MANUAL TÉCNICO DE LAS APLICACIONES

Realizado por:

Pablo Ortiz Gutiérrez

NOTAS

Se recomienda la lectura de este documento con el código fuente a mano.

APLICACIÓN DE ESCRITORIO

- PAQUETE CLIENTE

En este paquete se incluye la *Clase Cliente*.
La clase cliente es la primera que se ejecuta cuando se inicia la aplicación de escritorio. Es la encargada de desplegar el login, y intentarse conectar al servidor.

Tiene dos métodos estáticos que le sirven a clases del resto de la aplicación para obtener el objeto protocolo del cliente, o el propio socket del cliente.

- PAQUETE GENERAL

Este paquete incluye la clase *General*.
La clase General tiene dos métodos, encargados de devolver la salida/entrada del socket.

- PAQUETE INTERFAZ

Este paquete incluye todas las clases que tienen componentes visuales.
En este paquete se engloba prácticamente toda la aplicación.

La clase *PrincipalFrame* hace referencia al Jframe que se usa para todos los paneles.

La clase *Principal* es la pantalla principal de la aplicación. En ella se muestran los accesos a los diferentes servicios que ofrece la aplicación y a la configuración del perfil. Al acceder a ella se realiza una petición que nos da los servicios que ofrece

para saber a qué servicio debemos poder dejar acceder al usuario, y otra petición que nos devuelve si hay alguna notificación respecto del tiempo para las excursiones, si hay alguna notificación, se muestra en el label de notificaciones tiempo y se nos permite acceder a ellas (Clase *Notificaciones*).

El funcionamiento de todas estas clases de interfaz es muy parecido.

- Al iniciarse, si necesitan obtener unos datos básicos para funcionar (Por ejemplo, los precios en el servicio de alquiler, las excursiones programadas en las excursiones) se realiza una petición a la BD y se invoca al método *colocaValores()* que es el encargado de colocar los valores en su lugar.
- Cada vez que se necesita hacer una petición, se recupera el objeto *ClientProtocol* del método estático de la clase *Client* y se llama al método que necesitemos de *ClientProtocol*.
- Al método que necesitemos de *ClientProtocol* se le pasan los campos necesarios para realizar la petición. El método de *ClientProtocol* será el encargado de construir el JSON con la petición y los campos necesarios.
- Una vez construido el mismo método de *ClientProtocol* es el encargado de utilizando la clase *IO* invocar a su método *enviar*, para enviar el JSON construido, y invocar seguidamente al método *recibir*, que es el encargado de leer el JSON que nos devuelve el servidor y que a su vez el método que hayamos invocado de *ClientProtocol* devuelve.
- Dependiendo de si la petición es para reservar una clase, suspender una actividad, interpretaremos el resultado que nos da el servidor de manera que despleguemos un *Dialog* que informa del resultado de la operación, o si en cambio es una petición para obtener las excursiones pendientes, alquileres aprobados.. el resultado se utilizará para visualizarlos.

La clase *AlquilerSurf* puede cambiar los precios utilizando los textField. Una vez pulsado el botón guardar se mandará una petición al servidor para que los actualice.
Al iniciarse en el panel que está a la derecha se muestran los alquileres aprobados con la información del alquiler y del usuario que lo ha alquilado. De forma parecida a la que lo haría un ArrayAdapter+ListView utilizando un panel genérico para cada alquiler (*PanelSolicitud)*, que se construye y se añade a un panel general donde se colocan todos los alquileres.
Si a la hora de hacer la petición se observa que hay alguno nuevo se activa un label con una campana que una vez hagamos click sobre el, cambiará los alquileres aprobados por los alquileres pendientes de aprobar.

Este funcionamiento es el mismo en todos los paneles que muestran solicitudes ya sea de alquiler de surf, kayak o clases individuales.

