

Este es un documento explicativo del software RentSoft para su presentación en el IES Augustóbriga como trabajo final de grado.

# RentSoft

## Documentación

Iván Moreno Quirós

---

## TABLA DE CONTENIDO

DOCUMENTACIÓN RENTSOFT .....	3
1. IDENTIFICACIÓN DEL PROYECTO. ....	3
2. ANÁLISIS DE REQUISITOS. ....	3
2.1. OBJETIVOS.....	3
2.2. TECNOLOGÍAS UTILIZADAS.....	3
3. DISEÑO DE LA APLICACIÓN .....	4
3.1. DISEÑO DE LA BASE DE DATOS .....	4
3.1.1. PISTAS .....	4
3.1.2. USUARIOS .....	5
3.1.3. ALQUILER .....	6
3.2. DISEÑO DE LAS APLICACIONES .....	7
3.2.1. ADMINISTRADOR/ESCRITORIO .....	7
3.2.2. CLIENTE/ANDROID .....	15
3.2.3. SERVIDOR .....	19
3.2.4. MODELO .....	20
4. MANUALES.....	20
4.1. INICIO DE SESIÓN / LOGIN.....	20
4.2. PANTALLA DE GESTION DE USUARIOS. ....	21
4.2.1. INSERTAR .....	23
4.2.2. MODIFICAR .....	25
4.2.3. BORRAR .....	26
4.3. RESERVAR PISTA.....	26
4.4. ANDROID.....	28
5. CONCLUSIONES .....	32
6. PROPUESTA DE AMPLIACIÓN.....	32
6.1. INTERACTIVIDAD .....	32
6.2. AMBICIÓN .....	32
6.3. CONTROL .....	33
6.4. SEGURIDAD .....	33
7. BIBLIOGRAFÍA.....	33
8. ANEXOS.....	34
8.1. MODELOS .....	34

8.1.1. PISTAS .....	34
8.1.2. ALQUILER .....	35
8.1.3. USUARIOS .....	36
8.1.4. DETALLES .....	36

### 1. IDENTIFICACIÓN DEL PROYECTO.

Iván Moreno Quirós.

Desarrollo de aplicaciones multiplataformas (DAM).

IES Augustóbriga, Navalmoral de la mata, Cáceres, Extremadura.

Proyecto de renta de pistas deportivas para una entidad deportiva por la falta de información de los clientes que las reservan. Dado que muchos reservan y no asisten a la reserva, con este sistema se sabrá qué cliente ha reservado y cuándo, y así se podrán tomar medidas al respecto.

### 2. ANÁLISIS DE REQUISITOS.

---

#### 2.1.OBJETIVOS.

La aplicación tiene 3 partes diferentes:

- Por un lado, tiene un servidor con una base de datos que podrá subirse a cualquier dominio para acceder a él a través de una IP.
- Por otro lado, estará un cliente diseñado para labores administrativas dentro del servicio de atención al cliente, donde se podrán añadir, modificar y borrar nuevos clientes, así como reservar alguna pista en caso de que sea necesario.
- Y, por último, un cliente diseñado para terminales Android para poder reservar la pista una vez se esté inscrito en el club deportivo que corresponda.

Para reservar la pista se tendrá un máximo de dos días contando con el presente, o lo que es lo mismo: solo se podrá alquilar una pista el día en el que estés y el día siguiente a ese día. También se tendrá en cuenta que se alquilará desde que abra la entidad deportiva hasta la hora de su cierre y se dividirá por horas.

---

#### 2.2.TECNOLOGÍAS UTILIZADAS.

- Un servidor en Java hecho con socket atendiendo peticiones de cualquier cliente que entre a esa IP hecho con Eclipse y sin ninguna interfaz gráfica.
- Una base de datos en MySQL manejando el lenguaje SQL.

- Un cliente administrador hecho con Java con una interfaz gráfica hecha en NetBeans con la librería swing y una añadida llamada edisonCorX y la ayuda gráfica hecha con JavaHelp.
- También incluirá un proyecto Android para el cliente hecho con xml para la interfaz y con Java para los controladores en el IDE de Android Studio.
- Todo el proyecto se puede ver desde Git-Hub en la cuenta de @Ivii95 ya que he utilizado repositorios Git para el versionado de este proyecto.

### 3. DISEÑO DE LA APLICACIÓN

El diseño de esta aplicación está basado en tres tablas de la base de datos, ya que lo que resulta más práctico a la hora de guardar la información es hacer una buena relación entre las tablas.

#### 3.1.DISEÑO DE LA BASE DE DATOS

Para el diseño de la base de datos he preferido que no sea muy amplia y tenga una muy buena relación para que a la hora de guardar los datos sea lo más breve posible.

La base de datos está basada en una sola relación, puesto que no se necesita más para la realización de este proyecto.

Existe una relación principal: la de pistas (1: N ya que una pista puede tener varios usuarios, pero los usuarios solo pueden estar en una pista) con usuarios (1: N ya que un usuario solo puede estar en varias pistas, pero en distinta hora, pero a la misma hora no puede estar en dos pistas diferentes).

De estas dos relaciones se obtiene la relación N:N, que forma una nueva tabla a la que llamaremos “alquiler” y que servirá para distinguir las horas y los días que los usuarios usan las pistas.

##### 3.1.1. PISTAS

Como podemos comprobar, tenemos una tabla “pistas” que incluirá todas las que tengamos en esa institución o entidad deportiva. Esta tabla incluye:

- “id\_pista”: este dato servirá para distinguir todas las pistas que haya en la base de datos definidas.

- “num”: número que se refiere al número de pista que le quiera dar la institución o entidad deportiva a su pista por si ellos tienen una enumeración propia para las mismas.
- y por último tenemos el “tipo” de la pista que queremos guardar, que se guardará como una cadena de letras para así poder asignarle cualquier tipo de pista que quiera poner el administrador. En este programa trabajaremos pensando en 3 pistas estáticas de pádel.

---

### 3.1.2. USUARIOS

En esta tabla tendremos todos los registros para tener la información primordial de un cliente de la empresa donde se contrate este producto. Estos datos pueden adaptarse según las necesidades de cada institución o entidad deportiva.

Las más importantes serán nombradas a continuación y se podrán ver en la imagen.

- “id\_usu” para distinguir imparcialmente a cada usuario y que no se repitan. Este dato será un número, que será único y auto incrementable, y no hace falta que nadie lo introduzca.
- “nom\_usu” este dato definirá el nombre de usuario para acceder a la aplicación tanto móvil como administrador en el escritorio.
- “pass” este será el dato para la contraseña del usuario anteriormente nombrado.
- “admin” será el dato que guardara un 0 o un 1 para saber si es administrador y si puede acceder a la aplicación de escritorio.
- “correo” será el dato que guarde el correo electrónico del usuario.
- “nombre” será el dato que guarde el nombre del usuario guardado.
- “apellidos” será el dato que guarde los apellidos del usuario guardado.
- “tlf” será el dato que guarde el teléfono del usuario guardado.
- “sexo” será el dato que guarde si el usuario se identifica como hombre, mujer u otra sexualidad.
- “fech\_nac” será el dato que guarde el día de su nacimiento.
- “pais” será el dato que guarde el país en el que vive.

