Universidad de Valladolid

E.T.S. DE INGENIERIA INFORMATICA

Grado en Informática

Plataforma de juegos asíncronos para dispositivos móviles: Análisis crítico y propuesta.

Alumno: Adrián Mediavilla Zurita

Tutor: Jesús M. Vegas Hernández

Contenido

Índice de ilustraciones

Índice de tablas

1. Introducción

Desde el origen de los dispositivos móviles inteligentes como teléfonos y tabletas la oferta de aplicaciones no ha parado de crecer a un ritmo enorme. El entretenimiento siempre ha ocupado uno de los puestos más importantes en la demanda de aplicaciones, en un inicio la mayoría de los juegos ofrecían al usuario un modelo de juego muy similar al de plataformas mucho más anteriores como el ordenador o las videoconsolas.

Sin embargo la forma en que muchos usuarios usan su teléfono móvil no se asemeja al tipo de interacción que los juegos tradicionales necesitan: gran cantidad de tiempo dedicado de forma continua y frecuente para poder progresar. Hoy en día la mayoría de los usuarios usan el teléfono móvil muchas veces al día durante poco tiempo, modelo motivado, en gran parte, por las aplicaciones sociales y de mensajería.

Al intentar compatibilizar este modelo de interacción "intermitente" con el factor social cada vez más presente en los juegos y aplicaciones como, por ejemplo, jugar una partida con un amigo surge el concepto de juego asíncrono.

Un juego asíncrono es un formato que no requiere que los usuarios coincidan temporalmente para que la partida transcurra con normalidad. Las acciones de los jugadores modifican el estado de la partida quedando, normalmente, a la espera de que el o los oponentes realicen las suyas por ello muy frecuente que los juegos asíncronos se ajusten a un modelo de rondas o turnos.

En este proyecto se analizarán los requisitos necesarios para desarrollar una plataforma sobre la que desarrollar este tipo de juegos para posteriormente ofrecer una solución que englobará análisis, diseño, implementación y pruebas de la plataforma en cuestión y una pequeña aplicación de prueba.

ELEMENTOS DE UN JUEGO ASÍNCRONO

2. Elementos de un juego asíncrono

El primer paso que se ha realizado para poder realizar una propuesta para la plataforma de juegos asíncronos es analizar aquellas funcionalidades y requisitos de grano grueso que nuestro sistema debe reunir.

Para ello se ha estudiado numerosos juegos asíncronos, analizando aquellas funcionalidades que sería interesante que nuestro sistema incorporara y, una vez formada una idea más concreta sobre las funcionalidades existentes en la plataforma, explorar las opciones actualmente existentes en el mercado que puedan ayudarnos a resolver el problema.

Las diferentes funcionalidades, requisitos o subsistemas que han resultado del primer análisis son los siguientes:

- Modelo Asíncrono
- Usuarios
- Estadísticas y logros
- Rankings
- Almacenamiento de preferencias de usuario
- Mensajería entre usuarios
- Notificaciones push
- Sistema de partidas y emparejamiento de jugadores
- Soporte para los juegos

A. Modelo Asíncrono

Los juegos que albergará nuestra plataforma siguen un modelo asíncrono, es decir, cada jugador podrá realizar acciones y movimientos sin la necesidad de que el resto de jugadores coincidan estén usando la aplicación en ese mismo momento. Inclusive una partida podría empezar y terminar sin que los participantes hayan coincido usando la aplicación en el mismo momento.

Para este modelo resultan muy naturales aquellos juegos que tienen una estructura de rondas o turnos como el ajedrez, los juegos de cartas, la mayoría de juegos de tablero, etc.

La premisa de que los oponentes pueden no estar disponibles en ningún momento parece ser in indicador de que un modelo completamente distribuido muy probablemente no sea una opción, haciéndose necesario un servidor.

Con el servidor en escena se plantean dos opciones posibles:

- El servidor únicamente se encargue de almacenar y entregar los mensajes enviados por los clientes, siendo estos los encargados de mantener el estado de la partida. Este modelo sería el equivalente a jugar una partida de ajedrez por correo, nuestro servidor realizaría la función de servicio postal, siendo los clientes los encargados de actualizar su estado cada vez que reciben un nuevo mensaje (jugada en este caso) del oponente.
- El servidor mantiene una copia del estado de la partida, de esta forma filtra aquellos mensajes que sean incorrectos o fraudulento. Este modelo simplifica los clientes y su modelo para comunicarse ya que pueden confiar en que los mensajes que reciben son correctos y están validados a costa de un mayor coste computacional y almacenamiento en el servidor.

El primer modelo tiene varias implicaciones:

- Multi-dispositivo y borrado de datos: Situaciones en las que uno de los clientes no disponga de toda la información sobre la partida como puede ser que la partida se juegue desde varios dispositivos a la vez o borre la información local (desinstalando la aplicación por ejemplo) hace que esta opción requiera mecanismos adicionales para que los clientes puedan recuperar o reconstruir el estado de la partida.
- Seguridad: Es más vulnerable a usuarios maliciosos al perder la validación realizada por el servidor.
- Sincronismo: Presenta mayores dificultades para mantener el juego un estado sincronizado, especialmente si el juego no está basado en turnos, haciendo que el protocolo de comunicación entre los clientes se complique. Por ejemplo: Dos jugadores intentan realizar una jugada al mismo tiempo, si la jugada de uno de los jugadores hace que la del otro sea inválida. En el segundo modelo en servidor se encargaría de realizar las operaciones de forma atómica, denegando la segunda acción recibida.
- Generalización: Podemos ver el primer modelo como un caso especial del segundo modelo, en el cual el servidor da por bueno cualquier mensaje que recibe, y lo transmite al resto de jugadores.

Debido a estas razones, a priori, parece que la opción de un servidor más “listo” que entienda y valide las jugadas parece la opción más recomendable.

B. Usuarios

Una característica deseable la mayoría de los juegos y otras aplicaciones es dar la opción a los usuarios de registrarse y autenticarse en el sistema.

Aunque es frecuente que los servidores necesiten identificar los usuarios que usan las aplicaciones es común que registro por parte del usuario sea opcional, permitiendo a los usuarios acceder al sistema sin autenticarse o como usuarios anónimos generados automáticamente por el sistema.

Un sistema de usuarios abre nuestra aplicación a numerosas funcionalidades que aportan un valor adicional a nuestra aplicación:

- Autenticación en otros dispositivos, plataformas o tras un borrado de los datos del cliente.
- Almacenado remoto y sincronización en cliente de progresos, preferencias, estadísticas y otros datos.
- Interacción social, permitiendo a los usuarios descubrir, asociarse, invitar y estar en contacto con otros usuarios.
- Multiusuario en un mismo dispositivo mediante la gestión remota de los perfiles asociados a los diferentes usuarios.

Todas estas características son interesantes para los juegos asíncronos que se desarrollarán para nuestra plataforma siendo la principal la opción de continuar jugando las partidas en curso tras autenticarnos en el sistema con nuestro usuario.

Existen diferentes formas para registrar y autenticar a los usuarios. En la actualidad la más frecuente es registrar al usuario mediante un nombre de usuario, contraseña y correo electrónico, sirviendo este último para verificar el registro (enviando un email a la dirección proporcionada) y para restaurar el acceso a la cuenta para aquellos usuarios que olviden su contraseña.

Debido a los pasos que hay que dar para registrar una cuenta (formularios, confirmaciones, etc.) y las implicaciones de seguridad y privacidad asociadas el proceso de registro es evitado por una parte considerable de los usuarios, en caso de que sea opcional, siendo en último caso un factor de perdida de usuarios potenciales al encontrar este paso obligatorio. Para paliar este efecto cada vez es más frecuente que las aplicaciones permitan a sus usuarios identificarse mediante otras plataformas muy populares como Facebook, Twitter o Google+ mediante el protocolo OAuth, eliminando la necesidad de verificar el correo electrónico y ahorrando al usuario otra contraseña que recordar.

De este análisis podemos obtener varias funcionalidades que el sistema de usuarios de nuestra plataforma de juegos asíncronos podría reunir:

- Registro
- Autenticación

- Usuarios anónimo
- Acceso OAuth
- Servicios accesorios (recuperar contraseña, cambiar contraseña, etc.)

Más adelante estudiaremos más en profundidad la importancia e implicaciones de cada una.

C. Estadísticas y logros

Un logro es un reto planteado por el juego que generalmente está fuera de los objetivos planteados por las misiones y niveles del mismo. Un logro puede contemplar metas como objetivos secundarios, grados extra de dificultad como por ejemplo completar un nivel en un tiempo determinado, repetitivas como por ejemplo jugar un número determinado de partidas o pueden ser objetivos inherentes al juego en sí como completar un nivel.

El objetivo de los logros es extender la longevidad del juego más allá del objetivo general del juego, nivel o misión. De esta forma se ofrece a los jugadores no simplemente completar el juego, sino conseguir todos sus secretos y desafíos, fomentando que los jugadores experimenten con características o estilos de juego que normalmente no utilizarían.

Normalmente los logros aportan puntuación adicional y se comparten con la comunidad formando parte del perfil en línea del jugador, permitiendo a los usuarios comparar sus progresos.

El concepto de logro está íntimamente ligado al de estadística ya que podemos plantear un logro como la consecución de un objetivo en el valor de una o varias estadísticas.

Un sistema de estadísticas nos permite mantener una serie de valores asociados a un usuario, como por ejemplo el número de partidas que ha jugado o las balas disparadas en un videojuego tipo “shooter”.

De este análisis y del estudio del sistema de estadísticas y logros de otros juegos y plataformas como “Xbox life” o “Steam” se desprende que nuestra plataforma podría incluir las siguientes funcionalidades referentes al subsistema de estadísticas y logros.

- Gestión y sincronización de estadísticas de usuario.
- Logros.
- Logros ocultos (el usuario no conoce los objetivos y detalles hasta que los completa).
- Recompensas dentro del juego asociadas a logros.
- Consulta de estadísticas y logros de otros usuarios.

- Comparación entre las estadísticas y logros de usuario propios y las de otros usuarios.

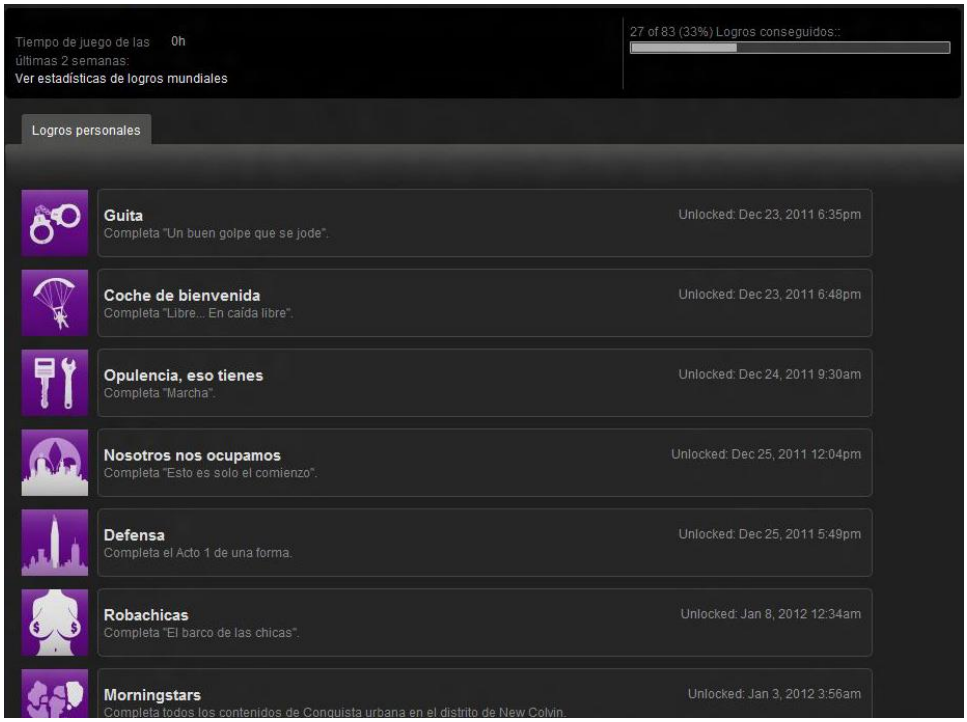


Ilustración 1: Pantalla de logros de Steam

D. Rankings

Un ranking consiste en una lista pública en la que posicionan los usuarios ordenados por su puntuación en el juego o en algún aspecto del mismo. El objetivo de los rankings es fomentar la competición entre usuarios.

También es frecuente que los juegos proporcionen rankings en los que únicamente se muestran las puntuaciones de los amigos de los usuarios, de esta forma se consigue que esta funcionalidad también sea atractiva para los jugadores que participan en la aplicación de una forma más casual.

En muchas ocasiones los juegos proporcionan un único ranking global basado en la puntuación total de los usuarios. Esto hace que el ranking no suponga un aliciente para aquellos usuarios hacen un uso más casual del juego.

Una de las funcionalidades más importantes de los rankings es la de permitir a los usuarios filtrar el número total de jugadores que participan en dicho ranking. De esta forma los usuarios pueden ajustar el alcance del desafío que escalar puestos en la clasificación supone. Alguno de estos filtros más frecuentes son: por nivel de dificultad del juego, por países, por servidores, por rango de tiempo, etc.

Ciertos juegos distinguen varios rankings en función del estilo de juego de los jugadores como por ejemplo en Travian, un juego de navegador basado en combates medievales, que dispone de rankings diferentes para los jugadores ofensivos y defensivos como puede verse en la figura de la derecha.

Agresores de la semana

No.	Jugadores	Puntos
1.	Trepoe	5885
2.	CFG	4648
3.	Tomatito	4507
4.	Eltontodelarpa	3986
5.	elb	3445
6.	Andromeda	3310
7.	lucius	2765
8.	Devil-x	2699
9.	roca	2660
10.	Icktorn	2628

Subidas de la semana

No.	Jugadores	Posiciones
1.	Scrott	4360
2.	control	3907
3.	chema_96	3859
4.	El lioo	3834
5.	Dark-yukai	3791
6.	ARodP11	3769
7.	baird	3745
8.	getyaya	3715
9.	DungaDunga	3714
10.	Arqtec	3651

Defensores de la semana

No.	Jugadores	Puntos
1.	Muerte	1077
2.	thalia	712
3.	MAKINATOR	631
4.	Agujeronegro	599
5.	Casio	589
6.	TRITON	555
7.	ezpok	494
8.	Colera	482
9.	dios de almas	472
10.	kikomatamoros	461

Atracadores de la semana

No.	Jugadores	Recursos
1.	zaw	2269252
2.	SKT1 PartinG	1872792
3.	Shinigami	1175514
4.	roca	1162931
5.	EAS	1117383
6.	TripleJ	1040453
7.	boikot	918515
8.	Icktorn	856084
9.	Eltontodelarpa	818071
10.	carletes	811831

Ilustración 2: Sistema de rankings de Travian

E. Notificaciones push

Existe una gran variedad de aplicaciones y juegos en los que el cliente requiere de algún mecanismo para descubrir si hay datos nuevos en el servidor y los futuros juegos de nuestra plataforma asíncrona son, sin duda, uno de ellos. Sería ideal que nuestro teléfono o tableta nos informara de que nuestro oponente ha finalizado su turno y que nos toca jugar.

Para llevar esta información generada en servidor hasta nuestro cliente existen varias aproximaciones:

- Conexiones permanentes: El cliente siempre está conectado al servidor, de esta forma cuando se produce una actualización para el usuario, el servidor puede usar este canal para notificar al cliente.
- Sondeo por parte del cliente: Cada cierto tiempo el cliente consulta las posibles novedades que se han podido generar desde el último sondeo.

Estas opciones tienen diferentes desventajas que hacen que no sean soluciones adecuadas en la mayoría de los casos para una aplicación móvil ya que consumen gran cantidad de batería, recursos, tráfico de red, requieren que el dispositivo salga de suspensión periódicamente, no se comportan bien en situaciones conectividad baja o intermitente, no

son completamente instantáneos, etc. Estos efectos negativos se acentúan cuando hay varias aplicaciones utilizando este tipo de aproximaciones.

Para dar solución a este problema los diferentes sistemas operativos para dispositivos móviles como Android y iOS proporcionan un servicio de notificaciones push consistente, en la mayoría de los casos, en una conexión permanente con un único servidor, encargado de hacer llegar las notificaciones de las diferentes aplicaciones instaladas. Este modelo permite una gestión mucho más optimizada y eficiente de la conexión, las salidas de suspensión de los equipos y por tanto de la batería y los recursos del dispositivo.

Por tanto, es altamente aconsejable, que nuestra plataforma proporcione mecanismos, tanto en servidor como en cliente, para enviar y recibir información de una forma sencilla y eficiente a los clientes mediante los servicios que proporcionan principalmente Apple y Google.

Tanto Google como de Apple permiten el envío de notificaciones bastante limitadas en cuanto a cantidad de información, esto hace que sea necesario contemplar dos tipos de notificaciones: aquellas que no requieren comunicar muchos datos incorporando por tanto la información en la propia notificación y aquellas que necesitan transmitir gran volumen de datos, para las que se requerirá que el cliente recupere la información tras recibir la notificación.

F. Sistema de partidas

Todos los juegos que albergará la plataforma requerirán de una serie de funciones encaminadas a la creación y gestión de las partidas. Por tanto deberemos analizar en profundidad estas necesidades para incorporar en la plataforma aquellas que sean más generales, disminuyendo de esta forma los tiempos y costes de los desarrollos. A su vez será necesario proporcionar puntos de extensión con los que cada juego pueda desarrollar sus funcionalidades específicas.

A partir del estudio de juegos asíncronos existentes se han recogido las siguientes funcionalidades a nivel de partidas que podría albergar la plataforma:

- Listar las partidas en las que participa un usuario, diferenciando aquellas en las que el usuario puede realizar alguna acción y en las que debe esperar para poder actuar, por ejemplo, porque es el turno del oponente.
- Crear una nueva partida
    - Con oponente(s) aleatorio(s)
    - Con otros jugadores invitados por el usuario

- Posibilidad de aceptar y rechazar tanto partidas a las que el usuario ha sido invitado como emparejamientos automáticos.
- Buscar jugadores en el sistema para jugar: amigos, jugadores recientes, contactos de Facebook, Google+, etc.
- Notificar a un oponente: avisándolo de que es su turno y que el usuario está esperando para poder jugar.
- Abandonar una partida o rendirse: Motivado bien porque el usuario no tiene ninguna posibilidad de ganar la partida como porque el oponente ha dejado de responder a las jugadas durante un periodo prolongado de tiempo, haciendo imposible o demasiado lento el desarrollo del juego.

Al igual que con el resto de puntos de este apartado, esta enumeración tiene como objetivo proporcionar una idea más aproximada del alcance y funcionalidades básicos de nuestra plataforma, más adelante se estudiarán estas características, así como aquellas que puedan aparecer posteriormente en profundidad.

G. Sistema de juego

Otro requisito indispensable para nuestra plataforma es que ofrezca la posibilidad a los clientes de enviar mensajes relativos al juego en sí mismo, es decir, por ejemplo realizar un movimiento, consultar las puntuaciones de los jugadores o conocer cuántos movimientos o acciones puede realizar el usuario antes de finalizar su turno.

Para este tipo de comunicación es fundamental que la plataforma ofrezca mecanismos sencillos de extensión, sobre los que cada juego concreto pueda desarrollar su lógica de una forma sencilla y eficiente. Por tanto la principal tarea de la plataforma en este aspecto es gestionar la autenticación de los mensajes, la asociación de los mismos a una partida y su redirección al componente adecuado para su tratamiento.

Cabe destacar que no todos los mensajes deben ir asociados necesariamente a una partida en curso, sino que ciertos juegos también podrían necesitar enviar mensajes a nivel global, por ejemplo para conocer el tiempo restante que queda para que finalice un torneo.

COMPARACIÓN DE PLATAFORMAS Y TECNOLOGÍAS EXISTENTES

3. Comparación de plataformas y tecnologías existentes

Tras el análisis preliminar de los subsistemas y funcionalidades que podría tener nuestra plataforma ahora tenemos una idea más concreta del problema, en este apartado nos centraremos en estudiar las plataformas y tecnologías ya existentes en el mercado. Una vez analizadas las opciones disponibles podremos determinar en qué grado se ajusta cada una a las necesidades de nuestro sistema de juegos asíncronos multijugador. De este análisis inicial se pueden esperar tres posibilidades:

- Alguna plataforma existente se ajuste correctamente a nuestras necesidades.
- Que una (o más) soluciones estudiadas aporte alguna de las funcionalidades requeridas para nuestro sistema. En este caso habría que estudiar cómo podríamos, si fuese posible, aprovecharlas e integrarlas en la solución final.
- Que ninguna alternativa considerada satisfaga nuestras necesidades, en este caso sería necesario plantear una solución totalmente personalizada que se ajuste de forma flexible a los requisitos de los futuros juegos que se desarrollarían sobre el sistema.

Las plataformas y tecnologías que se han tenido en cuenta son las siguientes:

- **Google Play Game Services**: SDK desarrollado por Google para agilizar el desarrollo de juegos para Android, iOS y web.
- **SmartFoxServer:** Plataforma desarrollada por gotoAndPlay enfocada a facilitar el desarrollo los aplicaciones online multijugador multiplataforma. SmartFoxServer está disponible para Adobe Flash/Flex/Air, HTML5, Android, Unity3D, Apple iOS, Java, Windows 8 y C++.
- **ES5 ElectroServer:** Plataforma desarrollada por Electrotank que, al igual que SmartFoxServer, está enfocada a facilitar el desarrollo de juegos online multijugador para navegador y plataformas móviles. Electrotank permite el desarrollo sobre Unity, Java, Android, iOS, Xna, FI y HTML5.
- **XMPP:** Extensible Messaging and Presence Protocol es un protocolo abierto y extensible basado en XML originalmente ideado para mensajería instantánea.
- **HTTP:** Hypertext Transfer Protocol es un protocolo de transferencia de hipertexto siguiendo el esquema petición-respuesta entre cliente y servidor.
- **Sockets de red:** Un socket es una interfaz por la cual dos equipos pueden intercambiar un flujo de datos

A. Google Play Game Services

1. Descripción

Google Play Game Services es un sistema desarrollado por Google enfocado a facilitar y agilizar el desarrollo de juegos proporcionando un API con el que manejar ciertas funcionalidades muy frecuentes en el desarrollo de este tipo de aplicaciones como son:

- Logros
- Rankings
- Almacenamiento de datos en la nube
- Comunicación multijugador en tiempo real

Ilustración 3: Logotipo de Google Play Game Services

Google proporciona este API mediante librerías para Android, iOS y Web. Todas las funcionalidades están disponibles para las tres plataformas excepto el módulo para multijugador en tiempo real, que sólo está disponible para Android.

El sistema es gratuito siempre y cuando no se superen los 50 millones de peticiones al API diarios, límite a partir del cual habría que negociar con Google el coste (del cual no se han encontrado precios orientativos, aunque tomando como referencias otras APIs similares de Google los precios una vez superados los límites gratuitos de cortesía suelen ser muy elevados).

La librería por la parte del cliente implementa un modelo “Send-and-forget”, es decir gestiona el envío de los datos al servidor, almacenando y reintentando los envíos en caso de que la conexión a Internet no esté disponible. Esto facilita en gran medida la programación del juego.

Para poder realizar llamadas al API es necesario que el usuario se autentifique en nuestra aplicación con su cuenta de Google mediante oAuth2. Esto impone una fuerte restricción a los juegos que utilizan esta plataforma ya que mientras el usuario no acceda con sus datos de Google el juego debe tener desactivadas estas funcionalidades (o implementarlas por su cuenta). Esta limitación supone una desventaja muy importante ya que generalmente los usuarios son reticentes a registrarse o introducir sus datos en aplicaciones hasta que no se han familiarizado con la aplicación y les gusta

Las librerías para plataformas móviles (Android e iOS) incorporan llamadas para mostrar las pantallas con los rankings y logros, no permitiendo al cliente controlar ni modificar estas

vistas para mantener un aspecto reconocible y único entre las distintas aplicaciones que usan este sistema.

A continuación se va a realizar un análisis detallado de cada subsistema de esta plataforma lo que nos permitirá conocer en qué grado se ajusta a nuestras necesidades, descubrir requisitos más detallados para nuestra plataforma, conocer cómo Google ha solucionado estos problemas lo que sin duda nos servirá de ayuda en caso de tener que implementar una solución personalizada.

a) Logros

Cada logro en el sistema de Google Play Game Services tiene los siguientes atributos:

- Id: Campo de texto único que identifica el logro
- Nombre
- Descripción
- Icono
- Orden en la lista: Valor numérico que permite controlar el orden en que aparecen los logros en la pantalla correspondiente.

Un logro puede estar en uno de estos tres estados:

- Oculto: El usuario no conoce los detalles del logro, únicamente conoce su existencia pero no puede acceder a su descripción ni objetivos.
- Revelado: El usuario conoce los detalles del logro pero aún no lo ha completado
- Desbloqueado: El usuario ha conseguido los objetivos del logro y por tanto aparece como completado.

Nota: *El progreso de los logros de un usuario se guarda de forma local en el cliente, permitiendo, por ejemplo, desbloquear un logro estando sin conexión de red. Cuando el usuario accede a Internet de nuevo el progreso se sincroniza.*

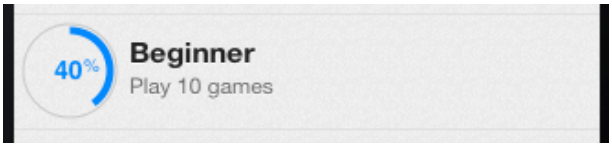


Ilustración 4: Representación de un logro incremental en Google Play Game Services

Cuando un usuario completa un logro la librería automáticamente notifica por pantalla al usuario el logro que acaba de conseguir, pero, salvo en el caso de la librería para clientes web, el API del sistema no permite acceder al estado de los logros, rankings, etc.

Logros Incrementales: El concepto de logro incremental consiste en un logro que requiere que el usuario alcance un cierto objetivo un número de veces determinadas. A medida que el usuario va consiguiendo esos objetivos el progreso del logro aumenta hasta que se desbloquea.

Un ejemplo de logro incremental sería "Completar 10 partidas" cada vez que el cliente completa dicho logro el contador de veces conseguidas aumenta, al llegar al 10 el usuario desbloquea el logro.

Puntos: Cada logro en el sistema de Google tiene asociado un valor de puntos. El valor en puntos de un logro debe ser múltiplo de 5 y no puede ser mayor de 200, además, la suma de puntos de todos los logros no puede superar los 1000 puntos. Esta equivalencia en puntos permite mostrar al usuario el progreso en cuanto a desbloqueo de logros se refiere.

Código: El código necesario para interaccionar con el subsistema de logros es realmente sencillo. Existiendo en el API tres llamadas:

- **unlockAchievement(achievementId):** Desbloquea el logro indicado
- **incrementAchievement(achievementId,count):** Incrementa un logro incremental la cantidad de veces indicada.
- Para los clientes móviles proporciona una llamada que muestra la pantalla de logros.

Una carencia importante del sistema de logros es que no permite otros usuarios ver el progreso del resto de jugadores.

Otro aspecto a destacar que el sistema de logros de Google Play no permite establecer niveles dentro de los logros. Es muy habitual que los juegos presenten gran cantidad de logros para dar aliciente hasta a los jugadores más fidelizados, con este sistema para establecer 10 logros consistentes en ganar 10, 20, 30,etc hasta 100 partidas, el usuario vería todos esos logros de forma simultánea. Una opción más deseable es que todos estos logros se agrupen dentro de una categoría, permitiendo al usuario consultar el último logro conseguido dentro de la categoría y los requisitos para desbloquear el siguiente nivel.

2. Rankings

El sistema de rankings de Google Play Game Services permite definir una o más listas de ranking. Cuando el usuario completa un nivel, misión, objetivo, etc. el cliente tiene que reportar la puntuación al sistema de rankings. Si la puntuación obtenida es mejor que el mejor valor existente hasta el momento, la nueva puntuación es remplazada con el nuevo valor. Esta gestión la realiza la librería de cliente automáticamente.

La puntuación de un ranking puede ser un valor numérico, un tiempo o una moneda, existiendo dos tipos de ponderaciones: “Larger is better” donde una puntuación es mejor cuanto más baja sea y “Smaller is better” donde una puntuación es mejor cuanto mayor sea su valor numérico.

A la hora de visualizar los rankings el sistema contempla dos tipos de filtros completamente gestionados por la librería del cliente, liberándonos de programarlo para cada aplicación:

- Temporal: El usuario puede filtrar los resultados por el momento en que se consiguió, pudiendo consultar el ranking con las mejores puntuaciones del día, semanales o globales
- Social: El sistema permite ver las puntuaciones de todos los usuarios o filtrar la lista, mostrando únicamente aquellos usuarios relacionados con el usuario mediante la red social Google +.

Un ranking consta de los siguientes atributos:

- Id: Identificador textual único.
- Nombre
- Icono
- Orden: Valor numérico que permite ordenar en qué orden aparecen los diferentes rankings de la aplicación entre ellos
- Límites: Valores máximos y mínimos de las puntuaciones que pueden aparecer en el ranking, permitiendo descartar automáticamente aquellos valores que son claramente fraudulentos.

El panel de administración el juego también permite personalizar el formato de las puntuaciones: valores decimales, tiempos, monedas, etc.

Código: Igual que ocurría con el sistema de logros las llamadas posibles al API de rankings son extremadamente sencillas, existiendo una llamada para mostrar la pantalla de alguno de los rankings (o todos) y otra llamada **submitScore(rankingId, score)** para modificar la puntuación del usuario en ese ranking.

3. Almacenamiento en la nube: Cloud Save

Un problema muy frecuente en los videojuegos consiste en el guardado y recuperado de información relativa al juego entre los diferentes dispositivos del usuario. En muchas ocasiones es deseable que el progreso, puntuaciones y demás elementos se sincronicen, por ejemplo, entre el teléfono y el tableta del usuario.

Para simplificar este proceso el sistema de Google incorpora un módulo llamado Cloud Save permitiendo sincronizar datos del usuario entre diferentes dispositivos y plataformas.

Esta librería presenta un API muy simplificado que resuelve bastante bien el problema de la sincronización:

Para cada usuario de la aplicación el sistema proporciona 4 “slots” de 128 KB cada uno (512KB en total) para los que se proporcionan dos llamadas: leer y escribir. Ambas operaciones se realizan a nivel de slot, es decir, no se puede leer o escribir una parte de un slot sino que hay que actuar sobre el slot completo. El formato de la información de un slot es un array de bytes.

La librería automáticamente gestiona el almacenamiento y envío de la información en el caso de que la red no esté disponible en ese momento, esto hace posible que se den situaciones de conflicto (similares a las que nos podemos encontrar en un control de versiones) si el usuario utiliza el sistema desde varios dispositivos.

Cuando se alcanza una situación de conflicto, el sistema realiza un callback proporcionando la información que se encuentra en el slot guardado en el servidor y la información de que la aplicación dispone de forma local. En esa llamada debemos proporcionar la información que se desea guardar como resolución de dicho conflicto. El algoritmo para resolver el conflicto debe ser lo suficientemente flexible y benévolo para que el usuario vea su objetos, progreso, y/o puntuación perdidos en el proceso.

Código: Al igual que el resto de subsistemas el código para utilizar Cloud Save está realmente simplificado:

Para guardar datos:

```
byte[] my_app_state = ... ;
mCloudSaveClient.updateState(this, key, my_app_state);
```

Para recuperar datos

```
byte[] state=mAppStateClient.loadState(this, key);
```

Para resolver los conflictos se implementa el callback onStateConflict:

```
@Override
public void onStateConflict(int stateKey, String versionId,
        byte[] localData, byte[] serverData) {

    byte[] resolvedData = resolveConflict(localData, serverData);
    mAppStateClient.resolveState(this, versionId, resolvedData);
}

byte[] resolveConflict(byte[] localData, byte[] serverData) {
    // consolidate conflicting data here as appropriate to your application
    return resolvedState;
}
```

Este subsistema de la librería aporta una funcionalidad muy interesante a la vez que resuelve un problema muy frecuente y laborioso de resolver de forma robusta. La única desventaja importante es la aplicación no puede superar de ninguna forma los 512KB de almacenamiento divididos en 4 slots, limitación que en ciertos juegos puede resultar insuficiente.

4. Multijugador en tiempo real

Google Play Game Services proporciona un API para crear de una forma sencilla una partida para que varios usuarios jueguen de forma simultánea. La librería de Google gestiona automáticamente todas las conexiones de red y se encarga de todos los aspectos de red a bajo nivel.

El sistema se organiza de forma similar a un sistema de chat, existiendo dos conceptos fundamentales:

- Sala (Room): Espacio virtual donde tiene lugar la partida. Los jugadores se pueden unir a una sala mediante una invitación o gracias al emparejador automático. Los jugadores en una misma sala pueden intercambiar información sobre la partida.

- Participante (Participant): Un jugador en una partida es un participante. Los jugadores pueden mandar invitaciones a otros usuarios en sus círculos Google+ para que se unan a sus salas o pueden solicitar un emparejamiento automático con un jugador aleatorio. Si dos jugadores son emparejados ambos se unen a una misma sala.