La clase *ConfiguracionPerfil* obtiene los datos del cliente al cargarse y los coloca en textField con un Button al lado de cada uno, al pulsar en el botón de algún textField, se pide un nuevo valor para ese campo, una vez introducido se valida que no esté vacío, y si no está vacío se guarda ese valor, una vez pulsemos el botón de guardar se actualizarán todos los datos cambiados realizando una petición al servidor mediante la clase *ClientProtocol.*

La clase *LimitedStyledDocument* es utilizada en los TextArea del registro, y en la descripción de las excursiones y las clases colectivas. Esta clase nos permite limitar el número de caracteres del TextArea a la cantidad que le pasemos mediante su constructor.

El funcionamiento de las clases *ExcursionPrincipal* y *ClasesColectivas*, es muy similar y funcionan de manera parecida a la de Alquileres. Al iniciarse se obtienen las excursiones o clases que tiene esa escuela, y se despliegan las creadas en el panel de la derecha, de la misma manera que se realiza en los alquileres, utilizando una clase Jpanel que representará a cada excursión o cada clase. Y mediante un jLabel que es un icono que se muestra en la parte superior derecha, nos permite cambiar entre las aprobadas y las programadas.

Cada Jpanel(*PanelExcursion)* que representa a una excursión o clase contiene dos botones, uno para cancelar esa actividad, y otro para programarla. Si pulsamos en programarla nos pide una fecha y un numero máximo de participantes, estos campos se validan que no estén vacíos y que la fecha no sea anterior a la actual, si la validación es correcta se procede a realizar una petición al servidor que guarde esa actividad programada.

El panel que se muestra en las excursiones o clases colectivas programadas es diferente. Cada panel de las programadas (*PanelExcProgramada)* muestra la fecha, el nombre de la actividad, el número maximo de participantes y número actual. Cuenta con un botón (*verParticipantes*) que nos mostrará en un nuevo Jframe el nombre y teléfono de los participantes que haya en esa actividad, otro botón (*Suspender)* , que al pulsarlo se nos pide un asunto y mensaje, que se enviará a cada participante para comunicar por que se ha suspendido y se suspenderá la actividad, es decir no se volverá a mostrar, y otro botón (*Mensaje),* que nos permite enviar un mensaje a cada uno de los participantes por si queremos comunicarles algo. Todo esto se realiza

haciendo peticiones al servidor de la manera que he explicado anteriormente utilizando *ClientProtocol*.

La clase *Notificaciones*, es parecida a la de excursiones o clases colectivas pero simplemente se muestra un panel que de la manera que hemos realizado antes el ArrayAdapter+ListView se muestran las notificaciones pendientes, cada *PanelNotificacion* nos permite suspender la actividad a la que corresponda pidiéndonos antes un mensaje y asunto para enviar a cada participante, o la opción de enviar un mensaje a todos los participantes.

- PAQUETE PROTOCOLO

Este paquete incluye la clase *ClientProtocol* . Esta clase contiene los métodos que utilizan todas las clases para comunicarse con el servidor. Cada método recibe los campos necesarios para realizar la petición, construye el objeto JSON que va a enviar, y utilizando la clase *IO* envía el JSON. Y seguidamente recibe lo que le devuelve el servidor, que es lo que cada método devuelve a la clase que haya llamado al método.

Ejemplo:

```
 public JSONObject registro(String nombre, String gerente, String latitud, String longitud, String direccion, String descripcion, String usuario, String contrasena, boolean colectivas, boolean individuales, boolean excursiones, boolean alquilerkayak, boolean alquilersurf ){

        try {
            JSONObject jregistro = new JSONObject();
            jregistro.put("peticion", "registro-escuela");
            jregistro.put("nombre", nombre);
            jregistro.put("gerente", gerente);
            jregistro.put("latitud", latitud);
            jregistro.put("longitud", longitud);
            jregistro.put("direccion", direccion);
            jregistro.put("descripcion", descripcion);
            jregistro.put("usuarioescuela", usuario);
            jregistro.put("contrasena", contrasena);
            jregistro.put("colectivas", colectivas);
            jregistro.put("individuales", individuales);
            jregistro.put("excursiones", excursiones);
            jregistro.put("alquilerkayak",alquilerkayak);
            jregistro.put("alquilersurf",alquilersurf);

            io.enviar(jregistro);

        } catch (JSONException ex) {
            Logger.getLogger(ClientProtocol.class.getName()).log(Level.SEVERE, null, ex);
        }
        return io.recibir();
    }
```

- PAQUETE SALIDA

Este paquete contiene la clase *IO* la cuál es la encargada de enviar y recibir del socket. Esta clase tiene dos métodos:

- enviar
- recibir

Utilizando los métodos de la clase *General* (explicada anteriormente) , de los que obtiene las tuberías de entrada y salida, escribe o lee de la tubería y lo devuelve a la clase que haya invocado al método, que siempre es *ClientProtocol*.

