhoneGap. En septiembre de 2011 se decide donar el código fuente a la fundación Apache y se convierte en un proyecto de código abierto.

En octubre de 2011, Adobe compra Nitobi, junto con la marca de Phonegap, aunque se decide continuar con el plan de mantener el código abierto y donarlo a la fundación Apache. Pero como PhoneGap ha pasado a ser una marca registrada de Adobe, para proteger la propiedad intelectual y en un futuro poder comercializar el producto añadiéndole características propias, en febrero de 2012 se decide renombrar el proyecto de código abierto a Cordova. El nombre no es más que una curiosidad, pues se trata del nombre de la calle donde estaban ubicadas las oficinas de Nitobi.

Y es aquí donde se halla la respuesta: al ser PhoneGap una marca registrada propiedad de Apache, este puede hacer con el software lo que quiera, aunque de momento sigue siendo de descarga gratuita al igual que Cordova y se utilizan del mismo modo, siendo PhoneGap nada más que una distribución de Cordova. No obstane PhoneGap, como única notable diferencia, aporta un servicio de compilación en la nube. Que aunque es gratuito para trabajar subiendo el código desde un único repositorio de github, si quieres ampliar las carácterísticas debes elegir la opción de pago. Esta opción de pago te permite ahorrarte la necesidad de instalar los SDK y las herramientas específicas de cada plataforma.

Descatar que a excepción del servicio de compilación en la nube, y a pesar del hecho de que phoneGap pertenezca a Adobe, la empresa se ha comprometido a que el software siempre siga siendo gratuito y de codigo abierto.

## 2.2.5 Node.js

Node.js es una librería y entorno de ejecución multiplataforma de entrada y salida dirigida por eventos, y por lo tanto asíncrona, que se ejecuta sobre el intérprete de Javacript. Esto posee varias ventajas ya de base: Javascript posee un modelo de eventos muy bueno, que resulta ideal para la programación asíncrona. Por otro lado Javascript, aunque antaño relegado para tareas menores de navegador, actualmente es tan capaz como otros lenguajes tradicionalmente enfocados para backend como Python, Ruby, o java. Por último, Javascript es un lenguaje conocido practicamente por todo desarrollador, lo que disminuye la curva de aprendizaje y facilita la construcción de aplicaciones.

## 2.2.5 Npm

NPM es el gestor de paquetes javascript de Node.js, cuyo nombre extendido “node package manager”, le fue otorgado una vez el programa ya estaba disponible para su uso mediante línea de comandos desde una terminal. Es decir, primero fue la abreviatura y posteriormente el nombre.

Más allá de esta pequeña curiosidad respecto al origen de su nombre, npm se usa como gestor respecto al manejo de dependencias y módulos de node.js. Los usos más comunes son publicación de nuevas herramientas, instalación y desarrollo de programas y extensión de herramientas de node.

## 3.2.8 webSockets

WebSocket es un protocolo de comunicación web bajo TCP, por el cual, a diferencia de la conexión habitual que venimos usando bajo HTTP, tenemos una comunicación bi-direccional. Esto significa, que a diferencia del modelo estándar de petición-respuesta por parte de cliente y servidor respectivamente, el servidor también puede ser el que inicie la comunicación, para pedir o comunicar al cliente en cualquier momento.

Por otro lado, webSocket utiliza conexión persistente. Esto quiere decir que una vez establecida la conexión mediante un intercambio de mensajes previos de protocolo (“handshake”) en HTTP, ya no es necesario volver a mandar esta información de control, y todo el intercambio de información puede hacerse directamente bajo TCP.

¿Qué ventajas posee esto? En aplicaciones donde el intercambio de información entre cliente y servidor es fluido, el aumento del rendimiento es notable.

Aunque webSocket con lo dicho ofrezca multitud de posibilidades para desarrollar, hay que tener en cuenta ciertas características de este sistema:

- WebSockets se inicia con un “handshake” HTTP, por lo que si no se cuenta con dicho protocolo, webSockets no sirve. Es decir, esta enfocado a aplicaciones web.

- La conexión TCP va por el puerto 80, el cual suele estar siempre en el punto de mira de los firewall, lo que nos da ciertas ventajas y algún inconveniente.

- Al igual que existe HTTPS, también existe WSS, pero si bien todos los navegadores soportan HTTPS, no todos los navegadores soportan WSS.

3.2.8 Socket.io

Websocket es todavía una tecnología muy nueva, que no está implementada en todos los navegadores (según una estadística de caniuse.com alrededor del 48% soportan webSockets). Es por ello que surgen librerías como socket.io que dan soporte practicamente a la totalidad de los usuarios.

“Socket.io es una libería en javascript que permite realizar aplicaciones en tiempo real sin preocuparse por el navegador en que se ejecute. Elimina todas las diferencias existentes entre los métodos de transporte de los navegadores unificándolos en una API.”