Ciclo de vida del sistema multijugador en tiempo real:

- Autenticación: Para que un usuario pueda participar en el sistema necesita estar autenticado en la aplicación con su cuenta de Google+.
- Inicialización de la sala: El jugador puede iniciar una nueva partida invitando a otros usuarios a unirse a su sala, solicitando un emparejamiento aleatorio o combinando invitaciones y emparejamiento automático en caso de que haya más de dos jugadores. Una sala se crea y se envían las invitaciones pertinentes. El cliente es notificado mediante la llamada **onRoomCreated**() de este evento.
- Aceptación de invitaciones: El jugador que creó la partida se une de forma automática a la sala, el resto de jugadores lo hacen al aceptar la invitación. Nuestro cliente es notificado cada vez que un nuevo jugador se une a la sala mediante la llamada **onConnected().**
- Interconexión de participantes: A medida que los usuarios se unen a la sala Google Play va comunicando los jugadores mediante una red de conexiones peer-to-peer. Cuando cada jugador está conectado con todos los demás el juego puede comenzar, los clientes son notificados mediante la llamada **onRoomConnected**(). A partir de este punto la implementación del juego depende de la aplicación, por ejemplo cada jugador podría mandar mensajes al resto que irían actualizando su estado local, o se podría establecer que uno de los jugadores fuese el "host" haciendo que fuese el encargado de actualizar a los demás sobre el estado de la partida.
- Juego: Una vez la partida ha comenzado los jugadores pueden abandonarla pero no se pueden unir nuevos jugadores. En esta fase los jugadores pueden intercambiar información hasta que la partida termine. Esta comunicación se realiza mediante las llamadas **sendRealiableRealTimeMessage** y **sendUnreliableRealTimeMessage** para enviar mensajes peer-to-peer y **onRealTimeMessageReceived** para recibirlos.
- Cierre de la sala: Una vez terminada la partida los clientes deben desconectarse llamando al método **leaveRoom()**, cuando todos los jugadores han abandonada la sala esta pasa a considerarse cerrada.

5. Conclusiones

La impresión obtenida del análisis de Google Play Game Services es muy buena en cuanto a su relación trabajo de programación requerido / resultado. Sin embargo la necesidad de que el usuario acceda con su cuenta de Google+ limita enormemente las posibilidades de su

librería, especialmente en iOS donde es menos probable que el usuario cuente con una cuenta registrada con que acceder.

Otro punto negativo de este SDK es que no permite ningún punto de enganche donde conectar la plataforma con otros subsistemas, como por ejemplo un sistema de usuarios externo a Google+.

Estas desventajas junto con la nula personalización indican que el sistema está pensado para simplificar el desarrollo de juegos relativamente sencillos cuyos requisitos se ajusten a lo que la plataforma ofrece y por tanto, aunque nos ha ayudado en gran medida en la recolección de los requisitos que nuestra plataforma final debería reunir.

A continuación se muestra una tabla resumen con las ventajas e inconvenientes de Google Play Game Services:

Ventajas	Inconvenientes
✓ Muy sencillo y rápido de integrar ✓ El sistema escala de forma automática ✓ Gratuito si el juego no se masifica ✓ Modelo send-and-forget ✓ Desarrollado por Google ✓ Adoptado por numerosas aplicaciones	✗ Personalización inexistente ✗ No permite el acceso a los datos de otros usuarios ✗ Solo cliente: No permite el acceso y modificación desde el servidor ✗ No extensible ✗ Personalización inexistente

Tabla 1: Resumen Google Play Game Services

B. SmartFoxServer

SmarFoxServer es una plataforma desarrollada por la compañía italiana "gotoAndPlay" enfocada a facilitar y acelerar el desarrollo de aplicaciones multi-usuario, especialmente de juegos online multijugador.

El kit de desarrollo está presente para numerosas plataformas y sistemas operativos como Adobe Flash, HTML5, Java, Android, iOS, Unity, C++ y HTML5.

Ilustración 5: Logotipo de SmartFoxServer

1. Arquitectura

SmartFoxServer está diseñado en un sistema por capas, en el nivel más bajo se encuentra "**BitSwarm"**, que se encarga de proporcionar conectividad TCP/UDP, gestionar sesiones, seguridad de red, calidad de servicio y flexibilidad frente a desconexiones mediante el sistema HRC (High-Resilient-Connections), servicios de clustering, monitorización … mediante un sistema altamente escalable.

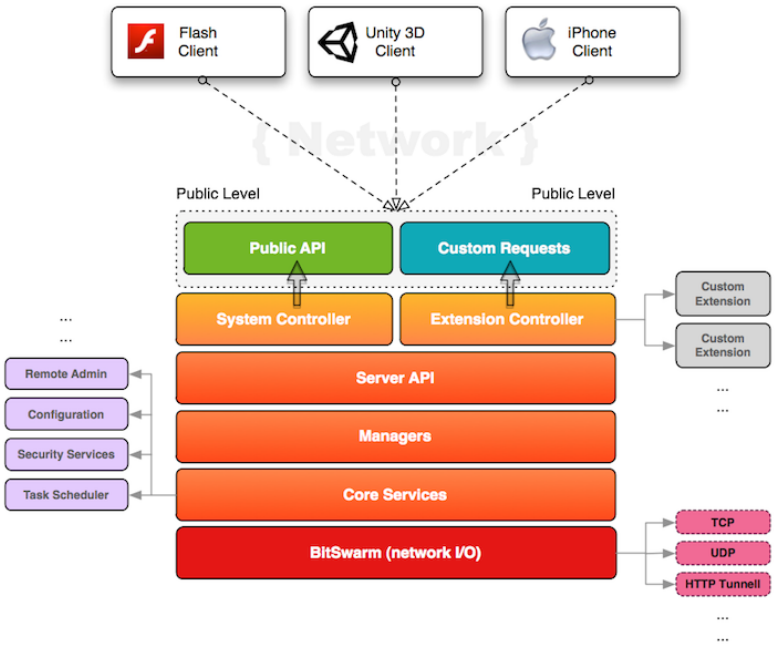


Ilustración 6: Arquitectura por capas de SmartFoxServer

Las capas "**Core Services**" y "**Managers**" se encargan de proveer una serie de servicios y controladores esenciales como configuración del sistema, gestión y autenticación de usuarios, seguridad, programación de tareas, administración remota, gestión de zonas y salas (de las que hablaremos más adelante), email, integración con bases de datos, etc.

SmartFoxServer proporciona a los desarrolladores la posibilidad de desarrollar y conectar con el sistema su propia lógica de la aplicación mediante la implementación de extensiones, que utilizan la capa del API de Servidor encargada de proporcionar mecanismos para gestionar las peticiones personalizadas por parte del cliente y los eventos del servidor.

Por último la plantea una capa de API de cliente, con la que interaccionan directamente los clientes y enfocada a ofrecer las funcionalidades de la plataforma accesibles por el cliente como crear una partida, interaccionar con otros usuarios, etc. de una manera sencilla.

En los puntos a continuación se estudiará la estructura y funcionalidades de la plataforma para poder determinar en qué grado se ajusta a las necesidades de nuestra plataforma de juegos.

2. El protocolo

La comunicación entre cliente y servidor se realiza mediante un protocolo binario optimizado para reducir el ancho de banda en la comunicación y aumentar la velocidad de parseo de los datos en cliente y servidor.

Los API de cliente y servidor proporcionan representaciones propias de los elementos Objeto y un Array que se serializan y comunican de forma transparente.

Esta conversión binaria supone un ahorro aproximado del 60% frente a la comunicación de los datos mediante texto, por ejemplo XML.

Adicionalmente cuando un mensaje supera una cierta longitud cliente y servidor lo comunican comprimido mediante GZip/Zip para reducir al máximo el ancho de banda y tiempo de transmisión necesarios.

3. Zonas y salas

SmartFoxServer organiza a los usuarios en un modelo jerárquico en el que la unidad más pequeña es la sala, que se agrupan en grupos y estos, a su vez, en zonas. El concepto de zona está orientado a separar diferentes aplicaciones y juegos que puede albergar un mismo servidor.

Una vez el cliente se ha unido a una zona, puede subscribirse y des-subscribirse de las diferentes zonas, recibiendo actualizaciones sobre nuevas salas creadas en el grupo, número de usuarios en las salas, etc. El objetivo de estos agrupamientos de salas es optimizar la cantidad de información que debe ser transmitida al usuario, permitiendo al cliente subscribirse solo a la información que le interese en cada momento.

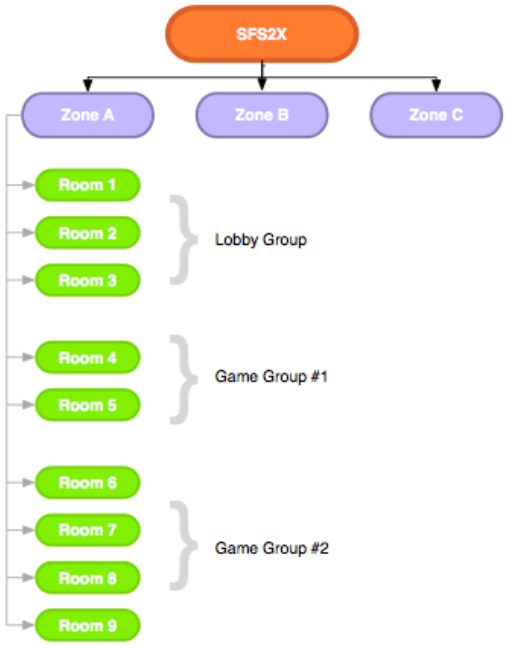


Ilustración 7: Modelo de Zonas y Salas de SmartFoxServer

Finalmente los usuarios pueden crear, unirse y abandonar las salas siempre que la configuración del servidor lo permita. Las salas pueden tener diferentes finalidades como permitir un chat entre usuarios, agrupar los jugadores esperando para jugar una partida, realizar la comunicación entre usuarios y con el servidor necesaria para el transcurso de un juego, comunicarse con una o varias extensiones del servidor, etc.

SmartFoxServer proporciona numerosas herramientas relativas a las zonas y salas como diferenciar entre salas regulares y salas de juego, asignando a los usuarios en estas últimas identificadores de jugador de forma automática y contemplando una diferenciación entre espectadores y jugadores que participan de forma activa en la partida.

4. Servidor y Extensiones

SmartFoxServer proporciona binarios para desplegar el servidor java en equipos Windows y Linux. La configuración del sistema está enormemente simplificada mediante los asistentes y los paneles de administración, evitando tener que manejar archivos de configuración en xml o texto.

Una vez desplegado el panel de administración permite configurar las zonas, salas, seguridad, permisos, módulos y demás elementos.

Con el servidor en funcionamiento el siguiente paso para implementar la lógica de nuestro juego o aplicación es desarrollar una o varias extensiones que entiendan los mensajes de extensión que envían los clientes y respondan de una forma acorde.

Las extensiones se desarrollan en java usando el "Extension API" y el "Java server-side API" el punto de entrada en una extensión es la recepción de un mensaje de usuario: **handleClientRequest**, a partir de la recepción de una solicitud la extensión analiza el tipo de la misma y sus parámetros y actúa en consecuencia, por ejemplo reflejando un movimiento en el estado de la partida.

5. Otras funcionalidades

SmartFoxServer proporciona numerosas funcionalidades accesorias para simplificar el desarrollo de las aplicaciones. A continuación se comentan brevemente las más significativas:

a) Buddy List

La última versión de esta plataforma proporciona un API llamado "Buddy API" enfocado a proporcionar una gestión de amigos, incluyendo la persistencia, los estados personalizados, eventos, posibilidad de establecer estados online/offline, mensajería entre amigos, etc.

b) BlueBox

BlueBox es una tecnología basada en HTTP que permite a los clientes conectar al servidor SmartFoxServer cuando, generalmente debido a firewalls o proxies, la conexión mediante sockets no está disponible. BlueBox encapsula el tráfico de la aplicación mediante un túnel HTTP utilizando mecanismos enfocados a eliminar la necesidad de que los clientes sondeen de forma continua al servidor buscando una comunicación lo más cercana posible al tiempo real.

c) RedBox

RedBox es un componente incorporado en la última versión de la plataforma que permite realizar la transmisión de audio y video en streaming. Cabe destacar que goToAndPlay no da soporte a esta funcionalidad.

d) HRC+

HRC son las siglas de High Resilient Connecition system, un sistema opcional que incorpora SmartFoxServer para gestionar las desconexiones involuntarias de los clientes, manteniendo su estado de forma transparente mientras reintenta la reconexión del servidor. Esta

característica depende en gran medida de la pila TCP del sistema operativo y la forma en que esta notifica a las aplicaciones las desconexiones abruptas.

Debido a la especial importancia de la gestión de las conexiones inestables en los dispositivos móviles se han realizado pruebas en situaciones de baja conectividad usan el sistema HRC+ y los resultados han sido bastante malos, no detectando las desconexiones en muchos casos, perdiendo paquetes y en ocasiones, incluso entrando en bloqueos que impedían al sistema SmartFoxServer seguir comunicándose con el servidor ni reiniciar la conexión.

e) Analytics

SmartFoxServer incorpora un módulo para transmitir a Google Analytics gran cantidad de métricas, permitiendo acceder a los informes que esta potente herramienta de Google puede proporcionar. Este módulo es opcional e incrementa entre 250 y 500€ el coste de la licencia de la plataforma.

6. Precios

goToAndPlay ofrece diferentes tipos de licencia en función de la cantidad de usuarios concurrentes (CCU) a los que se quiera dar servicio. Estas son los precios vigentes en Septiembre de 2013:

- 100 CCU: 350€
- 500 CCU: 750€ (+250€ con analytics)
- 2000 CCU: 1500€
- 5000 CCU: 2500€ (+500€ con analytics)
- CCU ilimitados: 3500€ (+500€ con analytics)

7. Conclusiones

La impresión general obtenida de SmartFoxServer ha sido bastante buena. Soluciona de forma correcta y eficiente los aspectos de gestión de zonas y salas, presencia de usuarios en las mismas y en el sistema y la comunicación entre usuarios y jugadores.

Las opciones que ofrece la plataforma en cuanto a personalización y extensión son también muy amplias y están bien diseñadas, por lo que, a priori, podría ser una solución excelente para un juego o chat en tiempo real.

SmartFoxServer también solucionaría los aspectos relacionados con la autenticación de usuarios, incluyendo, además de la posibilidad de extender y modificar su comportamiento, varias funcionalidades interesantes como el baneo de usuarios o protección anti flooding.

Sin embargo la plataforma presenta dos aspectos que hace que sea poco propicia para usarla como base en la construcción de nuestra plataforma:

- **Conexiones intermitentes**: Como la propia plataforma explica en su documentación del sistema HRC+ la gestión transparente de las desconexiones depende en gran medida de la pila TCP del dispositivo que ejecuta la aplicación y, para las pruebas realizadas con varios dispositivos Android, los resultados no han sido buenos. Esto no sería un problema importante en una aplicación de escritorio donde la conexión es relativamente estable, pero para el caso de los dispositivos móviles donde es extremadamente frecuente que el equipo cambie de red, antena, pierda cobertura, etc. este aspecto es muy importante.
- **Necesidad de usuarios conectados en tiempo real**: SmartFoxServer está concebido para permitir una comunicación rápida entre dos usuarios conectados a una misma sala donde se desarrolla la partida, generalmente manteniendo el estado de la partida en memoria. Este modelo choca completamente con el esquema de los juegos asíncronos donde la premisa es totalmente opuesta: los usuarios no tienen por qué coincidir temporalmente para que se desarrolle la partida.

  Para proporcionar la experiencia de un juego asíncrono deberíamos, por tanto, desarrollar extensiones que nos permitiesen guardar persistentemente el estado de la partida y consultar el estado de la misma así como recuperar la información pertinente cuando un usuario se conecta para poder conocer lo que ha pasado. Es decir, estaríamos usando SmartFoxServer únicamente para autenticar a nuestros usuarios y permitir la comunicación cliente-servidor en tiempo real, desaprovechando la mayoría de funcionalidades que la plataforma nos ofrece.

Debido principalmente a estas dos importantes desventajas parece que SmartFoxServer no proporciona una solución adecuada para nuestra plataforma de juegos asíncronos.

A continuación se presenta una tabla ilustrando las ventajas e inconvenientes de SmartFoxServer:

Ventajas	Inconvenientes
✓ Relativamente sencillo y rápido de integrar ✓ Paneles de administración fáciles de utilizar ✓ Gran cantidad de información sobre el estado del servidor en tiempo real ✓ Mantenido activamente por la compañía ✓ Resuelve eficientemente los aspectos relacionados con las gestiones de zonas, salas, presencia de usuarios, etc. ✓ Proporciona un sistema de autenticación y gestión de usuarios extensible ✓ Integración con Analytics	✗ **Requiere que los jugadores estén online** ✗ No almacena mensajes para usuarios desconectados (requeriría programar extensiones) ✗ Tiene coste económico ✗ No se puede acceder al código fuente ni modificar aspectos no contemplados en los mecanismos de extensión ✗ Gestión mejorable de conexiones con poca estabilidad en dispositivos Android ✗ Al final requiere un esfuerzo de programación para todo lo relacionado con las extensiones

Tabla 2: Resumen SmartFoxServer

C. Tecnologías para una solución personalizada

Como hemos visto en el estudio de las plataformas anteriores, ninguna encaja satisfactoriamente con las necesidades de nuestra plataforma de juegos asíncronos, por lo que lo más probable es que haya que plantear una solución personalizada.

En los siguientes puntos se estudiarán los principales protocolos que podrían ser empleados para realizar la comunicación entre cliente y servidor.

1. XMPP

Originariamente llamado Jabber, XMPP (Extensible Messaging and Presence Protocol) es un protocolo de comunicación basado en XML enfocado a la mensajería instantánea, información de presencia y mantenimiento de listas de contacto.

XMPP es un protocolo abierto y extensible que ha sido adoptado por numerosas empresas como Facebook, Tuenti, WhatsApp, Google, etc. Este último lo abandonó en 2013 en favor de su protocolo propietario Hangouts.

La arquitectura XMPP sigue un modelo descentralizado similar a la de correo electrónico, basada en nombre de dominio.

Existen numerosas implementaciones del estándar para clientes, servidores, librerías que pueden ser utilizadas de forma libre en su mayoría.

Ilustración 8: Logotipo XMPP

Además los estándares XMPP poseen robustos sistemas de seguridad como SASL y TLS para garantizar un intercambio de datos entre clientes y servidores seguros.

Las principales desventajas de este protocolo es la baja eficiencia en la transmisión de datos binarios, que deben ser codificados a formato base64 antes de ser transmitidos y la sobrecarga generada por los tráfico de datos de presencia, cercano al 70% del tráfico total de los servidores XMPP en el que el 60% de estos datos son redundantes.

Aunque el protocolo de transporte original de XMPP se basa en conexiones TCP de larga duración la comunidad XMPP ha desarrollado un transporte sobre HTTP para evitar el bloqueo de tráfico por los puertos XMPP (típicamente 5222 y 5223 para SSL) mediante dos sistemas:

- Polling: (Metodo deprecated en la actualidad) Los mensajes se almacenan en el servidor y son recuperados con sondeos regulares mediante las acciones HTTP ‘GET’ y ‘POST’.
- Binding: Se basa en el empleo de streams bidireccionales sobre HTTP síncrono (BOSH) que permite a los servidores entregar los mensajes en el cliente tan pronto como se envían, siendo más eficiente que el Polling.

XMPP cuenta con tres primitivas de comunicación: “message”, “presence” y “iq”.

- **Message:** Es el bloque de comunicación básico generalmente utilizado para la mensajería instantánea, chat en grupo, alertas, notificaciones, etc. Generalmente este tipo de mensajes no requieren confirmación de entrega por parte del receptor siendo un mecanismo para hacer llegar la información rápidamente de un punto a otro.
- **Presence:** Este tipo de bloques son utilizados para que las entidades en la red XMPP comuniquen su disponibilidad a otras entidades, permitiendo de esta forma conocer si un nodo está online y disponible para la comunicación. Los bloques presence son una indicación que indica de forma binaria si la entidad esta online u offline aunque el núcleo de presencia XMPP se extiende normalmente para soportar estados por parte de los usuarios.
- **IQ:** Los bloques IQ o Info/Query proporcionan una estructura para un formato de interacción solicitud/respuesta similar a los métodos GET, POST y PUT de HTTP.

2. HTTP

Hypertext Transfer Protocol es un protocolo usado en las transacciones World Wide Web. Define la sintaxis y la semántica que utilizan los elementos de software de la arquitectura web para comunicarse.

HTTP funcional con un formato petición respuesta entre cliente y servidor, el cliente envía un mensaje petición al servidor, quien responde con recursos como texto en HTML, ficheros y otro tipo de contenidos. La respuesta contiene información completa sobre el estado de la petición.

HTTP es un protocolo sin estado, es decir, no requiere que el servidor retenga información sobre los usuarios o sus estados a lo largo de múltiples peticiones, a pesar de ello existen diferentes mecanismos que permiten mantener el estado del usuario como variables ocultas en los formularios web, HTTP cookies o parámetros en las query.

HTTP dispone de diferentes métodos de petición. Inicialmente la especificación HTTP/1.0 definía los métodos GET, POST y HEAD a los que posteriormente la especificación HTTP/1.1 añadiría 5 nuevas opciones: OPTIONS, PUT, DELETE, TRACE y CONNECT.

- **GET:** Peticiona información sobre un recurso. Este tipo de peticiones deben solo recuperar información y no tener otros efectos.
- **HEAD:** Peticiona la misma información que GET pero únicamente recupera las cabeceras, ignorando el contenido del cuerpo de la petición. Es útil para recuperar la meta-información asociada.

- **POST:** Somete los datos a que sean procesados para el recurso identificado. Los datos se incluirán en el cuerpo de la petición. Esto puede resultar en la creación de un nuevo recurso o de las actualizaciones de los recursos existentes o ambas cosas.
- **PUT:** Petición para cargar y almacenar un recurso en la URI especificada, si el recurso ya existe este es remplazado.
- **DELETE:** Borra un recurso.
- **TRACE:** Petición que solicita al servidor que devuelva la misma información que ha enviado al cliente, este tipo de peticiones se usan generalmente con fines de comprobación y diagnóstico como por ejemplo comprobar los cambios realizados por servidores intermedios
- **OPTIONS:** Devuelve los métodos HTTP que el servidor soporta para una URL dada. También puede usarse para conocer la funcionalidad soportada por el servidor especificando '*' en lugar de un recurso específico.
- **CONNECT:** Petición que convierte la petición en un túnel TCP/IP, generalmente para facilitar la comunicación encriptada SSL (HTTPS) mediante un proxy HTTP sin encriptar.

4. Conclusiones

Tras analizar las diferentes plataformas y protocolos existentes los resultados no han sido demasiado buenos, ninguno de ellos se ha ajustado a las necesidades de nuestra plataforma asíncrona de una forma aceptable:

- **Google Play Game Services** nos ofrecía un sistema de logros y rankings perfectamente válido que, sin duda, podríamos haber utilizado en nuestra plataforma de no ser por la necesidad de que los usuarios accediesen con su cuenta de Google de manera obligatoria y por su extremada rigidez y falta de extensibilidad.
- **SmartFoxServer** proporciona una plataforma excelente para juegos en tiempo real, pero no considera en absoluto la comunicación asíncrona entre clientes desconectados, aunque su esquema de extensibilidad permitiría construir una solución para nuestra plataforma sobre SmartFoxServer, los requisitos de nuestra plataforma para juegos asíncronos nos obligaría a obviar la mayor parte del sistema, utilizándolo prácticamente como medio de transporte para las comunicaciones entre clientes y servidor.
- **XMPP**: XMPP sería una solución aceptable para la comunicación de nuestra plataforma, la amplia comunidad y popularidad con la que cuenta hace que exista gran cantidad de documentación, librerías y servidores disponibles que, sin duda, facilitarían el proceso de desarrollo.

  Sin embargo el modelo de conexión de larga duración que plantea el protocolo hace que sea necesaria una gestión avanzada de la conexión, especialmente en situaciones de baja conectividad de red, siendo esta situación muy frecuente en los dispositivos móviles. Esta labor de gestionar de una forma correcta las conexiones y desconexiones de los clientes plantea un trabajo no trivial adicional a considerar.

  Por otro lado, en el concepto general de juego asíncrono la comunicación entre cliente y servidor no es demasiado intensiva, por ejemplo en un juego de parchís por turnos cada jugador tirará un dado, elegirá la ficha que desea mover y excepcionalmente volverá a mover si come alguna ficha o saca un 6 en la tirada.

  Esta comunicación posiblemente podría realizarse en unas tres llamadas al servidor a lo largo de un par de minutos, quizás mantener una conexión permanente para una comunicación tan eventual sea demasiado costoso, tanto para el cliente como para el servidor.
- **HTTP:** Finalmente este es el protocolo que hemos elegido para el desarrollo de la plataforma, su esquema solicitud/respuesta encaja a la perfección con las necesidades del sistema en el que un jugador típicamente enviará sus movimientos o acciones y recibirá la confirmación por parte del servidor posiblemente informando del efecto de la acción en el estado de la partida.

Este esquema también encaja a la perfección para los servicios de consulta como rankings, estadísticas, etc.
Además, como se ha comentado en el punto anterior, el formato de comunicación para la gran mayoría de los juegos asíncronos no es demasiado intensivo, por lo que un único servidor web serviría para resolver las peticiones de miles de usuarios simultáneos.

SOLUCIÓN PERSONALIZADA PROPUESTA

5. Solución personalizada propuesta

Tras analizar todas las posibilidades llega el momento de plantear una solución que resuelva de forma correcta y eficiente nuestra plataforma de juegos asíncronos.

El primer paso a realizar es establecer el alcance del proyecto, el cual se ha definido teniendo en cuenta el tiempo para realizar el proyecto y priorizando aquellas funcionalidades que se han considerado como fundamentales frente a aquellas que son más accesorias.

Tras sopesar todas las posibilidades que se han ido viendo en otras plataformas y soluciones se ha decidido incluir en el proyecto las siguientes:

- Sistema de usuarios con registro, autenticación y soporte para usuarios anónimos.
- Sistema de estadísticas y logros
- Sistema de almacenamiento en la nube por parte del cliente
- Sistema de gestión de partidas y emparejamientos
- Sistema de notificaciones push
- Sistema de juego

Cabe destacar dos funcionalidades que no se han incluido por motivos temporales y de priorización de funcionalidades pero que también son muy interesantes y deberían tenerse en cuenta en trabajos futuros:

- Sistema de rankings y clasificaciones
- Sistema de chat

En esta sección se planteará el análisis y diseño realizado para las aplicaciones así como el código de una breve aplicación que, basándose en los mecanismos contemplados en la plataforma, permita jugar a un juego para dos jugadores de forma asíncrona: El tres en raya.

El proyecto se ha desarrollado dentro de la empresa Novagecko y la propuesta abarcará todos los aspectos relativos al cliente móvil Android, mientras que la parte de servidor será desarrollada por otros miembros de la empresa.

En primer lugar se establecerán las bases de comunicación entre cliente y servidor que será la que se utilice a lo largo de todos los subsistemas para posteriormente realizar el Análisis y diseño de cada módulo por separado. Finalizando en la aplicación de demostración que nos permita conectar todos los módulos.

A. Comunicación cliente-servidor

Tas la elección de HTTP como protocolo de comunicación el siguiente paso es definir el formato de la información que intercambiarán cliente y servidor. Inicialmente se consideraron XML y JSON y hemos elegido JSON para el intercambio de datos por ser un formato popular que usan la mayoría de servicios web actuales. Además existen multitud de librerías destinadas a su parseo de forma rápida y eficiente.

Para finalizar hemos definido el formato de los servicios web que dispondrá cada módulo de la siguiente forma: Los parámetros de entrada se enviarán al servidor por medio de los campos POST de la petición HTTP, utilizando texto plano para los campos de tipos básicos y texto en formato JSON para los campos complejos.

La respuesta del servidor será un texto en formato JSON con el campo “stat” de forma obligatorio. Este campo stat será un valor numérico similar al código de respuesta HTTP que permitirá identificar entre peticiones exitosas y peticiones erróneas, indicando en el campo stat el código del error.

El campo stat=0 corresponderá a una respuesta exitosa.

Un ejemplo de salida de un hipotético servicio de consulta de la hora sería:

{ “stat”:0, “time”:”11:23:42”}

En el anexo A pueden consultarse todos los códigos de respuesta utilizados.

B. Usuarios

En este apartado se desarrollará el subsistema de usuarios, inicialmente se especificarán los requisitos para posteriormente poder realizar el análisis y diseño del módulo, junto con la implementación y prueba de una pequeña demostración que agrupe todas las funcionalidades ofrecidas por el sistema de usuarios. El apartado de usuarios es de especial importancia ya que el resto los módulos y subsistemas lo usarán para aquellas peticiones que requieran autenticación.

La definición del protocolo asume un protocolo de transporte subyacente seguro, por lo que es frecuente el uso de TPC. Sin embargo también puede ser usado sobre protocolos no seguros como UDP (por ejemplo en SSDP).

1. Requisitos

UR 1. El sistema almacenará para cada usuario los siguientes atributos.

- Un identificador numérico único
- Un nombre de usuario único
- Un correo electrónico
- Contraseña
- Una URL con el avatar del usuario
- El país del usuario
- Los identificadores de usuario de aquellas plataformas enlazadas mediante OAuth

UR 2. Todos los atributos de un usuario son públicos a excepción de su correo electrónico, su contraseña y sus identificadores OAuth.

UR 3. El sistema permitirá a los usuarios registrarse y acceder mediante email/contraseña.

UR 4. El sistema permitirá a los usuario registrarse y acceder mediante el protocolo oAuth de servicios como Facebook, Twitter o Google+

UR 5. El sistema permitirá el registro de usuarios anónimos, a los usuarios “no anónimos” se los denominará registrados.

UR 6. El sistema permitirá diferenciar entre usuarios anónimos y usuarios registrados, tanto en cliente como en servidor.

UR 7. El sistema garantizará que todos los usuarios poseen los atributos que indica la tabla siguiente: (✓ Obligatorio, ? Opcional, ✗ No existente)

	Usuario Registrado	Usuario Anónimo
Identificador numérico	✓	✓
Nombre de usuario	✓	✗
Correo electrónico	✓	✗
Contraseña	?	✗
Avatar	?	?
País	?	?
Identificadores oAuth	?	✗

Tabla 3: Tipos de usuario

UR 8. El sistema deberá tener un sistema de validación de los email de usuario mediante el envío de un correo

UR 9. El sistema considerará los correos electrónicos obtenidos mediante oAuth automáticamente validados

UR 10. El sistema requerirá de forma obligatoria que aquellos usuarios que no se registren mediante oAuth establezcan una contraseña de acceso.

UR 11. El sistema podrá identificar el país del usuario mediante diferentes métodos: proporcionado por el cliente, geo-localizado a partir de la conexión del usuario o a través de plataformas de terceros como Facebook, Twitter o Gmail.

UR 12. El sistema proporcionará un mecanismo para convertir en usuario registrado a un usuario anónimo, este proceso se denominará “identificación de usuario anónimo”.

UR 13. En el proceso de registrar un usuario anónimo el sistema requerirá que se aporten, como mínimo, todos los datos que sean obligatorios para un usuario registrado

UR 14. El sistema permitirá a los clientes cambiar la contraseña de los usuarios.

UR 15. El sistema permitirá a los clientes recuperar la contraseña mediante el envío de un correo electrónico con las instrucciones para generar una nueva.

UR 16. Si un usuario sin contraseña desea recuperarla, el sistema proporcionará las instrucciones para generar una contraseña mediante correo electrónico.

UR 17. El sistema permitirá cambiar o generar una nueva contraseña a los usuarios registrados.

UR 18. El sistema requerirá la contraseña anterior (si existiera) en el proceso de cambio de contraseña por seguridad.

UR 19. El sistema permitirá comprobar si un email está registrado en el sistema.

UR 20. El sistema permitirá desvincular del usuario las identidades obtenidas de otras plataformas mediante oAuth.

UR 21. El sistema invalidará de forma automática las sesiones y tokens de acceso de un usuario tras un cambio de contraseña.

UR 22. El sistema no almacenará ni comunicará sin cifrar información sensible como correos electrónicos o contraseñas.

UR 23. El sistema proporcionará autenticación oAuth para que pueda ser usada por otros servicios y plataformas.

UR 24. El sistema requerirá que el usuario esté autenticado para cambiar la contraseña.

UR 25. El sistema requerirá que el usuario no esté autenticado para recuperar la contraseña.

UR 26. El sistema proporcionará autenticación a otros módulos o subsistemas mediante un token de acceso.

UR 27. El sistema permitirá la desconexión (log out) de un usuario en cualquier momento.

