

**PROYECTO DE APLICACIÓN MULTIPLATAFORMA: ‘GESTIÓN
DE ESCUELAS DE SURF DESDE UNA APLICACIÓN QUE
ENGLOBE A TODAS LAS ESCUELAS QUE DESEAN UNIRSE’**

NOMBRE DE LA APLICACIÓN: SURFSCHOOLS



Realizado por:

Pablo Ortiz Gutiérrez 2ºDAM

ÍNDICE

1. Introducción

- 1.1. Propósito**
- 1.2. Objetivo**
- 1.3. Temporalización**

2. Análisis

- 2.1. Documentación relevante**
- 2.2. Definiciones**
- 2.3. Requisitos funcionales y no funcionales**
- 2.4. Casos de usos**
- 2.5. Entidad – relación**

3. Diseño

- 3.1. Arquitectura hardware y software**
- 3.2. Diagramas de clases**
- 3.3 Relacional**

4. Pruebas

- 4.1. Base de datos vacía**
- 4.2. Base de datos con datos iniciales**
- 4.2. Inicio sin acceso a internet / servidor apagado**

5. Conclusión

- 5.1. Grado de consecución de los objetivos**
- 5.2. Dificultades encontradas**
- 5.3. Mejoras posibles**

6. Bibliografía

1. INTRODUCCIÓN

1.1. Propósito

El problema planteado y que se me plantea para realizar este proyecto es el siguiente. Son numerosos los surferos y gente amante de los deportes acuáticos que viajan de un lugar a otro en busca de mejores olas, conocer otras playas etc..

El problema que se encuentran muchas veces, es que no tienen ningún lugar en el que puedan ver las escuelas de cada playa, y mucho menos comparar precios, lo que conlleva a que con las prisas que tiene un viaje se acaban decantando por la primera escuela que ven sin tener tiempo para comparar precios.

Otro problema es la dificultad de reservar alquileres, o participaciones a excursiones sin presentarse físicamente en el establecimiento.

Un problema extra que he intentado solucionar, es el de las previsiones de olas y la realidad que se encuentran los surferos. Muchas veces las previsiones fallan y no se cumplen, por lo que he decidí añadir una funcionalidad la cuál es que cada escuela pueda determinar si en la playa en la que se encuentra se puede surfear o no en ese momento, y el cliente de un simple vistazo en las escuelas que sigue puede comprobar si puede ir a esa playa a surfear o no, sin el problema que supone realizar un viaje para luego encontrarse con que las previsiones no se cumplen.

Con este proyecto intento solucionar ese problema, congregando todas las escuelas posibles en una misma aplicación, y de forma que el usuario mediante un mapa pueda ver las escuelas que tiene disponibles, y poder seguirlas, participar en sus excursiones o reservar alquileres sin necesidad de estar físicamente en el establecimiento.

Una parte que podría ampliarse de mi idea sobre el proyecto es la posibilidad de pagar directamente las clases o los alquileres desde la propia aplicación.

1.2. Objetivo

El objetivo de este proyecto es crear 3 programas diferentes. Una aplicación servidor, una aplicación de escritorio y una app móvil que nos permita gestionar los servicios que ofrece una escuela.

En el servidor se encontraría la base de datos y la aplicación encargada de resolver las peticiones de la app y la aplicación de escritorio.

La aplicación de escritorio se encargaría de gestionar la propia escuela:

- Precios de los alquileres
- Gestionar alquileres
- Planear excursiones
- ...

La aplicación móvil estaría dirigida a los clientes:

- Seguir escuelas
- Reservar alquileres
- Participar en excursiones
- ..

El objetivo principal sería crear una plataforma que nos permita reunir a diferentes escuelas de surf a las que el cliente pudiera tener acceso y hacer uso de sus servicios.

1.3. Temporalización

Se incluye dentro de la carpeta de análisis del proyecto, un archivo con todos los sprints realizados durante la realización del proyecto.