Fuente: <http://www.antweb.es/programar/socket-io>

## 3.2.9 Express

Express es una infraestructura de aplicaciones web Node.js mínima y flexible que proporciona un conjunto sólido de características para aplicaciones web y móviles.

Se caracteriza por tener multitud de métodos de programa de utilidad HTTP y middleware que permiten la creación de una API sólida de forma rápida y sencilla. Además, proporciona una capa de carácterísticas de aplicación web básicas, que se añaden a las propias características de Node.js.

El concepto de express es similar a Jquery, es decir, si queremos escribir código de navegador, podemos hacerlo utilizando únicamente Javascript, pero nos llevará muchas más líneas y un esfuerzo extra que si lo hiciésemos utilizando Jquery. De la misma manera, express nos puede ayudar a crear una aplicación de servidor sin utilizar tanto código de Javascript, en Node.js.

Además, express es extensible, lo que quiere decir que se puede partir desde la base de una aplicación web muy sencilla y hacerla crecer a medida que se necesite, junto con código ya creado por otros programadores y extensiones que puedes encontrar en los repositorios de npm.

## 2.2.6 Heroku

Heroku es una plataforma como servicio de computación en la nube que soporta varios lenguajes de programación. En un principio fue diseñado únicamente para Ruby, pero actualmente ya son más de ocho lenguajes los que soporta, incluidos entre otros Python, Java, Ruby y Javascript.

El propósito de Heroku es que el programador emplee todo el tiempo posible en el diseño y en el desarrollo de sus aplicaciones, en vez de dedicarse a configurar servidores o preocuparse del despliegue.

Podemos utilizar Heroku para subir nuestras aplicaciones y dotarlas de servidor tanto de forma gratuita como de pago.

Las aplicaciones se inician desde un servidor Heroku usando Heroku DNS Server para apuntar al dominio donde se aloja la aplicación. Además, el servidor de Heroku cuenta con un servicio Github para manejar los distintos repositorios de las aplicaciones subidas por el usuario.

Además, cada vez cuenta con más funcionalidades extras propias del servidor llamadas add-ons que añaden funcionalidad extra a nuestras aplicaciones.

### 3.2.6.1 Dynos

Los Dynos son la estructura básica sobre la que se asienta el sistema de Heroku. Los Dynos son unidades de cómputo dentro de la plataforma.

Dentro del servicio gratuito se ofrecen 240 horas de computo gratuito gestionados por un único Dyno, que, no obstante, es suficiente para que cualquier desarrollador pueda probar sus aplicaciones y si está conforme poder ascender a una subscripción de pago. Lo cual es otra de las grandes virtudes de Heroku, la escalabilidad.

Heroku ofrece diferentes tipos de Dynos, cada uno con capacidades y memoria diferentes. Pero cambiar de un tipo de Dyno y cambiar de número de Dynos ¡es casi instantáneo! Se puede hacer en cualquier momento desde una línea de comandos o desde la propia interfaz de Heroku.

Estos Dynos además son independientes unos de otros, ya que si uno falla, no afecta en absoluto al resto que esté en ejecución y, además, son autogestionados, ya que si un Dyno por algún motivo presenta errores, se elimina y se crea otro nuevo automáticamente.

Por último, dentro de esta capacidad de autogestión de los Dynos, se destaca la capacidad de routing, que hace seguimiento a los Dynos para saber dónde están siendo ejecutados, y redirige el tráfico de los mismos en caso de ser necesario.

### 3.2.6.2 mLab MongoDB

Uno de los add-ons más destacados de Heroku es “mLab MongoDB”. Permite la creación y configuración de una base de datos MongoDB de una manera bastante sencilla.

Simplemente desde la propia de interfaz de tu aplicación, dentro de la interfaz de Heroku, tienes que añadir esta extensión y ya está, tienes una base de datos. O bueno, en realidad tienes la dirección de la base de datos. Pero con nodejs, la conexión la base de datos es bastante sencilla.

La extensión tiene su propia interfaz, que te ofrece de forma gratuita multitud de estadísticas ordenables por varios criterios, como diferentes escalas de tiempo y diferentes formas en que todos estos datos son presentados.

Entre estas estadísticas destacamos: operaciones por segundo, conexiones, búsquedas efectivas, cuellos de botella o estadísticas de uso de la CPU.

El add-on “mlab MongoDB” cuenta con planes de subscripción gratuita, de la misma forma que Heroku, pero ampliables instantáneamente a una subscripción de pago. No obstante, la subscripción gratuita cuenta con todas las funcionalidades necesarias para desarrollar cualquier aplicación de tamaño pequeño y medio, incluyendo 496 megabytes de almacenamiento, que no está nada mal.

