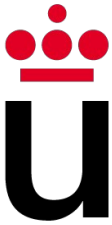


Universidad
Rey Juan Carlos

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

2 Objetivos

2.1 Lista numerada

2.2 Requisitos funcionales

2.3 Requisitos no funcionales

3 Descripción de la solución

3.1 Diseño

3.1.1 Metodología

En el diseño de la aplicación hemos estudiado principalmente dos modelos de desarrollo de software: desarrollo en cascada y desarrollo en espiral. Centrándonos finalmente en el desarrollo en espiral al considerar que era el que mejor se adaptaba a las necesidades de este proyecto.

A continuación, vamos a comentar los dos enfoques elegidos sobre el desarrollo de software y el porqué nos hemos decantado por el desarrollo en espiral.

3.1.2 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. Se finaliza con la elaboración de un *documento de especificación de requisitos*, que 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 independiente posible. De esta forma, en caso de tener un equipo, se optimiza el trabajo. Esta fase concluye con la elaboración de un documento de diseño de software, que contiene una descripción de 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, así como 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: Cuando cada uno de los elementos en los que se ha dividido el trabajo global se combinan, se realizan pruebas para asegurar que el software funciona sin errores y que cumple con los requisitos establecidos inicialmente.
- Implantación: Se trata de una de las fases finales del proyecto donde el software se pone en producción. Es una de la fases de más duración.
Se elabora una memoria de errores, fallos y mejoras para solucionar en la etapa de mantenimiento, o para implementar en futuras versiones del proyecto.
- Mantenimiento: Esta fase 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.

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 he 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 programa en un nuevo ciclo.
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 producto una vez terminado.

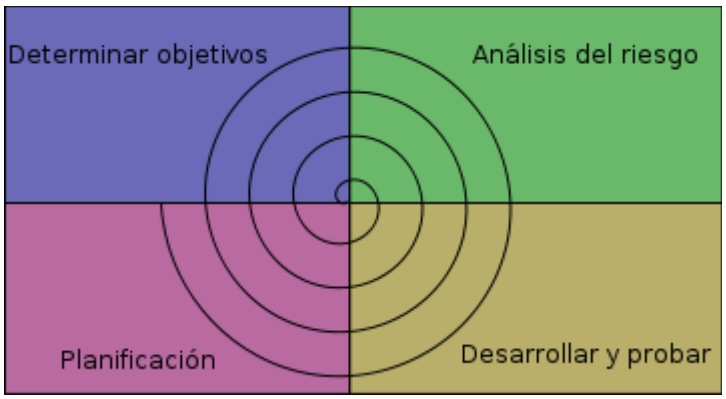
Es por todo lo mencionado anteriormente que para la elaboración del proyecto he usado un *diseño en espiral*.

3.1.2 Diseño 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 especificos. Se identifican los problemas y restricciones del proceso y del producto, y se crea un plan detallado de actuación consistente en la definición de recursos y tiempo necesarios. Es decir, se evalúan todos los requirimientos 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 todas las tareas para construir el nuevo nivel que se quiere implementar en la aplicación. Es en este punto, donde 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 de desarrollo general 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 producto totalmente terminado y funcional.



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 desarrollo de un software, en el acaso 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, en la aplicación, objeto de la presente memoria, he utilizado el modelo en espiral.

3.1.3 Diseño de la base de datos

Para el diseño de la base de datos hemos estudiado primero las bases de datos relacionales SQL y las bases de datos no relacionales, que diferencias existen entre ellas y cual es la mejor opción a utilizar deacuerdo a la tecnología que estamos utilizando.
Aunque finalmente se ha decidido utilizar una base de datos no relacional, en las subsecciones siguientes se van a describir brevemente ambas alternativas y las las razones del porqué del uso de este tipo de base de datos.

3.1.3.1 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 informacion 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 estan 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 Javacript. 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.

3.1.3.2 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 porque 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 estudiando 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 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 la colección utilizada en el proyecto:

User fields

{“_id”: string,

```
"usuario": string,
"pass": coded string,
"stats": {
        "wins": int,
        "losts": int,
        "draws": int,
        "total": int,
        "retired": int
        }
};
```

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 codificada ya que aunque la interacción del servidor con la base de datos está protegida por el propio servidor y la propia base de datos, no es algo sobre lo que tengamos control o que esté dentro de los límites de este proyecto.

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 en las transacciones de información que nos hagan plantearnos con profundidad las alternativas y características de otras bases de datos. Por lo tanto, haber optado por la sencillez de uso de MongoDB es una buena elección.

3.1.4 Imagen

3.2 Implementación

3.2.1 Javascript y html5

3.2.2 Jquery

3.2.3 Aplicaciones híbridas

3.2.4 Cordova vs Phonegap

3.2.5 Node.js

3.2.6 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 donde 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, busquedas 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.

3.2.8 webSockets

3.2.9 Express

3.2.10 Diseño de imágenes

3.3 Pruebas

3.3.1 Problemas-detalles

3.3.2 Encuestas

3.3.2.1 Pruebas de uso

3.3.2.2 Pruebas de diseño

4 Conclusiones

4.1 Competencias desarrolladas

4.2 Líneas de trabajo futuro

5 Bibliografía