APLICACIÓN SERVIDOR

- PAQUETE SERVER

La clase *MultiServer* es la que se ejecuta cuando se inicia el servidor. Esta clase crea un serverSocket y entra en un bucle en el cuál toda petición que le llegue se acepta y se crea un Thread de la clase *MultiServerThread.*

Se crea una hilo *MultiServerThread* por cada socket que llegue, esta clase es la encargada de leer y escribir en los flujos de entrada y salida.

El funcionamiento de *MultiServerThread* consiste en un bucle que cuando haya algo en la tubería de entrada lee, lo transforma en JSON y lo envía a *Protocol*. *Protocol* le devuelve un JSON con lo que tenga que devolver al cliente y *MultiServerThread* lo escribe en la tubería de salida.

La clase *Protocol*, recibe el JSON que llega desde el cliente, este JSON siempre contiene un valor que es “peticion” donde se encuentra el nombre de la petición, se realiza un case según este valor, cuando una petición entra en su case correspondiente, esta obtiene del JSON los valores necesarios, y llama al método de la clase *GestionaBD* correspondiente para cada petición y esta le devuelve lo que tiene que mandar que a su vez es devuelto de *Protocol* a *MultiServerThread.*

Ejemplo de case en *Protocol*:

```
case "login-escuela":
		nombreescuela = theInput.getString("nombre");
		contraescuela = theInput.getString("contrasena");
		output = bd.loginescuela(nombreescuela, contraescuela);

		break;
```

Por cada cliente se crea una instancia de la clase *GestionaBD*, esta clase contiene la conexión y los métodos que interactúan directamente con la BD. Los métodos de esta clase son llamados desde Protocol, y en los propios métodos es donde se construye el JSON que se le devolverá al cliente.

Ejemplo de método en *GestionaBD:*

```
JSONObject respuesta = null;
  try {

    respuesta = new JSONObject();
    respuesta.put("resultado", "incorrecto");
    String query = "SELECT * FROM escuela WHERE usuarioescuela = ? AND contrasenaescuela = ?";
    try {
      PreparedStatement preparedStatement = connection.prepareStatement(query);
      preparedStatement.setString(1, nombre);
      preparedStatement.setString(2, contrasena);

      ResultSet rs = preparedStatement.executeQuery();
      if (!rs.next()) {
        respuesta = new JSONObject();
        respuesta.put("resultado", "incorrecto");
      } else {
        respuesta = new JSONObject();
        respuesta.put("resultado", "correcto");
      }
    } catch (SQLException ex) {
      Logger.getLogger(GestionaBD.class.getName()).log(Level.SEVERE, null, ex);
    }

  } catch (JSONException ex) {
    Logger.getLogger(GestionaBD.class.getName()).log(Level.SEVERE, null, ex);
  }
  return respuesta;
}
```

La clase *CityName* es la encargada de cuando se tienen que obtener las notificaciones de tiempo, pasándole una dirección nos devuelve el nombre de la localidad en la que se encuentra haciendo una consulta a la api de google.

La clase *Meteo* también se utiliza cuando se obtienen las notificaciones de tiempo, y utiliza el nombre que nos proporciona la clase *CityName* para consultar una api del tiempo y obtener los datos meteorológicos necesarios de esa ciudad.
Para obtener la información meteorológica contamos con una tabla en la BD llamada “API” en la cuál se guardan nombre de localidad y código de la api. Esta tabla es consultada con el nombre que nos da *CityName* para obtener el código a utilizar en *Meteo.*

APLICACIÓN MÓVIL

- PAQUETE ACTIVITY

En este paquete se encuentran todos los activitys de la aplicación

El primer activity en ejecutarse es *SplashScreenActivity* , este activity se ejecuta al inicio y muestra una pantalla durante 5segundos para seguidamente dar la ejecución a *LoginActivity* este activity se encarga de realizar el Login.