- “comunidad\_auto” será el dato que guarde la comunidad autónoma del usuario guardado.
- “provincia” será el dato que guarde la provincia a la que pertenece el usuario guardado.
- “ciudad” será el dato que guarde la ciudad del usuario.
- “domicilio” será el dato que guarde la dirección de la vivienda del usuario guardado.

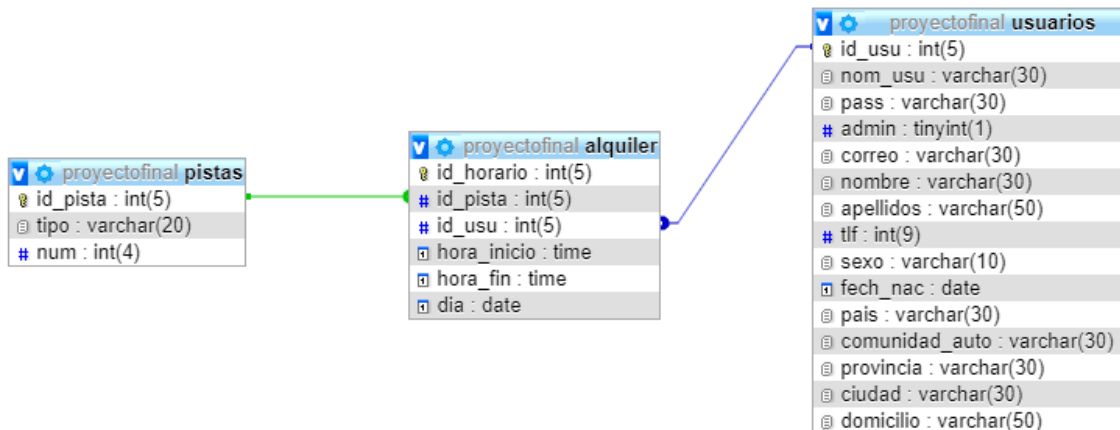
---

### 3.1.3. ALQUILER

Esta tabla se crea a partir de la relación anteriormente explicada: un usuario puede tener varias pistas y una pista puede tener varios usuarios el mismo día. Así surge esta tercera tabla con la intención de mezclar estas dos y, puesto que ya está creada, la aprovechamos para poner los horarios de las pistas a las que los usuarios pueden acceder para alquilarla. Esta tabla contiene los siguientes datos:

- “id\_horario” este dato guardara un id irrepetible para cada fila de la tabla.
- “id\_pista” este dato es una referencia directa al “id\_pista” de la tabla de pistas para que cuando se cree un alquiler se guarde una referencia de la pista donde se está haciendo ese alquiler.
- “id\_usu” este dato es una referencia directa al dato “id\_usu” de tabla usuarios para que cuando se cree un alquiler se sepa qué usuario está realizando este alquiler.
- “hora\_inicio” este dato guarda un tipo llamado time que marca la hora a la cual empieza la reserva.
- “hora\_fin” este dato guarda un tipo llamado “time” que marca la hora a la cual termina la reserva.
- “dia” este dato guarda el día en el que se realiza la reserva.

En esta foto se representan todas las tablas y lo explicado anteriormente.



**Fig.1 Modelo E/R de la base de datos.**

### 3.2.DISEÑO DE LAS APLICACIONES

Para empezar, mostraré los diferentes diagramas UML para que se vea bien el funcionamiento de la aplicación.

Antes de nada, debo señalar que el modelo de la aplicación se hace aparte, ya que todos los puntos explicados más adelante tienen el mismo sistema de modelos, es decir, tienen las mismas clases para que no falle en el intercambio de archivos.

#### 3.2.1. ADMINISTRADOR/ESCRITORIO

Este programa está hecho en Java y será multiplataforma, ya que con la clase “File” de java tenemos una herramienta muy potente en uno de sus métodos llamado “separator” que hará que los “slash” sean admitidos para el sistema operativo donde esté corriendo este programa. Esto, sumado al uso de rutas relativas en todos los procesos, hace que al unir todo en un mismo proyecto el programa funcione en cualquier plataforma en la que se lance.

Podemos ver que este programa está pensado para estar en la oficina o incluso al lado de las pistas donde se quiere reservar para que cualquier persona que esté cerca, siempre y cuando haya alguien del personal pendiente del programa, pueda realizar su alquiler. También está pensado por si existe algún recepcionista y también así se puedan recoger alquileres por teléfono si el recepcionista y el centro disponen de teléfono.



### 3.2.1.1. CONTROLADOR

En este apartado hablaremos sobre el administrador de las pistas orientado a la aplicación de escritorio.

Aquí vemos la clase Main que es con la que comienza el programa, esta clase llamará al controlador.

MainFlujo
<ul style="list-style-type: none"><li>+static Usuario usuarioRegistrado</li><li>+static boolean userOn</li></ul>
<ul style="list-style-type: none"><li>+static void main(String args)</li></ul>

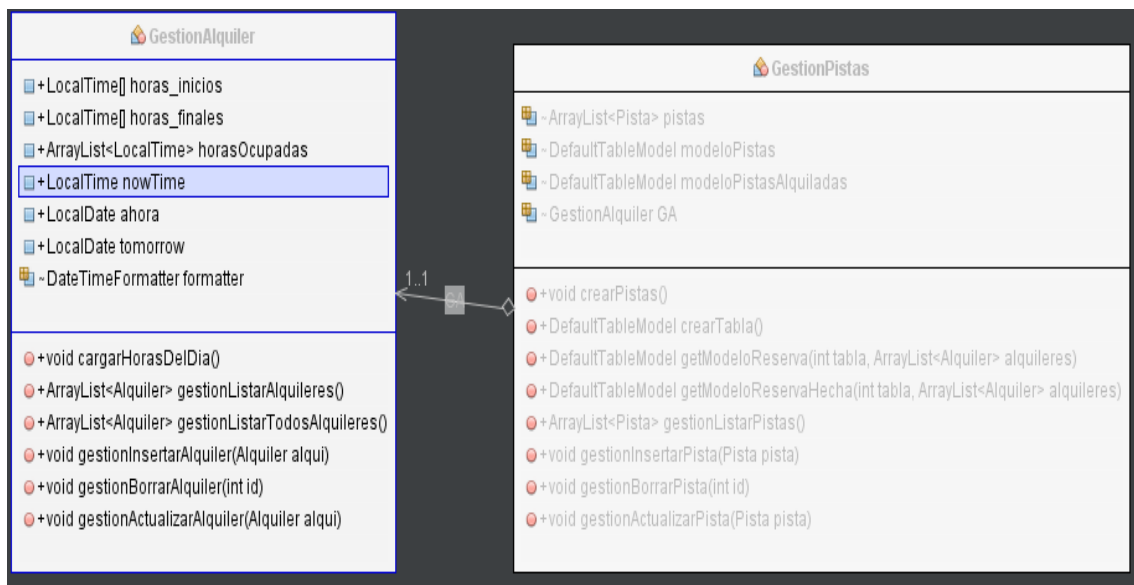
En el controlador ejecutamos el control de usuario de la aplicación con un usuario y una contraseña, siempre que este usuario sea administrador.