2. ANÁLISIS

2.1. Documentación relevante

No ha sido necesaria recabar mucha información sobre el funcionamiento de escuelas de surf y demás, debido a que al practicar deportes acuáticos mis propios conocimientos me han permitido resolver mucha de las dudas que he ido encontrando.

La única documentación que he visto necesaria buscar es respecto a programación de las funcionalidades del proyecto.

He necesitado consultar documentación para resolver mis dudas sobre estos conceptos:

- Parseador XML
- Comunicación con socket desde Android
- Validaciones y comparación de fechas
- ArrayAdapter/ListView en Android

La mayoría de esta documentación ha sido consultada en www.stackoverflow.com y en las guías oficiales de Android.

2.2. Definiciones

JAVA: es un lenguaje de programación de propósito general, concurrente, orientado a objetos que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. Su intención es permitir que los desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo (conocido en inglés como *WORA*, o "*write once, run anywhere*"), lo que quiere decir que el código que es ejecutado en una plataforma no tiene que ser recompilado para correr en otra.

JVM: es una máquina virtual de proceso nativo, es decir, ejecutable en una plataforma específica, capaz de interpretar y ejecutar instrucciones expresadas en un código binario especial (el bytecode Java), el cual es generado por el compilador del lenguaje Java.

IDE: es una aplicación informática que proporciona servicios integrales para facilitarle al desarrollador o programador el desarrollo de software.

Normalmente, un IDE consiste de un editor de código fuente, herramientas de construcción automáticas y un depurador. La mayoría de los IDE tienen auto-completado inteligente de código (*IntelliSense*). Algunos IDE contienen un compilador, un intérprete, o ambos, tales como NetBeans y Eclipse; otros no, tales como SharpDevelop y Lazarus.

NETBEANS IDE: es un entorno de desarrollo integrado libre, hecho principalmente para el lenguaje de programación Java. Existe además un número importante de módulos para extenderlo. NetBeans IDE es un producto libre y gratuito sin restricciones de uso.

MYSQL: es un sistema de gestión de bases de datos relacional desarrollado bajo licencia dual GPL/Licencia comercial por Oracle Corporation y está considerada como la base datos open source más popular del mundo, y una de las más populares en general junto a Oracle y Microsoft SQL Server, sobre todo para entornos de desarrollo web.

ANDROID: es un sistema operativo basado en el núcleo Linux. Fue diseñado principalmente para dispositivos móviles con pantalla táctil, como teléfonos inteligentes, *tablets* y también para relojes inteligentes, televisores y automóviles. Inicialmente fue desarrollado por Android Inc., empresa que Google respaldó económicamente y más tarde, en 2005, compró. Android fue presentado en 2007 junto la fundación del Open Handset Alliance (un consorcio de compañías de hardware, software y telecomunicaciones) para avanzar en los estándares abiertos de los dispositivos móviles.

ANDROID STUDIO: es el entorno de desarrollo integrado oficial para la plataforma Android. Fue anunciado el 16 de mayo de 2013 en la conferencia Google I/O, y reemplazó a Eclipse como el IDE oficial para el desarrollo de aplicaciones para Android. La primera versión estable fue publicada en diciembre de 2014.

CLIENTE/SERVIDOR: es un modelo de aplicación distribuida en el que las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores, y los demandantes, llamados clientes. Un cliente realiza peticiones a otro programa, el servidor, quien le da respuesta. Esta idea también se puede aplicar a programas que se ejecutan sobre una sola computadora, aunque es más ventajosa en un sistema operativo multiusuario distribuido a través de una red de computadoras.

SOCKETS: designa un concepto abstracto por el cual dos programas (posiblemente situados en computadoras distintas) pueden intercambiar cualquier flujo de datos, generalmente de manera fiable y ordenada.