En este mismo activity de *LoginActivity* se puede acceder al registro (*RegistroActivity*) mediante un intent al pulsar un botón o a cambiar el servidor usando el menú superior que también nos llevará al activity *CambioServidor*.

Los activity *AlquilerKayak* y *AlquilerSurf* son muy parecidos. Al iniciarse hacen una petición al servidor el cuál les devuelve los precios definidos por la escuela, si los precios aún no han sido definidos no permite al usuario realizar el alquiler. Si los precios si están definidos los coloca en dos label. Estos activity permiten al usuario introducir datos tales como son la cantidad, la fecha y si se van a realizar por horas, la hora. Con el escuchador que tiene en el editText de la cantidad, va actualizando el valor total que corresponde al precio que tendrá que pagar el cliente. Cuando el usuario pulsa alquilar, se validan los campos, y si los datos son correctos se procede a realizar una petición al servidor para que apunte ese alquiler.

El activity *ClasesIndActivity* es muy parecido a los dos anteriores, realiza la misma acción cuando se ejecutan (coger los precios del servidor) , pero en este caso simplemente se pide fecha, cantidad de horas y si quiere añadirlo una nota al respecto de la reserva. Cuando pulsa en reservar se validan los datos y si son correctos se apunta la reserva.

Los 3 activitys anteriores se ejecutan en el ámbito de una escuela, es decir, sólo se accede a ellos desde la propia interfaz de una escuela que haya elegido el usuario.

NOTA: Todas las fechas que se introducen en la aplicación son válidadas para no permitir la introducción de fechas anteriores a la actual a la hora de reservar o alquilar algún material.

El activity *Alquileres* nos muestra todos los alquileres activos que tiene el usuario independientemente de la escuela. Este activity obtiene del servidor, todos los

alquileres que están activo para ese usuario, y los despliega en pantalla con ayuda de un ArrayAdapter+ListView, dependiendo de si el alquiler esta pendiente de aprobación o pendiente de realización se nos permitirá cancelar esa petición, o marcarla como realizada.

El activity *ClasesIndPartActivity* nos muestra las solicitudes que tenemos activas sobre clases individuales, ya estén pendientes de aprobar o pendientes de realizar. Este activity recupera mediante un JSONArray, las peticiones que ha realizado ese usuario independientemente de la escuela, y con ayuda de un ArrayAdapter-ListView las muestra en la pantalla. Se nos permite marcar como realizado o cancelar las solicitudes.

Los activity *ColectivasEscuelaActivity* y *ExcursionesActivity* son muy parecidas. Su función es mostrar en un ArrayAdapter+ListView , el cuál rellenan con un ArrayList el cuál incluye las excursiones o clases colectivas programadas por la escuela y a las que el usuario no se ha apuntado aún, mediante una petición al servidor. La única función que se nos permite realizar es apuntarnos a la excursión que se nos muestre. Estos activitys se ejecutan en el ámbito de una escuela.

El activity *ConfiguraciónPerfil* nos permite modificar nuestros datos (nombre, telefono, contraseña y correo), se muestran unos checkbox al lado de cada editText que contiene ya nuestros datos actuales que se obtienen mediante una petición al servidor, al pinchar en el checkbox se nos permite cambiar el valor correspondiente. Hasta que no pulsemos actualizar no se actualizarán los datos en la base de datos, previa validación.

El activity *MapsActivity* nos cargará un mapa de GoogleMaps, en este activity se recuperarán del servidor un array con todas las escuelas que se encuentran registradas y su latitud y longitud, se pintará un marker en cada escuela por cada escuela, y se configura un escuchador para que cuando se pulse en él, se haga un intent al activity *DatosEscuela* que nos mostrará los datos principales de esa escuela (nombre, dirección y descripción) y la opción de poderla seguir.