UR 28. El sistema deberá ser desarrollado para Android

UR 29. La comunicación entre cliente y servidor se realizará mediante el protocolo HTTP usando JSON para el formato de los datos intercambiados

2. Análisis

a) *Casos de uso*

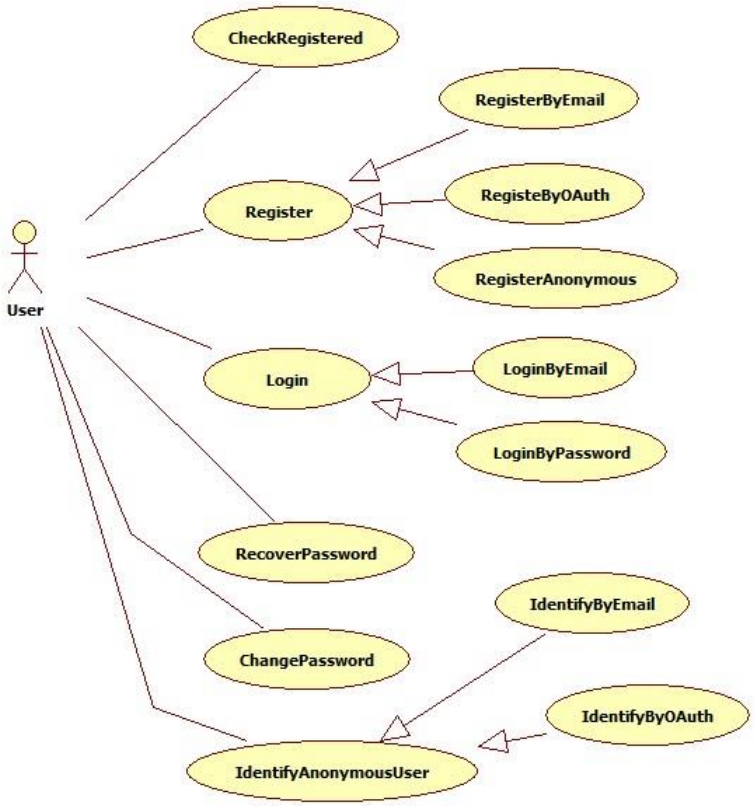


Ilustración 9: Diagrama de casos de uso del sistema de usuarios

UC- US-01	Check Registered	
Descripción	El sistema deberá comportarse tal y como se describe en el siguiente caso de uso cuando se solicite la comprobación de que un usuario está registrado en el sistema	
Precondición	Ninguna	
Secuencia normal	**Paso**	**Acción**
	1	El sistema comunica al servidor la dirección de correo a comprobar
	2	El servidor busca si el correo electrónico está registrado en el sistema
	3	El servidor comunica al cliente el resultado de la búsqueda
Postcondición	Se ha comprobado la existencia del usuario en el sistema	
Comentarios	Ninguno	

UC- US-02	Register by Email	
Descripción	El sistema deberá comportarse tal y como se describe en el siguiente caso de uso cuando se solicite el registro de un usuario	
Precondición	El usuario no está autenticado en el sistema	
Secuencia normal	**Paso**	**Acción**
	1	El usuario introduce su correo electrónico, su nombre de usuario y su contraseña
	2	El sistema comprueba que los datos para el registro son válidos
	3	El sistema registra el usuario los datos del registro
Excepciones	**Paso**	**Acción**
	2	Si el nombre de usuario ya existe el sistema comunica el error y el caso de uso queda sin efecto
	2’	Si el correo electrónico ya existe el sistema comunica el error y el caso de uso queda sin efecto
	2’’	Si los valores seleccionados no tienen formato válido el sistema comunica el error y el caso de uso queda sin efecto.
Postcondición	Se ha registrado al usuario en el sistema	
Comentarios	Ninguno	

UC- US-03	Register by oAuth	
Descripción	El sistema deberá comportarse tal y como se describe en el siguiente caso de uso cuando se solicite el registro de un usuario mediante un token oAuth de un sistema como Facebook o Gmail	
Precondición	El usuario no está autenticado en el sistema	
Secuencia	**Paso**	**Acción**

normal		
	1	El usuario proporciona su autenticación oAuth y su nombre de usuario deseado
	2	El sistema recupera el correo electrónico asociado a través del Access token
	3	El comprueba si los datos del registro son válidos
	4	El sistema registra al usuario y comunica al cliente los datos del registro
Excepciones	**Paso**	**Acción**
	3	Si el nombre de usuario ya existe el sistema comunica el error y el caso de uso queda sin efecto
	3’	Si el correo electrónico ya existe el sistema comunica el error y el caso de uso queda sin efecto
	3’’	Si los valores seleccionados no tienen formato válido el sistema comunica el error y el caso de uso queda sin efecto.
Postcondición	Se ha registrado al usuario en el sistema	
Comentarios	Ninguno	

UC- US-04	Register by oAuth	
Descripción	El sistema deberá comportarse tal y como se describe en el siguiente caso de uso cuando se solicite el registro de un usuario mediante un token oAuth de un sistema como Facebook o Gmail	
Precondición	El usuario no está autenticado en el sistema	
Secuencia normal	**Paso**	**Acción**
	1	El usuario proporciona su autenticación oAuth y su nombre de usuario deseado
	2	El sistema recupera el correo electrónico asociado a través del Access token
	3	El comprueba si los datos del registro son válidos
	4	El sistema registra al usuario y comunica al cliente los datos del registro
Excepciones	**Paso**	**Acción**
	3	Si el nombre de usuario ya existe el sistema comunica el error y el caso de uso queda sin efecto
	3’	Si el correo electrónico ya existe el sistema comunica el error y el caso de uso queda sin efecto
	3’’	Si los valores seleccionados no tienen formato válido el sistema comunica el error y el caso de uso queda sin efecto.
Postcondición	Se ha registrado al usuario en el sistema	
Comentarios	Ninguno	

UC- US-05	Register Anonymous	
Descripción	El sistema deberá comportarse tal y como se describe en el siguiente caso de uso cuando se solicite el registro de un usuario anónimo en el sistema	
Precondición	El usuario no está autenticado en el sistema	
Secuencia normal	**Paso**	**Acción**
	1	El usuario solicita el registro de un usuario anónimo
	2	El sistema comprueba que el registro puede efectuarse
	4	El sistema registra al usuario y comunica al cliente los datos del registro
Excepciones	**Paso**	**Acción**
	2	Si el regitro del usuario anónimo no puede registrarse el sistema notifica del error y el caso de uso queda sin efecto
Postcondición	Se ha registrado al usuario en el sistema	
Comentarios	Ninguno	

UC- US-06	Login by Email	
Descripción	El sistema deberá comportarse tal y como se describe en el siguiente caso de uso cuando se solicite el acceso en el sistema mediante nombre de usuario y contraseña	
Precondición	El usuario dispone de una cuenta en el sistema y no está autenticado en el mismo	
Secuencia normal	**Paso**	**Acción**
	1	El usuario solicita el acceso indicando su nombre de usuario y contraseña
	2	El sistema comprueba los datos de acceso
	3	El sistema concede acceso autenticado al usuario
Excepciones	**Paso**	**Acción**
	2	Si los datos de acceso no son válidos el sistema notifica del error y el caso de uso queda sin efecto
Postcondición	Se ha autenticado al usuario en el sistema	
Comentarios	Ninguno	

UC- US-07	Login by OAuth	
Descripción	El sistema deberá comportarse tal y como se describe en el siguiente caso de uso cuando se solicite el acceso en el sistema mediante un token de acceso OAuth (de Facebook o Gmail por ejemplo)	
Precondición	El usuario dispone de una cuenta en el sistema y no está autenticado en el mismo	
Secuencia normal	**Paso**	**Acción**
	1	El usuario solicita el acceso indicando su token de acceso
	2	El sistema comprueba el token de acceso contra el servidor proveedor del token, obteniendo su dirección de correo electrónico asociado
	3	El sistema concede acceso autenticado al usuario
Excepciones	**Paso**	**Acción**
	2	Si la validación del token de acceso no es válida el sistema notifica del error y el caso de uso queda sin efecto
Postcondición	Se ha autenticado al usuario en el sistema	
Comentarios	Ninguno	

UC- US-08	Recover Password	
Descripción	El sistema deberá comportarse tal y como se describe en el siguiente caso de uso cuando se solicite la recuperación de la contraseña de un usuario	
Precondición	El usuario dispone de una cuenta en el sistema y no está autenticado en el mismo	
Secuencia normal	**Paso**	**Acción**
	1	El usuario solicita la recuperación de su contraseña, proporcionando el email con que se registró en el sistema.
	2	El sistema comprueba los datos proporcionados
	3	El sistema envía un email con las instrucciones para generar una contraseña nueva al usuario
Excepciones	**Paso**	**Acción**
	2	Si el email no está registrado en el sistema notifica del error y el caso de uso queda sin efecto
	3	Si el envío del email no puede realizarse o falla el sistema notifica del error y el caso de uso termina sin efecto
Postcondición	Se ha enviado un email al usuario con las instrucciones para generar una nueva contraseña	
Comentarios	Ninguno	

UC- US-09	Recover Password	
Descripción	El sistema deberá comportarse tal y como se describe en el siguiente caso de uso cuando se solicite el cambio de la contraseña de un usuario	
Precondición	El usuario está autenticado en el sistema	
Secuencia normal	**Paso**	**Acción**
	1	El usuario proporciona su contraseña actual en el sistema y la nueva contraseña que desea establecer
	2	El sistema valida los datos proporcionados por el usuario
	3	El sistema actualiza la contraseña del usuario
Excepciones	**Paso**	**Acción**
	2	Si la contraseña actual no corresponde con la existente en el sistema se notifica del error y el caso de uso queda sin efecto
	2’	Si el formato de la nueva contraseña no es adecuado el sistema notifica del error y el caso de uso termina sin efecto
Postcondición	Se ha actualizado la contraseña del usuario	
Comentarios	Ninguno	

UC- US-10	Identify Anonymous By Email	
Descripción	El sistema deberá comportarse tal y como se describe en el siguiente caso de uso cuando se solicite la identificación de un usuario anónimo mediante email	
Precondición	El usuario está autenticado en el sistema y es anónimo	
Secuencia normal	**Paso**	**Acción**
	1	El usuario proporciona su correo electrónico, contraseña y nombre de usuario
	2	El sistema valida los datos proporcionados por el usuario
	3	El sistema actualiza el correo electrónico, contraseña y nombre de usuario del usuario anónimo, convirtiéndolo en usuario registrado
Excepciones	**Paso**	**Acción**
	2	Si el nombre de usuario ya existe el sistema comunica el error y el caso de uso queda sin efecto
	2’	Si el correo electrónico ya existe el sistema comunica el error y el caso de uso queda sin efecto
	2’’	Si los valores seleccionados no tienen formato válido el sistema comunica el error y el caso de uso queda sin efecto.
Postcondición	Se ha actualizado los campos email, contraseña y nombre de usuario de la cuenta y ahora es un usuario registrado.	
Comentarios	Ninguno	

UC- US-10	Identify Anonymous By OAuth	
Descripción	El sistema deberá comportarse tal y como se describe en el siguiente caso de uso cuando se solicite la identificación de un usuario anónimo mediante un token de acceso OAuth (de un servidor como Facebook o Gmail)	
Precondición	El usuario está autenticado en el sistema y es anónimo	
Secuencia normal	**Paso**	**Acción**
	1	El usuario proporciona su token de acceso y nombre de usuario
	2	El sistema valida los datos proporcionados por el usuario, extrayendo el correo electrónico asociado al token del acceso proporcionado
	3	El sistema actualiza el correo electrónico y nombre de usuario del usuario anónimo, convirtiéndolo en usuario registrado
Excepciones	**Paso**	**Acción**
	2	Si el nombre de usuario ya existe el sistema comunica el error y el caso de uso queda sin efecto
	2’	Si el correo electrónico ya existe el sistema comunica el error y el caso de uso queda sin efecto
	2’’	Si los valores seleccionados no tienen formato válido el sistema comunica el error y el caso de uso queda sin efecto.
Postcondición	Se ha actualizado los campos email, contraseña y nombre de usuario de la cuenta y ahora es un usuario registrado.	
Comentarios	Ninguno	

b) Diagrama de clases

A continuación se muestra el diagrama de clases de análisis del sistema de usuarios:

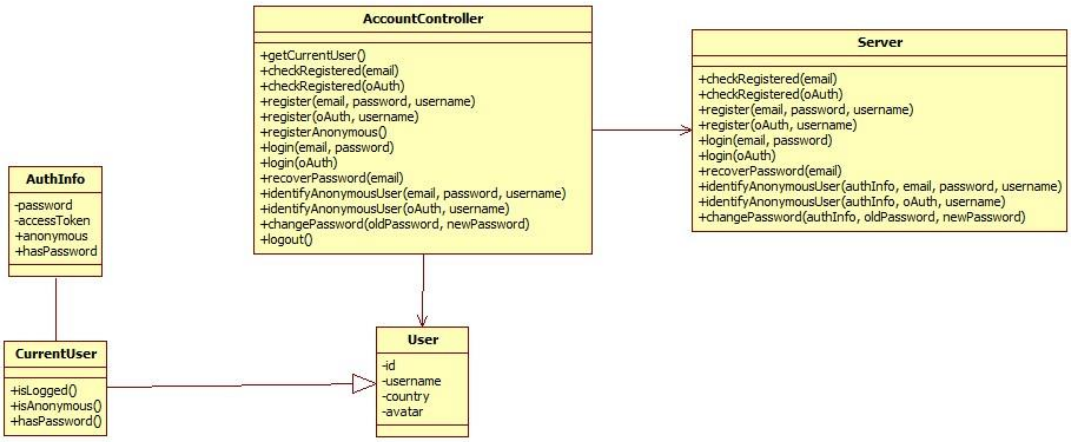


Ilustración 10: Diagrama de clases de análisis del sistema de usuarios

Descripción de las clases:

- **User**: Representa a un usuario del sistema permitiendo acceder a su información pública.
- **CurrentUser**: Clase que representa el usuario local en el sistema, que puede estar autenticado en el servidor y, en ese caso, tiene asociada una AuthInfo.
- **AuthInfo**: Representación de los datos de autenticación del CurrentUser. Incluye los credenciales de acceso como la contraseña o el accessToken (OAuth) y la información adicional sobre el usuario como si el usuario ha establecido su contraseña en el sistema o si es un usuario anónimo.
- **Server**: Clase que representa la interfaz pública (API) que el servidor de usuarios ofrece.
- **AccountController**: Controlador central del módulo de usuarios. Proporciona a la aplicación los métodos para conocer si el cliente esta autenticado en el sistema, registrarse, autenticarse, identificar usuarios anónimos, etc.

3. Diseño

a) Diagrama de clases

Librería:

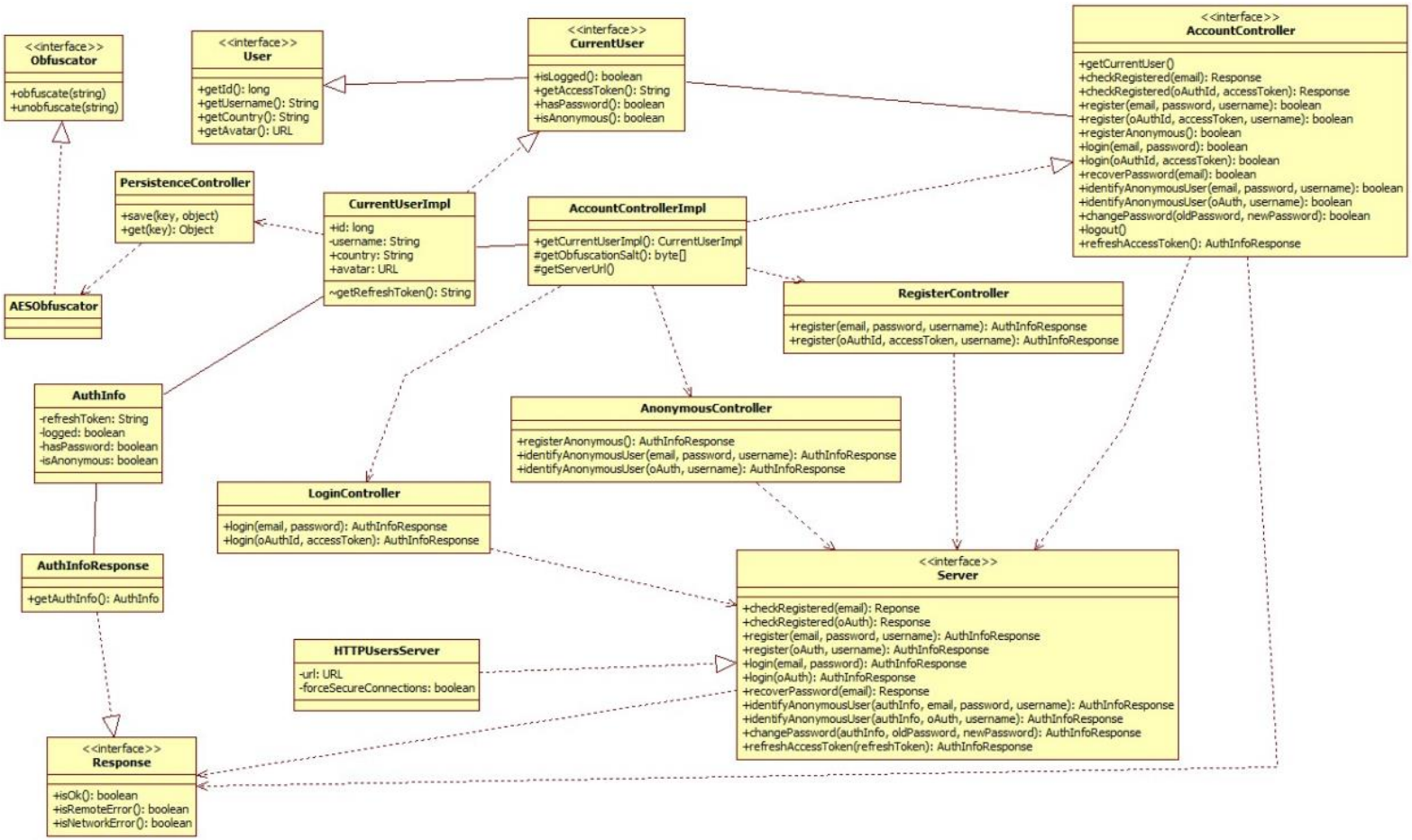


Ilustración 11: Diagrama de clases de análisis del sistema de usuarios

Descripción de las clases:

- **AccountController**: Interfaz principal del sistema con la que interaccionaran el resto de subsistemas.
- **AccountControllerImpl:** Clase que implementa AccountControler, delegando las diferentes funcionalidades en el LoginController, RegisterController, AnonymousController, etc.
- **LoginController:** Controlador encargado de las funciones de login en sus diferentes versiones.
- **RegisterController:** Controlador encargado de las funciones de registro en sus diferentes versiones.
- **AnonymousController:** Controlador encargado de las funciones relativas a usuarios anónimos, como registrarlos o identificarlos.
- **CurrentUser:** Interfaz que representa el usuario local aportando información como si está logueado o si es un usuario anónimo.
- **CurrentUserImpl:** Clase ActiveRecord encargada de implementar la lógica de CurrentUser y almacenar sus datos.
- **Response:** Representación de una respuesta a una petición, permite saber si se realizó con éxito la petición o si se produjo algún error.
- **AuthInfoResponse:** Response que incorpora la información sobre la autenticación de un usuario.
- **PersistenceController:** Clase encargada del almacenamiento persistente de datos, como el refreshToken o el accessToken.
- **Obfuscator:** Interfaz para las clases que ofrezcan ofuscación de datos, ofuscando los datos sensibles conseguimos que sean más difícilmente accesibles por fuentes no autorizadas.
- **AESObfuscator:** Clase que proporciona mecanismos de ofuscación mediante el esquema de cifrado AES.

Vistas:

Puesto que las vistas encargadas del registro, autenticación y gestión de usuarios son una labor inevitable en cualquier aplicación junto con la librería se incluye una aplicación de ejemplo con las vistas implementadas en clases Activity de Android para poder modificarlas con facilidad.

Este es el diagrama de clases planteado para la aplicación de autenticación de ejemplo:

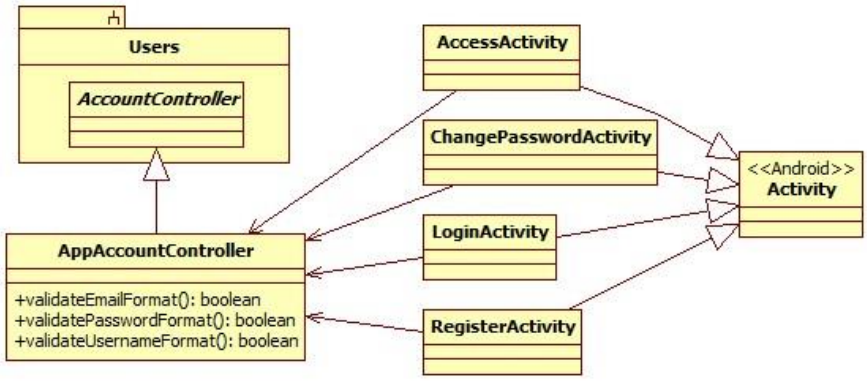


Ilustración 12: Diagrama de clases Activity para el módulo de usuarios

b) Proceso para las peticiones autenticadas

Cuando un usuario se registra o accede al sistema recibe dos valores que posteriormente utilizará para gestionar la autenticación:

- Access Token: Token que identifica únicamente a un usuario en el sistema. Tiene una validez limitada en el tiempo. Este token es el que se pasará al resto de subsistemas y con el podrán recuperar la información del usuario asociado.
- Refresh Token: Token utilizado a nivel interno por el sistema de usuarios, no es accesible externamente. Es utilizado para refrescar el access token cuando algún tipo expira o deja de ser válido por algún motivo. Acciones como generar o cambiar una contraseña invalidan el refresh token anterior.

c) Diagramas de secuencia

Debido a la sencillez y a que casi todos los métodos de este subsistema tienen un funcionamiento similar: realizar una llamada a un servicio web y actualizar la información de autenticación local. Únicamente se representarán dos diagramas de secuencia el de registro y el de cambio de contraseña, este último servirá para comprender el funcionamiento del refresh_token y el access_token.

Secuencia de Registro:

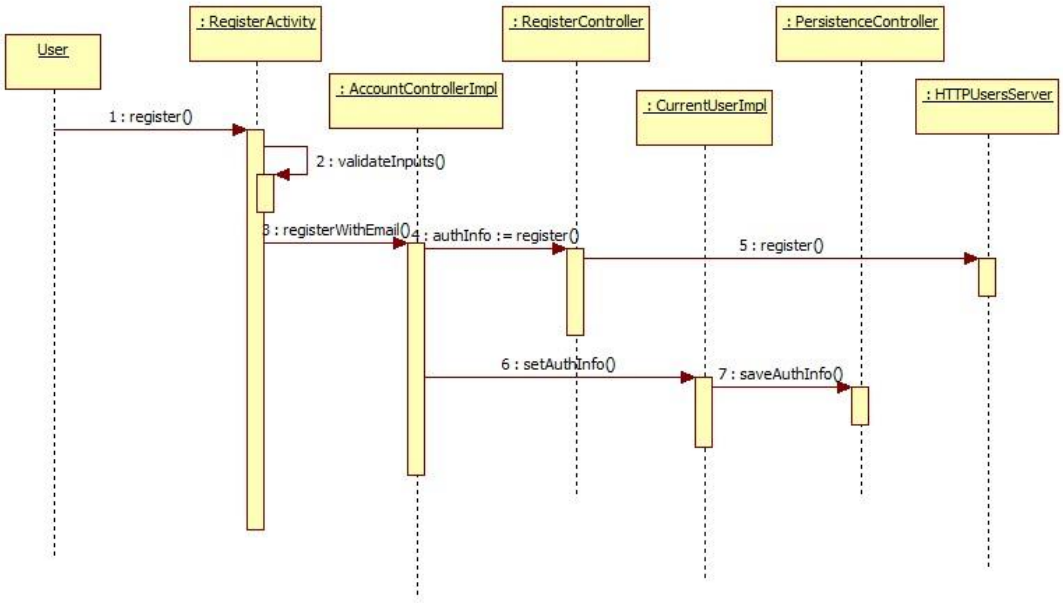


Ilustración 13: Diagrama de secuencia: Registro

Como se ha comentado anteriormente los diagramas de secuencia los servicios de identificar usuario anónimo, login se han omitido ya que a nivel de cliente son muy similares (exceptuando los parámetros y la dirección del servicio).

Secuencia de cambio de contraseña:

El cambio de contraseña es una llamada que requiere que el usuario esté autenticado, por ello requiere el envío del access_token así como la gestión de las respuestas asociadas a peticiones con autenticación, como por ejemplo que el access_token haya expirado.

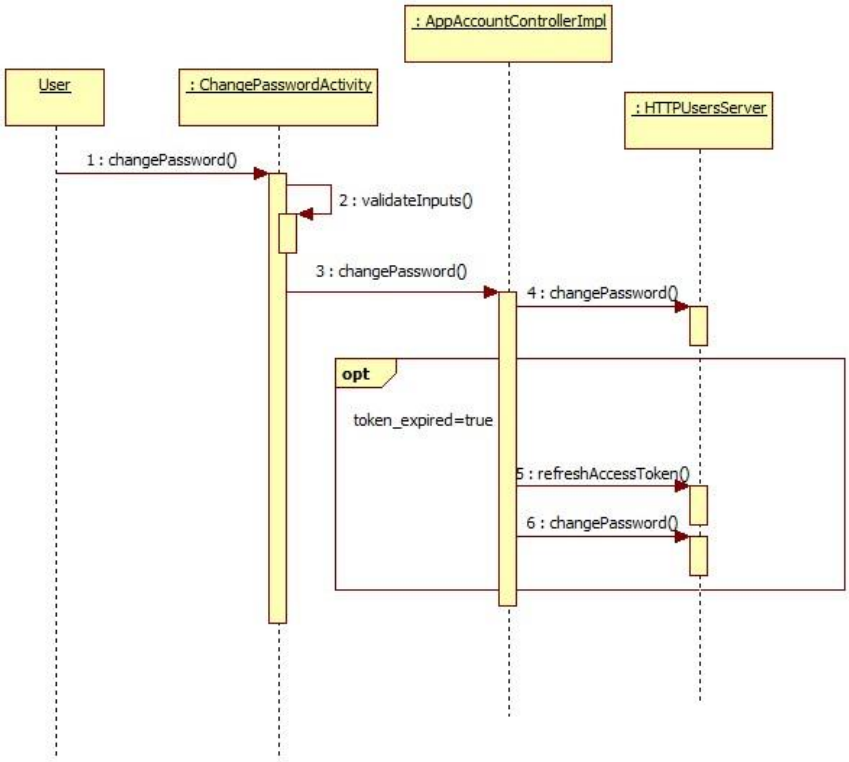


Ilustración 14: Diagrama de secuencia: Cambio de contraseña

4. Implementación

El proceso de autenticación para los usuarios puede verse en el siguiente diagrama de flujo. Nótese que en él no se ha tenido el acceso e identificación de usuarios anónimos.

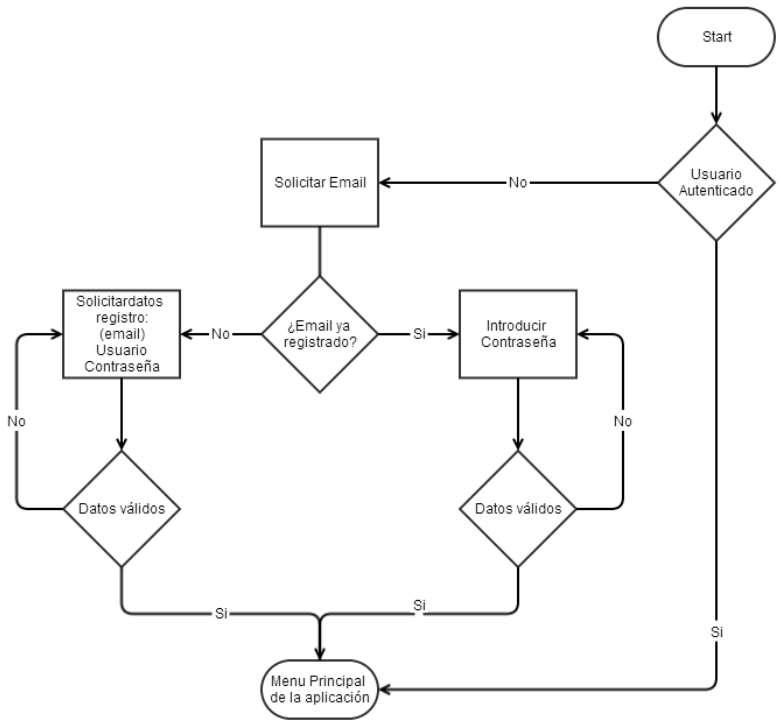


Ilustración 15: Proceso de autenticación

Con el fin de facilitar el proceso minimizando la información que se requiere en cada pantalla para autenticar al usuario inicialmente se accede a una pantalla: AccessActivity donde únicamente se solicita el email o se da la opción de acceder como usuario anónimo. Una vez introducido la aplicación comprueba si el email ya está registrado en el sistema, si no lo está se presenta la pantalla de registro (RegisterActivity) con la información que se requiere para todo usuario registrado: nombre de usuario, email y contraseña. Por el contrario si el email ya está registrado únicamente se solicita la contraseña (LoginActivity) y también se ofrece la opción de recuperar la contraseña si el usuario la ha olvidado.

RedirectActivity se encarga de redirigir al usuario a la pantalla de acceso o a la pantalla principal de la aplicación en función de si el usuario está autenticado o no, tanto en el arranque inicial de la aplicación como durante el proceso de login o registro.

En la pantalla del menú principal se puede cambiar la contraseña accediendo a la pantalla ChangePasswordActivity, donde se solicita la contraseña actual y la nueva.

Finalmente si el usuario autenticado es anónimo la pantalla principal también ofrece la posibilidad de identificarse, aportando los mismos datos que en el registro: nombre de usuario, email y contraseña.

La imagen a continuación muestra el modelo de navegación por las pantallas implicadas en la autenticación de usuarios.

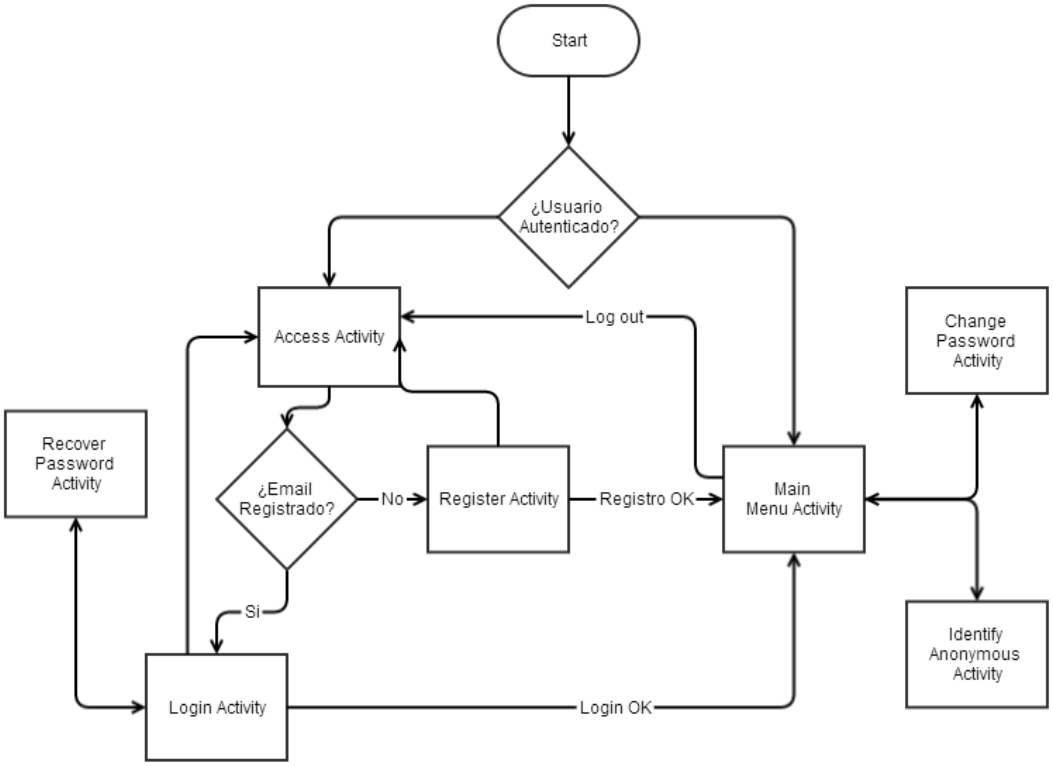
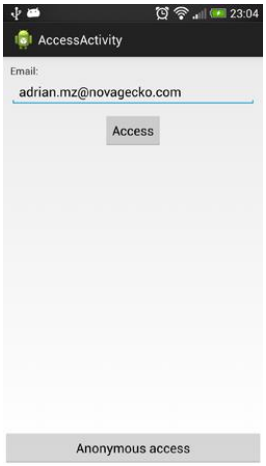


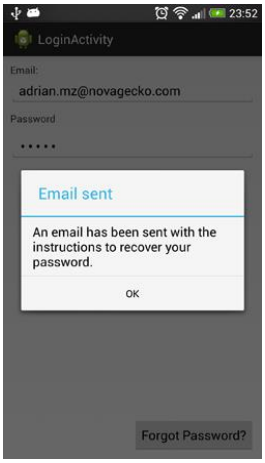
Ilustración 16: Diagrama de navegación autenticación de usuarios

Pantallas del sistema de autenticación de usuarios:



Access Activity

Login Activity

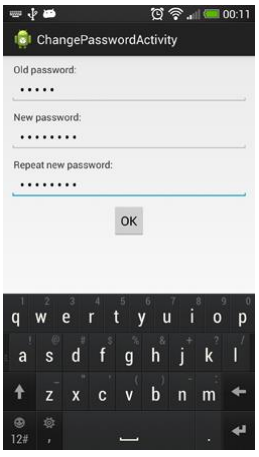


Recover Password

Register Activity

Main Activity (Anonónimo)

Main Activity (Registrado)



Change Password Activity

Idenfity Anonymous Activity

5. Pruebas

A continuación se listan las pruebas que se han realizado sobre el sistema de autenticación de usuarios:

Prueba	Resultado
Acceso como usuario anónimo: ✓ Identificador de usuario válido ✓ Nombre de usuario nulo ✓ Bandera isAnonymous activada ✓ Bandera hasPassword desactivada ✓ Access_token válido	OK
Identificación de un usuario anónimo: ✓ Falla sin email ✓ Falla con email con formato incorrecto ✓ Falla sin contraseña ✓ Falla con contraseña muy corta ✓ Falla sin usuario ✓ Falla con usuario con formato muy corto ✓ Identifica con entradas correctas ✓ Falla con email existente ✓ Falla con usuario existente ✗ Banderas isAnonymous y hasPassword actualizadas tras identificación ✓ Banderas isAnonymous y hasPassword actualizadas: Corregido	1 Fallo Solucionado
Registro de usuario: ✓ Falla sin email ✓ Falla con email con formato incorrecto ✓ Falla sin contraseña ✓ Falla con contraseña muy corta ✓ Falla sin usuario ✓ Falla con usuario con formato muy corto ✓ Registra con entradas correctas ✓ Falla con email existente ✓ Falla con usuario existente ✓ Cuentra creada y accesible tras el registro ✓ Nueva cuenta creada con valores correctos: o Id de usuario o Nombre de usuario o Contraseña o Access_token o Bandera isAnonymous=false o Bandera hasPassword=true	OK
Login de usuario: ✓ Falla con email / contraseña inválidos ✓ Accede con email / contraseña válidos ✓ Recupera datos de cuenta correctamente	OK

Acceso aplicación: ✓ Detecta correctamente si el usuario está autenticado o no y lo redirige a la pantalla adecuada ✓ Redirige a la pantalla principal tras login. ✓ Redirige a la pantalla principal tras registro. ✓ Redirige a la pantalla de acceso tras logout.	OK
Cambio de contraseña ✓ Requiere contraseña anterior ✓ Falla con contraseña anterior inválida ✓ Valida “repetir contraseña” ✓ Falla con contraseña nueva inválida (demasiado corta) ✓ Falla si contraseña nueva= contraseña anterior ✓ Cambia la contraseña para entradas correctas ✓ Accede con la nueva contraseña	OK
Recuperación de contraseña ✓ Envía email de recuperación.	OK

C. Estadísticas y logros

1. Requisitos

SAR 1. El sistema permitirá el uso del sistema de estadísticas a cualquier usuario, aunque no esté autenticado en el sistema.