2.3. Requisitos funcionales y no funcionales

2.3.1. Requisitos funcionales

COMUNES

- **Función inicio de sesión**
 - El programa mostrará un formulario donde se deberá introducir el usuario y contraseña, al intentar iniciar sesión se comprobará si son correctos los datos, y si son correcto se mostrará la pantalla principal de la aplicación
- **Función registro**
 - El programa mostrará un formulario donde se pedirán los datos para realizar una nueva alta de usuario, al proceder al registro si los datos no existen ya y son todos correctos se realizará el registro.
- **Función configuración de perfil**
 - El programa mostrará un formulario similar al de registro con los datos actuales del usuario ya definidos, los cuáles el usuario podrá cambiar. En el caso de las escuelas también podrá cambiar los servicios que ofrece.

APLICACIÓN ESCRITORIO

- **Función posibilidad surfear**
 - La escuela puede decidir si en ese momento en la playa en la que se encuentra se puede surfear o no con dos botones que se muestran en la pantalla principal. La posibilidad de surfear en esa escuela nos servirá para mostrarse posteriormente a los clientes que sigan a esa escuela.
- **Función nueva actividad**
 - Mediante un formulario el usuario de la escuela podrá crear una nueva actividad que podrá ser una clase colectiva o una excursión, y que posteriormente podrá programar para alguna fecha informando también del número máximo de participantes.
- **Función programar actividad**
 - En el mismo panel que se muestran las actividades ya creadas, se encuentra la opción de programar cualquiera de ella para una fecha que sea posterior a la actual y incluyendo también un número máximo de participantes.
- **Función suspender actividad**
 - Al suspender una actividad programada se pide al usuario un mensaje y un asunto que será enviado a los participantes de la misma para informar del motivo por el cuál se suspende.

- **Función aprobar alquiler**
 - El usuario de la escuela puede ver en un panel todas las solicitudes que reciba de alquiler, ya sea de surf o de kayak, y puede aprobarlas.
- **Función suspender alquiler**
 - El usuario de la escuela marca como suspendido el alquiler.
- **Función ver notificaciones de tiempo**
 - Cuando las condiciones meteorológicas máximas que hemos definido al crear una excursión sean superadas, podremos acceder al apartado donde observamos las notificaciones por tiempo, donde nos informará de la información meteorológica prevista y la que nosotros indicamos al crear la excursión. Desde esta pantalla también se nos permitirá suspender la excursión o mandar un mensaje a todos los participantes.
- **Función aprobar clase**
 - El usuario puede ver en un panel todas la clases individuales que se le pidan y las puede aprobar desde la propia solicitud.
- **Función suspender clase**
 - El usuario de la escuela marca como suspendida una clase
- **Función enviar mensaje**
 - Esta función forma parte de la de suspender una actividad, ya que necesita enviar un mensaje, también puede enviar mensaje desde el panel en el que se muestran las actividades programadas.

APLICACIÓN MÓVIL

- **Función ver mensajes**
 - El usuario tiene un activity en el que se le muestran todos los mensajes recibidos, destacando aquellos que aún no ha visto
- **Función alquilar tabla/kayak**
 - En el ámbito de una escuela el usuario podrá realizar un alquiler rellenando un formulario, que se validará antes de realizar la petición.
- **Función apuntarse actividad**
 - En el ámbito de una escuela el usuario puede ver las actividades programadas que tiene dicha escuela y apuntarse a ellas.
- **Función cancelar participación en actividad**
 - El usuario tiene un activity en el cuál se muestran todas las actividades en la que participa, desde ahí puede suspender su participación.
- **Función pedir clase**
 - En el ámbito de una escuela el usuario podrá pedir una clase individual en un día determinado rellenando un formulario, que se validará antes de realizar la petición.

- **Función ver clases**
 - El usuario tiene un activity en el cuál se muestran todas las clases que tiene pendiente de aprobar, o pendiente de realizar.
- **Función ver alquileres**
 - El usuario tiene un activity en el cuál se muestran todos los alquileres que tiene pendiente de aprobar, o pendiente de realizar.
- **Función ver actividades**
 - El usuario tiene un activity en el cuál se muestran todas las actividades en las que participa.
- **Función ver mapa**
 - El usuario tiene un activity en el que se muestra un mapa con todas las escuelas registrada en ese momentos, a las que podrá acceder y comenzar a seguir.
- **Función ver mis escuelas**
 - El usuario cuenta con un activity principal en el que se muestran las escuelas que sigue el usuario y que le permite el acceder a ellas.