# 3 Análisis de requisitos

Una vez planteados los objetivos que se pretenden alcanzar con el proyecto, surge la necesidad de realizar un análisis de requisitos del sistema. En este análisis se plantea un estudio profundo de las necesidades tecnológicas del proyecto y se especifica las características operacionales que tendrá el software a desarrollar. Es decir, se describe el plan de proyecto a seguir.

Es necesario ser lo más riguroso posible en este análisis y definir claramente las propiedades que se deben satisfacer para obtener una interacción eficaz con los demás elementos del sistema (usuario, hardware, otros elementos de software…). Pero, sobre todo, en el caso particular de este proyecto, la interacción de la aplicación con el usuario debe estar clara y bien definida. Esto será crucial para llevar la aplicación a buen término.

Bajo estas premisas el análisis de requisitos puede dividirse en dos partes que se detallan a continuación: requisitos funcionales y requisitos no funcionales.

## 3.1 Requisitos funcionales

Los requisitos funcionales son aquellos que definen toda la funcionalidad que debe tener la aplicación desde un punto de vista cualitativo.

La mayoría de los objetivos básicos de diseño se detallarán en este apartado, así como todos aquellos relacionados con la interacción del usuario con la aplicación.

La siguiente lista representa toda la funcionalidad básica que hemos tenido en cuenta desde un principio. Engloba todos los objetivos mínimos necesarios para poder llegar a considerar la aplicación como terminada y dotada de la adecuada funcionalidad, y es el punto de partida para poder realizar un nuevo ciclo de mejora de la aplicación:

* Redimensionado. La aplicación, dentro de unos límites razonables, debe poder adaptarse a diferentes tamaños de pantalla sin perjuicio de su funcionalidad
* Interacción con el usuario. La aplicación será intuitiva para el usuario, contando además con suficientes cuadros de ayuda para que el usuario pueda usar y experimentar la aplicación desde un primer momento.
* Apariencia. En la medida de lo posible, la aplicación será atractiva, con colores agradables. Los elementos que se puedan usar deben parecer que realmente se pueden usar.
* Dividiremos el diseño de la interfaz de la aplicación en tres partes bien diferenciadas.
* En primer lugar, en la pantalla inicial, se mostrarán todos aquellos botones relacionados con la plataforma de juego.
  + Un primer botón principal de partida rápida, para en el caso de no tener preferencias de qué partida jugar, entrar a jugar la partida que tiene más posibilidades de empezar antes.
  + Un segundo botón principal de crear partida.
  + Un tercer botón principal para acceder a las salas de espera con todas las partidas creadas a la espera de completar el número de jugadores necesario.
  + Un botón de login/leave y otro de registro.
  + Un botón de clasificación que, al menos mostrará los primeros clasificados y dentro del menú que despliega este botón, botones extra para ordenar la lista de diferentes modos.
  + Un botón de “settings”, en el cual guardar algunos parámetros por defecto como un “autologin” al iniciarse la aplicación.
  + Un botón de instrucciones, en el que se describirá brevemente y de la manera más ilustrativa posible, como interactuar con los diferentes elementos que se encontrará el usuario en la aplicación.
  + Un botón de cómo jugar, en el cual se detallarán las instrucciones del juego.
  + Una pantalla de información, donde se indica la partida a la que se entrará en caso de pulsar partida rápida.
* En segundo lugar, dentro del mismo contexto que la primera pantalla, tendremos una pantalla secundaria donde todas las funcionalidades de la primera pantalla quedarán ocultas. A está interfaz secundaria accederemos mediante pulsar alguno de los botones de la pantalla principal. Y contendrá al menos los siguientes elementos:
  + Una interfaz para poder crear una partida.
  + Una interfaz que listará todas las partidas creadas que están esperando que se unan jugadores para poder empezar la partida.
  + Un botón de regreso a la pantalla principal
* En tercer lugar y de forma totalmente diferenciada de la interfaz anterior, tendremos la pantalla destinada a la partida. Podrá contener algunos botones en el caso de ser necesarios y será el núcleo de la aplicación.
* Usaremos animaciones y elementos adecuados para representar los tiempos de espera, bien a causa del tiempo de respuesta del servidor, bien por la aplicación.

## 3.2 Requisitos no funcionales

Los requisitos no funcionales son todos aquellos que se enfocan en las herramientas o procedimientos para lograr hacer funcionar la aplicación. Se trata de explicar cómo va a realizarse la implementación teniendo en cuenta los objetivos de diseño y funcionalidad anteriormente marcados.

Sin entrar en detalles concretos acerca de la tecnología a emplear, se detalla el funcionamiento interno de la aplicación, el flujo de trabajo de los diferentes sistemas que se quieren emplear y como fluye la información por el sistema.