SAR 2. El sistema solo permitirá el envío y sincronización a aquellos usuarios autenticados en el sistema (registrados o anónimos)

SAR 3. El sistema deberá contemplar estadísticas de usuario con los siguientes atributos:

- Identificador numérico único.
- Tipo de valor: numérico, porcentual o booleano.
- Valor actual
- Valor máximo
- Valor mínimo
- Título
- Descripción (Opcional)
- Icono (Opcional)
- Visibilidad: visible o no visible para el usuario
- Puntos asociados al valor actual.

SAR 4. El valor en puntos de una estadística se calcula multiplicando sus puntos por su valor. En el caso de estadísticas booleanas su valor será 0 si son false y en el caso de estadísticas porcentuales el valor en puntos será proporcional al porcentaje de la estadística

SAR 5. El sistema permitirá a un usuario consultar sus estadísticas.

SAR 6. El sistema permitirá a los usuarios consultar las estadísticas de otros usuarios.

SAR 7. La actualización de estadísticas desde el cliente debe realizarse de tal modo que no se pierdan datos si se envían modificaciones desde varios clientes asociados al mismo usuario.

SAR 8. La configuración del sistema de estadísticas podrá realizarse por código o leyendo un texto en formato JSON.

SAR 9. El valor de una estadística del sistema se actualizará de tres modos:

- Actualizada desde servidor: El servidor dispone del valor (o la información necesaria para derivarlo. El cliente no puede modificar el valor remoto de la estadística.
- Actualizada mediante deltas: El cliente comunica al servidor valores diferenciales que se acumulan con el valor actual remoto.
- Actualizada mediante valores absolutos: El cliente comunica al servidor el nuevo valor absoluto de la estadística, remplazando el valor existente.

SAR 10. El cliente almacenará de forma persistente la información necesaria para informar al servidor de los cambios en las mismas.

SAR 11. El sistema contemplará logros de usuario con los siguientes atributos:
- Identificador numérico único.
- Título
- Descripción completa (Opcional)
- Descripción abreviada (Opcional)
- Icono (Opcional)
- Visibilidad
- Puntos asociados

SAR 12. El sistema deberá permitir la consulta del estado actual de los logros, tanto propios como de otros usuarios.

SAR 13. Cada logro tendrá asociada uno o más pares estadística-valor. El logro se considerará completado cuando cada estadística alcance o supere el valor indicado.

SAR 14. El sistema permitirá conocer el porcentaje de completitud obteniendo el porcentaje completado de cada par estadística-valor asociada y calculando su media aritmética

SAR 15. El sistema deberá contemplar logros que no se puedan perder una vez conseguidos y logros que se pierden si se dejan de dar las condiciones necesarias.

SAR 16. Cada logro podrá tener un logro asociado como "anterior"

SAR 17. Los valores de visibilidad de un logro serán:
- Siempre visible
- Invisible hasta que el logro anterior se complete
- Invisible hasta que se complete.

SAR 18. El sistema ofrecerá un servicio de sincronización de estadísticas entre cliente y servidor.

SAR 19. El sistema permitirá la modificación de todas las estadísticas, incluyendo las "actualizadas desde el servidor" a nivel local por parte del cliente.

SAR 20. Al recibir las estadísticas del servidor el cliente sobrescribirá aquellas que sean "actualizadas desde el servidor".

SAR 21. Tras una modificación de una estadística, el cliente detectará si se ha completado algún logro asociado a la misma y, en caso de ser así, lo notificará.

SAR 22. El sistema se integrará con el módulo de usuarios y usará sus tokens de acceso para la comunicación autenticada entre cliente y servidor.

SAR 23. El sistema deberá borrar del cliente toda la información relativa a las estadísticas y logros cuando este cierre su sesión.

2. Análisis

a) Casos de Uso

El diagrama a continuación muestra los casos de uso para este subsistema de estadísticas. El conjunto de casos de uso no es muy grande y comprende: Actualizar una estadística,

consultar el estado de las estadísticas del sistema, consultar el estado de los logros del sistema y sincronizar estadísticas y logros con el servidor.

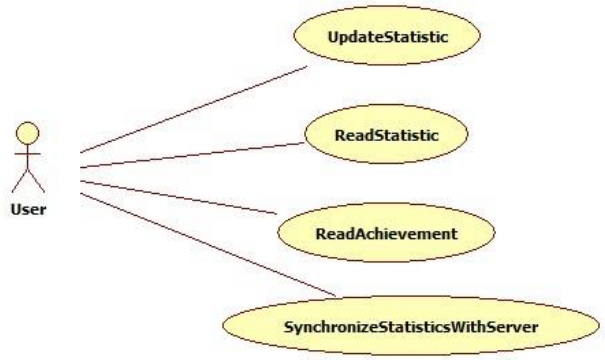


Ilustración 17: Diagrama de casos de uso de Estadísticas y logros

UC- SL-01	Updade Statistic	
Descripción	El sistema deberá comportarse tal y como se describe en el siguiente caso de uso cuando se solicite la modificación de un valor de una estadística	
Precondición	El usuario está autenticado en el sistema de usuarios	
Secuencia normal	**Paso**	**Acción**
	1	El usuario realiza alguna acción que requiera la modificación de una estadística
	2	El sistema de recibe la solicitud de modificar una estadística a un nuevo valor
	3	El sistema comprueba que el nuevo valor está entre los valores aceptables para la estadística
	4	El sistema actualiza y guarda persistentemente el valor de la estadística
	5	Si la estadística es del tipo "modificada desde el cliente" el sistema guarda la información como pendiente de envío al servidor
	6	El sistema comprueba los logros asociados a la estadística
	7	Si algún logro se ha completado a consecuencia de la modificación el sistema lo notifica
Excepciones	**Paso**	**Acción**
	3	Si el valor solicitado no está dentro de los valores mínimo y máximo el caso de uso termina y queda sin efecto
Postcondición	El valor de la estadística está actualizado al nuevo valor, guardado persistentemente y se pendiente de envío para la próxima sincronización con el servidor	
Comentarios	Ninguno	

UC- SL-02	Read Statistic	
Descripción	El sistema deberá comportarse tal y como se describe en el siguiente caso de uso cuando el usuario consulte el valor de una o más estadísticas	
Precondición	El usuario está autenticado en el sistema de usuarios	
Secuencia normal	**Paso**	**Acción**
	1	El usuario solicita consultar el estado de una o varias estadísticas
	2	El sistema recupera la información sobre las estadísticas solicitadas
	3	Si existe información pendiente de envío al servidor el sistema recuperará el valor pendiente para el servidor.
	4	El sistema muestra la información de las estadísticas solicitadas.
Postcondición	El usuario ha consultado el estado de las estadísticas	

UC- SL-02	**Read Achievement**	
Descripción	El sistema deberá comportarse tal y como se describe en el siguiente caso de uso cuando el usuario consulte el valor de uno o más logros	
Precondición	El usuario está autenticado en el sistema de usuarios	
Secuencia normal	**Paso**	**Acción**
	1	El usuario solicita consultar el estado de uno o varios logros
	2	El sistema recupera la información sobre los logros seleccionados
	3	El sistema recupera las estadísticas asociadas a los logros
	4	Si existe información pendiente de envío al servidor el sistema recuperará el valor pendiente para el servidor.
	5	El sistema calcula el estado de los logros a partir de sus estadísticas relacionadas
	6	El sistema muestra el estado de los logros solicitados al usuario
Postcondición	El usuario ha consultado el estado de las logros	
Comentarios	Ninguno	

UC- SL-03	Synchronize with server	
Descripción	El sistema deberá comportarse tal y como se describe en el siguiente caso de uso cuando se solicite la sincronización de estadísticas con el servidor	
Precondición	El usuario está autenticado en el sistema de usuarios	
Secuencia normal	**Paso**	**Acción**
	1	El usuario u otro subsistema solicita la sincronización de estadísticas con el servidor
	2	El sistema recupera la información pendiente de envío para el servidor
	3	El sistema envía los datos al servidor
	4	El servidor devuelve el estado actualizado de las estadísticas del usuario.
	5	El sistema actualiza las estadísticas locales con los nuevos valores recibidos.
	6	El sistema marca la información pendiente de envío como enviada con éxito
Excepciones	**Paso**	**Acción**
	2	Si no hay información pendiente, no se envía ningún dato al servidor, únicamente se recupera la información del mismo.
	3	Si se produce algún error en la comunicación con el servidor se notifica del error y este caso de uso termina sin efecto.
Postcondición	Las estadísticas del usuario están sincronizadas entre cliente y servidor	

3. Diseño

a) Diagrama de clases

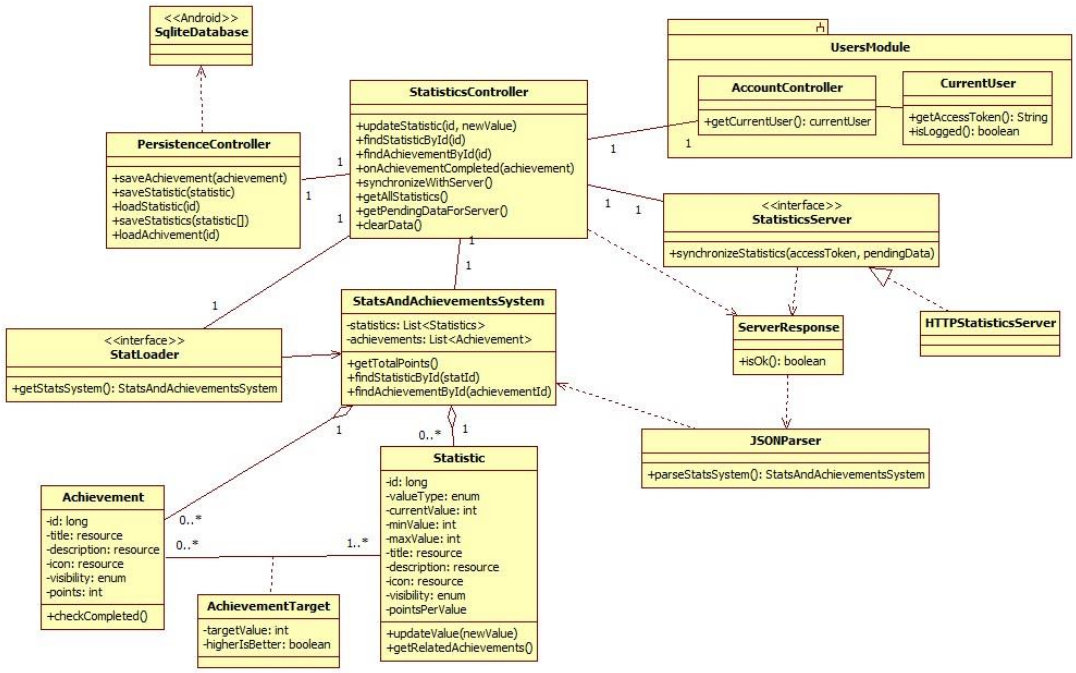


Ilustración 18: Diagrama de clases de Estadísticas y Logros

Descripción de las clases:

- **Statistic:** Representa una estadística de usuario, puede tener asociado uno o más logros.
- **Achievement:** Representa un logro en el sistema, tiene que tener asociado al menos una estadística junto con el valor necesario de la estadística para completar el logro.
- **AchievementTarget:** Clase asociación que representa el valor que requiere, en un logro, una estadística para considerarse completado. También tiene una bandera higherIsBetter que permite conocer si el valor debe ser superior (higherIsBetter=true) o ser inferior (higherIsBetter=false)
- **StatisticsController:** Clase controlador que ofrece la funcionalidad de este módulo al resto de subsistemas de la aplicación. Dispone de métodos de consulta para estadísticas y logros tanto de forma individual como colectiva, ofrece el método updateStatistic para actualizar el valor de una estadística y finalmente un método para borrar los datos locales.

- **PersistenceController:** Clase encargada del almacenamiento persistente de las estadísticas y los logros. Proporciona métodos para cargar y guardar ambos tipos de objetos en persistencia.
- **StatiticsServer:** Representación del servidor de estadísticas, ofrece un único método que recibe la información pendiente desde cliente (puede ser nula) y devuelve las estadísticas del usuario
- **StatsLoader:** Interfaz utilizada para inyectar las dependencias de la creación e inicialización del sistema de estadísticas. Cada aplicación aportará su implementación de esta interfaz.
- **ServerResponse:** Esta clase representa una respuesta desde servidor.
- **JSONParser:** Puesto que la comunicación entre cliente y servidor se realiza en formato JSON esta clase se encarga de parsear y transformar las respuestas del servidor a objetos de la lógica de la aplicación.

b) Diagramas de secuencia

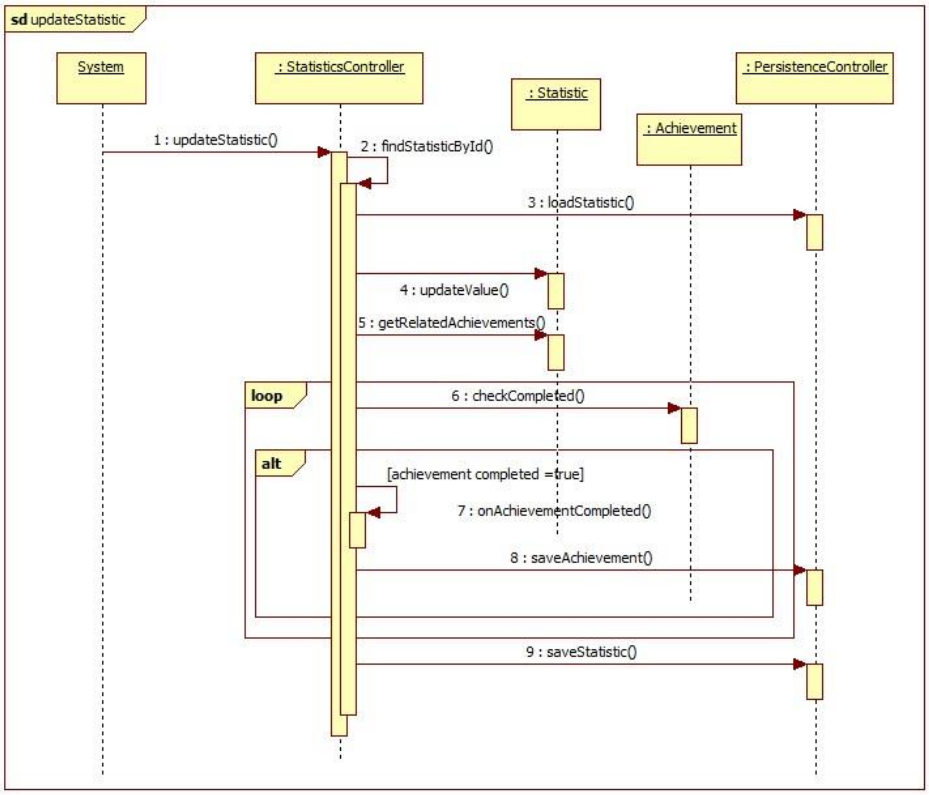


Ilustración 19: Diagrama de secuencia: UpdateStatistic

Synchronize Statistics: En este diagrama se puede ver el proceso de sincronización de estadísticas con el servidor.

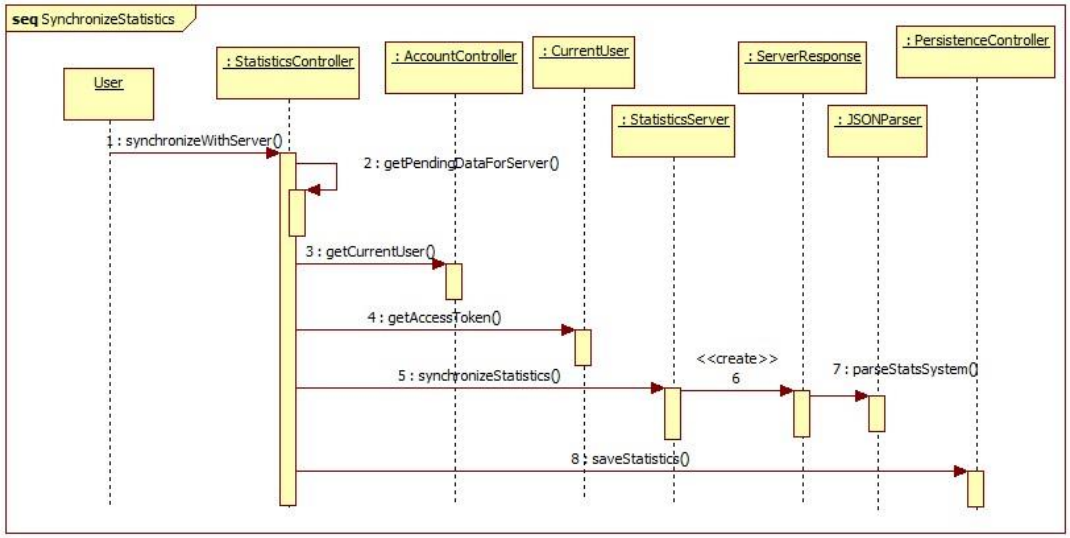


Ilustración 20: Diagrama de secuencia de Sychronize Statistics

4. Implementación

Configuración inicial

A continuación se describe el proceso de implementación de un sistema de estadísticas en una Aplicación Android. Para ello debemos sobrescribir el método **getStatsLoader**() de nuestro **StatsController**.

En nuestro ejemplo devuelve una instancia de **HardcodedStatLoader** que nos permite definir las estadísticas y logros que forman parte de nuestro sistema. En el código a continuación se definen 6 estadísticas, con id's del 1 al 6. Las propiedades de cada estadística se establecen al ser creadas por el objeto StatsGenerator. Puede consultarse el código adjunto con el proyecto que incorpora un ejemplo aislado del uso del sistema de estadísticas.

Posteriormente se crean 3 logros, asociando todos ellos a la estadística con identificador=1, estableciendo los valores requeridos para completar cada logro en 600,2600 y 3600 respectivamente. Por supuesto la complejidad de esta configuración dependerá de la cantidad de estadísticas y logros en el sistema pero una vez realizado nuestro cliente estará listo para manejar las estadísticas definidas de forma local y sincronizar los valores con el servidor cuando se indique mediante el método **updateStatsFromServer().**

```
@Override
protected void initSystem(StatsAndAchievementsSystem statsSystem) {
          StatsGenerator generator = new StatsGenerator(getFactory());
          statsSystem.addStat(generator.createStat(1));
          statsSystem.addStat(generator.createStat(2));
          statsSystem.addStat(generator.createStat(3));
          statsSystem.addStat(generator.createStat(4));
          statsSystem.addStat(generator.createStat(5));
          statsSystem.addStat(generator.createStat(6));

          Achievement achievement1 = generator.createAchievement(1,
StatsControllerImpl.this);
          achievement1.addRelatedStatistic(statsSystem.getStat(1), 600);
          statsSystem.addAchievement(achievement1);

          Achievement achievement2 = generator.createAchievement(2,
StatsControllerImpl.this);
          achievement2.addRelatedStatistic(statsSystem.getStat(1), 2600);
          statsSystem.addAchievement(achievement2);

          Achievement achievement3 = generator.createAchievement(3,
StatsControllerImpl.this);
          achievement3.addRelatedStatistic(statsSystem.getStat(1), 3600);
          statsSystem.addAchievement(achievement3);
          statsSystem.getStat(1).addRelatedAchievement(achievement1);
          statsSystem.getStat(1).addRelatedAchievement(achievement2);
          statsSystem.getStat(1).addRelatedAchievement(achievement3);
}
};
}
```

Para recibir los eventos de logro completado debemos sobrescribir el método onAchievementCompleted(Achievement achievement) de nuestro controlador de estadísticas. Una respuesta frecuente en las aplicaciones es mostrar un mensaje flotante sobre la pantalla (Toast en el caso de Android) notificando el evento.

5. Pruebas

A continuación se listan las pruebas que se han realizado sobre el sistema de estadísticas y logros:

Prueba	Resultado
Inicialización de estadísticas: ✓ Carga de sistema de estadísticas por código ✓ Carga de sistema de estadísticas por JSON ✓ Lanza excepción si no se define el sistema de estadísticas ✓ Valores por defecto de las estadísticas correctos	OK
Modificación de estadísticas: ✓ Modificación de estadísticas numéricas correcto ✓ Modificación de estadísticas porcentuales correcto	1 Fallo Solucionado

✓ Modificación de estadísticas booleanas correcto ✓ Guardado persistente tras modificación de una estadística correcto ✓ Ignora las modificaciones de estadísticas para valores fuera del rango [min,max] ✓ Comprobación de logros asociados a la estadística modificada ✓ No comprobación de logros no asociados a la estadística modificada ✓ Cálculo de porcentaje completado para logros asociados a estadísticas "higher is better" × Cálculo de porcentaje completado para logros asociados a estadísticas "lower is better": fallo, se estaba usando el mismo algoritmo. ✓ Cálculo de porcentaje completado para logros asociados a estadísticas "lower is better": Corregido	
Lectura de estadísticas y logros: ✓ Acceso a estadísticas individuales por identificador, correcto ✓ Acceso a logros individuales por identificador, correcto. ✓ Acceso a todas las estadísticas en bloque, correcto. ✓ Acceso a todos los logros en bloque, correcto. ✓ Estadísticas con información pendiente para el servidor muestran el valor pendiente como valor actual hasta que se envían. ✓ Lanza excepción al acceder a logros inexistentes ✓ Lanza excepción al acceder a estadísticas inexistentes ✓ Lanza excepción al acceder a un sistema de estadísticas sin inicializar.	OK
Sincronización de estadísticas ✓ Generación de valores pendientes para el envío al servidor. ✓ Envío únicamente de la información modificada. ✓ Recepción y parseo de los nuevos valores, correcto. ✓ Almacenamiento persistente de los nuevos valores. ✓ Conserva los datos pendientes si, mientras se está sincronizando, una estadística es modificada. × Funcionamiento adecuado sin acceso a la red. Fallo: provoca un cierre forzado ✓ Funcionamiento adecuado sin acceso a la red solucionado. Modificada la gestión de las excepciones de entrada salida en el método de sincronización con el servidor	1 Fallo solucionado

D. Notificaciones Push

1. Requisitos

PUR 1. El sistema se integrará con el servicio de envío y recepción de notificaciones push Google Cloud Messaging (GCM).

PUR 2. El sistema deberá gestionar de forma sencilla para los clientes el acceso a la red GCM).

PUR 3. El sistema se integrará con el sistema de usuarios desarrollado en este documento.

PUR 4. El sistema gestionará de forma transparente al cliente la recogida y envío al servidor de la aplicación del código de identificación del terminal en el servicio GCM para el envío de mensajes push.

PUR 5. El sistema registrará en el servidor de la aplicación los terminales en el servicio GCM de tres modos diferentes:

- Registro anónimo: Los terminales se registran sin estar asociados a ningún usuario del sistema. Estos usuarios únicamente podrán recibir notificaciones "broadcast".
- Registro autenticado: Los terminales se registran únicamente si el usuario está autenticado en el sistema (aunque el usuario sea anónimo). De esta forma se podrán dirigir notificaciones específicas a este usuario.
- Registro preferiblemente autenticado: Los terminales realizan un registro autenticado si el usuario se ha identificado en el sistema, en caso contrario realiza un registro anónimo.

PUR 6. En caso de no poder comunicar al servidor de la aplicación la ID de registro de un terminal, el cliente reintentará el envío de forma automática en el futuro.

PUR 7. El sistema permitirá el des-registro de los terminales en el sistema de notificaciones push, borrando su información tanto del servidor GCM como del servidor de la aplicación.

PUR 8. El sistema proporcionará algún mecanismo para conocer si un terminal soporta el envío de notificaciones push.

PUR 9. El sistema notificará al cliente de los mensajes recibidos a través de la red GCM.

PUR 10. El sistema almacenará cada terminal a un único usuario, remplazando si fuese necesario las entradas más antiguas.

PUR 11. Un usuario podrá tener asociado un número indeterminado de terminales.

PUR 12. Un terminal solo podrá tener asociado un usuario en cada momento.

2. Análisis

a) Casos de uso

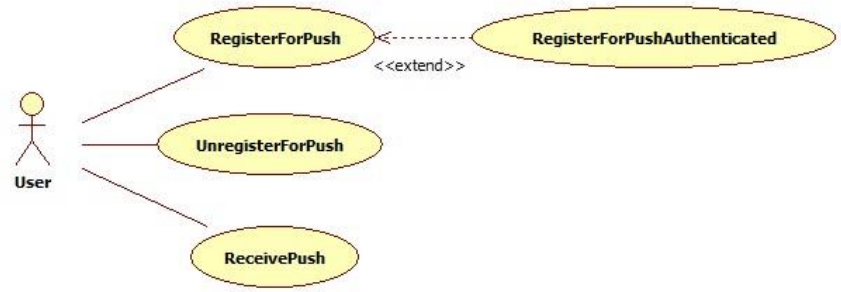


Ilustración 21: Casos de uso del sistema de notificaciones push

UC- PUSH-01	Register for push	
Descripción	El sistema deberá comportarse tal y como se describe en el siguiente caso de uso cuando se solicite el alta del terminal en el sistema de notificaciones push.	
Precondición	El terminal es compatible con el sistema de notificaciones GCM	
Secuencia normal	**Paso**	**Acción**
	1	El usuario requiere el registro de su terminal para la recepción de notificaciones push.
	2	El sistema contacta con el servicio Google Cloud Services y recupera un identificador para el envío de notificaciones.
	3	El sistema almacena persistentemente el indicador
	4	El sistema comunica al servidor de la aplicación el identificador para el envío de notificaciones.
Excepciones	**Paso**	**Acción**
	1a	Si el sistema ya tiene un identificador para notificaciones y ha sido comunicado al servidor el caso de uso termina con éxito
	1b	Si el sistema ya tiene un identificador para notificaciones pero no ha sido comunicado al servidor se omite su recuperación del sistema GCM
	2	Si el sistema no puede obtener un identificador el caso de uso termina quedando sin efecto
	4	Si el sistema consigue comunicar el identificador al servidor almacena como pendiente de envío el mismo y el caso de uso queda sin efecto.
Postcondición	El terminal está registrado tanto en los servidores de Google (GCM) como en los de la aplicación con su identificador para notificaciones.	
Comentarios	Ninguno	

UC- PUSH-02	Register for push authenticated	
Descripción	El sistema deberá comportarse tal y como se describe en el siguiente caso de uso cuando se solicite el alta del terminal en el sistema de notificaciones push asociándolo a su usuario en el sistema.	
Precondición	El terminal es compatible con el sistema de notificaciones GCM y el usuario está autenticado en el sistema	
Secuencia normal	**Paso**	**Acción**
		La secuencia normal y las excepciones es la misma que en el caso de uso **UC-PUSH-02 Register for push** a excepción de las pre y post condiciones y el paso siguiente:
	4’	El sistema comunica al servidor de la aplicación el identificador para el envío de notificaciones junto con la autenticación de usuario.
Postcondición	El terminal está registrado tanto en los servidores de Google (GCM). El terminal está registrado en el servidor de la aplicación y asociado al usuario autenticado	
Comentarios	Ninguno	

UC- PUSH-04	Unregister for push	
Descripción	El sistema deberá comportarse tal y como se describe en el siguiente caso de uso cuando se solicite la baja del terminal en el sistema de notificaciones push	
Precondición	El terminal está dado de alta en el sistema de notificaciones push	
Secuencia normal	**Paso**	**Acción**
	1	El usuario solicita la baja del sistema de notificaciones push
	2	El sistema comunica al servicio de Google GCM la baja.
	3	El sistema comunica al servidor de la aplicación la baja del identificador para notificaciones
	4	El sistema elimina la información local asociada al registro de notificaciones
Postcondición	El terminal ya no está registrado en el servicio de Google (GCM). El terminal ya no está registrado en el servidor de la aplicación para recibir notificaciones push. El terminal no recibirá (o ignorará) futuros mensajes hasta que vuelva a darse de alta en el sistema	
Comentarios	Ninguno	

UC- PUSH-03	Receive Push	
Descripción	El sistema deberá comportarse tal y como se describe en el siguiente caso de uso cuando el servidor de la aplicación solicite el envío de una notificación push a un terminal registrado en el sistema	
Precondición	El terminal está dado de alta en el sistema de notificaciones push	
Secuencia normal	**Paso**	**Acción**
	1	El servidor solicita al servicio GCM el envío de la notificación indicando su contenido y el identificador push de su receptor
	2	El sistema GCM entrega la notificación al terminal
	3	El sistema cliente es notificado del mensaje recibido
Excepciones	**Paso**	**Acción**
	2	Si el terminal no está disponible el caso de uso queda sin efecto
Postcondición	El cliente ha recibido la notificación	
Comentarios	Ninguno	

3. Diseño

a) Diagrama de clases

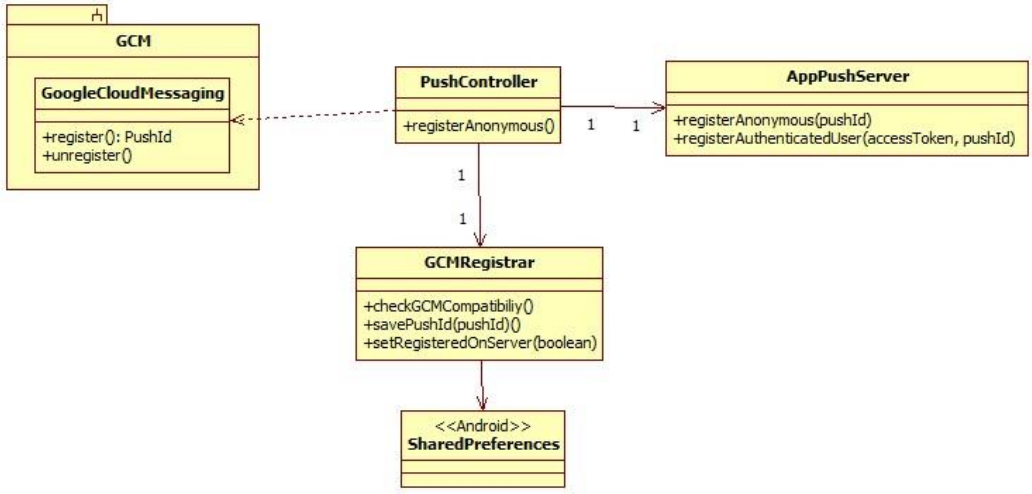


Ilustración 22: Diagrama de clases del sistema de notificaciones push

b) Diagramas de secuencia

Register for push:

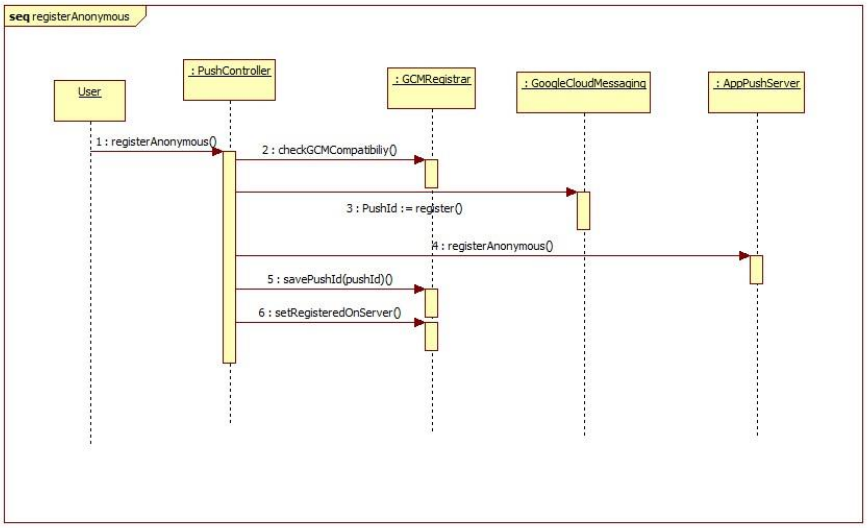


Ilustración 23: Diagrama de secuencia: Register for push

Register for push authenticated

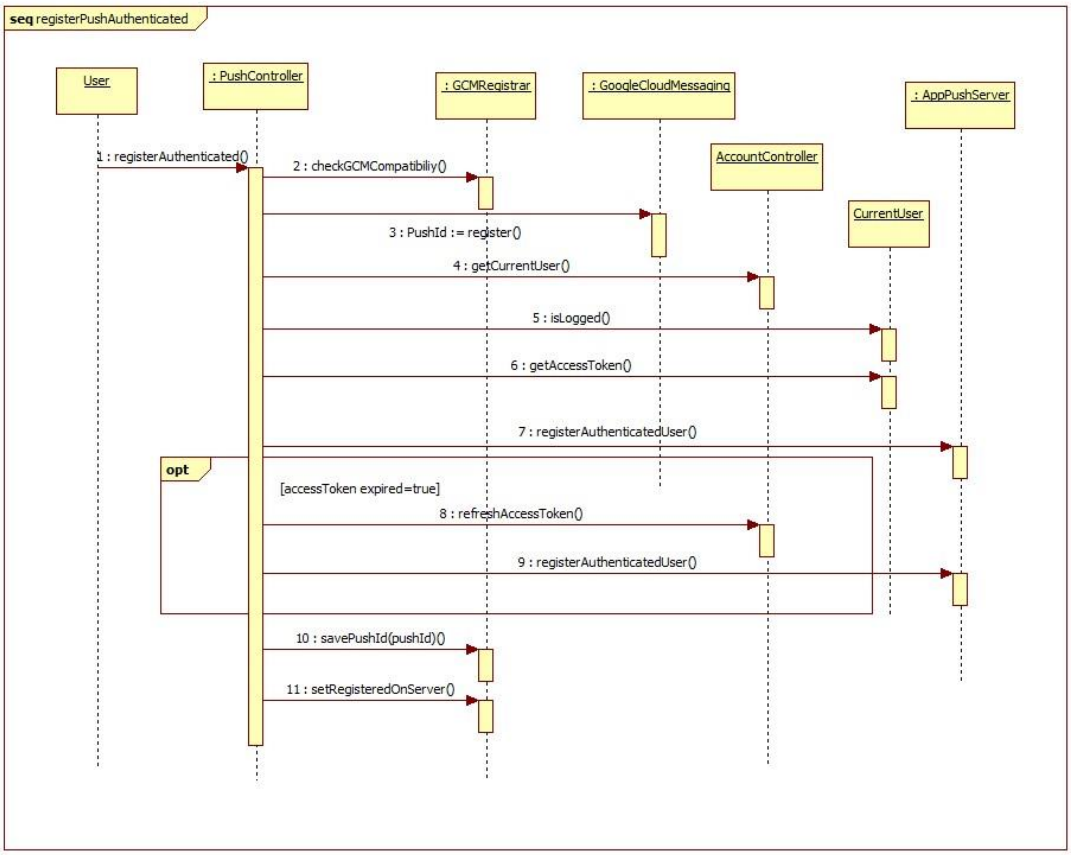


Ilustración 24: Diagrama de secuencia: Register for push authenticated

Unregister for push

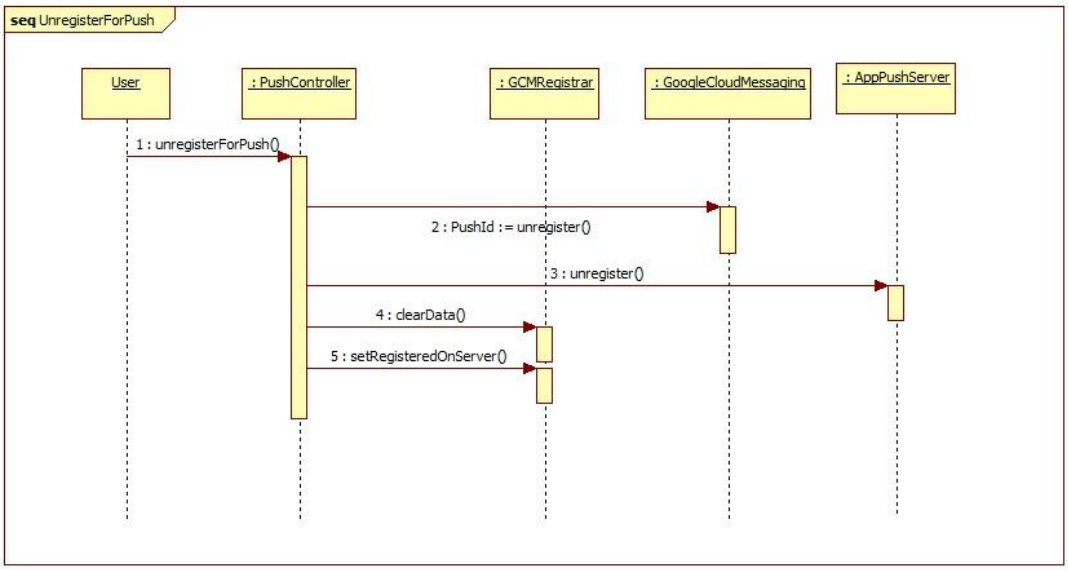


Ilustración 25: Diagrama de secuencia: Unregister for push

4. Implementación

La implementación de este módulo en una aplicación Android es bastante sencillo:

- Registro de la aplicación en Google Play Services: En primer lugar es necesario conseguir un "sender ID" que se obtiene al registrar el proyecto en la página web Google APIS (https://code.google.com/apis/console/).
- En segundo lugar se necesita sobre-escribir el método **getSenderIds**() del PushController para que devuelva el valor obtenido en el paso 1
- Para notificar la llegada de mensajes push basta con sobrescribir el método **onMessagereceived**() de la clase PushController.

5. Pruebas

Las pruebas de este subsistema han sido especialmente sencillas debido a la poca cantidad de servicios y posibles entradas/salidas en las pruebas.

Cabe destacar que durante las pruebas la mayoría de los mensajes se han recibido de instantáneamente, mientras que un pequeño porcentaje (inferior al 5%) se han recibido con unos pocos segundos, llegando incluso a algún minuto de retraso.

Prueba	Resultado
Registro anónimo para push: ✓ Obtención del pushId de los servidores GCM correcta ✓ Comunicación del pushId a al servidor de la aplicación correcta ✓ Almacenamiento persistente del pushId satisfactorio ✓ Reintento de envío del pushId a servidor de la aplicación tras fallo de conexión correcto.	OK
Registro autenticado para push ✓ Obtención del pushId de los servidores GCM correcta ✓ Comunicación del pushId a al servidor de la aplicación junto con la autenticación en el sistema de usuarios (accessToken) correcta ✓ Almacenamiento persistente del pushId satisfactorio ✓ Reintento de envío del pushId a servidor de la aplicación tras fallo de conexión correcto.	OK
Des-registro para push ✓ Baja de los servidores GCM con éxito ✓ Baja del servidor de la aplicación exitosa ✓ Borrado de los datos locales tras des-registro satisfactorio.	OK
Envío de notificaciones push ✓ Recepción de notificaciones para dispositivos conectados a Internet correcto y rápido ✓ Recepción tardía para dispositivos desconectados de la red satisfactoria	OK

E. Almacenamiento en la Nube

1. Requisitos

CSR 1. El sistema proporcionará un sistema de almacenamiento compartido.

CSR 2. El sistema dividirá el almacenamiento en slots, la unidad mínima de información que puede leerse o escribirse.

CSR 3. Un slot contará con los siguientes atributos

- Tipo
- Identificador
- Versión
- Id propietario
- Contenido

CSR 4. El sistema incrementará la versión de un slot cada vez que se escriba.

CSR 5. El sistema se integrará con el sistema de usuario para realizar la comunicación con el servidor que requiera autenticación.

CSR 6. Un slot se identificará de forma única para cada usuario por su tipo e identificador.

CSR 7. El contenido de un slot tendrá formato binario.

CSR 8. El sistema permitirá a un usuario acceder (leer y escribir) a sus slots.

CSR 9. El sistema no permitirá el acceso de ningún tipo a los slots de otros usuarios.

CSR 10. El sistema permitirá listar los slots con los que cuenta el usuario.

CSR 11. El sistema aportará las siguientes operaciones:

- Listar los slots disponibles en el servidor.
- Leer el contenido de un slot (de forma local o remota)
- Escribir el contenido de un slot (de forma local o remota)
- Sincronizar un slot, combinando el contenido local con el remoto.
- Borrar los datos locales en el cliente.

CSR 12. El sistema contemplará un sistema de resolución de conflictos para solucionar colisiones entre versiones modificadas de forma concurrente en varios clientes.

CSR 13. El sistema combinará y resolverá los conflictos de forma automática y transparente para el cliente las versiones locales y remotas

CSR 14. El sistema combinará, resolverá los conflictos y gestionará el reenvío de aquellos slots que hayan sido rechazados por el servidor por estar desfasados con la versión actual.

CSR 15. El sistema detectará e ignorará si los datos pendientes de envío son revertidos (es decir se vuelve a escribir en el slot lo que había en última versión recuperada de servidor)

2. Análisis

a) Casos de uso

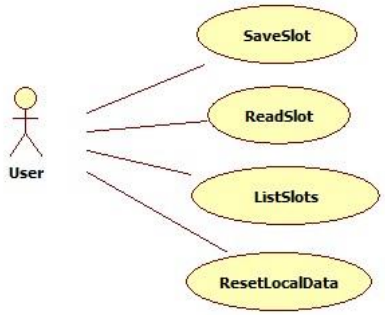


Ilustración 26: Casos de uso del sistema de almacenamiento en la nube

UC- CS-01	Read Slot	
Descripción	El sistema deberá comportarse tal y como se describe en el siguiente caso de uso cuando el usuario solicite el acceso a un slot	
Precondición	El usuario está autenticado en el sistema de usuarios	
Secuencia normal	**Paso**	**Acción**
	1	El usuario solicita el acceso a un slot especificando su autenticación de usuario, el tipo e identificador del slot y si se permite el acceso a la red para resolver la petición.
	2	Si se permite el acceso a la red el sistema solicita al servidor la versión más reciente del slot.
	3	Si la versión recuperada es diferente a la versión local en el cliente el sistema resuelve los posibles conflictos que pudiesen darse y guarda el slot actualizado
	4	Si no se permite el acceso a la red o este no está disponible, el sistema recupera la versión local del slot más actualizada existente.
	5	El sistema devuelve el contenido del slot
Excepciones	**Paso**	**Acción**
	3	Si no existe ninguna versión local la versión entrante del servidor únicamente se guarda.
Postcondición	Se ha recuperado la versión más actualizada del contenido del slot, del servidor si es posible o de la información local en caso contrario.	
Comentarios	Ninguno	

UC- CS-02	Save Slot	
Descripción	El sistema deberá comportarse tal y como se describe en el siguiente caso de uso cuando el usuario solicite la escritura de un slot	
Precondición	El usuario está autenticado en el sistema de usuarios	
Secuencia normal	**Paso**	**Acción**
	1	El usuario solicita el acceso a un slot especificando su autenticación de usuario, el tipo, identificador y contenido del slot y si se permite el acceso a la red para resolver la petición.
	2	Si se permite el acceso a la red el sistema solicita al servidor la escritura del slot.
	3	El servidor verifica que el cliente está intentando escribir desde la última versión del servidor
	4	El servidor actualiza el slot remoto
	5	El sistema confirma la operación al cliente.
Excepciones	**Paso**	**Acción**
	2	Si la red no se pudo completar la operación de red o no se permite el acceso a la misma, el sistema guarda el slot en local y lo marca como pendiente de envío al servidor y el caso de uso termina sin efecto
	3	Si el cliente no se encuentra actualizado el servidor notifica del error y el caso de uso queda sin efecto
Postcondición	El slot ha sido guardado en el servidor o en local.	
Comentarios	Ninguno	

UC- CS-03	List Slots	
Descripción	El sistema deberá comportarse tal y como se describe en el siguiente caso de uso cuando el usuario solicite el listado de los slots en el servidor	
Precondición	El usuario está autenticado en el sistema de usuarios	
Secuencia normal	**Paso**	**Acción**
	1	El usuario solicita el listado de sus slots al servidor, especificando su autenticación de usuario.
	3	El servidor recupera y devuelve la lista de los slots que posee el usuario, incluyendo para cada uno su tipo e identificador.
Postcondición	El cliente ha recuperado la lista de los slots en el servidor	
Comentarios	Ninguno	

UC- CS-04	Reset Local Data	
Descripción	El sistema deberá comportarse tal y como se describe en el siguiente caso de uso cuando se solicite el borrado de la información local, por ejemplo porque el usuario cierra su sesión.	
Precondición	Ninguna	
Secuencia normal	**Paso**	**Acción**
	1	El usuario u otro subsistema solicitan el borrado de la información local relativa al sistema de almacenamiento. También se indica si se quiere forzar el borrado, es decir borrar la información aunque haya datos pendientes que no se puedan enviar al servidor
	2	Si hay información pendiente el sistema lo comunica al servidor
	3	El sistema borra toda la información local del usuario
Excepciones	**Paso**	**Acción**
	2	Si no se consigue comunicar la información pendiente al servidor el sistema notifica del error y si se seleccionó la opción de borrado forzoso el caso de uso continua, en caso contrario el caso de uso finaliza sin efecto.
Postcondición	Se ha borrado toda la información local de los slots y versiones del usuario	
Comentarios	Ninguno	

3. Diseño

a) Diagrama de clases

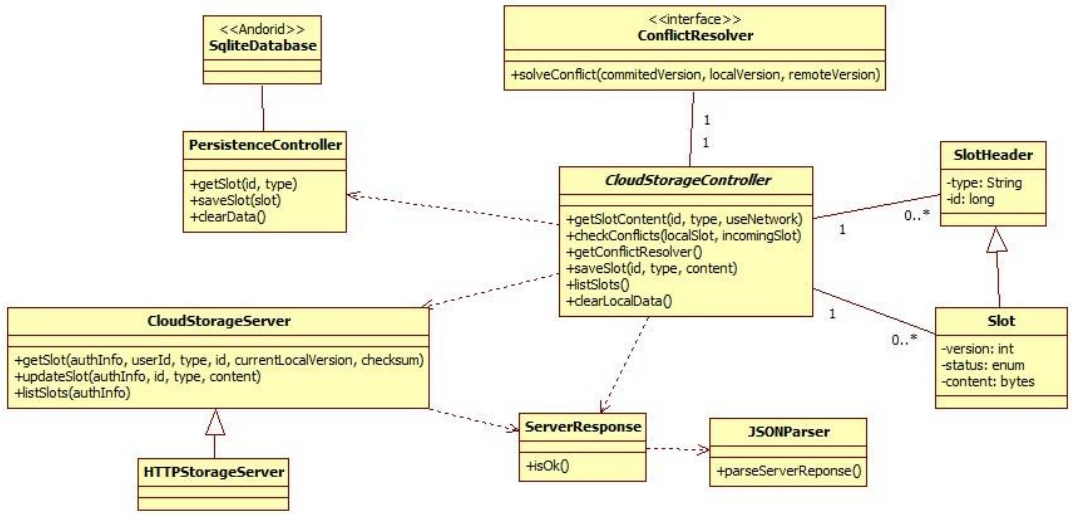


Ilustración 27: Diagrama de clases del sistema de almacenamiento en la nube

Definición de las clases:

- **CloudStorageController:** Controlador principal del subsistema, presentando su funcionalidad al resto de la aplicación. Ofrece las funcionalidades de listar los slots disponibles, leer un slot, guardar un slot y borrar la información local.
- **Slot:** Representa un slot en el sistema.
- **SlotHeader:** Representa la cantidad de información necesaria para conocer la existencia de un slot, por ejemplo el método listSlots devuelve valores de este tipo.
- **PersistenceController:** Controlador encargado del guardado y almacenamiento local peristente de slots.
- **CloudStorageServer:** Interfaz de comunicación con el servidor del almacenamiento en la nube, en este caso la realización se hace mediante conexiones HTTP.
- **ConflictResolver:** Clase implementada por cada aplicación que usa este subsistema y encapsula el código necesario para resolver los conflictos entre las versiones de slots que puedan surgir en la comunicación con el servidor.
- **ServerResponse:** Clase encargada de modelar una respuesta desde el servidor
- **JSONParser:** Clase encargada de transformar las respuestas del servidor en formato JSON a objetos del modelo de la aplicación.

b) Diagramas de secuencia

List Slots:

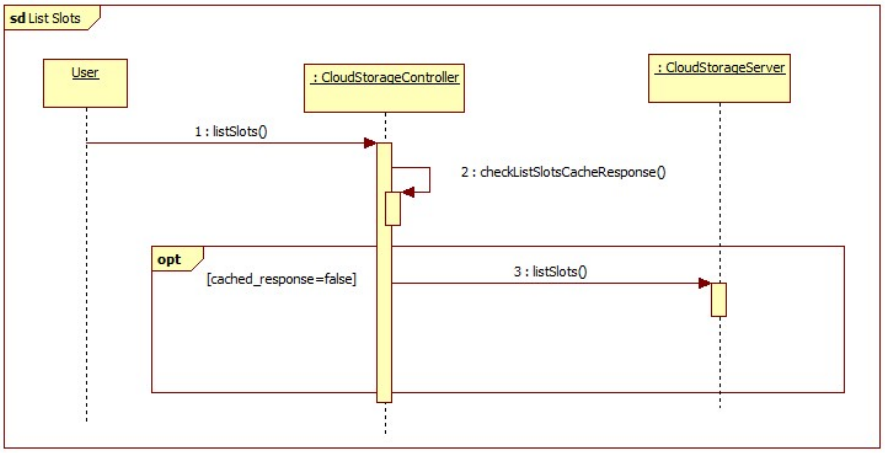


Ilustración 28: Diagrama de secuencia "List Slots"

Read Slot:

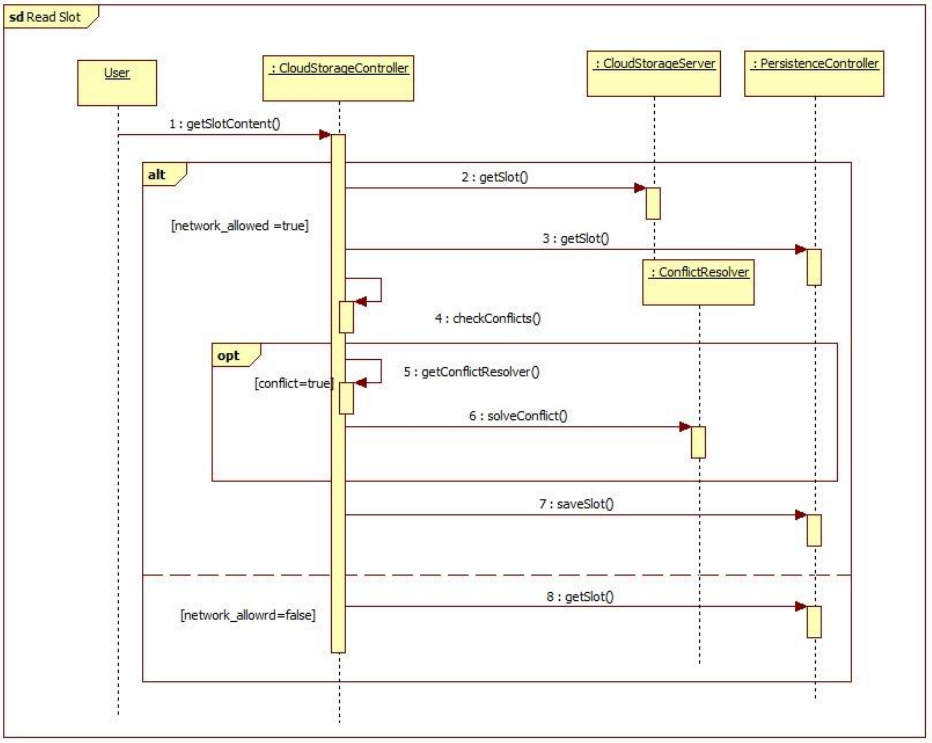


Ilustración 29: Diagrama de secuencia "Read Slot"

Save Slot:

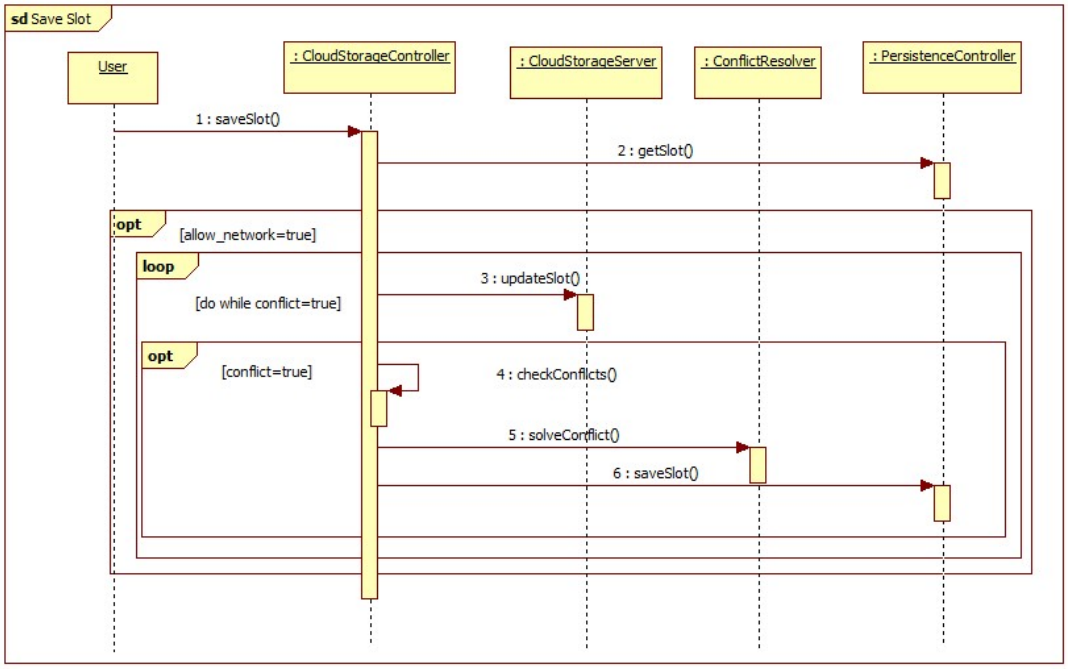


Ilustración 30: Diagrama de secuencia "Save Slot"

Reset Local Data:

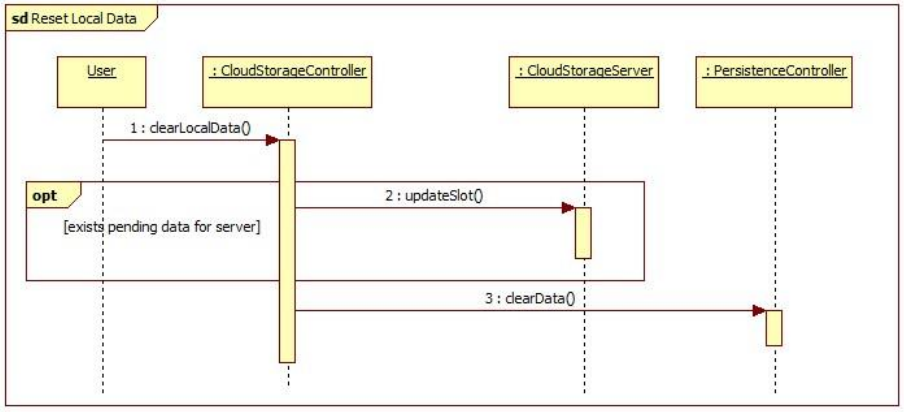


Ilustración 31: Diagrama de secuencia "Reset Local Data"

c) Conflictos

En este apartado se describirá en que consiste y cuando se produce un conflicto: Una situación de conflicto sucede cuando un cliente intenta escribir un servidor pero no lo hace partiendo de la versión actual en el mismo.

La siguiente tabla ilustra el proceso de generación del conflicto mostrando las diferentes versiones que va teniendo cada cliente y el servidor

Paso	A	B	C	D	E	F
Servidor	1	2	3	4	4	5
Cliente A	1	2	3		**4'**	5
Cliente B			3	4		

Explicación: El cliente A comienza editando un slot hasta que en el paso C escribe la versión 3 en el servidor. En ese punto el cliente B actualiza su estado recuperando la versión 3 y modificándolo, escribiendo la versión 4 en el servidor.

La siguiente vez que el cliente A intenta escribir, su versión local es la 3 por lo que intenta escribir la versión 4, que colisiona con la versión 4 que escribió el cliente B.

Para que el cliente A pueda volver a escribir necesita resolver el conflicto, para el cual dispone de la siguiente información:

- Última versión conocida del servidor (versión 3)
- Versión local (4')
- Versión remota (4)

El objetivo es que partiendo de esa información el cliente A combine los datos de las tres de la forma más adecuada para el usuario.

4. Implementación

La implementación e integración de este subsistema en una aplicación Android es extremadamente sencilla, es suficiente con especificar la dirección del servidor y la clase encargada de resolver los conflictos.

En la aplicación de ejemplo se presentará una implementación concreta, que usará el sistema de almacenamiento en la nube para mantener una lista de amigos, sin la intervención del servidor (salvo para almacenar la información)

5. Pruebas

Para probar este módulo se ha desarrollado un proyecto con una serie de casos de test en JUnit que permiten la prueba automatizada del subsistema.

El proyecto de prueba puede consultarse en el proyecto CloudStorageTest adjunto con el código fuente del proyecto

Los aspectos probados en esta suite son:

- Carga de un slot desde el servidor sin datos iniciales en cl cliente.
- Borrado de los datos locales.
- Lectura de un slot inexistente sin conexión con el servidor.
- Lectura de un slot inexistente con conexión con el servidor.
- Guardado de un slot nuevo en local.
- Actualización de un slot existente en local desde servidor.
- Sobre-escritura y remplazado de datos pendientes del servidor al escribir un slot.
- Actualización de un slot con un conflicto
- Guardado de un slot vacío
- Revertir el contenido de un slot.
- Sincronizar el contenido de un slot

F. Sistema de partidas y emparejamientos

1. Requisitos

MHR-1. El sistema se integrará con el sistema de usuarios para autenticar a sus jugadores.

MHR-2. El sistema permitirá solicitar un emparejamiento para una nueva partida en dos modalidades:

- Emparejamiento aleatorio
- Emparejamiento por invitación

MHR-3. Los jugadores de una partida se organizarán en equipos.

MHR-4. Cada jugador debe pertenecer a un equipo (y sólo uno)

MHR-5. Una partida deberá tener como mínimo un equipo.

MHR-6. Cada equipo deberá tener como mínimo un jugador.

MHR-7. Una partida no tendrá un número máximo de jugadores y equipos indeterminado.

MHR-8. Los equipos se identificarán por un valor entero incremental empezando en 1 (equipo 1, equipo 2, equipo3, etc.)

MHR-9. Cada jugador se asociará con su usuario del sistema de usuarios.

MHR-10. Los jugadores se identificarán por un valor entero incremental empezando en 1 (jugador 1, jugador 2, jugador 3, etc.)

MHR-11. Una partida contará con los siguientes atributos

- Identificador numérico único
- Tipo (para distinguir distintos tipos de partida)
- Fase
  - Pendiente de aceptación
  - En preparación
  - En juego
  - Finalizada
- Ronda
- Información del turno (indicando equipos y/o jugadores que pueden actuar)
- Información de extensión: Paquete de información definido por cada aplicación concreta con la lógica e información propia de la misma.
- Fecha de la última acción en la partida
- Fecha de creación

MHR-12. El sistema permitirá a un usuario consultar las partidas en las que participa y ha participado.

MHR-13. El sistema permitirá a los usuarios aceptar o rechazar las partidas en las que se le haya emparejado sin que lo soliciten (por ejemplo por invitación).

MHR-14. El sistema permitirá la búsqueda de usuarios por nombre y por correo electrónico.

MHR-15. El sistema se integrará con Facebook para permitir la invitación de amigos a jugar.

MHR-16. El sistema tendrá un límite máximo de partidas simultáneas que puede jugar un usuario.

MHR-17. El sistema notificará al usuario cuando sea emparejado con un contrincante y cuando el estado de alguna partida sea modificado.

MHR-18. El sistema permitirá a los usuarios abandonar una partida si lleva más de un tiempo determinado (configurable) sin actividad.

MHR-19. El sistema almacenará los oponentes recientes de un usuario y los propondrá como sugerencia cuando el usuario inicie una nueva partida.

MHR-20. Se permitirá configurar en el sistema si dos jugadores pueden tener más de una partida en común.

MHR-21. El sistema borrará de forma automática las partidas que lleven terminadas o inactivas más de un tiempo configurable en el sistema.

MHR-22. El sistema se comunicará con el servidor mediante el protocolo HTTP usando JSON como formato de los datos.