APLICACIÓN SERVIDOR

- **Función gestionar sockets**
 - La aplicación servidor gestiona la llegada de nuevos sockets, aceptandolos y creando un hilo independiente para cada uno.
- **Función gestionar base de datos**
 - La aplicación servidor cuenta con un protocolo que traduce las peticiones para que una clase que gestiona la base de datos realice las operaciones necesarias.

2.3.2. Requisitos de ejecución

Para ejecutar la aplicación escritorio se necesitará:

- Tener instalado JRE
- Acceso a internet
-

Para ejecutar la aplicación servidor necesitará:

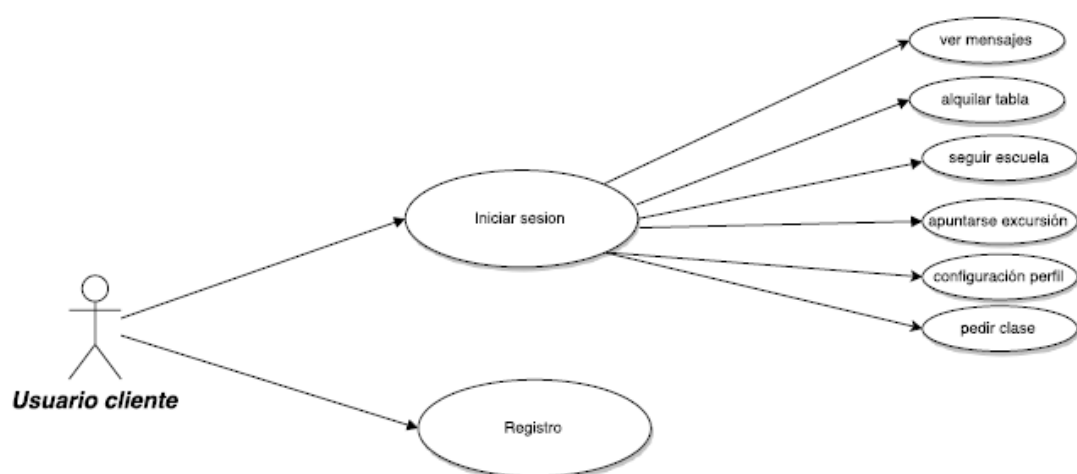
- Tener instalado JRE
- Acceso a internet
- Servidor mysql

Para ejecutar la aplicación móvil necesitará:

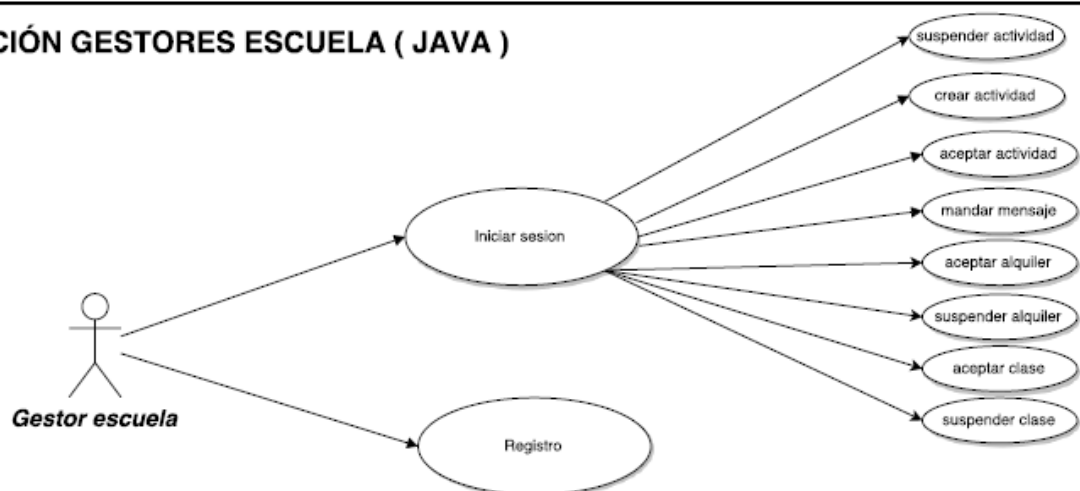
- Acceso a internet

2.4. Casos de uso

APLICACIÓN USUARIOS (ANDROID)