En la siguiente lista se enumeran, seguidos de una breve descripción todos los requisitos funcionales que se han creído relevantes para este proyecto:

* Separaremos claramente y de forma diferenciada la aplicación en dos partes, para poder trabajar la funcionalidad implícita de cada una de forma diferenciada.
  + La primera parte tendrá que ver con las dos primeras pantallas, cuyo enfoque está dirigido a un sistema que tendrá funcionalidad de plataforma.
  + La segunda parte de la funcionalidad de la aplicación aparecerá una vez iniciado el juego en la tercera pantalla, y tratará todo el flujo de información que se produce por parte del usuario al accionar los distintos elementos disponibles.
* La aplicación debe ser escalable. Nuevas características deben poder ser añadidas sin perjuicio de las ya existentes.
* Multiplataforma. La aplicación debe funcionar en todos los dispositivos móviles y, además, poder funcionar como aplicación web en un navegador.
* Interacción con el servidor. La aplicación debe ser lo más autosuficiente posible, tratando de enviar y recibir datos lo menos posible.
* Velocidad de respuesta por parte de servidor. Esta debe ser lo suficientemente rápida y en caso de no serlo, el usuario debe tener la sensación de que la aplicación no se ha quedado colgada. Esto lo haremos con comunicación asíncrona entre servidor y aplicación.
* Usaremos un servicio de computación en la Nube, que sirva como plataforma para poder alojar el servidor de la aplicación. Trataremos que al mismo tiempo ese mismo servicio proporcione facilidades para la creación de una base de datos.
* La base de datos será no relacional, ya que la flexibilidad y la escalabilidad de la misma es un requisito imprescindible.

# 4 Descripción de la solución

En esta parte de la memoria del proyecto vamos a describir como se ha llevado a cabo la realización del proyecto, objeto de esta memoria.

## 4.1 Diseño

El diseño de la implementación de la solución se ha dividido en varias partes: metodología, diseño y pruebas.

### 4.1.1 Metodología

En cuanto a la metodología, lo entendemos como el modelo de desarrollo de software utilizado en el proyecto.

Hemos estudiado dos enfoques que se podían ajustar a las necesidades de este, el *desarrollo en cascada* y el *desarrollo en espiral*, acabándonos de decidir finalmente por este último al considerar que es el que mejor se adapta a las necesidades de este proyecto.

A continuación, vamos a describir los dos enfoques brevemente y porque nos hemos decantado por uno u otro.

#### 4.1.1.1 Desarrollo en cascada

El desarrollo en cascada es el enfoque metodológico que ordena rigurosamente las etapas del ciclo de vida del software, de tal forma que el inicio de cada etapa debe esperar a la finalización de la inmediatamente anterior. Este desarrollo presenta una estructura rígida donde se presentan las siguientes fases o etapas:

- Análisis de requerimientos: En esta etapa se analizan los requerimientos finales que debe tener la aplicación. Detalla lo que debe hacer y las características finales del sistema sin entrar en detalles internos.

- Análisis de diseño: Se descompone el proyecto en elementos más pequeños con los cuales pueda trabajarse por separado de la forma más independientemente posible. De esta forma, en caso de tener un equipo, se optimiza el trabajo. Se describe la estructura global del proyecto y la especificación de lo que tiene que hacer cada una de sus partes; así como la manera en que se combinan unas con otras.

- Diseño detallado: Se realizan los algoritmos necesarios para el cumplimiento de los requerimientos del programa y se analizan las herramientas a usar durante la etapa de codificación.

- Codificación: Es la etapa de programación. Se desarrolla el código y se realizan pruebas para la detección y corrección de errores.

- Pruebas: Se realizan pruebas para asegurar que el software funciona sin errores y que cumple con los requisitos establecidos inicialmente. Esta fase se inicia cuando cada uno de los elementos en los que se ha dividido el trabajo global se combinan.

- Implantación: Se trata de una de las fases finales del proyecto donde el software se pone en producción. Es una de las fases de más duración, pues no suele terminar hasta que el software deja de utilizarse.

- Mantenimiento: Al igual que la fase de implantación, es de gran duración no termina hasta que el software deja de utilizarse o se dispone de una nueva versión. Se realiza corrección de errores solo detectados en la etapa de producción y pequeños cambios de la versión original del proyecto.

**Análisis**

El *desarrollo en cascada* es el más utilizado. Es el ideal para proyectos rígidos donde los requerimientos están muy especificados y se conocen las herramientas a utilizar durante todo el desarrollo del programa.

El producto final es de una calidad muy alta y es fácil de entender por el cliente, que en todo momento conoce el estado de desarrollo.

Sin embargo, presenta ciertas desventajas por las cuales se ha decidido no utilizarlo en la elaboración de este proyecto:

1.- Sin una especificación realmente detallada desde un principio es difícil seguir esta estructura, corriendo el peligro de hacer saltos entre fases.

2.- Para proyectos relativamente pequeños se tarda mucho en completar todo el ciclo de desarrollo.

3.- Pueden surgir nuevas ideas durante el desarrollo que no pueden ser implementadas hasta la finalización del proyecto.