MHR-23. El sistema se implementará en Android (Java)

MHR-24. El sistema no permitirá la creación de nuevas partidas con oponentes aleatorios a usuarios pendientes de emparejamiento

2. Análisis

a) Casos de uso

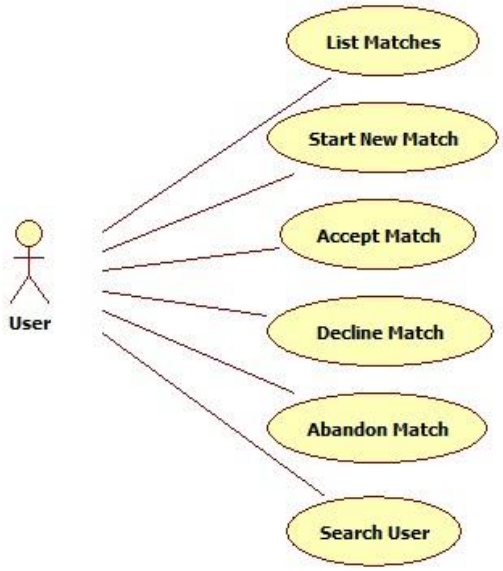


Ilustración 32: Diagrama de casos de uso del módulo de partidas

UC- MS-01	List Matches	
Descripción	El sistema deberá comportarse tal y como se describe en el siguiente caso de uso cuando se solicite el listado de las partidas de un usuario	
Precondición	El usuario está autenticado en el sistema de usuarios	
Secuencia normal	**Paso**	**Acción**
	1	El usuario solicita al servidor el listado de las partidas en las que participa o ha participado.
	2	El sistema recupera las partidas del usuario y las comunica al cliente
	3	El sistema genera la información de las partidas a partir de la respuesta del servidor.
	4	El sistema muestra las partidas al usuario
Postcondición	Se han recuperado las partidas en las que participa o ha participado el usuario.	
Comentarios	Ninguno	

UC- MS-02	Start new match	
Descripción	El sistema deberá comportarse tal y como se describe en el siguiente caso de uso cuando se solicite el inicio de una nueva partida	
Precondición	El usuario está autenticado en el sistema de usuarios	
Secuencia normal	**Paso**	**Acción**
	1	El usuario solicita al servidor el inicio de una nueva partida, especificando opcionalmente los identificadores de los usuarios con los que desea jugar
	2	Si se especificaron oponentes el servidor valida la existencia y disponibilidad de los mismos
	3	Si el sistema encuentra un emparejamiento de forma instantánea crea una partida con los usuarios apropiados
	4	Si el sistema no encuentra usuarios con los que emparejar al usuario se introduce al usuario en la cola de emparejamiento para que se cree una partida cuando sea posible.
Excepciones	**Paso**	**Acción**
	2	Si alguno de los oponentes seleccionados no existen o no están disponibles para ser emparejados el sistema informa del error y el caso de uso termina sin efecto
Postcondición	Se ha creado una nueva partida o se ha introducido al usuario en una cola para emparejarlo cuando sea posible	
Comentarios	Ninguno	

UC- MS-03	Accept match	
Descripción	El sistema deberá comportarse tal y como se describe en el siguiente caso de uso cuando el usuario solicite la aceptación de un emparejamiento en una partida	
Precondición	El usuario está autenticado en el sistema de usuarios y tiene alguna partida pendiente de aceptación	
Secuencia normal	**Paso**	**Acción**
	1	El usuario selecciona que acepta el emparejamiento
	2	El sistema comunica al servidor que el usuario acepta dicho emparejamiento
	3	El servidor actualiza el estado del jugador en la partida
	4	Si todos los usuarios han aceptado la partida el sistema actualiza el estado de la partida para que los jugadores puedan empezar a jugar
Excepciones	**Paso**	**Acción**
	2	Si la partida ya había sido aceptada/rechazada el sistema notifica del error y el caso de uso termina sin efecto
Postcondición	Se ha actualizado el estado del usuario en la partida a “aceptado”	
Comentarios	Ninguno	

UC- MS-04	Decline match	
Descripción	El sistema deberá comportarse tal y como se describe en el siguiente caso de uso cuando el usuario solicite la cancelación de un emparejamiento en una partida	
Precondición	El usuario está autenticado en el sistema de usuarios y tiene alguna partida pendiente de aceptación	
Secuencia normal	**Paso**	**Acción**
	1	El usuario selecciona que declina el emparejamiento
	2	El sistema comunica al servidor que el usuario declina dicho emparejamiento
	3	El servidor actualiza el estado del jugador en la partida
	4	El servidor actualiza el estado de la partida a terminada, notificando del hecho al resto de participantes
Excepciones	**Paso**	**Acción**
	2	Si la partida ya había sido aceptada/rechazada el sistema notifica del error y el caso de uso termina sin efecto
Postcondición	Se ha actualizado el estado del usuario en la partida a “rechazada” y la partida ha sido terminada	
Comentarios	Ninguno	

UC- MS-05	Abandon match	
Descripción	El sistema deberá comportarse tal y como se describe en el siguiente caso de uso cuando el usuario solicite el abandono de una partida	
Precondición	El usuario está autenticado en el sistema de usuarios y participa en una partida	
Secuencia normal	**Paso**	**Acción**
	1	El usuario comunica al sistema que desea abandonar una partida
	2	El sistema comprueba que se cumplen las condiciones para que el usuario pueda abandonar la partida
	3	El sistema actualiza el estado del jugador en la partida a "abandonado"
	4	El sistema actualiza el estado de la partida a "finalizado"
Excepciones	**Paso**	**Acción**
	2	Si las condiciones para el abandono no se cumplen el sistema informa del error y el caso de uso termina sin efecto
Postcondición	Se ha actualizado el estado del usuario en la partida a "abandonado" y la partida ha sido terminada	
Comentarios	Ninguno	

UC- MS-06	Search User	
Descripción	El sistema deberá comportarse tal y como se describe en el siguiente caso de uso cuando se realice la búsqueda de un usuario en el sistema.	
Precondición	El usuario está autenticado en el sistema de usuarios	
Secuencia normal	**Paso**	**Acción**
	1	El usuario introduce los criterios de búsqueda del usuario, como su nombre de usuario o correo electrónico
	2	El sistema busca en aquellos usuarios que cuadren con el criterio de búsqueda
	3	El sistema devuelve al cliente los resultados de la búsqueda
	4	El cliente muestra los resultados
Postcondición	El cliente ha recuperado una lista de usuarios que coinciden con los criterios de búsqueda	
Comentarios	Pueden no encontrarse ningún usuario con el criterio especificado	

3. Diseño

a) Diagrama de clases

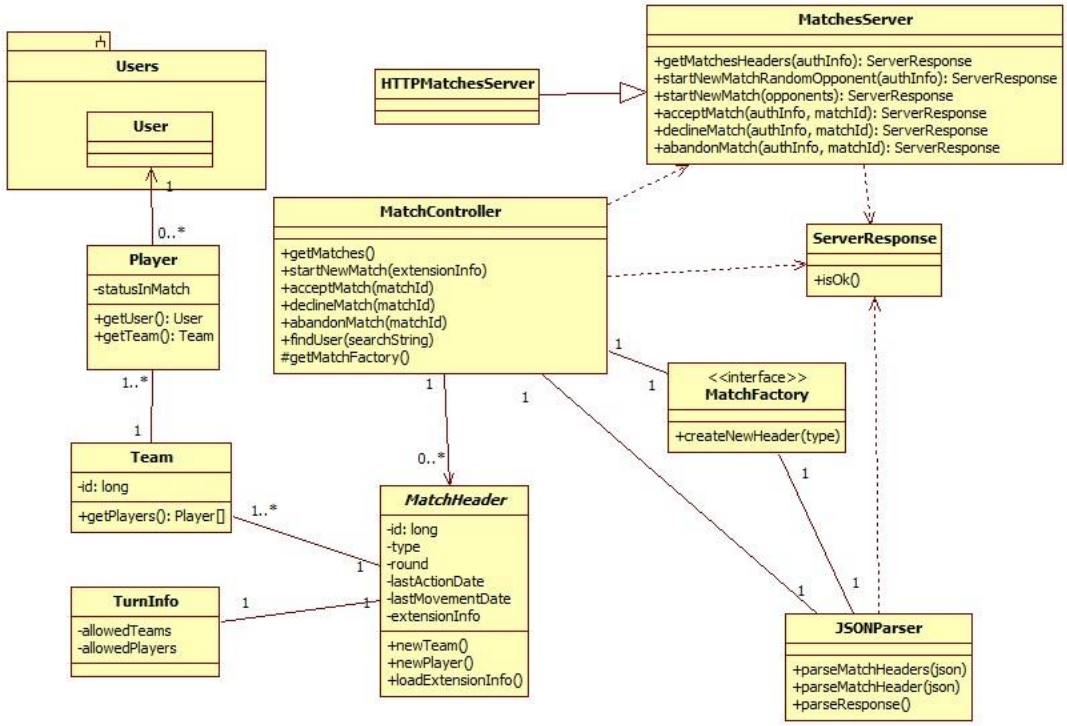


Ilustración 33: Diagrama de clases del módulo de partidas

Descripción de las clases

- **MatchController:** Controlador principal del subsistema, proporciona la interfaz para que la aplicación concreta use las funcionalidades del mismo como listar partidas, crear una partida nueva y aceptar/declinar partidas.
- **MatchesServer:** Representación en el sistema del API ofrecido por el servidor de partidas.
- **JSONParser:** Clase encargada de convertir los datos en formato JSON entrantes desde servidor a objetos.
- **MatchHeader:** Esta clase representa la información principal de una partida. La cantidad de información recogida en esta clase es la suficiente para representarla en los menús de la aplicación. De aspectos relativos al mecanismo del juego y el desarrollo de la partida se encargará el submódulo "Juego"
- **Team:** Representa a un equipo participante en una partida. Toda partida dispone de al menos un equipo y estos están formados por, al menos, un jugador.

- **Player:** Representa a un usuario que participa como jugador en la partida. Cada jugador tiene asignado el índice de jugador en la partida
- **TurnInfo:** Clase encargada de representar la información sobre el turno actual en que se encuentra la partida. En la versión básica esta clase cuenta con la información sobre que equipos y/o jugadores pueden realizar acciones
- **MatchFactory:** Interfaz utilizada para inyectar las dependencias relativas a la creación de las instancias de MatchHeader.

b) Diagramas de secuencia

List Matches:

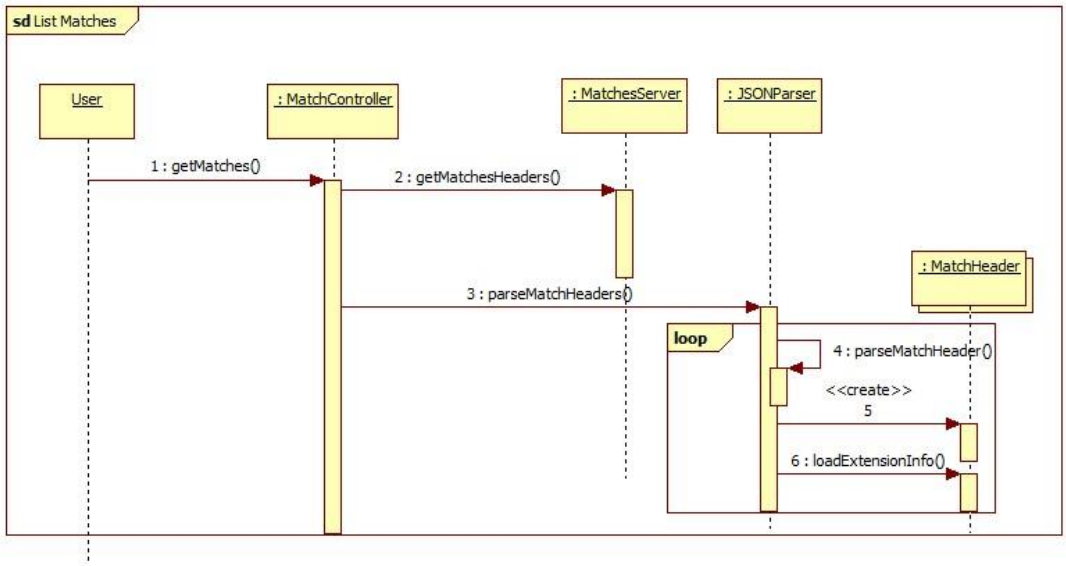


Ilustración 34: Diagrama de secuencia "List matches"

Start New Match:

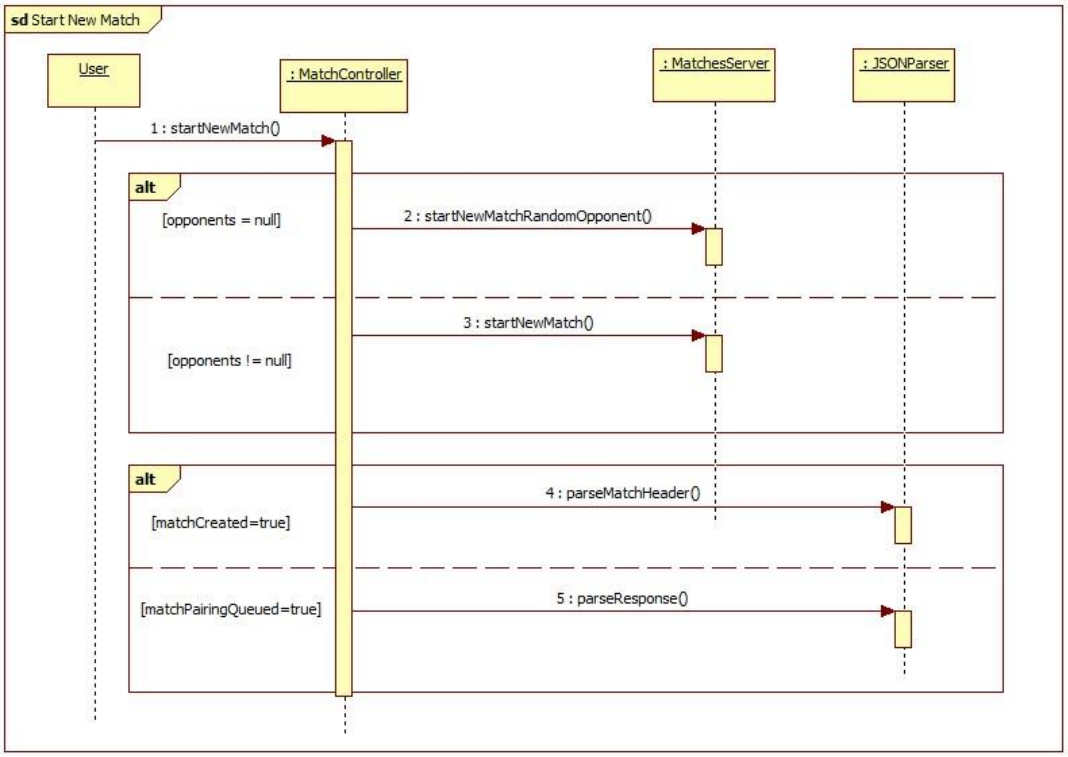


Ilustración 35: Diagrama de secuencia "Start New Match"

Accept Match

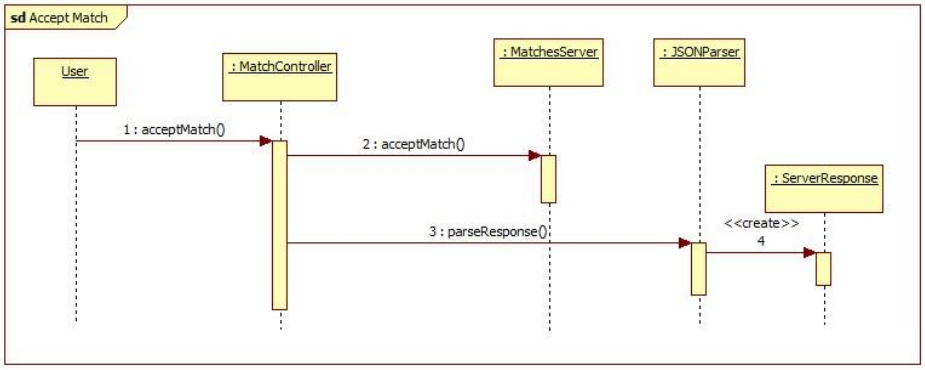


Ilustración 36: Diagrama de secuencia "Accept Match"

Decline Match

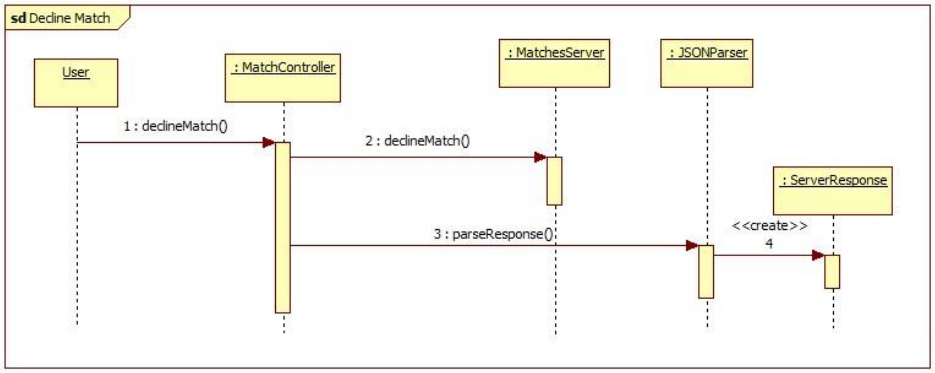


Ilustración 37: Diagrama de secuencia "Decline Match"

Abandon Match

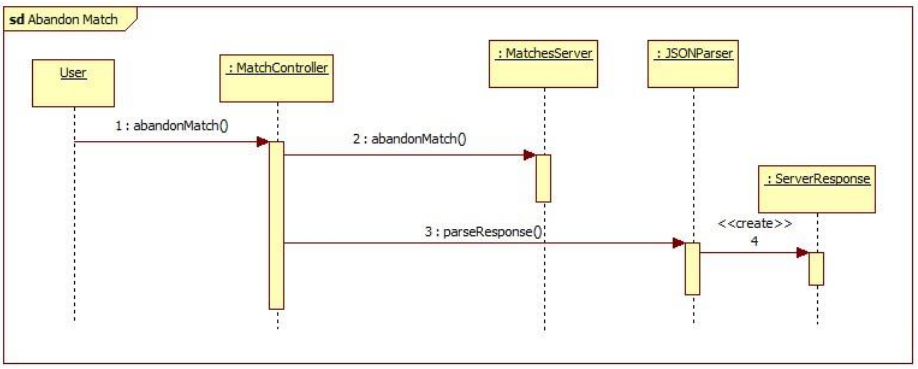


Ilustración 38: Diagrama de secuencia "Abandon Match"

c) API del Servidor

4. Implementación

La implementación e integración del módulo de partidas es sencillo y requiere poco código

En primer lugar es necesario extender la clase MatchController, indicando mediante sus métodos abstractos la dirección del servidor de partidas y la "MatchFactory" con la que se crearán las instancias concretas de MatchHeader que veremos a continuación.

En segundo lugar es necesario extender la clase MatchHeader con la lógica propia de la aplicación concreta. En ella implementaremos el método loadExtensionInfo() encargado de entender e interpretar la información específica de la partida que llega de servidor.

5. Pruebas

A continuación se detallan las pruebas que se han llevado a cabo para este subsistema

Prueba	Resultado
Listado de partidas ✓ Devuelve lista vacía si no hay partidas del usuario ✓ Devuelve las partidas en que participa un usuario correctamente	OK
Creación de partidas ✓ Evita correctamente la creación de una nueva partida si el usuario está pendiente de emparejamiento. ✓ Inserta en cola correctamente a los usuarios que soliciten un emparejamiento con usuario aleatorio ✓ Inserta en empareja satisfactoriamente por invitación ✓ Da error si se intenta invitar a un usuario inexistente	OK
Aceptación y rechazo de partidas ✓ Requerido tras el emparejamiento. ✓ La partida pasa a fase “en juego” tras ser aceptada. ✓ La partida pasa a fase “finalizada” tras ser rechazada ✓ Únicamente se aceptan o rechazan partidas que estén en fase de “aceptación” el resto de peticiones son ignoradas	OK
Abandono de partidas ✓ No se permite el abandono de partidas que no estén inactivas ✓ Se permite el abandono de partidas inactivas ✓ Al abandonar una partida esta pasa a estado finalizado	OK

G. Sistema de juego

1. Requisitos

GMR1. El sistema permitirá a los jugadores de las partidas enviar sus movimientos.

GMR2. El sistema proporcionará mecanismos de extensión con el fin de que cada aplicación pueda añadir su propia lógica al juego.

GMR3. El sistema permitirá a los jugadores de una partida recuperar el estado detallado de la misma.

GMR4. El cliente almacenará localmente la información local de una partida

GMR5. El sistema actualizará el estado de los clientes de dos formas:

- Actualización absoluta: El cliente recibe el estado absoluto de la partida, sobrescribiendo su información local con la información entrante.
- Actualización mediante deltas: El cliente recibe la información que ha sido modificada desde la versión de que dispone en local, realizando los cambios pertinentes en su estado para llegar a nuevo estado de la partida.

GMR6. El sistema implementará mecanismo de detección y corrección de errores en la actualización de partidas mediante deltas.

GMR7. El sistema proporcionará mecanismos de comunicación mediante mensajes especializados para cada aplicación entre los clientes y el servidor encargado de gestionar la partida

GMR8. El sistema autenticará a los usuarios mediante el módulo de usuarios desarrollado en este documento

GMR9. El sistema se integrará con el módulo de partidas para una gestión uniforme de las partidas (a nivel externo) y juego (a nivel interno)

GMR10. El sistema se implementará en Android (Java)

GMR11. La comunicación entre cliente y servidor se realizará mediante HTTP usando JSON como formato de los datos.

2. Análisis

a) Casos de uso

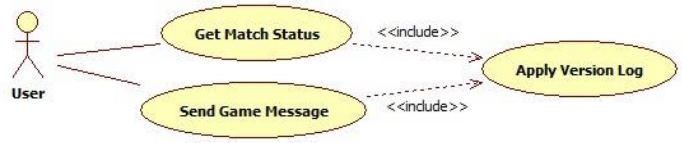


Ilustración 39: Diagrama de casos de uso del módulo de juego

UC- GS-01	Get Match Status	
Descripción	El sistema deberá comportarse tal y como se describe en el siguiente caso de uso cuando el usuario solicite la actualización del estado de una partida	
Precondición	El usuario está autenticado en el sistema de usuarios y participa como jugador en la partida	
Secuencia normal	**Paso**	**Acción**
	1	El usuario solicita al servidor la actualización del estado de una partida, indicando cual es la versión de la partida que el cliente tiene en local.
	2	Si la versión del cliente es válida, el servidor devuelve un VersionLog, en caso contrario devuelve el estado completo en el que se encuentra la partida.
	3	Si devuelve un estado completo el cliente actualiza su versión local sobrescribiendo la información local.
	4	El sistema guarda persistentemente el nuevo estado de la partida en la base de datos
Secuencia alternativa	**Paso**	**Acción**
	3	Si el servidor devuelve un VersionLog el sistema actualiza el estado como se indica en el siguiente caso de uso: Include “Apply versión log”
Postcondición	El cliente tiene el estado de la partida actualizado	
Comentarios	Si el cliente no tiene ninguna información local el servidor devolverá un estado completo.	

UC- GS-02	Send Game Message	
Descripción	El sistema deberá comportarse tal y como se describe en el siguiente caso de uso cuando el usuario solicite el envío de un mensaje al sistema de juego	
Precondición	El usuario está autenticado en el sistema de usuarios y participa como jugador en la partida	
Secuencia normal	**Paso**	**Acción**
	1	El usuario solicita al servidor el envío de juego, especificando la partida y los parámetros del mensaje.
	2	El servidor interpreta y realiza el efecto del mensaje
	3	El sistema devuelve el nuevo estado completo o un VersionLog que permita actualizar el estado local de la partida con los cambios causados por el mensaje
	4	Si se devuelve un VersionLog, el sistema actualiza el estado como se indica en el siguiente caso de uso: Include “Apply versión log”
	5	Si la aplicación del VersionLog falla el sistema repeticiona el estado completo de la partida como especifica el caso de uso: Include “Get Match Status”
Secuencia alternativa	**Paso**	**Acción**
	4	Si devuelve un estado completo el cliente actualiza su versión local sobrescribiendo la información local y guardándolo persistentemente en la base de datos
Postcondición	El mensaje ha sido recibido en servidor y el cliente se ha actualizado reflejando los efectos del mensaje en la partida	
Comentarios	Ninguno	

UC- GS-03	Apply Version Log	
Descripción	El sistema deberá comportarse tal y como se describe en el siguiente caso de uso cuando se requiera la aplicación de un VersionLog para actualizar el estado de una partida	
Precondición	El usuario está autenticado en el sistema de usuarios. Se dispone de la partida en un estado correcto pero no actualizado.	
Secuencia normal	**Paso**	**Acción**
	1	Se solicita la aplicación del versión log
	2	El sistema guarda el estado de la partida
	3	El sistema itera el VersionLog, aplicando para cada Versión todas sus acciones sobre la partida
	4	El sistema comprueba que tras la aplicación de todas las acciones el checksum de la partida sea el mismo que el que indica el VersionLog
	5	Si la actualización se ha realizado con éxito el sistema guarda persistentemente en base de datos el nuevo estado
Secuencia alternativa	**Paso**	**Acción**
	5	Si se produce algún error y los checksum no coinciden el sistema restaura el estado de la partida a través de la versión guardada en el paso 2 y el caso de uso termina sin efecto
Postcondición	La partida se ha actualizado al nuevo estado con éxito	
Comentarios	Ninguno	

3. Diseño

a) Diagrama de clases

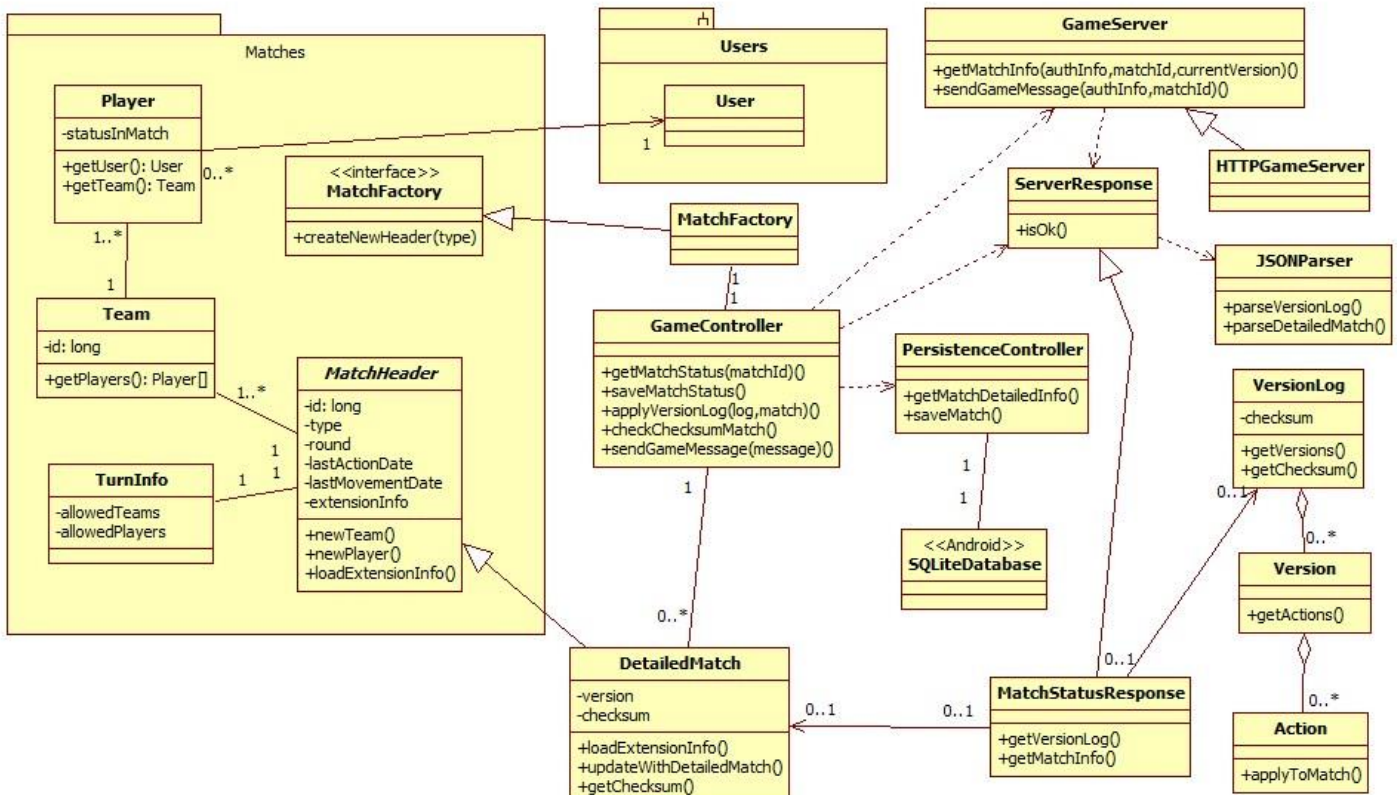


Ilustración 40: Diagrama de clases del módulo de Juego

Descripción de las clases

- **GameController:** Controlador principal del subsistema, aporta la funcionalidad del mismo al resto de subsistemas de la aplicación, ofreciendo las funciones de actualizar el estado de la partida, aplicar un VersionLog y enviar un mensaje de juego.
- **DetailedMatch:** Clase que extiende la cabecera de una partida añadiendo la lógica necesaria para llevar el estado de la partida.
- **Game Server:** Representación del servidor de juegos, proporciona los métodos necesarios para la comunicación cliente -> servidor: getMatchInfo y sendGameMessage.
- **Persistence Controller:** Controlador encargado del almacenamiento persistente de los estados de la partida mediante una base de datos SQLite.
- **JSONParser:** Clase encargada de interpretar las respuestas del servidor en formato JSON y convertirlas a objetos del modelo de la aplicación.
- **MatchStatusResponse:** Esta clase representa una respuesta del servidor con información para actualizar el estado de una partida. Esta información puede ser:
    - Un VersionLog que permita actualizar paso a paso el estado de la partida
    - Un estado completo que permita la actualización atómica de la partida
- **VersionLog**: Log con las diferentes versiones del servidor. Incluye el valor de checksum que debería tener la partida tras aplicarse el log para realizar comprobaciones de error.
- **Version**: Una versión se genera cada vez que una partida cambia de estado por la interacción de un jugador. Cada versión está compuesta de una o más acciones.
- **Acción:** Representación de un cambio básico que puede sufrir la partida, como por ejemplo una modificación de puntuación o un cambio de turno.

b) Diagramas de secuencia

Get Match Status:

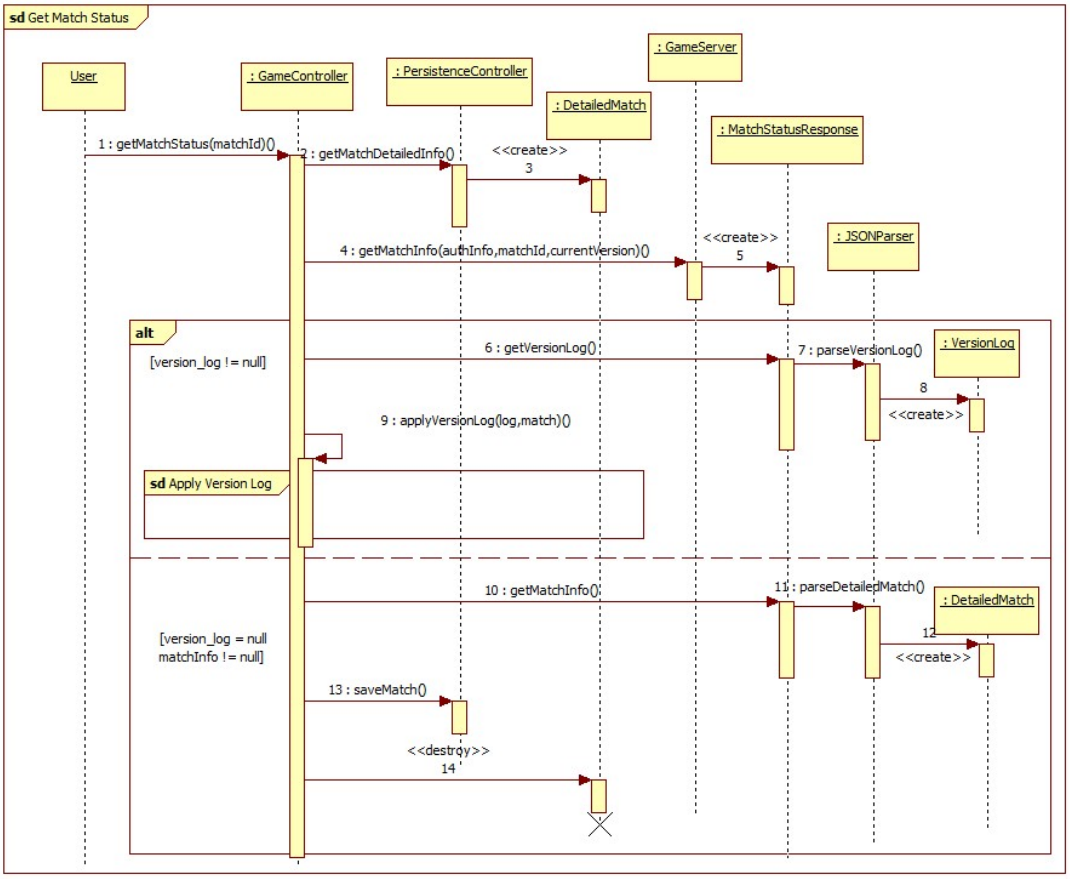


Ilustración 41: Diagrama de secuencia "Get Match Status"

Send Game Message

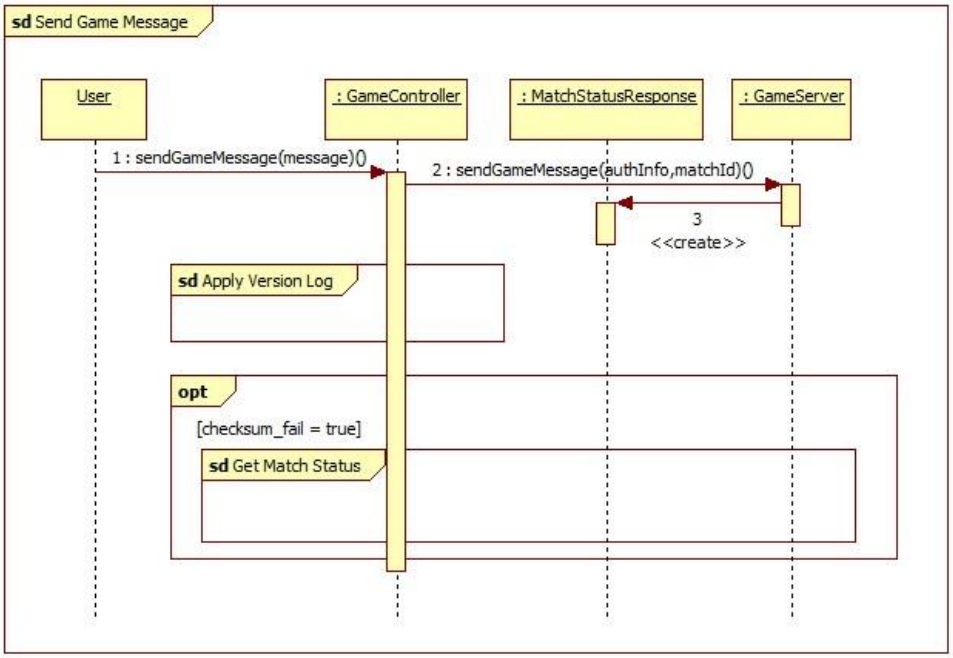


Ilustración 42: Diagrama de secuencia "Send Game Message"

Apply Version Log

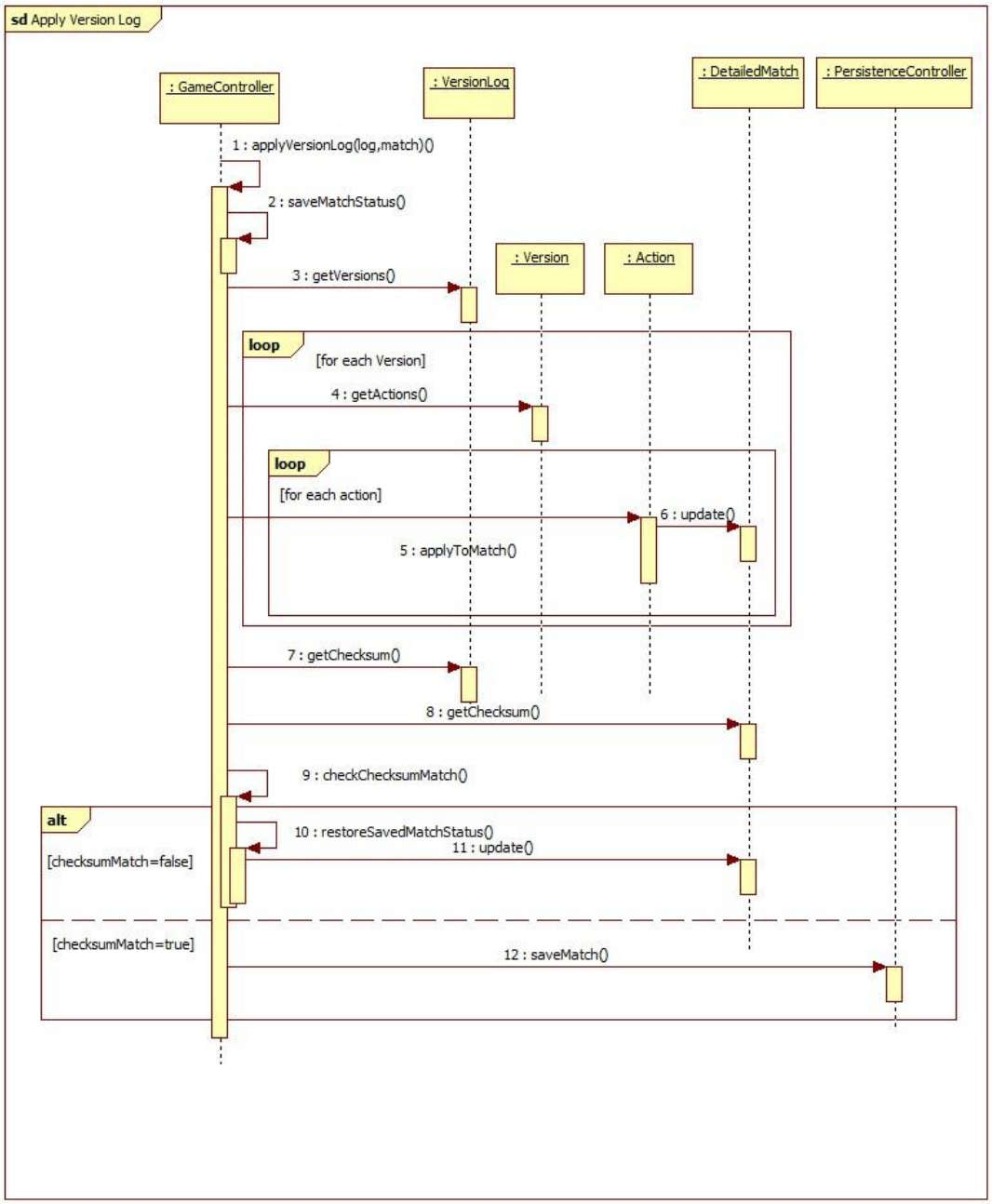


Ilustración 43: Diagrama de secuencia "Apply Version Log"

c) API del Servidor

4. Implementación

La implementación de este subsistema depende completamente del juego concreto a desarrollar por lo que se verá un ejemplo de implementación en la aplicación "Tres en raya" que veremos a continuación.