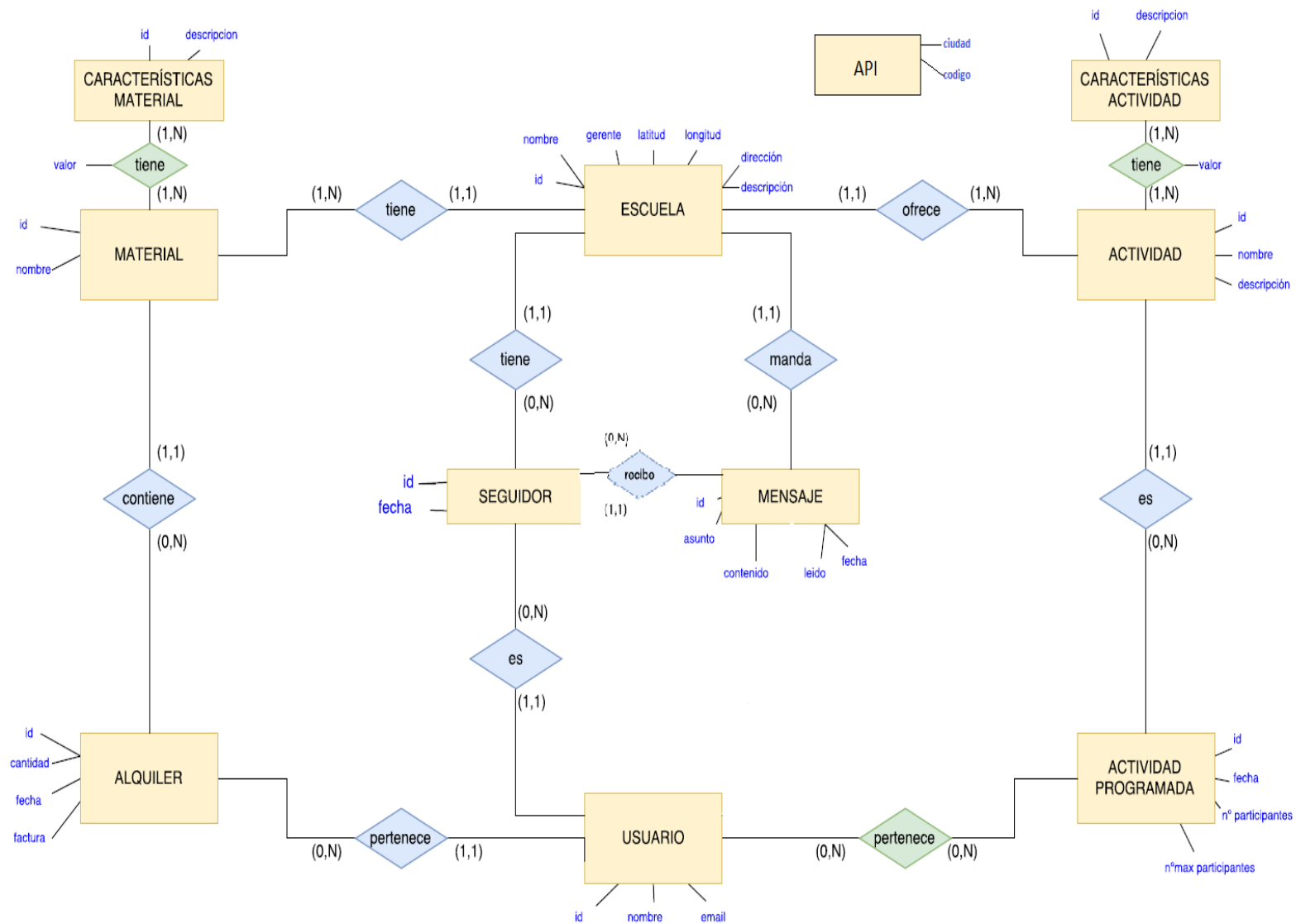


APLICACIÓN GESTORES ESCUELA (JAVA)