4.- Pueden surgir problemas con las herramientas planteadas para la elaboración del software que difícilmente pueden ser sustituidas por otras.

5.- Es difícil incorporar nuevas características al proyecto una vez terminado.

#### 4.1.1.2 Desarrollo en espiral

En el diseño en espiral, el proyecto se compone de un conjunto de actividades desarrolladas de forma secuencial. La espiral completa es el producto final y cada ciclo de la espiral es cada una de las actividades de las que se compone, las cuales no conocemos todas al inicio del proyecto.

Cada uno de los ciclos se puede dividir en cuatro fases: definición de objetivos, evaluación y reducción de riesgos, desarrollo y validación, y planificación.

En este diseño, como hemos comentado antes, las actividades no están fijadas a priori, sino que se eligen en función del análisis de riesgo de un nuevo ciclo, comenzando por la iteración anterior. Es decir, se parte de un objetivo primario funcional y una vez se ha completado, se hace un análisis para estudiar el siguiente objetivo.

Se trata del enfoque de desarrollo de software más realista actualmente. Es el más indicado cuando no se requiere de una definición completa de los requerimientos del software para empezar a funcionar. Además, es ideal cuando tampoco se depende de una implementación y un diseño rígidos.

Este diseño puede combinarse con otros modelos de proceso de desarrollo (cascada, evolutivo…) para la realización de cada ciclo.

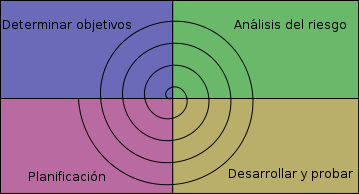
Cada actividad o ciclo del diseño en espiral se compone de las siguientes tareas:

- Definición de objetivos: Para cada fase del proyecto se definen objetivos específicos. Se identifican los problemas y restricciones del proceso, y se crea un plan detallado de actuación consistente en la definición de recursos y tiempo necesarios. Es decir, se evalúan todos los requerimientos que se quieren implementar en el ciclo.

- Análisis de los riesgos: Se lleva a cabo un análisis detallado de cada uno de los riesgos del proyecto y se definen los pasos necesarios para poder reducir dichos riesgos.

- Desarrollo: Consiste en realizar todas las tareas para construir el nuevo nivel que se quiere implementar en la aplicación. Dependiendo de los riesgos identificados, se plantean diferentes modelos para reducción de los mismos. Para cada ciclo se puede aplicar un modelo de desarrollo diferente dentro del modelo general de desarrollo en espiral.

- Planificación: El proyecto se revisa y se toma la decisión de continuar con una nueva iteración de la espiral. Con cada iteración se construyen sucesivas versiones del software, cada vez más completo y, al final, se obtiene el software totalmente terminado y funcional.



**Análisis**

En particular, en el desarrollo de mi aplicación, he usado el modelo en espiral, ya que considero que es el enfoque más realista en el cual puedo contextualizar mi aplicación.

He comenzado con una idea general sobre dónde quería llegar, y una vez he completado la primera iteración funcional, se han diseñado nuevas actividades.

El modelo en espiral es un buen modelo, ya que puede aplicarse a lo largo de toda la vida del software. Incorpora objetivos de calidad y gestión de riesgos a corto plazo, y elimina errores y alternativas que no son muy atractivas al comienzo.

Sin embargo, siendo riguroso, este modelo también presenta algunas desventajas, de entre las cuales voy a enumerar las siguientes:

1.- Requiere bastante habilidad para evaluar los riesgos de cada ciclo. Habilidad que es necesaria para poder llegar a buen término.

2.- A la hora de evaluar los riesgos, si un riesgo importante no es detectado y gestionado a tiempo, podrían surgir problemas muy graves que lleven a desechar el ciclo completo.

3.- Para proyectos muy grandes, es un modelo muy difícil de aplicar, ya que se desconoce el número de iteraciones que van a ser necesarias.

4.- En cada iteración, el tiempo de desarrollo es cada vez más elevado.

Pese a los problemas que pueden surgir por la utilización de este modelo, en el caso que nos ocupa, algunas de las desventajas no son tan relevantes. Al tratarse de un proyecto relativamente pequeño, el número de iteraciones que hagamos no es algo que nos tenga que preocupar en ningún momento, ya que en todo momento podemos abarcar una visión global de todo el trabajo sin demasiadas complicaciones. Y tiene como ventaja tener siempre una versión funcional del proyecto, con el objetivo final siempre alcanzable y presente en el horizonte.

Por otro lado, en el caso de haber planteado erróneamente los riesgos y encontrarnos en un punto de difícil solución, podemos deshacer rápidamente el ciclo y abarcar el problema de otro modo.

Y, por último, aunque si bien es cierto que la fase de desarrollo, en cada iteración, se va haciendo exponencialmente más compleja, las ventajas de este modelo compensan sobradamente este problema.