Controlador
<ul style="list-style-type: none"><li>~ static Socket skCliente</li><li>- boolean salir</li><li>+ String usuario</li><li>+ String pass</li><li>+ static ObjectInputStream flujoObjEntrada</li><li>+ static ObjectOutputStream flujoObjSalida</li><li>+ static DataInputStream flujo_entrada</li><li>+ static DataOutputStream flujo_salida</li><li>- final char[] CONSTS_HEX</li></ul>
<ul style="list-style-type: none"><li>+ Controlador()</li><li>+void inicializarConexiones()</li><li>+ boolean gestionLOG()</li><li>+void gestionSalir()</li><li>+ static String encriptaEnMD5(String stringAEncriptar)</li><li>+void startReport()</li></ul>

GestionUsuario
<ul style="list-style-type: none"><li>+GestionUsuario()</li><li>+Usuario getUsuario(int id)</li><li>+ArrayList&lt;Usuario&gt; gestionListarUsuarios()</li><li>+void gestionInsertarUsuario(Usuario usu)</li><li>+void gestionBorrarUsuario(int id)</li><li>+void gestionActualizarUsuario(Usuario usu)</li></ul>

En esta imagen vemos a la clase de GestionUsuario, que servirá para conectar el programa con el servidor y realizar las peticiones que queramos para los usuarios.

En la siguiente imagen podemos ver las dos clases de gestión que son GestionAlquiler y GestionPistas, que tienen relación puesto que se complementan una a otra a la hora de obtener las horas y las pistas. También uso métodos para rellenar las listas en la GestionPistas, algo que de otro modo resultaría inútil en esta aplicación. Lo correcto sería hacer otra clase para eso y realizar las pistas dinámicas y que el usuario pudiera añadirlas, modificarlas y borrarlas.



También tenemos una tercera clase de gestión que se llama GestionUsuario para transmitir datos entre el servidor y el programa.

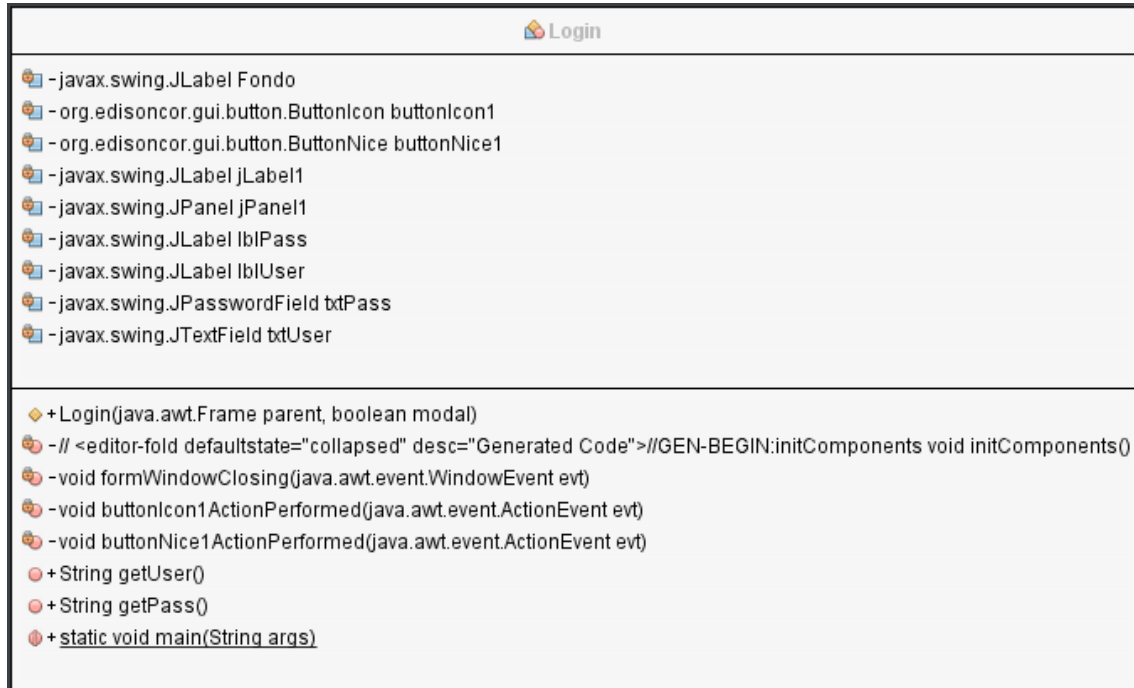
Y por último tenemos UtilidadesPantalla que servirá para ejecutar métodos en todas las vistas, y que al ser los mismos, podemos usarlo varias veces escribiéndolos una sola vez, lo que llamamos abstracción.



### 3.2.1.2. VISTA

Todo este apartado estará compuesto por vistas del proyecto, o lo que es lo mismo, las partes visibles del proyecto y las que verá al final el cliente.

La primera vista que mostraremos será el control de usuario(login).



Aquí se puede ver cómo recogemos el usuario y la contraseña y lo mandamos al controlador. Después se abrirá la pantalla principal.



Aquí podemos ver los diferentes métodos que controlan la clase principal que llamaremos tabla. El objeto principal será el GestionUsuario que vimos en el controlador. Este método permitirá tanto añadir como borrar o actualizar los usuarios. También llamará a la clase de ReservaPadel al pulsar el botón de “Reservar Pista”. La clase ReservaPadel será la siguiente:



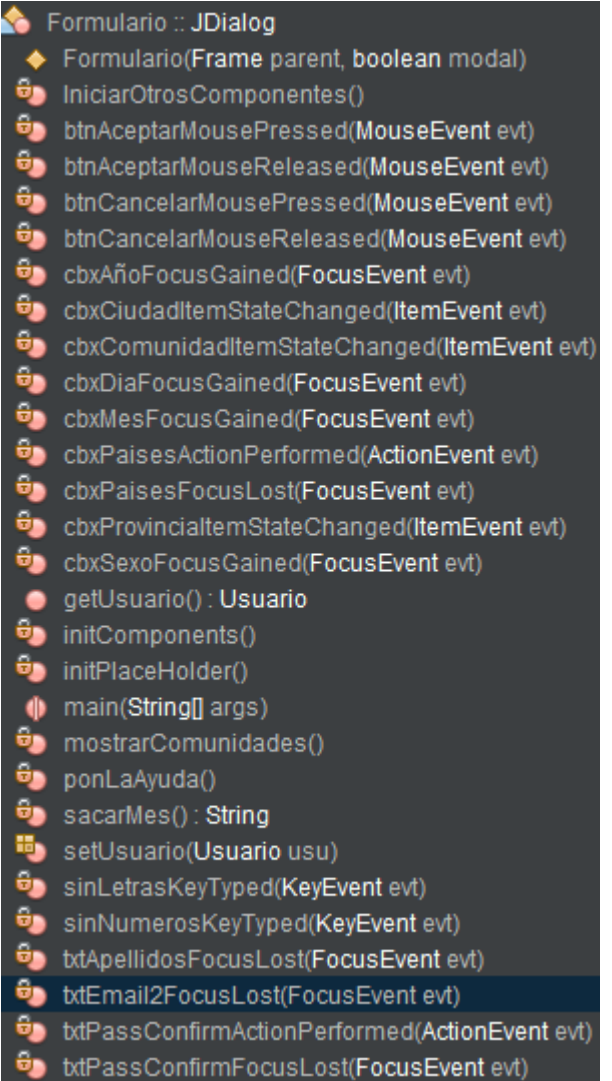
Aquí accedemos a las dos clases de gestión tanto de las pistas como del alquiler. Según la pista pulsada accederemos a la clase de gestión de pistas para añadirle un nuevo modelo a la tabla y así cambiar los datos sin tener que cambiar de tabla.