2.5. Modelo entidad – relación

ENTIDAD - RELACIÓN PROYECTO SURFSCHOOLS - PABLO ORTIZ



3. DISEÑO

3.1. Arquitectura hardware y software de la solución

El software de la solución se divide en tres aplicaciones, dos aplicaciones java una servidor y otra de escritorio, y una aplicación móvil.


Las librerías externas que se usan son :


- Java JSON
- MySQL JDBC Driver

Las dos aplicaciones que se conectan con el servidor tienen clases apartes que se encargan de la salida y entrada de información, y otra clase que es el protocolo, que es la encargada de construir los JSON que se envían para realizar las peticiones.

3.2. Modelado funcional de la solución. Diagramas de clase

CLASES PRINCIPALES APLICACIÓN ESCRITORIO

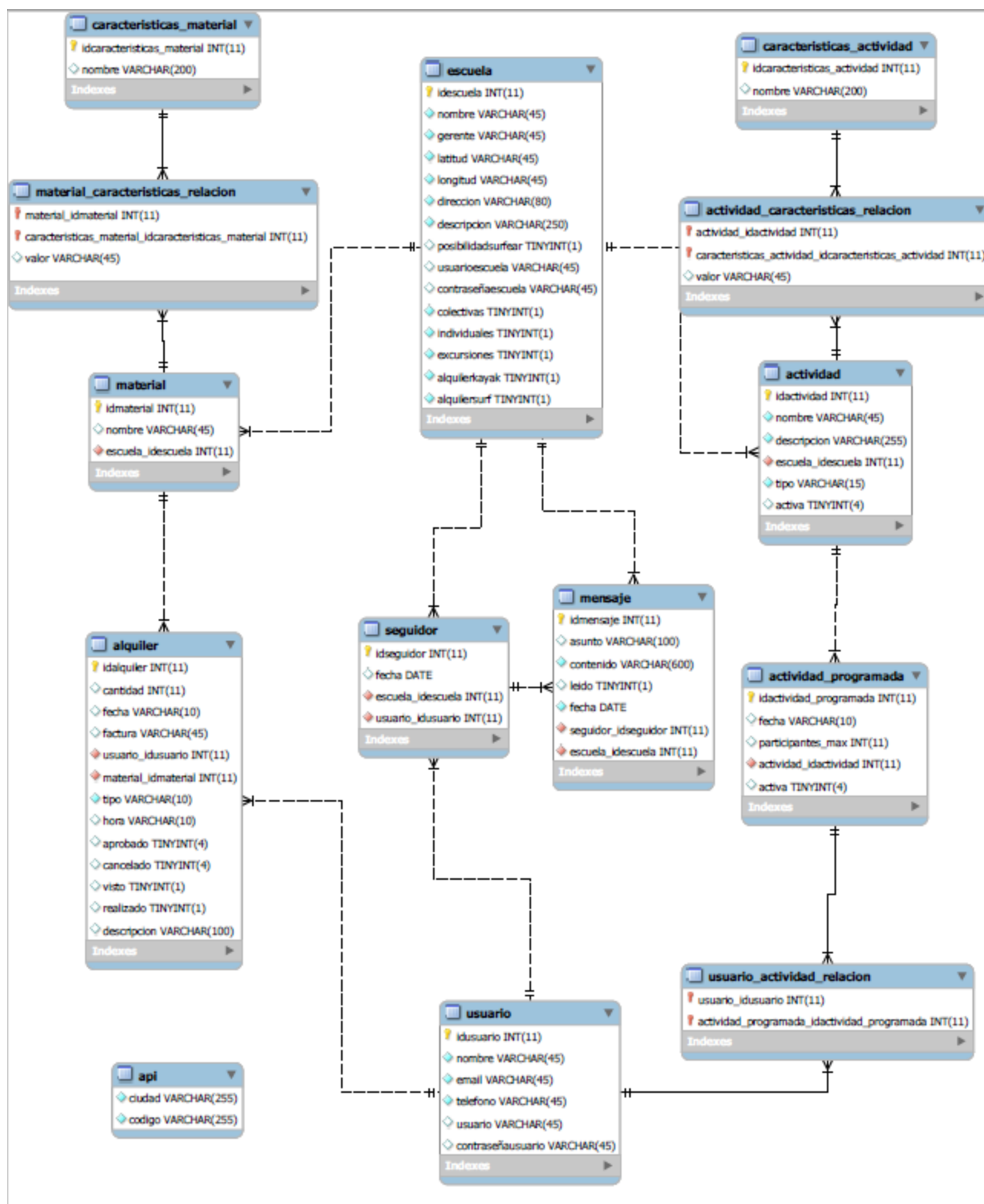
 Client
<i>Attributes</i>
<code>public String ip = "localhost"</code> <code>public int port = 4444</code> <code>public Socket socket</code> <code>public String nombreescuela</code>
<i>Operations</i>
<code>public void main(String args[0..*])</code> <code>public ClientProtocol getClientProtocol()</code> <code>public Socket getClientSocket()</code>