Por todo lo expuesto anteriormente, nos hemos decantado por utilizar el modelo en espiral.

### 4.1.3 Diseño de la base de datos

Para el diseño de la base de datos hemos hecho primero un estudio de las diferentes alternativas y para posteriormente decantarnos por una de ellas.

Hemos estudiado las bases de datos relacionales SQL y las bases de datos no relacionales, que diferencias existen entre ellas y cuál es la mejor opción a utilizar de acuerdo a la tecnología que estamos utilizando.

Aunque finalmente se ha decidido utilizar una base de datos no relacional, hablaremos de ambas, cuáles son sus ventajas e incovenientes y un pequeños argumentario de porqué nos hemos decidido por las bases de datos no relacionales. Finalmente, mostraremos la colección que usamos en la base de datos.

#### 4.1.3.1 Bases de datos relacionales (MySQL)

Las bases de datos relacionales, almacenan los datos en tablas y utilizan lenguaje estructurado de consulta para el acceso a la base de datos. Se llaman bases de datos relacionales ya que se establecen asociaciones entre tipos de datos que son parecidos, de esta forma, la duplicidad de la información es mínima.

La principal opción que se estudió como base de datos relacional fue MySQL, que es una base de datos relacional de código abierto desarrollada por Oracle.

En MySQL se debe predefinir con antelación el esquema que se usará en la base de datos. Los datos están organizados en tablas que tienen un esquema fijo, y cada registro siempre tiene la misma forma y las mismas columnas. Los campos son fijos y no se pueden variar bajo ningún concepto.

Uno de los motivos que me hizo desechar esta opción de base de datos fue que el acceso a ella se realiza mediante lenguaje estructurado SQL, frente a otras alternativas que utilizan lenguaje JavaScript. Lo cual lleva implícito una curva de aprendizaje más elevada.

Por otro lado, en nuestro caso, la base de datos únicamente se va a utilizar para guardar información relacionada con los usuarios que utilizan la aplicación, así como algunas estadísticas, con lo cual una opción sencilla es siempre recomendable.

#### 4.1.3.2 Bases de datos no relacionales (MongoDB)

En las bases de datos no relacionales no existen las tablas, lo que tenemos son colecciones de documentos, que son algo parecido a archivos JSON.

Además, no tiene por qué existir ningún tipo de relación entre los datos de una colección u otra, aunque se pueden simular.

Como base de datos no relacional se ha estudiado MongoDB que utiliza lenguaje Javascript. MongoDB es una base de datos de código abierto desarrollada por la compañía MongoDB, inc.

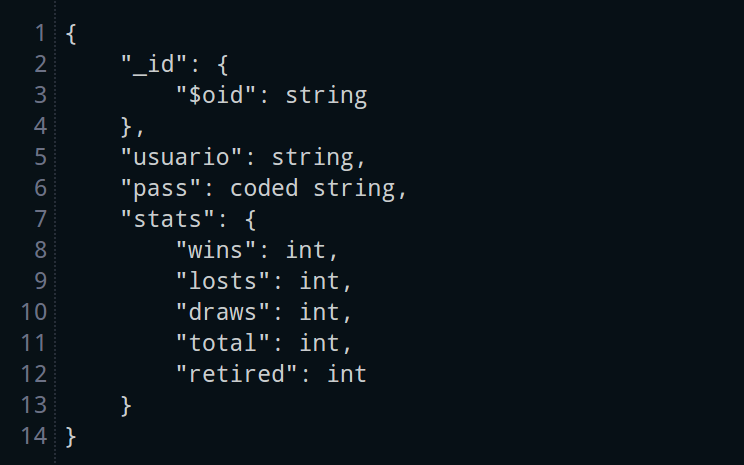
En MongoDB los datos se almacenan de forma binaria para aumentar el rendimiento de la base de datos. Es por esto, que las colecciones que hemos dicho y que son algo parecido a documentos JSON, en MongoDB se conozcan como BSON o binary JSON.

MongoDB usa esquemas dinámicos, lo que quiere decir que te permite almacenar información sin antes definir la estructura o el modelo de tu base de datos, o los campos o el tipo de información de tus colecciones de datos.

MongoDB se trata de una base de datos muy sencilla con la cual se puede trabajar muy fácilmente y, en comparativa con otras bases de datos, relacionales o no, es tan potente o más que cualquiera de ellas.

Es por todo lo mencionado anteriormente que la base de datos elegida para el proyecto es MongoDB.

A continuación, se detalla el modelo utilizado en el proyecto:



La colección se compone de un identificador, el cual es asignado automáticamente por la base de datos cuando se crea un nuevo registro; un usuario, una contraseña, que se almacena codificada desde el servidor; y un registro de estadísticas, que almacena el número de partidas ganadas, perdidas, empatadas, abandonadas y un total de ellas, que ha tenido el usuario.

Como se puede observar, el aspecto es el de un archivo JSON.