Por último, pondré las últimas vistas, que son diálogos para meter o sacar información de los usuarios. Empezaré con el más grande, el Formulario. Aunque el propio programa no me permitió sacar el UML, dejaré una captura de todos los métodos y dejaré las variables por motivos obvios.

Como podemos observar en esta clase hacemos un control de errores muy estricto y siempre que se puede en tiempo real, ya que capturamos los eventos para que el usuario sepa que se está equivocando en el momento en que lo está escribiendo.

Esto es primordial ya que un fallo en la inserción de los datos puede ser muy perjudicial tanto para el programa como para el cliente, puesto que puede que quiera acceder al usuario y los datos estén incompletos.

También podemos ver que el único método que es público es el “getUsuario()”. Este método lo tendremos que llamar desde la clase principal para obtener el objeto usuario que creamos en esta clase.



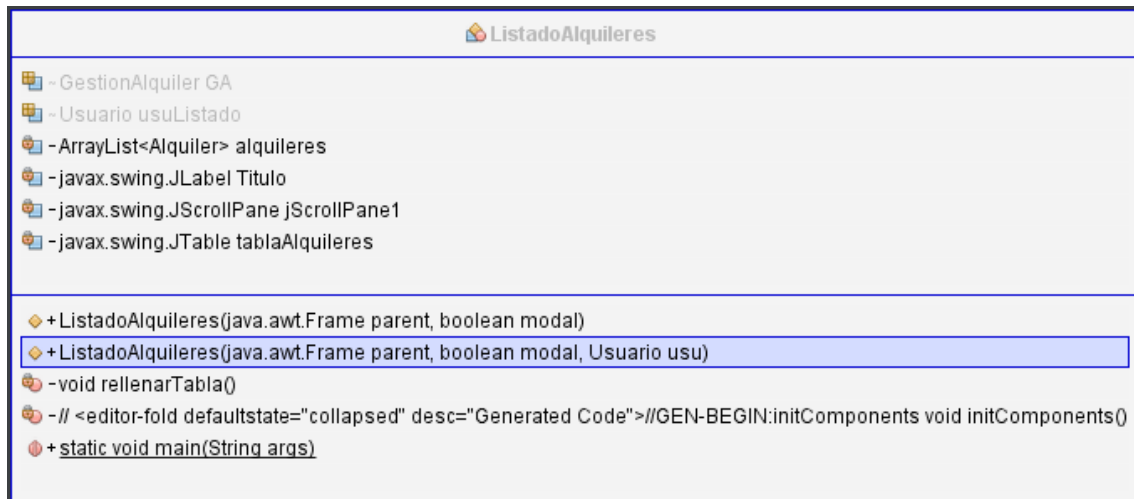
```
Formulario :: JDialog
  Formulario(Frame parent, boolean modal)
  IniciarOtrosComponentes()
  btnAceptarMousePressed(MouseEvent evt)
  btnAceptarMouseReleased(MouseEvent evt)
  btnCancelarMousePressed(MouseEvent evt)
  btnCancelarMouseReleased(MouseEvent evt)
  cbxAñoFocusGained(FocusEvent evt)
  cbxCiudadItemStateChanged(ItemEvent evt)
  cbxComunidadItemStateChanged(ItemEvent evt)
  cbxDiaFocusGained(FocusEvent evt)
  cbxMesFocusGained(FocusEvent evt)
  cbxPaísesActionPerformed(ActionEvent evt)
  cbxPaísesFocusLost(FocusEvent evt)
  cbxProvinciaItemStateChanged(ItemEvent evt)
  cbxSexoFocusGained(FocusEvent evt)
  getUsuario() : Usuario
  initComponents()
  initPlaceholder()
  main(String[] args)
  mostrarComunidades()
  ponLaAyuda()
  sacarMes() : String
  setUsuario(Usuario usu)
  sinLetrasKeyTyped(KeyEvent evt)
  sinNumerosKeyTyped(KeyEvent evt)
  txtApellidosFocusLost(FocusEvent evt)
  txtEmail2FocusLost(FocusEvent evt)
  txtPassConfirmActionPerformed(ActionEvent evt)
  txtPassConfirmFocusLost(FocusEvent evt)
```

Una clase prácticamente calcada de esta última sería la de modificar.



Aquí podemos ver que es prácticamente igual, pero sin tantos controles de errores, puesto que rellenamos todos los campos con los datos que ya tenemos del usuario que queremos modificar, algo que resulta más fácil para el usuario al no tener que rellenar todos los campos en blanco.

Y por último tenemos un listado de todas las reservas de un usuario seleccionado. Para seleccionarlo solo tenemos que hacer doble clic en la tabla de los usuarios. Aquí tenemos el modelo.



### 3.2.2. CLIENTE/ANDROID

Para empezar, explicaré brevemente el proceso del programa.

Se trata de una app que empieza con una cabecera del programa. Al pulsarla se conecta con el servidor: si este no pasa de esa ventana significa que no tiene acceso al servidor. Una vez conectado accederá al login de la aplicación y, rellenando los datos correctos, podrá acceder a las pistas y al seleccionar una pista podrá ver las listas de horas disponibles. Al seleccionar una hora a la que alquilar te preguntará para confirmar y una vez aceptado alquilará.

Me ha sido muy difícil sacar los UML del proyecto en Android Studio, por eso he subido una captura del directorio de carpetas.