El activity *MisEscuelas* es el interfaz principal de la aplicación, en el se muestran todas las escuelas a las que sigue el usuario mediante un ArrayAdapter+ListView que es recuperado en forma de JSONArray del servidor. Este activity cuenta con un menú superior el cuál nos permite movernos por todas las funcionalidades de la aplicación. Al pulsar en una de las escuelas, se realiza un intent a *EscuelaUsuario*, desde este activity tendremos accesos a los diferentes servicios que ofrezca la escuela en la que hemos pulsado. Desde este activity podremos acceder a reservar alquileres, clases

individuales o ver las clases colectivas o excursiones que tiene programada esa escuela. Este activity tambien tiene la opción de dejar de seguir a esa escuela, pulsando un botón (*UNFOLLOW)* que mandará una petición al servidor para que nos borré como seguidor de esa escuela.

El activity *MensajesActivity* nos muestra mediante un ArrayAdapter+ListView, los mensajes que ha recibido el cliente, su asunto y el nombre de la escuela de la cuál ha sido recibido. En primer lugar se muestran los que el usuario aún no ha marcado como visto. Esto se realiza con un order by a la hora de realizar la consulta en la base de datos (Servidor) y nos devuelve un JSONArray que es el que utiliza el ArrayAdapter, en el cuál los elementos superiores serán los mensajes aún no marcados como visto.

- PAQUETE ADAPTERS

Sobre este paquete simplemente voy a explicar el funcionamiento de los Adapters, ya que todos son prácticamente iguales.

Un adapter es el encargado de colocar los atributos de un objeto en un layout predefinido, de forma que al recibir varios objetos, puedan ser colocado en una lista descendente todos los layout predefinidos que corresponderían cada uno a un objeto.

Tenemos un layout predefinido para cada elemento, que es el que utiliza las clases que heredan del ArrayAdapter, y también tenemos en el activity en el cuál vayan a ser mostrados un layout con un ListView donde serán desplegados los elementos.

PAQUETE COM.PABLO.SURFSCHOOLS

Este paquete contiene las clases generales de la aplicación y el activity *MapsActivity* debido a que no me permitía moverlo de esta ubicación.

La clase *Alquiler* hace referencia a un alquiler, y será la clase utilizada por *AlquilerAdapter* para obtener los atributos de cada Alquiler.

La clase *Colectiva* hace referencia a una clase colectiva, y será la clase utilizada por Colectiva*Adapter y ColectivaPartAdapter* para obtener los atributos de cada clase colectiva.

La clase *Escuela* hace referencia a una escuela y será la clase utilizada por Escuela*Adapter* para obtener los atributos de cada escuela a la hora de mostrarlo en *Mis Escuelas*.

La clase *Excursion* hace referencia a una excursión, y será la clase utilizada por *ExcursionAdapter y ExcursionPartAdapter* para obtener los atributos de cada excursion.

La clase *Individual* hace referencia a una clase individual, y será la clase utilizada a la hora de realizar el adapter que muestre todas las clases individuales que tiene activa el usuario en el activity *ClasesIndvPartActivity.*

La clase *Mensaje* hace refencia a un mensaje, y será la clase utilizada por *MensajeAdapter* para mostrar todos los mensajes que tiene el usuario en el activity *MensajesActivity.*

- PAQUETE CONEXION

En este paquete se muestran las clases que utilizamos para conectarnos con el servidor.

La clase *Client* es la primera que se ejecuta para establecer la conexión con el socket teniendo su IP y PUERTO. Se tiene un *ClientProtocol* que es igual que el de la aplicación escritorio, traduce las peticiones que realiza la aplicación, en JSON que son mandados al servidor.

Las clases *Input* y *Output* son AsyncTask y son los encargados de enviar o recibir del socket del servidor, android obliga a hacer estas operaciones en AsyncTask, el Asynctask *Input* se encarga de recibir el JSON que mande el servidor y el AysncTask *Output*se encarga de mandar al servidor las peticiones que construye el *ClientProtocol*.

La clase *General* es usada por las clases *Input* y *Output* para obtener las tuberías necesarias del socket del servidor. En esta clase también se guardan variables estáticas que nos siven durante la ejecución como por ejemplo el nombre de usuario que ha hecho login.