La contraseña está protegida mediante un algoritmo de codificación estandar (AES), y se guarda codificada en la base de datos. Es una aproximación a un buen nivel de seguridad, ya que la propia plataforma de Heroku en la que está alojado el servidor tiene sus propios mecanismos de seguridad, y la base de datos que utilizamos, no es más que una extensión instalada en esta plataforma.

Como se puede ver, los datos que se almacenan en la base de datos no son de gran complejidad, ni es necesaria una gran robustez de las transacciones de información que nos hagan plantearnos con profundidad las alternativas y características de otras bases de datos. Por lo tanto, la sencillez de uso de la base de datos de MongoDB, es la que nos ha llevado a decantarnos por esta opción.

### 4.1.4 Imagen

El diseño gráfico es una parte que ha consumido gran parte del tiempo de desarrollo de la aplicación, tanto por el diseño en sí como por la búsqueda y edición de imágenes que se usan en la aplicación.

El diseño de la aplicación se ha dividido en dos partes. Por un lado la *pantalla principal* que actúa de plataforma para los usuarios y por otro lado, la *pantalla de juego* juego en sí y a la partida. Para ambas, se ha trabajado enteramente con HTML5 y hojas de estilo CSS3.

La **plataforma** se compone de diferentes contenedores, los cuáles algunos son pulsables y se puede interactuar con ellos.



Aunque la mayoría del lenguaje HTML está presente desde el principio, hay cierto contenido que se genera dinámicamente mediante código javascript, como es el ranquing de jugadores, la lista de partidas que hay creadas y el contenedor de partida rápida, entre otros.

La **interfaz de juego,** se compone principalmente del elemento canvas de HTML y algún contenedor que usamos para mostrar recuadros de texto. En el canvas dibujamos todos los elementos que están relacionados con el juego como las cartas, órganos, baraja y pila de descartes, cartas de la mano etc, y otros elementos como el cronómetro, turno de los jugadores, nombre de los jugadores.. etc. Aunque nos seguimos sirviendo de otros recuadros HTML, para elementos que no pueden ser representados fácilmente en canvas, como recuadros de texto y botones pulsables.



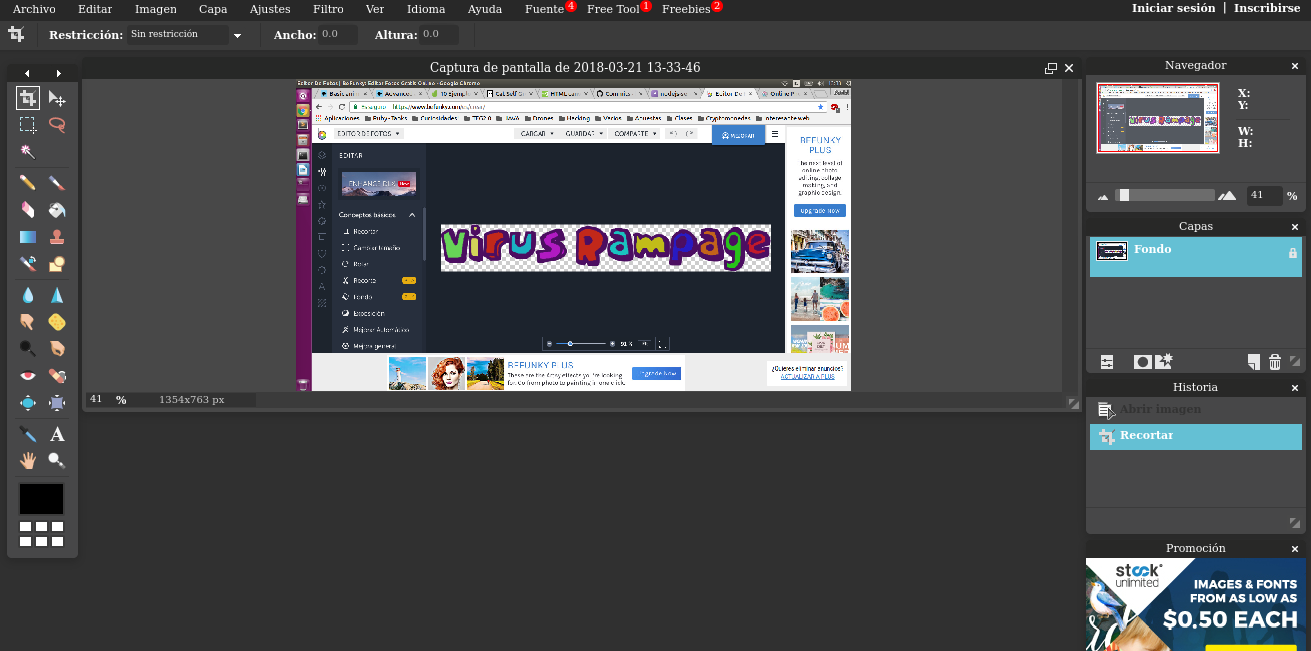
La **edición de imágenes** es otro de los puntos que ha llevado bastante tiempo. La aplicación usa numerosas imágenes para backgrounds de elementos divs, para los iconos de la aplicación, para logotipos y para cada una de las diferentes cartas del juego.