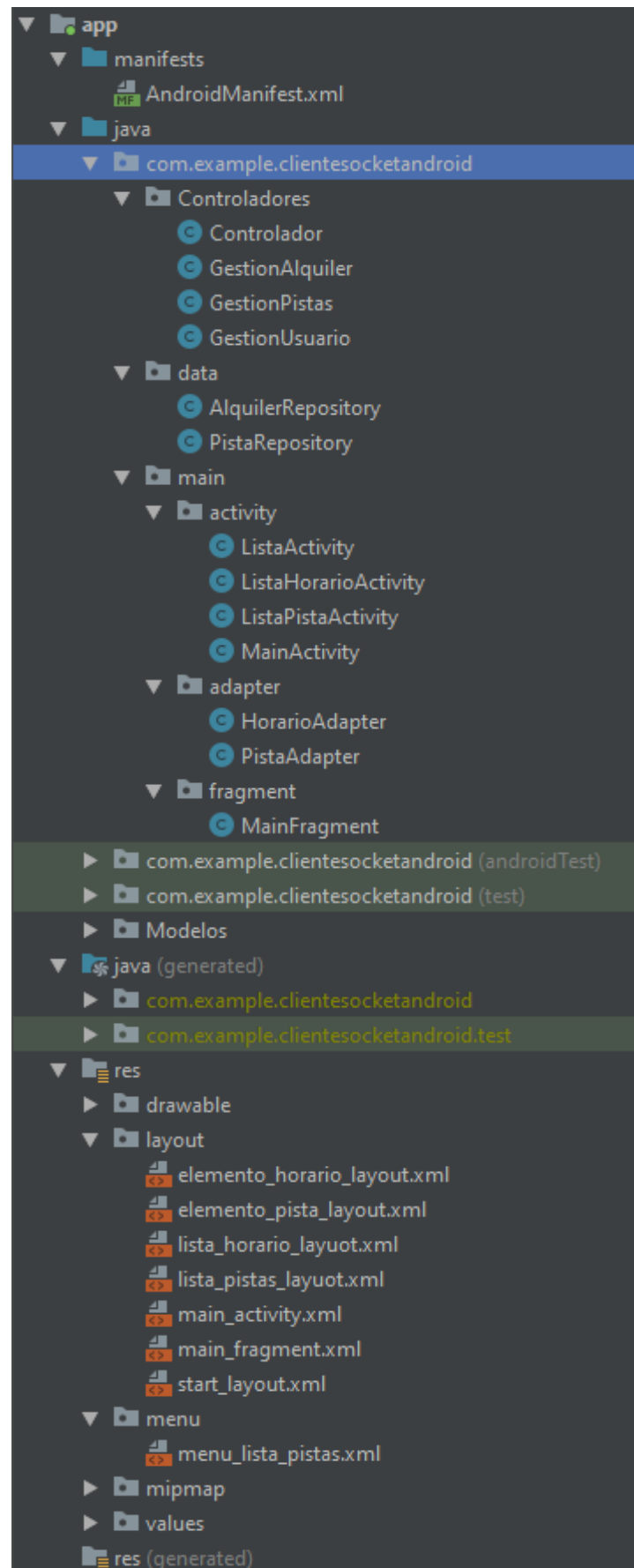
Para empezar, tendremos el `AndroidManifest.xml`, que es donde se pedirán los permisos de accesos y se declaran las actividades. Una actividad en Android es todo aquello que el usuario puede ver a través de una pantalla, puede incluso interactuar con ella.

Después tenemos los paquetes de los controladores que serán los mismos que la aplicación de escritorio.

Tenemos el paquete `data` que contiene los repositorios de los datos. Esto es básicamente un almacenamiento de los datos en memoria RAM que nos envía el servidor.

Después entramos en el `main` de la app donde tenemos las actividades, que son las vistas de la aplicación donde se le asignan los layout que podemos ver más abajo. Tenemos también los adaptadores, que son los que conectan las listas que tenemos en los repositorios en la vista y, por último, vemos el `MainFragment` que es el estilo, esquema y modelo que seguirá esa lista para verse.

Finalmente, un poco más abajo tenemos los Modelos que serán los mismos que las demás aplicaciones (Véase Anexo 1 Pág. 34).



Voy a enseñar unas porciones de código para que se vea el trabajo realizado. En primer lugar, el login que habrá en el menú principal.

Como podemos observar recogemos el usuario y la contraseña de los EditText y llamamos al método login, que estará implementado en el controlador de esta aplicación, al igual que en la aplicación de escritorio.

```
public void loginTry(View view) {
    EditText textField = getView().findViewById(R.id.userFile);
    EditText passField = getView().findViewById(R.id.passwordFile);
    // Llama a un metodo de login
    boolean isLoginCorrect = c.gestionLOG(textField.getText().toString(), passField.getText().toString());
    //isLoginCorrect = UsersRepository.login(textField.getText().toString(),passField.getText().toString());
    /*AlertDialog.Builder dialog =*/
    new AlertDialog.Builder(getActivity())
        .setTitle("Login")
        .setMessage(isLoginCorrect ? "correcto" : "necorrecto")
        .create();
    if (isLoginCorrect)
        startActivity(new Intent(getActivity(), ListaPistaActivity.class));
    else {
        Toast.makeText(this.getContext(), text: "Usuario o contraseña incorrecta", Toast.LENGTH_LONG).show();
    }
}
```

Seguimos con el método para obtener los horarios y que controla: que sean de una pista, que no se repitan y que se eliminen los que ya están alquilados. Todo eso tanto el mismo día que se ejecuta la aplicación como el día siguiente a la misma ejecución.

```

static public List<Alquiler> getHorariosByNum() {
    ArrayList<Alquiler> alquileres = null;
    ArrayList<Alquiler> horas = null;
    ArrayList<Alquiler> horasReales = new ArrayList<>();
    GestionAlquiler GA = new GestionAlquiler();
    horas = GA.cargarHorasDelDia();
    alquileres = GA.gestionListaAlquileres();
    for (int i = 0; i < horas.size(); i++) {
        if (horas.get(i).p.num == pistaSelec.getNum()) { //NUMERO DE PISTA
            if (horas.get(i).getDia().equals(LocalDate.now())) {
                if (horas.get(i).getHoraInicio().getHour() > LocalTime.now().getHour() + 1) {
                    if (alquileres.isEmpty()) {
                        horasReales.add(horas.get(i));
                    } else {
                        for (int j = 0; j < alquileres.size(); j++) {
                            if (horas.get(i).dia.equals(alquileres.get(j).dia)) { //DIA DE HOY
                                if (horas.get(i).getHoraInicio().getHour() != alquileres.get(j).getHoraInicio().getHour()) {
                                    horasReales.add(horas.get(i));
                                    j = alquileres.size();
                                }
                            } else {
                                horasReales.add(horas.get(i));
                                j = alquileres.size();
                            }
                        }
                    }
                }
            } else {
                if (alquileres.isEmpty()) {
                    horasReales.add(horas.get(i));
                } else {
                    for (int j = 0; j < alquileres.size(); j++) {
                        if (GA.tomorrow.equals(alquileres.get(j).dia)) { //DIA DE MAÑANA
                            if (!horas.get(i).getHoraInicio().equals(alquileres.get(j).getHoraInicio())) {
                                horasReales.add(horas.get(i));
                                j = alquileres.size();
                            }
                        }
                    }
                }
            }
        }
    }
    return horasReales;
}

```

Aquí tenemos el adaptador capturando el clic cuando pulsamos en algún archivo de la lista.

```

public void onBindViewHolder(@NonNull HorarioAdapter.ViewHolder holder, final int position) {
    holder.txtHoraInicio.setText(" Hora comienzo: ".concat(horasLista.get(position).getHoraInicio().toString()));
    holder.txtHoraFin.setText(" Hora finalización: ".concat(horasLista.get(position).getHoraFin().toString()));
    holder.txtFecha.setText(" Fecha: ".concat(horasLista.get(position).getDia().toString()));
    holder.itemView.setOnClickListener((view) -> {
        AlertDialog.Builder dialogol = new AlertDialog.Builder(callerActivity);
        dialogol.setTitle("Señor/a: " + horasLista.get(position).usu.getNombre() + "-" + horasLista.get(position).getUsu().getApellidos());
        dialogol.setMessage("¿ Seguro de que desea reservar la pista" + horasLista.get(position).p.num + " a las " + horasLista.get(position).horaInicio +
            " el dia " + horasLista.get(position).getDia() + " ?");
        dialogol.setCancelable(false);
        dialogol.setPositiveButton( text "Confirmar", new DialogInterface.OnClickListener() { //SI
            public void onClick(DialogInterface dialogol, int id) {
                GA.gestionInsertarAlquiler(horasLista.get(position));
                Toast.makeText(callerActivity, text "Reservada a las " + horasLista.get(position).horaInicio + " el dia " + horasLista.get(position).getDia()
                    + " en la pista" + horasLista.get(position).p.num, Toast.LENGTH_LONG).show();
                Intent in = new Intent(callerActivity, ListaPistaActivity.class);
                callerActivity.startActivity(in);
            }
        });
        dialogol.setNegativeButton( text "Cancelar", new DialogInterface.OnClickListener() { //NOO
            public void onClick(DialogInterface dialogol, int id) {
            }
        });
        dialogol.show();
    });
}