 ClientProtocol
<i>Attributes</i>
<code>package Socket socket</code>
<i>Operations</i>
<code>public ClientProtocol(Socket socket)</code> <code>public JSONObject registro(String nombre, String gerente, String latitud, String longitud, String</code> <code>public JSONObject setAprobado(int idalquiler)</code> <code>public JSONObject getAlertas(String escuela)</code> <code>public JSONObject suspenderActividad(String escuela, int id, String mensaje, String asunto)</code> <code>public JSONObject enviarMsn(String escuela, int id, String mensaje, String asunto)</code> <code>public JSONObject setCancelado(int idalquiler)</code> <code>public JSONObject cancelarActividad(int id)</code> <code>public JSONObject getParticipantes(int id)</code> <code>public JSONObject programarActividad(int idactividad, String fecha, String participantes)</code> <code>public JSONObject getActividadesProgramadas(String nombreescuela, String tipo)</code> <code>public JSONObject newExcursion(String nombre, String descripcion, String vientomax, String</code> <code>public JSONObject newColectiva(String nivel, String descripcion, String precio, String usuario)</code> <code>public JSONObject getExcursiones(String nombreescuela)</code> <code>public JSONObject getColectivas(String nombreescuela)</code> <code>public JSONObject login(String nombre, String contrasena)</code> <code>public JSONObject getPrecios(String usuario)</code> <code>public JSONObject getAlquileres(String usuario, String tipoalquiler)</code> <code>public JSONObject setPosibilidadSurfear(String posibilidad, String usuario)</code> <code>public JSONObject datosEscuela(String nombre)</code> <code>public JSONObject actualiza(String nombre, String gerente, String latitud, String longitud, Stri</code> <code>public JSONObject setPrecios(String usuario, String preciohora, String preciodia, String alquil</code> <code>public void salir()</code>

[illegible]

 MultiServerThread
<p><i>Attributes</i></p> <pre>private Socket socket = null</pre>
<p><i>Operations</i></p> <pre>public MultiServerThread(Socket socket) public void run()</pre>

3.3. Modelado de datos. Modelo relacional. Diccionario de Datos



4. PRUEBAS

4.1. Base de datos vacía

Al iniciar la aplicación con la base de datos completamente vacía no nos dará problema para registrarnos o iniciar sesión, pero si nos da problema cuando queremos cambiar el precio de los alquileres o de las excursiones.

Debido a que la forma en la que está diseñada en la base de datos, hace necesaria la importación de unos datos sql para la perfecta ejecución de la aplicación.

En la importación también se encuentran los datos sobre la api del tiempo que se usan para comprobar la meteorología de cara a las excursiones programadas.