5. Pruebas

A continuación se detallan las pruebas realizadas sobre este subsistema

Prueba	Resultado
Actualización del estado de una partida ✓ Funcionamiento correcto sin información local. ✓ Funcionamiento correcto con información local sobrescrita por la recepción de un estado completo ✓ Detección correcta de la necesidad de aplicación de un VersionLog a partir de una respuesta de servidor.	OK
Envío de un mensaje de juego ✓ Envío y recepción de la información de extensión correcto. ✓ Modificación remota del estado de la partida correcto ✓ Devolución correcta de la respuesta de servidor con la información necesaria para que el cliente se actualice.	OK
Aplicación de un VersionLog ✓ Parseo de las versiones entrantes del servidor correcto ✓ Aplicación de una acción individual sobre una partida correcta ✓ Aplicación de una versión individual sobre una partida correcta ✓ Aplicación de un VersionLog completo sobre una partida correcto ✓ Generación de checksums correcto ✓ Comprobación de checksums entre partida y VersionLog satisfactorio. ✓ Reversión del estado si los checksum no coinciden o se produce algún error en la ejecución correcto.	OK

APLICACIÓN DE EJEMPLO: "TRES EN RAYA"

6. Aplicación de Ejemplo: “Tres en raya”

A. Introducción

En esta sección se presentará un prototipo de ejemplo de una aplicación que se ejecuta sobre la plataforma que se ha desarrollado desarrollando en este proyecto.

Esta aplicación permitirá a los usuarios jugar partidas del famoso juego “Tres en raya” de forma asíncrona. Permitiendo que cada jugador realice sus movimientos cuando sea posible y después espere a que los oponentes respondan a sus jugadas.

B. Características

Estas son las principales características que aporta cada subsistema de nuestra plataforma a la aplicación de “Tres en raya”:

- Usuarios:
  - Registro de usuarios
  - Autenticación de usuarios
  - Soporte para usuarios anónimos
  - Cambio y recuperación de contraseña
- Estadísticas y logros:
  - Gestión de estadísticas de usuarios
  - Integración de logros
  - Notificación automática de logros
- Almacenamiento en la nube:
  - Gestión de amigos
- Partidas:
  - Listado de partidas
  - Creación de una partida
  - Aceptación / rechazo de partidas
  - Búsqueda de oponentes
- Juego:
  - Desarrollo de la partida

C. Usuarios

En lo referente al registro, autenticación y gestión de usuarios las necesidades de la aplicación se ajustan a la perfección al esquema de pantallas especificado en el módulo de usuarios (para más información consultar el apartado implementación del módulo "Usuarios"):

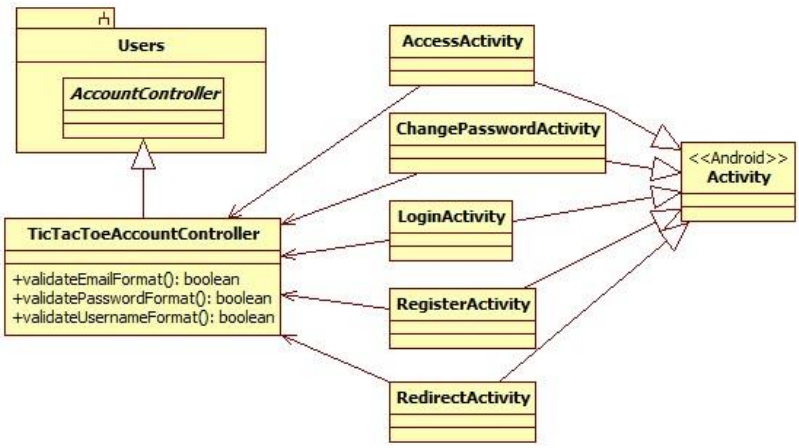


Ilustración 44: Diagrama de clases de Usuarios para Tres en Raya

La dirección url del servidor para las peticiones del subsistema de usuarios es:

```
http://tictactoe.novagecko.com/user/
```

El controlador de la aplicación incorpora, al igual que la aplicación de autenticación de ejemplo, los métodos para validar el formato de los emails, contraseñas y nombres de usuario introducidos en la aplicación. La validación realizada para cada campo es la siguiente:

- Email: El texto tiene formato válido de email.
- Contraseña: El texto tiene una longitud mínima de 4 caracteres.
- Nombre de usuario: El texto tiene una longitud mínima de 4 caracteres.

D. Estadísticas y logros

Los servicios web del sistema de estadísticas se encuentran en la siguiente url:

http://tictactoe.novagecko.com/stats/

Para este sistema se han definido las siguientes estadísticas de usuario (con su identificador de estadística):

- Id=100: Partidas jugadas
- Id=200: Partidas ganadas
- Id=300: Partidas perdidas
- Id=400: Porcentaje de partidas ganadas

Asociados a las mismas se definen los siguientes logros:

- Id 101-104: Partidas Jugadas: Jugar 1,5,15,30
- Id 201-204: Partidas ganadas: Ganar 1,2,5,10,
- Id 301-304: Partidas perdidas: Perder 1,2,5,10
- Id 401-404: Porcentaje de partidas ganadas: Conseguir 20%,40%,60%,80%.

La generación de las estadísticas en el cliente se ha realizado utilizando un HardcodedStatLoader como puede verse a continuación, su código aunque algo extenso es bastante sencillo de comprender.

```
new HardcodedStatLoader() {

@Override
protected void initSystem(StatsAndAchievementsSystem statsSystem) {
          StatsGenerator generator = new StatsGenerator(getFactory());
          initGamesPlayedStats(statsSystem, generator);
          initGamesWonStats(statsSystem, generator);
          initGamesLostStats(statsSystem, generator);
          initGamesWonPercentStats(statsSystem, generator);
}

private void initGamesPlayedStats(
                    StatsAndAchievementsSystem statsSystem,
                    StatsGenerator generator) {
          Stat stat = generator.createStat(STAT_GAMES_PLAYED);
          statsSystem.addStat(stat);
          // STAT_GAMES_PLAYED achievements
          Achievement level1 = generator
                              .createAchievement(ACHIEVEMENT_GAMES_PLAYED_LVL1,
                                                  StatsControllerImpl.this);
          Achievement level2 = generator
```

```
                        .createAchievement(ACHIEVEMENT_GAMES_PLAYED_LVL2,
                                StatsControllerImpl.this);
        Achievement level3 = generator
                        .createAchievement(ACHIEVEMENT_GAMES_PLAYED_LVL3,
                                StatsControllerImpl.this);
        Achievement level4 = generator
                        .createAchievement(ACHIEVEMENT_GAMES_PLAYED_LVL4,
                                StatsControllerImpl.this);
        // Associate the achievements to the statistic
        level1.addRelatedStatistic(stat, 1);
        level2.addRelatedStatistic(stat, 5);
        level3.addRelatedStatistic(stat, 15);
        level4.addRelatedStatistic(stat, 30);

        // Add the achievements to the system
        statsSystem.addAchievement(level1);
        statsSystem.addAchievement(level2);
        statsSystem.addAchievement(level3);
        statsSystem.addAchievement(level4);
}

private void initGamesWonStats(
                StatsAndAchievementsSystem statsSystem,
                StatsGenerator generator) {
        Stat stat = generator.createStat(STAT_GAMES_WON);
        statsSystem.addStat(stat);
        // STAT_GAMES_PLAYED achievements
        Achievement level1 = generator.createAchievement(
                        ACHIEVEMENT_GAMES_WON_LVL1, StatsControllerImpl.this);
        Achievement level2 = generator.createAchievement(
                        ACHIEVEMENT_GAMES_WON_LVL2, StatsControllerImpl.this);
        Achievement level3 = generator.createAchievement(
                        ACHIEVEMENT_GAMES_WON_LVL3, StatsControllerImpl.this);
        Achievement level4 = generator.createAchievement(
                        ACHIEVEMENT_GAMES_WON_LVL4, StatsControllerImpl.this);
        // Associate the achievements to the statistic
        level1.addRelatedStatistic(stat, 1);
        level2.addRelatedStatistic(stat, 2);
        level3.addRelatedStatistic(stat, 5);
        level4.addRelatedStatistic(stat, 10);

        // Add the achievements to the system
        statsSystem.addAchievement(level1);
        statsSystem.addAchievement(level2);
        statsSystem.addAchievement(level3);
        statsSystem.addAchievement(level4);
}

private void initGamesLostStats(
                StatsAndAchievementsSystem statsSystem,
                StatsGenerator generator) {
        Stat stat = generator.createStat(STAT_GAMES_LOST);
        statsSystem.addStat(stat);
        // STAT_GAMES_PLAYED achievements
        Achievement level1 = generator.createAchievement(
                        ACHIEVEMENT_GAMES_LOST_LVL1, StatsControllerImpl.this);
```

```
        Achievement level2 = generator.createAchievement(
                        ACHIEVEMENT_GAMES_LOST_LVL2, StatsControllerImpl.this);
        Achievement level3 = generator.createAchievement(
                        ACHIEVEMENT_GAMES_LOST_LVL3, StatsControllerImpl.this);
        Achievement level4 = generator.createAchievement(
                        ACHIEVEMENT_GAMES_LOST_LVL4, StatsControllerImpl.this);
        // Associate the achievements to the statistic
        level1.addRelatedStatistic(stat, 1);
        level2.addRelatedStatistic(stat, 2);
        level3.addRelatedStatistic(stat, 5);
        level4.addRelatedStatistic(stat, 10);

        // Add the achievements to the system
        statsSystem.addAchievement(level1);
        statsSystem.addAchievement(level2);
        statsSystem.addAchievement(level3);
        statsSystem.addAchievement(level4);
}

private void initGamesWonPercentStats(
                StatsAndAchievementsSystem statsSystem,
                StatsGenerator generator) {
        Stat stat = generator.createStat(STAT_GAMES_WON_PERCENT);
        statsSystem.addStat(stat);
        // STAT_GAMES_PLAYED achievements
        Achievement level1 = generator.createAchievement(
                        ACHIEVEMENT_GAMES_PLAYED_PERCENT_LVL1,
                        StatsControllerImpl.this);
        Achievement level2 = generator.createAchievement(
                        ACHIEVEMENT_GAMES_PLAYED_PERCENT_LVL2,
                        StatsControllerImpl.this);
        Achievement level3 = generator.createAchievement(
                        ACHIEVEMENT_GAMES_PLAYED_PERCENT_LVL3,
                        StatsControllerImpl.this);
        Achievement level4 = generator.createAchievement(
                        ACHIEVEMENT_GAMES_PLAYED_PERCENT_LVL4,
                        StatsControllerImpl.this);
        // Associate the achievements to the statistic
        level1.addRelatedStatistic(stat, 20);
        level2.addRelatedStatistic(stat, 40);
        level3.addRelatedStatistic(stat, 60);
        level4.addRelatedStatistic(stat, 80);

        // Add the achievements to the system
        statsSystem.addAchievement(level1);
        statsSystem.addAchievement(level2);
        statsSystem.addAchievement(level3);
        statsSystem.addAchievement(level4);
}
};
```

Aunque todas estas estadísticas serán modificadas desde el servidor, el cliente las modificará localmente cuando una partida termine para que el cliente reciba las notificaciones de los logros conseguidos en tiempo real.

Como puede verse en el siguiente esquema, el sistema de estadísticas apenas ha requerido la implementación o creación de tres clases adicionales para aportar la funcionalidad del sistema

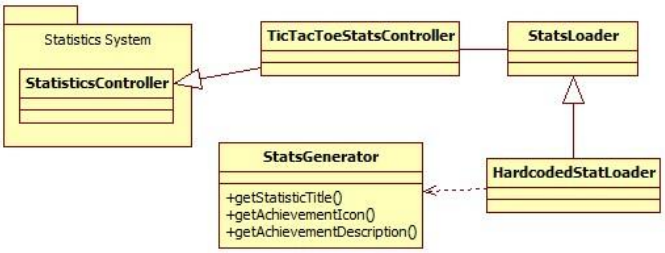


Ilustración 45: Clases extensión del módulo Estadísticas

Por último es necesario implementar en la clase TicTacToeStatsController el método onAchievementCompleted que será llamado automáticamente cuando la modificación de una estadística. En esta aplicación mostraremos un mensaje Toast de Android que nos permite mostrar un mensaje sobre la pantalla durante unos pocos segundos:

```
@Override
public void onAchievementCompleted(final Achievement achievement) {
         String message = getContext().getString(
                           R.string.achievement_completed_message,
                           getContext().getString(

         achievement.getFullDescriptionResource()));
         Toast.makeText(getContext(), message, Toast.LENGTH_LONG).show();
}
```

E. Almacenamiento en la nube

Para la aplicación “Tres en raya” usaremos el servicio de almacenamiento en la nube que ofrece la plataforma para que el usuario pueda guardar su lista de jugadores amigos de forma persistente, remota y mantenida a lo largo de los múltiples dispositivos con los que el usuario pueda ejecutar la aplicación.

Los servicios web del sistema de almacenamiento en la nube se encuentran en la siguiente url:

`http://tictactoe.novagecko.com/cloudstorage/`

Cada usuario únicamente necesitará un slot para almacenar su lista de amigos, este será (elegido arbitrariamente) el slot con tipo=”friends” e id=1.

El siguiente punto a considerar es el formato de los datos que se van a guardar: para cada usuario su identificador numérico único y su nombre.

Puesto que no vamos a hacer un uso intensivo de este almacenamiento el rendimiento no es un factor determinante, por lo que usaremos como formato de datos JSON codificando un array de pares “id,username” y guardaremos sus bytes en el sistema.

El último paso para configurar el sistema de almacenamiento en la nube es definir el algoritmo de resolución de conflictos Como se describe en el apartado “Resolución de conflictos” del módulo “Almacenamiento en la nube” en un conflicto contaremos con la última versión que había en el servidor (lista A), la versión actual local (lista B) y la versión remota (que ha escrito otro cliente, lista C).

El algoritmo de resolución de conflictos utilizado es el siguiente:

- Partir de la lista C.
- Encontrar los elementos que han sido eliminados localmente y por tanto estén en la lista A y no en la B, eliminando estos elementos de la lista C.
- Encontrar los elementos que han sido añadidos localmente estando, por tanto, en la lista B pero no en la A y añadirlos a la lista C.

Este algoritmo no es perfecto pero es perfectamente aceptable.

El diagrama de clases implementado para este subsistema queda por tanto de la siguiente manera:

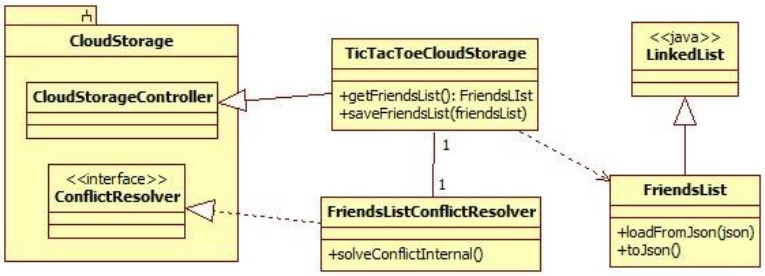


Ilustración 46: Diagrama de clases del módulo de almacenamiento en la nube para gestión de amigos

F. Gestión de partidas y juego

Debido a la necesidad de implementar la lógica de cada aplicación y modelo de juego el módulo de gestión de partidas y el de juego son los que, inevitablemente, necesitan un mayor trabajo.

Lo primero que hay que hacer es extender los controladores de juego y partida de la plataforma e indicarles donde se van a encontrar los servicios web:

`http://tictactoe.novagecko.com/match/`

`http://tictactoe.novagecko.com/game/`

Posteriormente extenderemos un objeto de tipo Match y dos de tipo MatchInfo, que hemos llamado TicTacToeMatch, TicTacToeMatchHeader, TicTacToeMatchDetail. Sobre estas clases construiremos posteriormente toda la lógica necesaria para desarrollar la partida de tres en raya.

Lo siguiente que requerirán los métodos abstractos de los controladores es que especifiquemos el MatchFactory, de esta forma se inyectarán las dependencias necesarias para que se puedan crear nuestros objetos TicTacToeMatch, TicTacToeMatchHeader y TicTacToeMatchDetail dentro del código de la librería de la plataforma, que de otra forma no podría conocerlos.

En este punto todavía queda trabajo por hacer, pero ya tenemos parte de la plataforma funcionando. Las partidas son “tontas” y todavía no conocen nada sobre el tres en raya pero esto no es un problema para que podamos conocer las partidas en las que estamos participando o, incluso, emparejarnos a jugar con otros usuarios.

La lógica necesaria para el juego del tres en raya es realmente sencilla, únicamente necesitamos mantener el estado del tablero, que hemos representado como una matriz de enteros cuyo valor representan el identificador de jugador del propietario de la celda, siendo cero cuando la celda esté vacía.

También es necesario definir qué tipo de mensajes de juego podrán mandar los clientes al servidor. En este ejemplo tan sencillo hay un único mensaje que se puede mandar:

- Tomar una celda: Representará la acción en la que un usuario pone una ficha sobre una de las celdas del tablero y únicamente requerirá que el cliente especifique las coordenadas de la misma (y un identificador para conocer el tipo de mensaje, permitiendo añadir nuevos mensajes en el futuro).

Por ello el parámetro "message" del método **sendGameMessage** de nuestro **GameController** será un objeto JSON con los siguientes campos:

- Id: Id del tipo del mensaje = tc (abreviatura de "tomar celda" o "take cell")
- x: coordenada X de la celda.
- y: coordenada Y de la celda.

La representación textual del parámetro sería esta:

```
{
   "id": "tc",
   "x": 1,
   "y": 0
}
```

También es necesario definir las acciones que pueden darse durante el juego. Estas acciones permitirán al cliente actualizarse mediante deltas a los nuevos estados, esta actualización por deltas en un juego con una lógica tan sencilla y pequeña no ganará en eficiencia pero nos permitirá conocer con facilidad los eventos que van ocurriendo en la partida y de esa forma poder animarlos en la interfaz de la aplicación.

Para el desarrollo del juego se requieren tres acciones:

Acción	Descripción	Representación
		{ "id": "tc", "x": 1, "y": 0 }
Cambio de turno	El turno de la partida cambia, especificando el equipo (y por tanto el jugador) con el turno activo	{ "id":"ctn", "tms":[1], }
Fin de partida	La partida finaliza, especificando el equipo (y por tanto el jugador) ganador	{ "id":"end", "win":1, }

Por último es necesario definir el formato del valor de checksum, que permita comparar estado de la partida y detectar errores. Las variables en el estado de la partida son el propietario de cada celda del tablero y el ganador de la partida (en caso de que haya terminado)

Puesto que el tablero tiene un número fijo y pequeño de celdas (9) y podemos representar su propietario con un valor del 0 al 2 (0= no propietario, 1= propietario el jugador 1, 2 =

propietario el jugador 2) usaremos la siguiente estructura para el checksum: **abcdefghiX** donde las letras de la “a” a la “i” representan los propietarios de las celdas (recorridas por filas de arriba abajo) y la X el ganador de la partida (-1 si no ha terminado, 0 en caso de empate)

Por tanto el checksum del estado de la imagen donde los círculos corresponden al jugador 1 y las cruces al jugador 2 sería:
1222112111

Por tanto este sería el diagrama de clases de la extensión que hemos realizado:

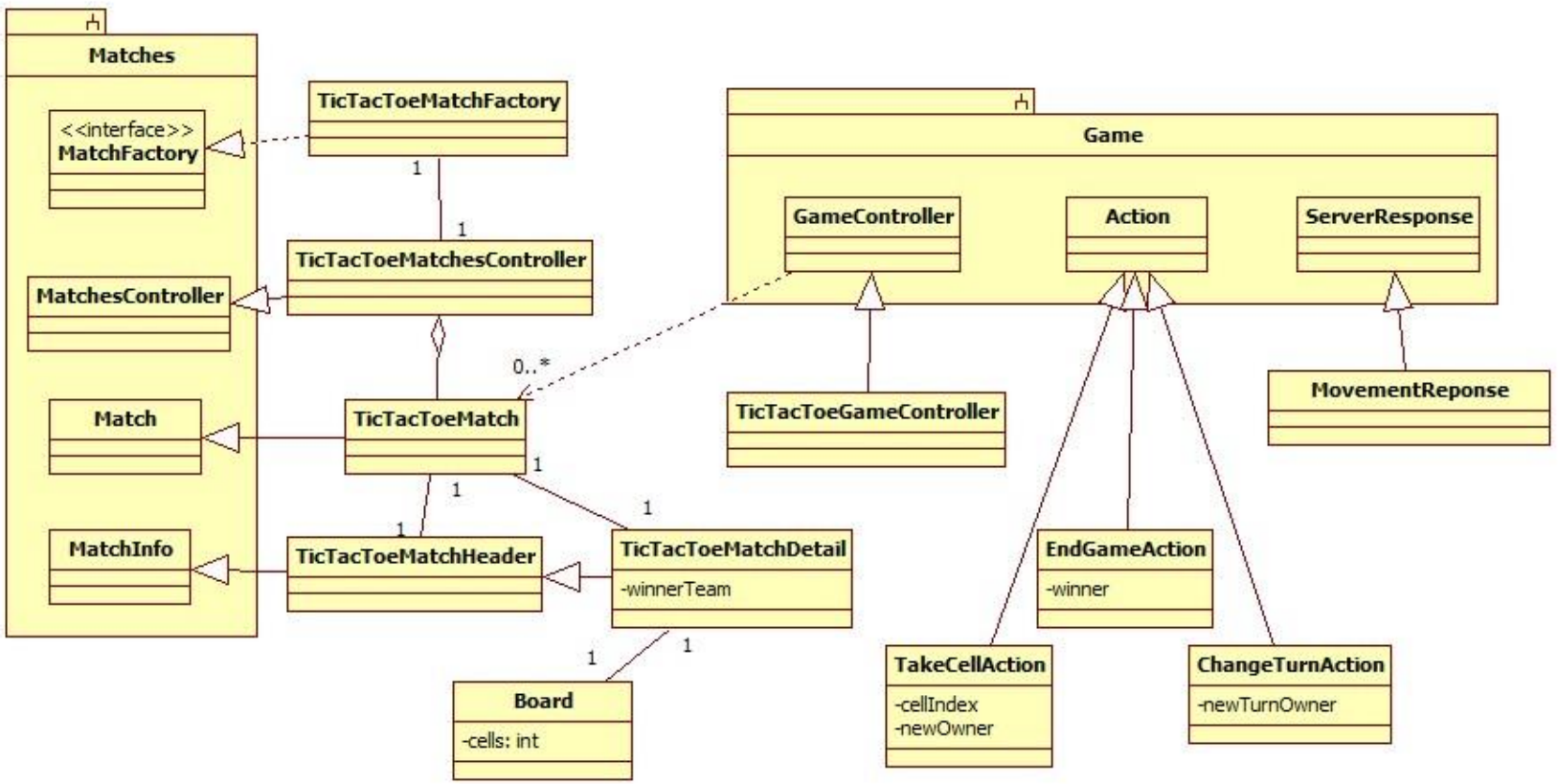


Ilustración 47: Diagrama de clases de extensión del juego tres en raya

G. Push

Los servicios web del sistema de push se encuentran en la siguiente url:

```
http://tictactoe.novagecko.com/push/
```

Para este subsistema no requerimos más que especificar esta url y el ”senderId” para que el cliente pueda registrarse en los servidores de Google. Con esto ya estaremos recibiendo mensajes push en nuestro dispositivo mediante la llamada **onMessageReceived()**.

La función del PushController será por tanto identificar los diferentes mensajes que podemos recibir y entregárselos al componente adecuado.

Básicamente existen dos situaciones en las que el servidor puede querer comunicar una notificación push a un usuario:

- Cuando un jugador recibe el turno en una partida
- Cuando una partida termina, anunciando el ganador si lo hubiere

El formato de los mensajes push es el siguiente:

Id	Parámetros	Descripción
msc	match_id	Notifica que el estado de una partida ha cambiado y ahora es el turno del jugador.
end	match_id winner	Notifica el fin de una partida, indicando el ganador en el campo winner si lo hubiere o un valor nulo en caso de empate

H. Manual de usuario

En este apartado se detallará el modo de uso de la aplicación a nivel de usuario:

1. Instalación

2. Acceso

La primera pantalla que se presenta al abrir la aplicación es la pantalla de "**Acceso**". Si el usuario ya estuviese identificado en el sistema de una sesión anterior esta pantalla es omitida, llevando directamente al usuario a la pantalla "**Principal**".

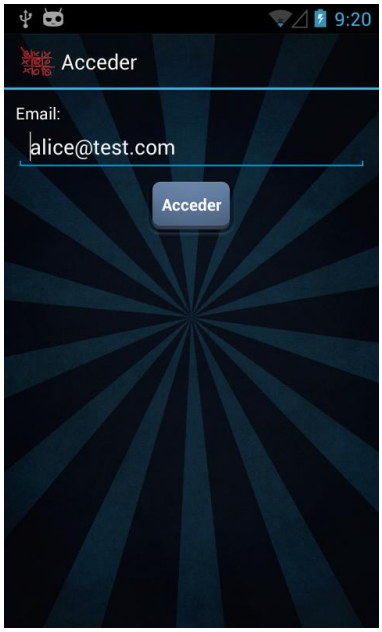
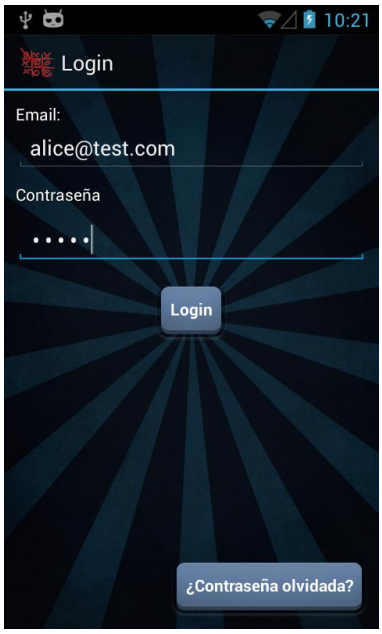


Ilustración 48: Pantalla de Acceso

En la pantalla "Acceso" se debe introducir el email y pulsar "**Acceder**". Al hacerlo el sistema comprobará si el correo está registrado en el sistema, dependiendo de dicha comprobación se mostrará una pantalla u otra:

- Si el email ya está registrado en el sistema el sistema mostrará la pantalla de "**Login**"
- Por el contrario si el email no se encuentra en el sistema el sistema ofrecerá al usuario la opción de registrarse, mostrando la pantalla de "**Registro**"

En la pantalla de "**Login**" únicamente se debe introducir la contraseña y pulsar el botón "Login" para que el sistema compruebe los credenciales de acceso. En caso de que los credenciales sean correctos permitirá el acceso a la pantalla "**Principal**".



Esta pantalla también ofrece la opción "¿Contraseña olvidada?" que permite restablecer la contraseña de la cuenta en caso de no ser recordada. Al seleccionar esta opción el sistema enviará un correo electrónico con las instrucciones para establecer una nueva contraseña.

3. Menú principal

La pantalla "**Principal**" permite de un vistazo conocer el estado de las partidas en las que se está participando, que aparecen ordenadas en una lista clasificada en cuatro categorías:

- Partidas nuevas: En esta categoría se mostrarán las partidas que se hayan iniciado pero requieran la confirmación por parte de algún usuario para

empezar. Una partida en este estado permitirá ser aceptada o rechazada mediante sendos botones "Aceptar" y "Rechazar"

- Tu turno: Esta categoría engloba todas aquellas partidas en las que el usuario puede jugar porque le toca el turno.

- Turno del oponente: Esta categoría todas las partidas en las que se espera que el oponente juegue
- Partidas acabadas: Categoría con todas las partidas acabadas, permitiendo al usuario consultar sus últimos emparejamientos y el resultado final de la partida.

Esta pantalla también presenta varios botones:

- Estadísticas y logros: Permite consultar el estado de las estadísticas y logros del juego.
- Recargar: Refresca la lista de partidas para consultar posibles cambios en la misma
- Nueva partida: Permite comenzar una nueva partida accediendo a la pantalla **"Nueva Partida".**
- Cambio de contraseña: Para cambiar la contraseña de la cuenta.
- Salir: Cierra la sesión y vuelve a la pantalla de "**Acceso**"

Ilustración 49: Pantalla de Login

Ilustración 50: Ilustración 54: Pantalla Principal

4. Nueva partida

La pantalla "**Nueva Partida**" permite comenzar un juego nuevo. Se puede seleccionar la opción "**Oponente Aleatorio**" para unirse a la cola de emparejamientos. El sistema creará una partida tan pronto como disponga de un oponente válido y comenzará la partida.

Por otro lado es posible que queramos jugar con algún amigo o un jugador con el que lo hayamos pasado bien anteriormente. Esta pantalla permite buscar a los usuarios del sistema por nombre e incluso mantener una lista de amigos para poder crear nuevas partidas con estos usuarios.

Para añadir un jugador a la lista de amigos basta con pulsar sobre el botón ★.
Para eliminar un jugador de la lista de amigos hay que pulsar el botón correspondiente: ★.

La lista de amigos se guarda automáticamente en el servidor y se restaura si accedes al juego desde otro dispositivo o tras borrar los datos de la aplicación.

Ilustración 51: Pantalla Nueva Partida

5. Juego

Para acceder a cualquiera de las partidas es suficiente con tocar sobre la misma y se mostrará la pantalla de "**Juego**" como la que puede verse en la captura de pantalla de la derecha.
En ella se desarrolla la partida. En la cabecera de la pantalla un texto indica quien posee el turno en cada momento y el resto de la pantalla muestra el estado actual del tablero.
Durante su turno, un jugador puede colocar su ficha en cualquier casilla del tablero que no esté ocupada, pasando el turno a su oponente.
Gana el primero que consiga colocar tres fichas en línea, ya sea vertical, horizontal o diagonalmente. Cuando un usuario lo consiga la aplicación muestra una ventana indicando el ganador.

Ilustración 52: Pantalla de juego

6. Logros y estadísticas

En esta pantalla podemos conocer todas las estadísticas asociadas a nuestra cuenta, como el número de partidas que hemos jugado o las que hemos ganado.

Además cada estadística tiene asociado cuatro niveles de logros: bronce, plata, oro y diamante. Que podremos conseguir a medida que vayamos jugando partidas y utilizando la aplicación.

Cuando se completa un nuevo logro la aplicación lo notifica mediante un mensaje flotante en la pantalla como el de la imagen a continuación:

Ilustración 53: Mensaje Logro completado

Ilustración 54: Pantalla Estadísticas y Logros

CONCLUSIONES

7. Conclusiones y trabajo futuro

En el inicio del proyecto se identificaron aquellas características de los juegos multijugador asíncronos, identificando las funcionalidades que podría incluir un juego de este estilo, estableciendo prioridades para diferenciar entre elementos necesarios y deseables para una plataforma orientada a facilitar el desarrollo de este tipo de juegos.

Una vez establecido el marco de comparación se analizaron las principales opciones existentes en el mercado para la implementación de juegos online multijugador: Google Play Game Services y SmartFoxServer, estudiando en qué medida cada una de las soluciones se ajustaba a las necesidades de la plataforma de juegos asíncronos. El resultado de la comparativa fue que ninguna de las soluciones existentes resolvía los problemas que un juego asíncrono plantea, principalmente por estar orientadas a un modelo de comunicación en el que los participantes tienen que estar conectados en el mismo momento para interactuar. Pese a ello cabe destacar la buena impresión obtenida de la plataforma SmartFoxServer en cuanto a su arquitectura y extensibilidad.