```

Y aquí tenemos el funcionamiento de una actividad que llama a una lista RecyclerView. Creamos un objeto View y le añadimos un LinearLayout para que sea vertical y le pasamos la lista junto con el flujo de la actividad.

```

public class ListaPistaActivity extends AppCompatActivity {

    RecyclerView pistasView = null;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.lista_pistas_layout);

        pistasView = findViewById(R.id.listaView);
        pistasView.setLayoutManager(new LinearLayoutManager( context: this));
        pistasView.setAdapter( new PistaAdapter(PistaRepository.pistas(), callerActivity: this)); //THIS EL FLUJO DE LA APP
    }
}

```

### 3.2.3. SERVIDOR

En este apartado veremos que solo hay una clase general movida por hilos (*Threads*). Llamaremos hilo a cada programa ejecutado que quiera acceder a la base de datos, con lo cual hará una llamada al servidor.

Dentro del método “run()” se ve cómo están todos los métodos de gestión a los que se podrá acceder según el protocolo que envíe el cliente. Los protocolos son variables globales dentro del modelo.

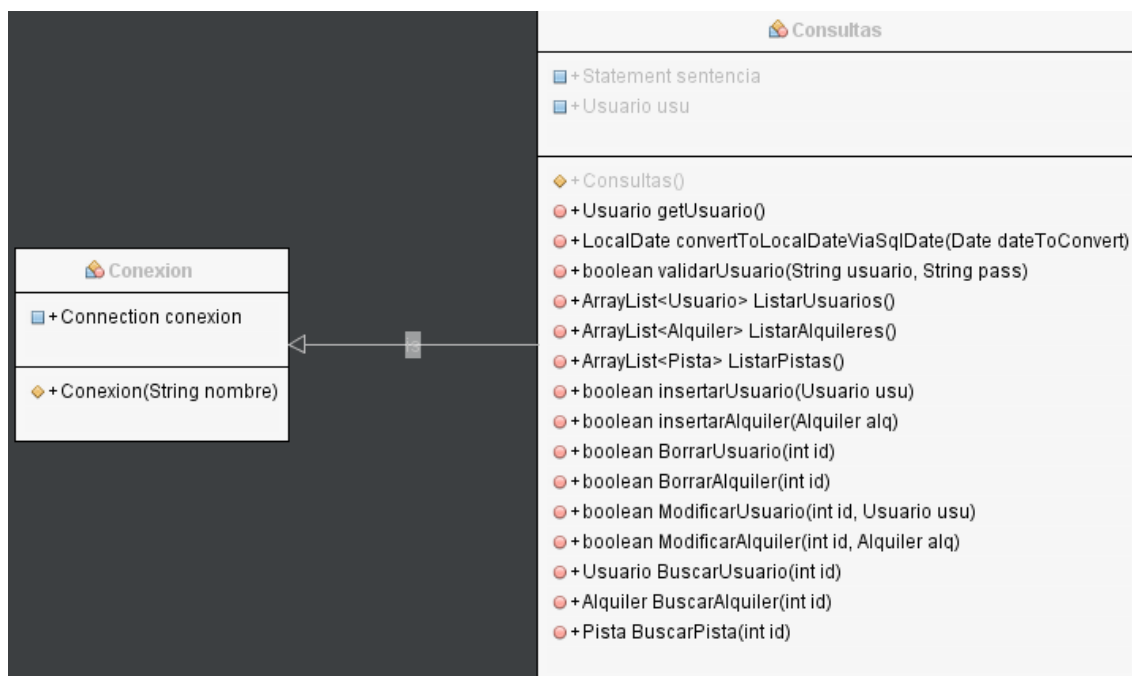
Para obtener algún mensaje de este servidor hay que ejecutarlo desde un terminal. Muestra los clientes que se conectan y desde la IP que viene.

 **Servidor**

-  - DataInputStream flujo\_entrada
-  - DataOutputStream flujo\_salida
-  - ObjectInputStream flujoObjEntrada
-  - ObjectOutputStream flujoObjSalida
-  - Socket skCliente
-  - Consultas consultas
-  - static String nombre

-  + **Servidor(Socket cliente)**
-  + **void run()**
-  - void gestionLOG()
-  - void gestionListarUsuarios()
-  - void gestionListarAlquileres()
-  - void gestionListarPistas()
-  - void gestionInsertarUsuario()
-  - void gestionInsertarAlquiler()
-  - void gestionBorrarUsuario()
-  - void gestionBorrarAlquiler()
-  - void gestionActualizarUsuario()
-  - void gestionActualizarAlquiler()
-  - void gestionListarUsuario()
-  - void gestionListarAlquiler()
-  - void gestionSalir()
-  - void iniciarConexiones()
-  + static void main(String[] args)

También tendremos clases aparte, que son utilizadas para el acceso del servidor a la base de datos.



Aquí podemos ver que siempre que se llama a la clase consulta esta crea una conexión a la base de datos y una vez obtenidos los datos necesarios cierra la conexión para aprovechar al máximo el rendimiento del sistema gestor de base de datos.

#### 3.2.4. MODELO

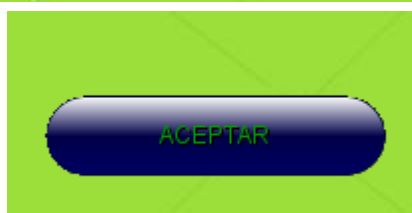
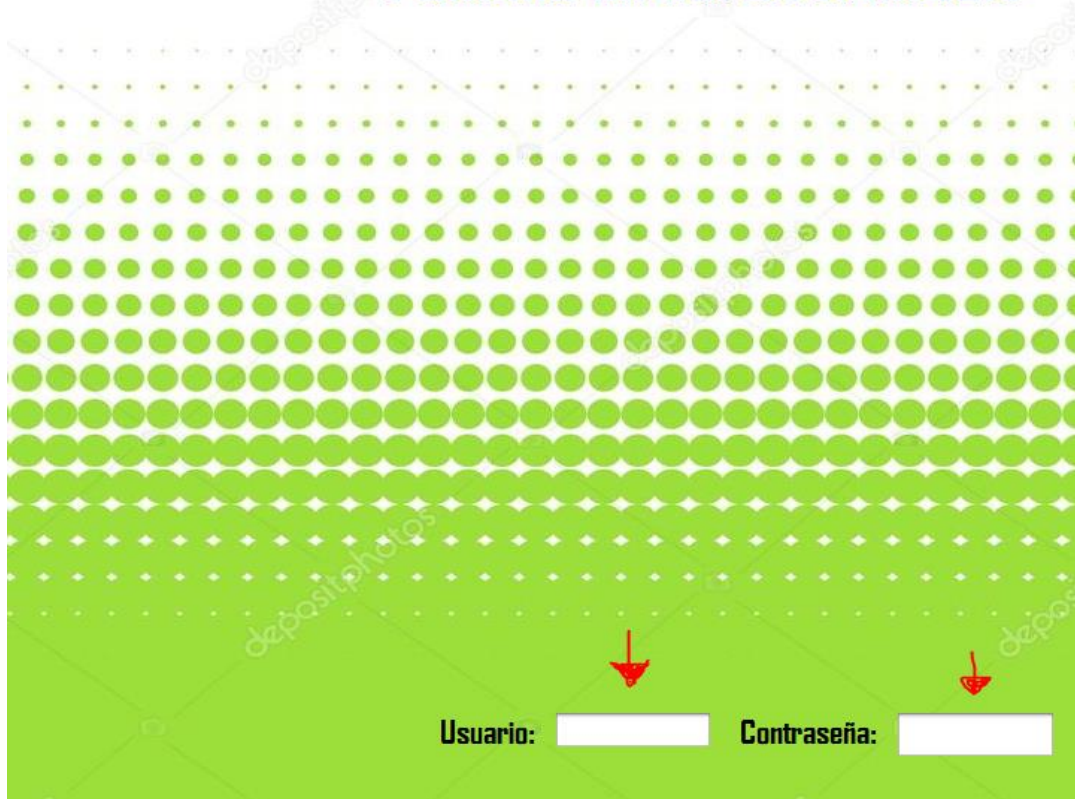
En este cuadro veremos lo que contiene un objeto de cada relación de la base de datos. (Véase Anexo 1 Pág. 34).