4.2. Base de datos con datos iniciales introducidos.

Al iniciar la base de datos con los datos iniciales importados, el funcionamiento es totalmente correcto, permitiendo tanto el cambio de precio de alquileres y excursiones, como el correcto funcionamiento del resto de las funcionalidades.

4.3. Inicio de aplicación sin acceso a internet / servidor apagado

Cuando se intenta arrancar la aplicación sin tener acceso a internet, o estando el servidor apagado se informa al usuario mediante un dialog de que no se ha podido conectar al servidor. Si esto pasa mientras se ejecuta la aplicación aparece un mensaje de error en un dialog y se cierra la aplicación.

5. CONCLUSIONES

5.1. Grado de consecución de los objetivos inicialmente planteados

La mayoría de objetivos han sido realizados con éxitos pero algunas funcionalidades no ha dado tiempo a ser implementadas y se incluirán dentro de las propuestas de mejora y posibles ampliaciones.

Los objetivos han sido realizados a un 90% sin dejar ninguna funcionalidad básica del proyecto sin realizar

5.2. Dificultades encontradas

Las dificultades encontradas han sido:

- Uso de sockets en Android:
 - El problema que encontré es que para enviar y recibir datos de un socket en Android debes hacerlo obligatoriamente en un AsyncTask.
- Campos insuficientes en la BD
 - Al ir implementando las funcionalidades he ido viendo que el diseño de la BD no estaba realizado del todo correctamente, y he tenido que añadir varios campos a algunas tablas mientras se realizaba el proyecto
- ArrayAdapter-ListView:
 - Una de las cosas que más tiempo me ha llevado es realizar la funcionalidad que tienen el ArrayAdapter-ListView para una aplicación escritorio, ya que estas clases solo se encuentran disponibles para Android. La solución final fue realizada con paneles con layout relativos los cuales se colocan en lista en otros paneles también con layout relativos.
- Obtención datos de la API:
 - Para obtener datos de la API, anteriormente yo lo había realizado desde una aplicación android, a la hora de realizarlo para una aplicación de escritorio el parseador XML que usaba en Android no se encontraba disponible, y tuve que cambiar la forma de parsearlo con otra librería.

5.3. Propuestas de mejora y posibles ampliaciones

- Agregar un sistema de puntos por el cuál el cliente vaya ganando puntos según realiza alquileres o participa en excursiones que posteriormente puedan ser canjeados por descuentos en alquileres, o incluso alquileres gratis dependiendo el número de puntos.
- Poder añadir nuevo material para alquilar o otro tipo de actividades:
 - Esta posible mejora ya tiene implementada su parte en BD. La BD está diseñada de manera que se puedan agregar nuevas actividades y materiales ya que las características de los mismos se encuentran en tablas separadas, pero no se encuentra implementado en código.
- Actualizaciones automáticas:
 - En la versión que se encuentra actualmente el proyecto, has de recargar el activity, o la clase para que se actualicen los datos. Una mejora sería mejorar el sistema de peticiones y comunicación con el servidor para que cada cierto tiempo se actualizara automáticamente.
- Mejora en el diseño:
 - Mejoraría el diseño y imagen, sobretodo de la aplicación móvil, ya que debido al tiempo, el diseño a sido a lo que menos tiempo le he dedicado.

- Incluir la posibilidad de insercción de escuelas internaciones.
 - Ahora mismo la aplicación solo permite escuelas dentro del territorio nacional debido a que la información de la API del tiempo sólo permite España. También habría que hacer la aplicación multilenguaje.
- Utilizar DatePicker en Android
 - A la hora de reservar los alquileres o las clases individuales, en vez de tener que introducir a mano la fecha, usar un DatePicker

6. BIBLIOGRAFÍA Y RECURSOS ON-LINE

- Numerosas páginas de www.stackoverflow.com
- Cursos android de www.udacity.com
- www.lawebdelprogramador.com
- <https://developer.android.com/index.html>
- Tutoriales de www.youtube.com