Ante la necesidad de tener que implementar una solución a medida se consideraron las principales tecnologías sobre las que se podría construir un sistema personalizado y se concluyó que un modelo REST sobre HTTP era la opción más adecuada ya que en los juegos asíncronos multijugador no es necesaria una comunicación en tiempo real ni requieren un intercambio intensivo de mensajes.

La solución personalizada planteada consta de seis módulos:

- Usuarios
- Estadísticas y logros
- Notificaciones Push
- Almacenamiento en la nube
- Módulo de partidas
- Módulo de juego

Cada subsistema se integra con el resto, pero a su vez se han mantenido independientes y con un bajo acoplamiento. Esta fue sin duda la tarea más complicada, junto con la elección de los alcances de cada módulo, optando en muchas ocasiones por el compromiso entre sistemas sencillos y con funcionalidades útiles para el desarrollo de futuras aplicaciones para la plataforma.

Finalmente se ha realizado una aplicación sencilla de ejemplo que permite jugar al conocido juego de tablero “Tres en raya” en ella, además de la funcionalidad básica de creación y

desarrollo de partidas se ha incluido registro y autenticación de usuarios, gestión de amigos mediante el almacenamiento en la nube, comunicación cliente → servidor mediante notificaciones "push" y un sistema sencillo de estadísticas y logros usando las herramientas proporcionadas por la plataforma.

No obstante el proyecto no está, ni de lejos, cerrado, quedando numerosas funcionalidades que se han quedado por el camino debido al tiempo limitado y a la priorización de tareas como por ejemplo:

- **Sistema de Rankings**: Este subsistema se quedó fuera en el planteamiento del alcance del proyecto por motivos temporales y por ser, de entre el resto de módulos, el que nos pareció en principio más accesorio. Pero añadir un sistema de rankings con las funcionalidades analizadas en este documento sería, sin duda, una excelente añadido a la plataforma.
- **Integración con redes sociales**: Aunque el sistema de usuarios contempla el registro y autenticación mediante OAuth, permitiendo el acceso mediante las principales redes sociales con facilidad hay muchas funcionalidades que podrían añadirse para aumentar el grado de interacción social de las aplicaciones como compartición de progresos y estados del juego en las redes sociales, integración de la foto de perfil de las redes sociales con los avatares del sistema, etc.
- **Panel de administración**: Toda aplicación con un cierto éxito requerirá de ciertas tareas de gestión y moderación de la comunidad, como banear usuarios, gestionarlos, modificarlos, obtener información, etc. La realización de un panel de administración que se integre con los diferentes subsistemas de la plataforma y permita gestionar los aspectos más comunes en el mantenimiento de la aplicación.
- **Espectadores**: Una funcionalidad que incluyen cada vez más juegos y que, por ejemplo, la plataforma SmartFoxServer contemplaba es la incorporación de usuarios a las partidas que no participan sino que únicamente contemplan en desarrollo de la partida. Esta sería una funcionalidad a considerar para trabajos futuros.
- **Chat**: Ofrecer la posibilidad de hablar entre ellos a dos jugadores enfrascados en una partida trepidante o sencillamente conversar en una sala de chat con más usuarios es una funcionalidad estupenda que podría considerarse para el futuro.
- **Sistema de emparejamiento**: En el sistema actual el algoritmo de emparejamiento para jugadores al azar es realmente sencillo: Cuando el sistema encuentra la cantidad necesaria de usuarios que desean jugar se crea una nueva partida. Este proceso puede mejorarse en gran medida añadiendo al sistema de emparejamiento un grado extra de complejidad, de forma que considere otros aspectos como el tipo de partida o la experiencia y perfil del usuario para generar emparejamientos más equilibrados mejorando así la experiencia de juego en las aplicaciones.

- **Seguridad**: Aunque se han desarrollado las que se han considerado como medidas de seguridad básicas para el funcionamiento seguro de la aplicación como el almacenamiento y comunicación de información sensible encriptada, la seguridad es un aspecto en el que siempre se puede trabajar más. Hay muchos aspectos de seguridad en los que se podría trabajar, como la detección de usuarios multicuenta, auto-detección de ataques a los servidores de la aplicación, protección anti flooding, etc.

Bibliografía

Libros

- Jim Arlow, Ila Neustadt. "UML 2 (Programación)", Anaya 2006.
- Craig Larman. "UML y patrones", Prentice Hall, 1999.
- Mark L. Murphy. "The busy Coder's Guide: Android Development", CommonsWare, 2008.
- Jason Morris "Android User Interface Development", Packt 2011.
- Peter Saint-Andre. Kevin Smith. Tronçon, Remko: "*XMPP: The Definitive Guide*" O'Reilly 2009.
- Dave Smith, Jeff Friesen. "Android Recipes: A Problem-Solution Approach", aPress 2011.

Referencias Web

- AES: Wikipedia http://en.wikipedia.org/wiki/Advanced_Encryption_Standard (Última visita 1-9-2013)
- ElectroServer 5 http://www.electrotank.com/es5.html (Última visita 1-9-2013)
- Facebook Developers https://developers.facebook.com/ (Última visita 1-9-2013)
- Google Play Game Services: https://developers.google.com/games/services/ (Última visita 1-9-2013)
- Gmail API: https://developers.google.com/gmail/oauth_overview (Última visita 1-9-2013)
- HTTP: Wikipedia http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol (Última visita 1-9-2013)
- Logro: Wikipedia http://en.wikipedia.org/wiki/Achievement_(video_gaming) (Última visita 1-9-2013)
- Network Socket http://en.wikipedia.org/wiki/Network_socket (Última visita 1-9-2013)
- OAuth: Wikipedia http://en.wikipedia.org/wiki/OAuth (Última visita 1-9-2013)
- Push: Wikipedia http://es.wikipedia.org/wiki/Tecnolog%C3%ADa_Push (Última visita 1-9-2013)
- SmartFoxServer http://www.smartfoxserver.com/ (Última visita 1-9-2013)
- Stack Overflow http://stackoverflow.com/ (Última visita 1-9-2013)

- XMPP:Wikipedia
  http://en.wikipedia.org/wiki/Extensible_Messaging_and_Presence_Protocol
  (Última visita 1-9-2013)

Apéndices

Apéndice A: API del servidor

En este apéndice se describirán los servicios web ofrecidos por el servidor para cada uno de los módulos de la plataforma.

Nota: Salvo que se indique lo contrario todos los campos de entrada son parámetros POST y la salida del servicio es un JSON codificando los pares clave/valor de las salidas

A.1: Sistema de usuarios

Método	**Check register by email**		
Ubicación	/check_register_by_email		
Descripción	Este servicio permite conocer cuál es el estado de un usuario en el sistema a partir de su dirección de email. Los estados posibles para un usuario son: • El usuario no está registrado • El usuario está registrado, pero no tiene contraseña. En este caso una nueva contraseña es generada y se le envía al usuario por email. • El usuario está registrado y tiene contraseña		
Entradas	**Nombre**	**Formato**	**Descripción**
→	email	String	Email del usuario
Salidas	**Nombre**	**Formato**	**Descripción**
←	stat	Integer	Código de estado de la petición
←	user_status	Integer	0= No registrado 1= Registrado pero sin contraseña 2= Registrado con contraseña

Comentarios	Ejemplo de salida:
	{
	"stat": 0,
	"user_status": 0,
	}		

Método	**register_by_email**		
Ubicación	/register_by_email		
Descripción	Registra a un usuario en el sistema con el método clásico usuario / email / contraseña.		
Entradas	**Nombre**	**Formato**	**Descripción**
→	email	String	Email del usuario
→	password	String	Contraseña del usuario
→	username	String	Nombre de usuario
Salidas	**Nombre**	**Formato**	**Descripción**
←	stat	Integer	Código de estado de la petición
←	require_activation	Integer	0= La cuenta ha sido creada y activada 1= La cuenta ha sido creada pero requiere activación por correo electrónico

Comentarios	Ejemplo de salida:
	{
	"stat": 0,
	"require_activation": 0,
	}		

Método	**register_anonymous**		
Ubicación	/register_anonymous		
Descripción	Registra a un usuario anónimo en el sistema		
Entradas	**Nombre**	**Formato**	**Descripción**
→	---	---	---
Salidas	**Nombre**	**Formato**	**Descripción**
←	stat	Integer	Código de estado de la petición
←	login_info	JSON	Información de autenticación

Comentarios	Ejemplo de salida:
	{
	"stat": 0,
	"login_info": {
	(((INCLUDE LOGIN INFO)))
	}
	}		

Método	**identify_anonymous_by_email**		
Ubicación	/identify_anonymous_by_email		
Descripción	Identifica a un usuario anónimo en el sistema ya registrado previamente asignándole un email, nombre y contraseña		
Entradas	**Nombre**	**Formato**	**Descripción**
→	email	String	Email del usuario
→	password	String	Contraseña del usuario
→	username	String	Nombre de usuario
→	access_token	String	Autenticación del usuario en el sistema
Salidas	**Nombre**	**Formato**	**Descripción**
←	stat	Integer	Código de estado de la petición
←	login_info	JSON	Información de autenticación

Comentarios	Ejemplo de salida:
	{
	"stat": 0,
	"login_info": {
	(((INCLUDE LOGIN INFO)))
	}
	}		

Método	**login_by_email**		
Ubicación	/login_by_email		
Descripción	Servicio para hacer login en el sistema mediante email. Devuelve los datos necesarios para futuras peticiones que requieran autenticación así como para renovar la sesión		
Entradas	**Nombre**	**Formato**	**Descripción**
→	email	String	Email del usuario
→	password	String	Contraseña del usuario
Salidas	**Nombre**	**Formato**	**Descripción**
←	stat	Integer	Código de estado de la petición
←	login_info	JSON	Información de autenticación

Comentarios	Ejemplo de salida:
	{
	"stat": 0,
	"login_info": {
	(((INCLUDE LOGIN INFO)))
	}
	}		

Método	**refresh_access_token**		
Ubicación	/refresh_access_token		
Descripción	Este servicio refresca el access_token usando el refresh_token. El access token se usa en todos los aquellos servicios que requieran que el usuario este identificado		
Entradas	**Nombre**	**Formato**	**Descripción**
→	refresh_token	String	Token de refresco privado
Salidas	**Nombre**	**Formato**	**Descripción**
←	stat	Integer	Código de estado de la petición
←	login_info	JSON	Información de autenticación

Comentarios	Ejemplo de salida:
	{
	"stat": 0,
	"login_info": {
	(((INCLUDE LOGIN INFO)))
	}
	}		

Método	**change_password**		
Ubicación	/change_password		
Descripción	Modifica la contraseña de un usuario		
Entradas	**Nombre**	**Formato**	**Descripción**
→	access_token	String	Autenticación del usuario en el sistema
→	old_pass	String	Contraseña actual
→	new_pass	String	Nueva contraseña
Salidas	**Nombre**	**Formato**	**Descripción**
←	stat	Integer	Código de estado de la petición
←	login_info	JSON	Información de autenticación

Comentarios	Ejemplo de salida:
	{
	"stat": 0,
	"login_info": {
	(((INCLUDE LOGIN INFO)))
	}
	}		

Método	**recover_password**		
Ubicación	/recover_password		
Descripción	Servicio para recuperar la contraseña, si el email coincide con alguno existente en la base de datos se enviará un correo al usuario con las instrucciones para recuperar su contraseña		
Entradas	**Nombre**	**Formato**	**Descripción**
→	email	String	Email del usuario
→	locale	String	Idioma del cliente (para enviar el email en ese mismo idioma)
Salidas	**Nombre**	**Formato**	**Descripción**
←	stat	Integer	Código de estado de la petición

Comentarios	Ejemplo de salida:
	{
	"stat": 0
	}		

Login Info:

El login info consiste en el paquete de información básica que el cliente debe conocer para cualquier usuario autenticado. Consta de:

Clave	**Descripción**
user_id	Identificador de usuario
username	Nombre público del usuario
access_token	Token para peticiones como usuario autenticado
refresh_token	Token para refrescar el access_token
has_password	Indica si el usuario tiene o no password (default=true)
is_anonymous	Indica si es anónimo (default=false)

Un ejemplo del loginInfo en formato Json:

```
{
   "user_id": 928725,
   "username": "Heyson"
   "access_token": "XXXXXXXXXXXX"
   "refresh_token": "YYYYYYYYYYY"
   "has_password": 1
   "is_anonymous": 0
}
```

A.2: Sistema de estadísticas y logros

Método	**send_stats**		
Ubicación	/send_stats		
Descripción	Actualiza el valor de las estadísticas indicado en el servidor (aquellos que hayan cambiado) y devuelve los valores actualizados (todos)		
Entradas	**Nombre**	**Formato**	**Descripción**
→	access_token	String	Autenticación del usuario en el sistema

→	stats	String (JSON)	JsonArray con los nuevos valores:
			{
			"id": 1,
			"value": 712
			}

Salidas	**Nombre**	**Formato**	**Descripción**
←	stat	Integer	Código de estado de la petición

←	stats	JSON	JSONArray con los valores actualizados
			{
			"id": 1,
			"value": 712
			}

Comentarios	Ejemplo de salida:
	{
	"stat": 0,
	"stats": [
	{
	"id": 1,
	"value": 23
	},
	{
	"id": 2,
	"value": 156
	}
	]
	}		

A.3: Notificaciones Push

Método	**register**		
Ubicación	/ register		
Descripción	Registra a un dispositivo en el sistema de push		
Entradas	**Nombre**	**Formato**	**Descripción**
→	access_token	String	Autenticación del usuario en el sistema
→	push_id	String	Identificador del dispositivo en GCM
Salidas	**Nombre**	**Formato**	**Descripción**
←	stat	Integer	Código de estado de la petición

Comentarios	Ejemplo de salida:
	{
	"stat": 0
	}		

Método	**unregister_by_push_id**		
Ubicación	/unregister_by_push_id		
Descripción	Da de baja a un dispositivo buscándolo a través de su push_id. Una vez dado de baja, el dispositivo no recibirá más mensajes hasta que vuelva a registrarse usando el servicio de registro.		
Entradas	**Nombre**	**Formato**	**Descripción**
→	access_token	String	Autenticación del usuario en el sistema
→	push_id	String	Identificador del dispositivo en GCM
Salidas	**Nombre**	**Formato**	**Descripción**
←	stat	Integer	Código de estado de la petición

Comentarios	Ejemplo de salida:
	{
	"stat": 0
	}		

Método	**register_anonymous**		
Ubicación	/register_anonymous		
Descripción	Registra a un usuario en el sistema de push de manera anónima, es decir, no lo asocia con ningún usuario del sistema de usuarios, por lo que únicamente podrá recibir mensajes push “broadcast”		
Entradas	**Nombre**	**Formato**	**Descripción**
→	push_id	String	Identificador del dispositivo en GCM
Salidas	**Nombre**	**Formato**	**Descripción**
←	stat	Integer	Código de estado de la petición

Comentarios	Ejemplo de salida:
	{
	"stat": 0
	}		

A.4: Almacenamiento en la Nube

Método	**list_slots**		
Ubicación	/list_slots		
Descripción	Recupera la lista de slots sin su contenido (slot_type,slot_id) de un usuario		
Entradas	**Nombre**	**Formato**	**Descripción**
→	access_token	String	Autenticación del usuario en el sistema
→	slot_type	String	Parámetro para filtrar el listado por tipo de slot, slot_type=0 ignora este filtro
Salidas	**Nombre**	**Formato**	**Descripción**
←	stat	Integer	Código de estado de la petición
←	stat	Integer	Json con una lista de pares slot_type,slot_id.

Comentarios	Ejemplo de salida:
	{
	"stat": 0,
	"slots_headers": [
	{
	"slot_type": 1,
	"slot_id": "preferences"
	},
	{
	"slot_type": 2,
	"slot_id": "favourites"
	}
	]
	}		

Método	**get_slot**		
Ubicación	/get_slot		
Descripción	Recupera la información de un slot del servidor, también permite indicar la versión y checksum actual en el cliente para ahorrar tráfico si no hay actualizaciones		
Entradas	**Nombre**	**Formato**	**Descripción**
→	access_token	String	Autenticación del usuario en el sistema
→	slot_type	String	Categoría tipo del slot
→	slot_id	String	Identificador del slot dentro de la categoría
→	current_version	String	Versión actual en el cliente
→	current_checksum	String	Checksum actual en el cliente
Salidas	**Nombre**	**Formato**	**Descripción**
←	stat	Integer	Código de estado de la petición
←	slot	JSON	Json con la información del slot

Comentarios	Ejemplo de salida:
	{
	"stat": 0,
	"slot": {
	"version": 0,
	"slot_type": 1,
	"slot_id": "foo",
	"user_id": 1,
	
	"content":"c2Rhc2Rhc2Rhc2Rhc2Rhc2Q4eWF0ZmRhNnNmZHRnODBmc2RhOHlnZnNkMGhmc2RhMGh1ZnNkYTBoZnNkYTBoZnNkYTBoZnNkYQ=="
	}
	}		

Método	**update_slot**		
Ubicación	/update_slot		
Descripción	Actualiza un slot con la id indicada. En caso de conflicto la petición es rechazada y se debe analizar el contenido actual del slot, combinar ambos valores y repeticionar actualizando su current_version a la versión del servidor.		
Entradas	**Nombre**	**Formato**	**Descripción**
→	access_token	String	Autenticación del usuario en el sistema
→	slot_type	String	Categoría tipo del slot
→	slot_id	String	Identificador del slot dentro de la categoría
→	version	String	Version que el cliente tiene del slot tras la modificación, es decir, si el cliente tiene la versión 5 y modifica el contenido mandará un 6
→	content	String	Nuevo contenido del slot en base64
→	override	Boolean	Permite ignorar la validaciones de versión y sobrescribir el valor directamente
Salidas	**Nombre**	**Formato**	**Descripción**
←	stat	Integer	Código de estado de la petición
←	slot	JSON	Json con la información del slot remoto en caso de conflicto. NOTA: Este campo solo aparecerá en caso de conflicto

Comentarios	Ejemplo de salida:
	{
	"stat": 0,
	"slot": {
	"version": 0,
	"slot_type": 1,
	"slot_id": "foo",
	"user_id": 1,
	
	"content":"c2Rhc2Rhc2Rhc2Rhc2Rhc2Q4eWF0ZmRhNnNmZHRnODBmc2RhOHlnZnNkMGhmc2RhMGh1ZnNkYTBoZnNkYTBoZnNkYTBoZnNkYQ=="
	}
	}		

A5: Sistema de Partidas

Método	**get_matches_headers**		
Ubicación	/get_matches_headers		
Descripción	Este servicio devuelve las partidas en las que está participando (o ha participado) un jugador junto con la información necesaria para representarlas en la pantalla de selección de partidas		
Entradas	**Nombre**	**Formato**	**Descripción**
→	access_token	String	Autenticación del usuario en el sistema
→	include_finished	String	Incluir partidas terminadas en el resultado
Salidas	**Nombre**	**Formato**	**Descripción**
←	stat	Integer	Código de estado de la petición
←	matches	JSON	son con un array de partidas, para cada partida se dará la siguiente información (consultar la documentación del módulo para más info sobre los parámetros): • id • phase • round • current_turn • players • version • checksum

Comentarios	Ejemplo de salida:
	{
	"stat": 0,
	"matches": [
	{
	"id": 1,
	"type": 0,
	"name": null,
	"version": 9,
	"round": 3,
	"phase": "game",
	"creation_timestamp":1369202389,
	"last_move_timestamp":1369214107,
	"current_turn": {
	"allowed_teams": [
	1
	]
	},
	"max_players": 2,
	"max_teams": 2,		

```
        "players": [
          {
            "player_id": 1,
            "team_id": 1,
            "user_id": 825624,
            "username": "Redneck_298",
            "avatar": "http://goo.gl/s75oG",
            "is_ready": 1
          },
          {
            "player_id": 2,
            "team_id": 2,
            "user_id": 873372,
            "username": "LinuxYedi",
             "avatar": "http://goo.gl/EWO2n",
             "is_ready": 1
          }
        ],
        "extension_info": {
          "players_scores": {
            "1": 374,
            "2": 423
          }
        }
      }
    ]
}
```

Método	**request_new_match**		
Ubicación	/request_new_match		
Descripción	El usuario solicita iniciar una nueva partida. En los parámetros de llamada puede especificar los oponentes y los parámetros de la partida si los hubiere. Si no se especifican oponentes se emparejará de forma aleatoria. Este servicio responde inmediatamente con un stat =0 si todo ha ido bien y el cliente deberá pedir los match_headers de nuevo. Esto permite que el emparejamiento pueda funcionar de diferentes maneras: • Que se cree una partida y se le asignen los jugadores si hay oponentes inmediatamente • Que, mientras está emparejándose a ese jugador, se levante una bandera de "emparejamiento pendiente" que el cliente represente como considere oportuno • Que se cree una partida sin oponentes y cuando haya oponentes se les asigne a la partida. Este último caso es más complicado de implementar pero permitiría al usuario ir jugando su primer turno en lo que se le encuentra un oponente. En el caso de que se pueda realizar el emparejamiento de forma instantánea (por ejemplo porque hay gente esperando emparejarse, o porque se han especificado los oponentes)		
Entradas	**Nombre**	**Formato**	**Descripción**
→	access_token	String	Autenticación del usuario en el sistema
→	opponents_info	JSON	Json Array con los user_id a incluir en la partida
→	extension_info	JSON	Información adicional necesaria para crear la partida (como modos de juego u otros parámetros para la partida). Este campo puede ser nulo si no es necesario (o si se desea usar los parámetros por defecto)
Salidas	**Nombre**	**Formato**	**Descripción**
←	stat	Integer	Código de estado de la petición
←	header	JSON	Cabecera de la partida en caso de que se crease instantáneamente

Comentarios	Ejemplo de salida:
	{
	"stat": 0,
	"header": ((match_info)),
	}		

Método	**accept_match**		
Ubicación	/accept_match		
Descripción	Con este servicio el usuario acepta la participación en una partida. Típicamente porque el usuario ha sido emparejado por el sistema o a petición de otro usuario y se requiere que el usuario acepte el emparejamiento antes de empezar a jugar		
Entradas	**Nombre**	**Formato**	**Descripción**
→	access_token	String	Autenticación del usuario en el sistema
→	match_id	Integer	Id de la partida que se acepta
→	extension_info	JSON	Información adicional
Salidas	**Nombre**	**Formato**	**Descripción**
←	stat	Integer	Código de estado de la petición
←	header	JSON	Cabecera de la partida tras crearse

Comentarios	Ejemplo de salida:
	{
	"stat": 0,
	"header": ((match_info)),
	}		

Método	**decline_match**		
Ubicación	/decline_match		
Descripción	Con este servicio el usuario rechaza la participación en una partida. Típicamente porque el usuario ha sido emparejado por el sistema o a petición de otro usuario y se requiere que el usuario acepte el emparejamiento antes de empezar a jugar		
Entradas	**Nombre**	**Formato**	**Descripción**
→	access_token	String	Autenticación del usuario en el sistema
→	match_id	Integer	Id de la partida que se rechaza
→	extension_info	JSON	Información adicional
Salidas	**Nombre**	**Formato**	**Descripción**
←	stat	Integer	Código de estado de la petición

Comentarios	Ejemplo de salida:
	{
	"stat": 0
	}		

Método	**abandon_match**		
Ubicación	/abandon_match		
Descripción	Servicio para abandonar una partida, una partida podrá abandonarse por motivos como: • Está inactiva • El equipo que la intenta abandonar no está en su turno activo • Ha pasado un tiempo razonable desde que se recibió el último movimiento para considerarla inactiva		
Entradas	**Nombre**	**Formato**	**Descripción**
→	access_token	String	Autenticación del usuario en el sistema
→	match_id	Integer	Id de la partida que se abandona
Salidas	**Nombre**	**Formato**	**Descripción**
←	stat	Integer	Código de estado de la petición

Comentarios	Ejemplo de salida:
	{
	"stat": 0
	}		

Método	**search_users**		
Ubicación	/search_users		
Descripción	Mediante este servicio el usuario recuerda a el/los oponentes con el turno activo que les toca mover y que está a la espera.		
Entradas	**Nombre**	**Formato**	**Descripción**
→	access_token	String	Autenticación del usuario en el sistema
→	search_info	Integer	son Array Con los valores a buscar. Por ejemplo ["john", "user@example.com"]
Salidas	**Nombre**	**Formato**	**Descripción**
←	stat	Integer	Código de estado de la petición

Comentarios	Ejemplo de salida:
	{
	"stat": 0,
	“users”:[{
	"user_id": 231264,
	"username": "John",
	"avatar_url": null,
	"country": "es"
	}]
	}		

A.6: Sistema de Juego

Método	**send_extension_message**		
Ubicación	/send_extension_message		
Descripción	Envía un mensaje al servidor, como por ejemplo un movimiento o una query		
Entradas	**Nombre**	**Formato**	**Descripción**
→	access_token	String	Autenticación del usuario en el sistema
→	version	String	Versión de la partida que tiene el cliente en el momento de enviarlo
→	message	JSON	Json definido por la extensión con la información del mensaje
Salidas	**Nombre**	**Formato**	**Descripción**
←	stat	Integer	Código de estado de la petición
←	match_status	JSON	Estado de la partida (Para más información consultar la documentación del módulo de Juego)
←	version_log	JSON	Version Log (Para más información consultar la documentación del módulo de Juego)

Comentarios	Ejemplo de salida:
	{
	"stat": 0,
	"match_status":"(((Ver Game Status)))"
	"version_log":"(((Ver Version Log)))"
	}		

Método	**get_match_status**		
Ubicación	/get_match_status		
Descripción	Devuelve la información detallada del estado la partida. Opcionalmente permite al cliente comunicar su estado actual y si es correcto el servidor responderá que todo está bien pero no devolverá el Game Status.		
Entradas	**Nombre**	**Formato**	**Descripción**
→	access_token	String	Autenticación del usuario en el sistema
→	match_id	Integer	Identificador de la partida a consultar
→	checksum	String	Checksum del cliente
→	version	Integer	Versión del cliente
Salidas	**Nombre**	**Formato**	**Descripción**
←	stat	Integer	Código de estado de la petición
←	match_status	JSON	Estado de la partida (Para más información consultar la documentación del módulo de Juego)
←	version_log	JSON	Version Log (Para más información consultar la documentación del módulo de Juego)

Comentarios	Ejemplo de salida:
	{
	"stat": 0,
	"match_status":"(((Ver Game Status)))"
	"version_log":"(((Ver Version Log)))"
	}		

Apéndice B: Código de respuestas de los servicios web

Éxito:

Peticiónes exitosas

Rangos de código:**0-999**

Nombre	Código
RESPONSE_OK	0

Permanentemente no disponible

Grupo de respuesta para errores que indiquen que el servicio ya no funcionará en adelante salvo que alguna condición cambie

Rangos de código:**1000-1999**

Nombre	Código
ERROR_PERMANENT_UNAVAILABLE_GENERIC	1000
ERROR_CLIENT_UPDATE_REQUIRED	1001
ERROR_SERVICE_PERMANENTLY_MOVED	1002
ERROR_SERVICE_PERMANENTLY_DISABLED	1003

Temporalmente no disponible

Grupo de errores que indican que un servicio está temporalmente fuera de servicio

Rangos de código:**2000-2999**

Nombre	Código
ERROR_TEMPORALLY_UNAVAILABLE_GENERIC	2000
ERROR_IN_MAINTENANCE	2001
ERROR_TEMPORALLY_DISABLED	2002

Errores de protocolo

Grupo de respuestas para los errores de protocolo

Rangos de código:**3000-3999**

Nombre	Código
ERROR_INVALID_INPUTS_GENERIC	3000
ERROR_USERNAME_MISSING	3001
ERROR_USERNAME_INVALID_FORMAT_INTERNAL	3002
ERROR_EMAIL_MISSING	3003
ERROR_EMAIL_INVALID_FORMAT_INTERNAL	3004
ERROR_PASSWORD_MISSING	3005
ERROR_PASSWORD_INVALID_FORMAT_INTERNAL	3006
ERROR_ACCESS_TOKEN_MISSING	3007
ERROR_ACCESS_TOKEN_INVALID_FORMAT_INTERNAL	3008
ERROR_USER_ID_MISSING	3009
ERROR_USER_ID_INVALID_FORMAT_INTERNAL	3010
ERROR_SECRET_MISSING	3011
ERROR_SECRET_INVALID_FORMAT_INTERNAL	3012
ERROR_TOKEN_MISSING_OR_INVALID_FORMAT	3013
ERROR_PASSWORD_RECOVERY_TOKEN_INVALID_FORMAT_INTERNAL	3014
ERROR_PASSWORD_RECOVERY_TOKEN_INVALID	3015
ERROR_INVALID_DEVICE_ID	3016
ERROR_INVALID_OS_CODE	3017
ERROR_PARSE	3018
ERROR_ACTIVATED_INVALID_FORMAT	3019
ERROR_ANONYMOUS_INVALID_FORMAT	3020
ERROR_INVALID_MATCH_ID	3021
ERROR_INVALID_VERSION_NUMBER	3022
ERROR_INVALID_EXTENSION_MESSAGE	3023
ERROR_INVALID_PUSH_ID	3024
ERROR_TOO_MANY_SEARCH_ITEMS	3025
ERROR_INVALID_OPONENTS	3026
ERROR_INVALID_CLOUD_STORAGE_SLOT_ID	3027
ERROR_INVALID_CLOUD_STORAGE_SLOT_TYPE	3028
ERROR_INVALID_CLOUD_STORAGE_SLOT_CONTENT	3029
ERROR_CLOUD_STORAGE_USER_HAS_TOO_MANY_SLOTS	3030
ERROR_CLOUD_STORAGE_SLOT_NOT_FOUND	3031
ERROR_INVALID_IMAGE_ID	3032
ERROR_INVALID_QUANTITY	3033
ERROR_ELEMENT_HAS_OWNER	3034
ERROR_USER_ALREADY_IN_MATCH_QUEUE	3036

Not Found Errors

Grupo de errores para indicar que algo no ha podido ser encontrado, imposibilitando la realización de la petición.

Rangos de código:**4000-4999**

Nombre	Código
ERROR_NOT_FOUND_GENERIC	4000
ERROR_USER_NOT_FOUND	4001
ERROR_EMAIL_NOT_FOUND	4002
ERROR_MATCH_NOT_FOUND	4003

Errores de servidor

Grupo de códigos de error para los errores internos de servidor

Rangos de código:**5000-5999**

Nombre	Código
ERROR_SERVER_ERROR_GENERIC	5000
ERROR_DATABASE_CONNECTION_FAILED	5001
ERROR_EXECUTING_QUERY	5002
ERROR_STATEMENT_PREPARE_FAILED	5003
ERROR_STATEMENT_BIND_PARAM_FAILED	5004
ERROR_STATEMENT_BIND_RESULT_FAILED	5005
ERROR_STATEMENT_EXECUTION_FAILED	5006
ERROR_SENDING_EMAIL	5007
ERROR_MODULE_NOT_FOUND	5008

Quota Errors

Conjunto de códigos para errores relacionados con cuotas o límites temporales.
Rangos de código:**6000-6999**

Nombre	Código
ERROR_QUOTA_ERROR_GENERIC	6000
ERROR_GENERIC_USER_MUST_WAIT	6001
ERROR_TOO_MANY_RECENT_ACCESS_TOKENS	6002

Errores de permisos

Grupo de códigos para representar errores de permisos

Rangos de código:**7000-7999**

Nombre	Código
ERROR_PERMISSION_GENERIC	7000
ERROR_AUTH_EXPIRED	7001
ERROR_AUTH_INVALID	7002
ERROR_AUTH_REQUIRED	7004
ERROR_OPERATION_NOT_ALLOWED	7010
ERROR_GENERIC_BANNED	7990

Errores Condicionales

Conjunto de códigos para errores condicionales

Rangos de código:**8000-8999**

Nombre	Código
ERROR_COULD_NOT_CONNECT	8000
ERROR_USERNAME_ALREADY_TAKEN	8001
ERROR_EMAIL_ALREADY_TAKEN	8002
ERROR_ALREADY_REGISTERED	8003
ERROR_SAME_NEW_AND_OLD_PASSWORD	8004
ERROR_PASSWORD_IS_WEAK	8005
ERROR_NETWORK_UNAVAILABLE	8006
ERROR_ACTION_ALREADY_IN_PROGRESS	8007
ERROR_DEFAULT_ERROR	8008
ERROR_INVALID_AVATAR_URL	8009
ERROR_ACTION_ALREADY_PERFORMED	8010
ERROR_PLAYER_IS_OUTDATED	8011
ERROR_NOT_PLAYER_TURN	8012
ERROR_INVALID_STATE	8013
ERROR_CONFLICT	8014

Apéndice B: Códigos de respuesta de los servicios web

Errores de entradas de usuario

Conjunto de códigos para representar errores de entrada de datos por parte del usuario
Rangos de código:**9000-9999**

Nombre	Código
ERROR_USER_INPUT_GENERIC	9000
ERROR_USERNAME_INVALID_FORMAT	9001
ERROR_EMAIL_INVALID_FORMAT	9002
ERROR_PASSWORD_INVALID_FORMAT	9003
ERROR_PASSWORD_INVALID	9004

Errores de servidores externos

Códigos para la representación de errores de servidores externos

Rangos de código:**10000-10999**

Nombre	Código
ERROR_EXTERNAL_SERVER_ERROR_GENERIC	10000
ERROR_REMOTE_SERVER_VALIDATION_FAILED	10001
ERROR_OAUTH_TOKEN_INVALID_OR_EXPIRED	10002