## 4. MANUALES.

### 4.1. INICIO DE SESIÓN / LOGIN

Para empezar, veremos dos cuadros de texto:

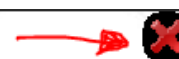
## Pantalla inicial de RentSoft



Uno de los cuadros sirve para poner el nombre de usuario del administrador y otro para poner la contraseña de ese usuario. Una vez insertado pulsaríamos el botón aceptar para acceder a la página principal del programa.

También dispone de un botón arriba a la derecha del programa para cerrarlo en caso de arrepentimiento.

## Pantalla inicial de RentSoft



### 4.2. PANTALLA DE GESTION DE USUARIOS.

En esta pantalla como podemos comprobar tenemos lo principal del programa, que son los usuarios.

Nombre	Apellidos	Usuario	Domicilio	Telefono
ejemplo1	ejemplo ejemplo	e	C/Antonio Huertas Portal 1 Piso 1ºB	123456789
ivan	moreno quiros	ivan	a	195

Cuando pulsamos dos veces en un usuario nos muestra todas las veces que ha reservado una pista para que sepamos la experiencia de este usuario con la empresa. Se muestra la pista en la que estaba la hora a la que la reservó y el día. Podemos pulsar en el título de la tabla para ordenarlos.

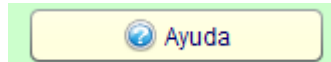
Usuario: ivan-moreno quiros		
Pista	Hora	Dia
1-padel	20:00	2019-10-21
1-padel	22:00	2019-10-21
2-padel	22:00	2019-11-07



aprenderlo.

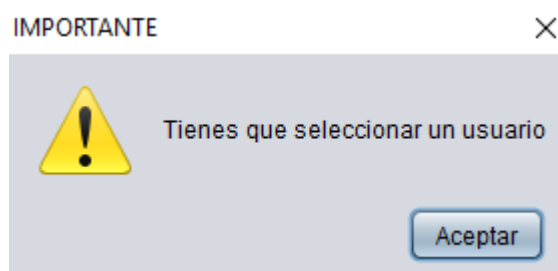
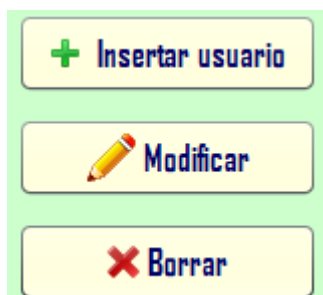
Tenemos un reloj que marca la hora para no perder la noción del tiempo cuando el programa se deje abierto. En principio quería implementar que el programa te avisase cuando llegue la hora de cualquier sesión, pero necesito hacer un minutero constante en el programa y no dispongo de esos conocimientos ni tiempo para

Tenemos también el botón de ayuda en la parte inferior derecha de la ventana. Esta



ayuda está implementada en todo el programa al pulsar F1 en cualquier ventana para guiarte.

Tenemos tres botones abajo a la izquierda para la gestión de usuarios. Siempre que pulsemos uno de estos botones tendremos que seleccionar antes un usuario de la tabla, menos en el caso de añadir, que no hará falta tener seleccionado nada (todo esto está controlado).



Y por último tenemos el botón de “RESERVAR PISTAS” en la parte inferior de la pantalla. Para usarlo tendremos que seleccionar un usuario de la tabla y pulsar el botón para que se abra la ventana de las pistas.

## RESERVAR PISTAS

### 4.2.1. INSERTAR

Cuando se pulsa el botón “Insertar” se abre un formulario de ingreso con muchos datos y su correspondiente control de errores. También tiene metida una clase que se llama “TextPrompt” que hace que se muestre un ejemplo de lo que hay que ingresar para que no dé ningún fallo. Disponemos también de un botón de ayuda en la parte superior izquierda de la pantalla por si tenemos alguna duda.

## Crea tu cuenta de Usuario

 Ayuda

En este apartado tenemos tres fases:

#### 4.2.1.1. CUENTA

Aquí hay que ingresar todos los datos de la cuenta para poder acceder luego desde esta otra aplicación como la de Android.

Cuenta			
Usuario:	Contraseña:	Correo de recuperación:	
<input type="text" value="Ejemplo"/>	<input type="text" value="Contraseña"/>	<input type="text" value="Confirmar Contraseña"/>	<input type="text" value="Ejemplo"/> @ <input type="text" value="Ejemplo.com"/>

Cabe destacar que al poner mal la segunda contraseña saltará automáticamente el título en rojo indicando que las contraseñas no coinciden. Todo el formulario tiene controles de errores como este.

#### 4.2.1.2. DATOS PERSONALES

Aquí hay que ingresar todos los datos personales de usuario que queremos insertar por si queremos ponernos en contacto con él o si hay algún problema.




## Datos personales

Nombre:   Sexo:


Fecha de Nacimiento:    Telefono:

Ubicación:   Domicilio:



comunidad autónoma.

Cabe señalar que si se modifica el país se cambiará la bandera. Si seleccionamos España nos mostrará automáticamente otro combo para seleccionar la



Una vez seleccionada la comunidad autónoma nos mostrará otro combo con todas las provincias de esa comunidad autónoma seleccionada.



Al haber seleccionado la provincia nos mostrará todas las ciudades de esa provincia.

#### 4.2.1.3. REQUISITOS

---

En esta sección tenemos que aceptar que se puedan usar esos datos para el beneficio de la empresa y también si queremos que ese usuario pueda acceder al administrador o solo sea un cliente.



**Requisitos**

☐ Acepto los términos y condiciones al crear mi usuario.

☐ Pulsa para hacer Administrador.