Esta tarea resultaba especialmente un problema, ya que el conocimiento previo en el área de edición de imágenes era nulo, y adentrarse en herramientas profesionales como photoshop y similares no tenía ningún sentido para los objetivos del proyecto.

Después de estudiar diversas alternativas, e incluso plantearse derivar el trabajo a un ilustrador profesional, se ha hecho uso de dos herramientas online de edicion de imágenes con las cuales se ha solventado el problema de forma muy notable:

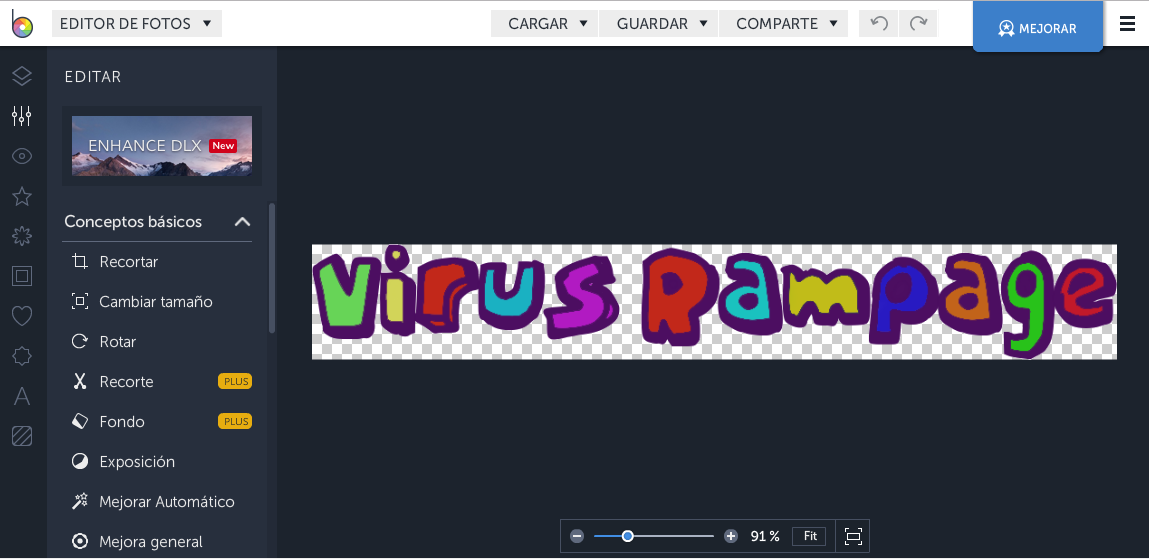
- La primera herramienta es el **editor de imágenes de pixlr** (<https://pixlr.com/editor/>) . Cubre una parte de la de funcionalidad de photoshop pero de forma más sencilla. Se ha utilizado principalmente para recortar imágenes y vectorizarlas, así como para modificar el peso (en kilobits) y el tamaño (en píxeles) de las mismas. Además, como algunas imágenes eran creadas y obtenidas a partir de la unión y edición de otras, hemos hecho uso de esta herramienta para tal fin.

De este editor, destacar que se trata de un software gratuito.



- La segunda herramienta de edición de imágenes que se ha usado se trata del e**ditor de imágenes de befunky** (<https://www.befunky.com/es/crear/>). Contiene funcionalidad para hacer diseños y collages, aunque nosotros solo hemos utilizado el editor, el cual cubre la otra parte de las necesidades de herramienta profesional de edición de software que necesitábamos, la edición, mejora y retoque de imágenes.

Una vez seleccionábamos las imágenes deseadas (a menudo previo paso por el *editor de pixlr)* era necesario aplicarle una serie de mejoras y filtros para optimizarlas, pues muchas imágenes eran obtenidas mediante fotografías o tenían poca calidad.



## 4.2 Implementación

4.2.1 Instalación y configu

4.2.x Creación del proyecto con Apache Cordova

4.2.x Diseño de lienzos superpuestos y canvas

4.2.x Movilidad de objetos y detección de colisiones

4.2.x Lógica del juego

4.2.x Renderizado de imágenes

### 4.2.x Instalación del servidor y base de datos

4.2.x Funcionamiento del servidor

## 4.3 Pruebas

## 4.3.1 Problemas-detalles

## 4.3.2 Pruebas de uso

## 4.3.3 Pruebas de diseño

# 5 Conclusiones

## 5.1 Competencias desarrolladas

## 5.2 Líneas de trabajo futuro

# 6 Bibliografía