**Cancelar** **Confirmar**

Los botones están creados por el programador y están insertados en el proyecto a través de un “.jar” llamado “btn.jar” que, una vez añadido a la paleta de NetBeans, tiene una serie de métodos para modificar su comportamiento, lo que hace que al seleccionarlos dé una sensación de profundidad.

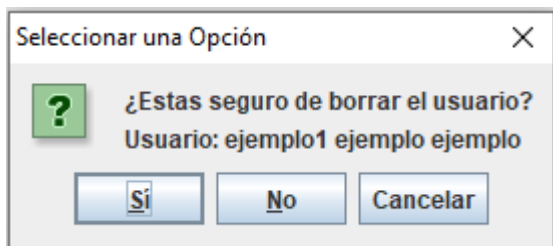
Al pulsar “Confirmar” se validan todos los campos. En caso de pulsar “Cancelar” se cancela el formulario y te vuelve a la página principal.

---

#### 4.2.2. MODIFICAR

Cuando se pulsa el botón “Modificar” se abre un pequeño formulario con los datos rellenados, aquí no hay muchos controles de errores ya que en principio solo se querría cambiar algún dato erróneo o equivocado.

### 4.2.3. BORRAR



Cuando se pulsa el botón de “Borrar” se abre un diálogo que te pregunta si de verdad quieres borrar el usuario y, si se acepta, lo borrará de todos lados.

### 4.3.RESERVAR PISTA

En este apartado podemos ver las diferentes pistas. Dejamos un espacio en la parte derecha para que se puedan añadir más en caso de necesitarlo. Como se puede ver en la siguiente imagen son pistas de Pádel y por eso están representadas con ese fondo.



Cuando pulsamos una de las pistas tenemos un título justo debajo que nos indicará en qué pista estamos actualmente para que sepamos cuál vamos a alquilar.



Según la pista que pulsemos se mostrarán los horarios disponibles de esa pista seleccionada. También tendrá en cuenta la hora del día en la que se encuentra para que si se abre el programa a una hora no te muestre horarios anteriores del mismo día, puesto que sería totalmente imposible alquilarla.

Día	Hora Inicio	Hora Final
----- 2019-11-07 -----	----- 2019-11-07 -----	----- 2019-11-07 -----
2019-11-07	14:00	15:00
2019-11-07	15:00	16:00
2019-11-07	16:00	17:00
2019-11-07	17:00	18:00
2019-11-07	18:00	19:00
2019-11-07	19:00	20:00
2019-11-07	20:00	21:00
2019-11-07	21:00	22:00
2019-11-07	22:00	23:00
----- 2019-11-08 -----	----- 2019-11-07 -----	----- 2019-11-07 -----
2019-11-07	09:00	10:00
2019-11-07	10:00	11:00
2019-11-07	11:00	12:00
2019-11-07	12:00	13:00
2019-11-07	13:00	14:00
2019-11-07	14:00	15:00

Pulsando una hora de la tabla y también el botón con una flecha a la derecha se pasará esa hora a la base de datos dando por hecho que el usuario seleccionado anteriormente hizo la reserva.



En la otra tabla más a la derecha se mostrarán las horas que el usuario ha reservado. Pulsando el botón con la flecha hacia la izquierda se borrará la hora que tiene reservada, la devolverá a la tabla original y la borrará de la base de datos.

Día	Hora Inicio	Hora Final
2019-11-07	22:00	23:00

También se mostrará el usuario al cual se le está haciendo la reserva.

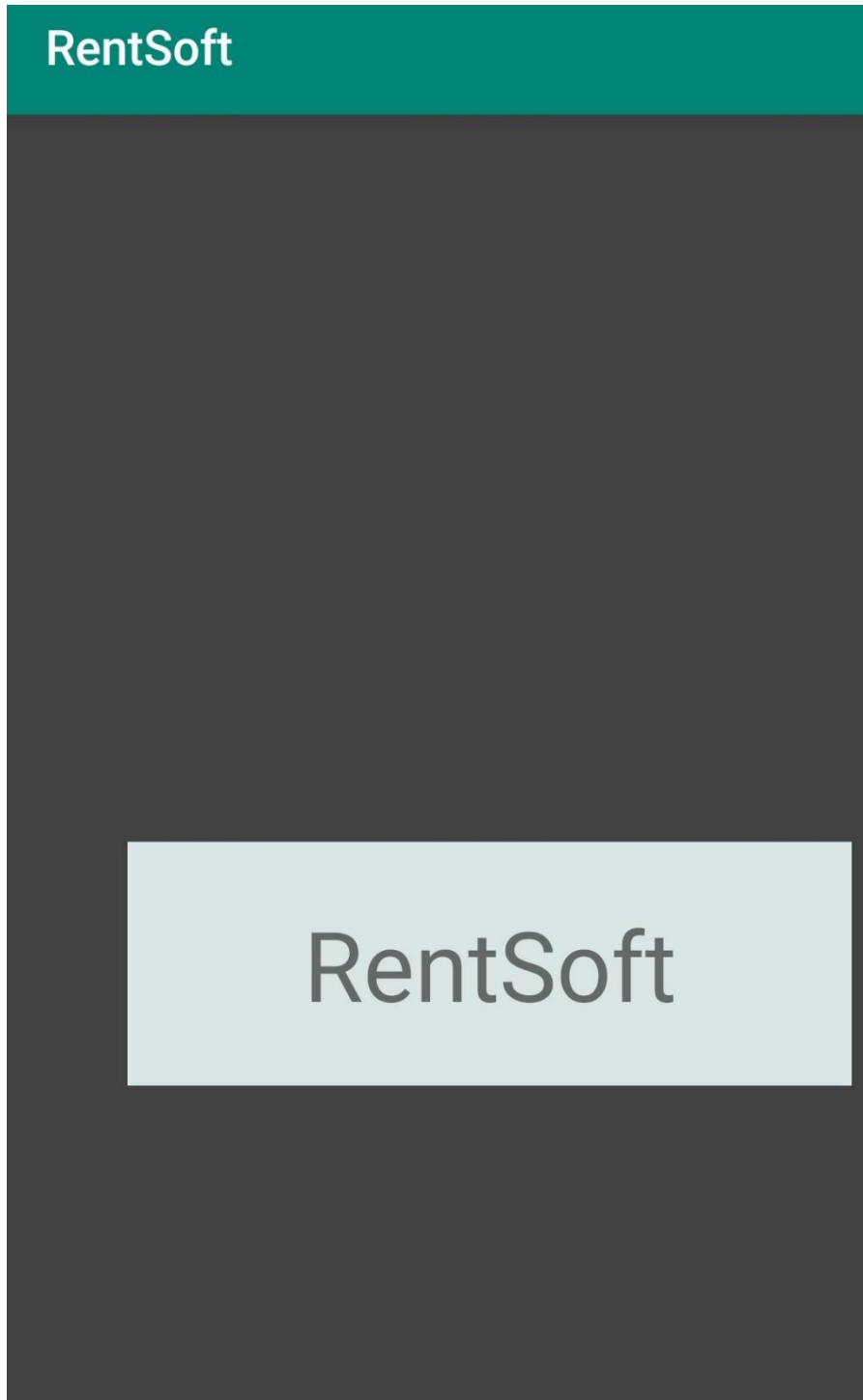
**Tus reservas:** Nombre: ivan moreno quiros

También contiene el botón de ayuda en la parte derecha de la pantalla.

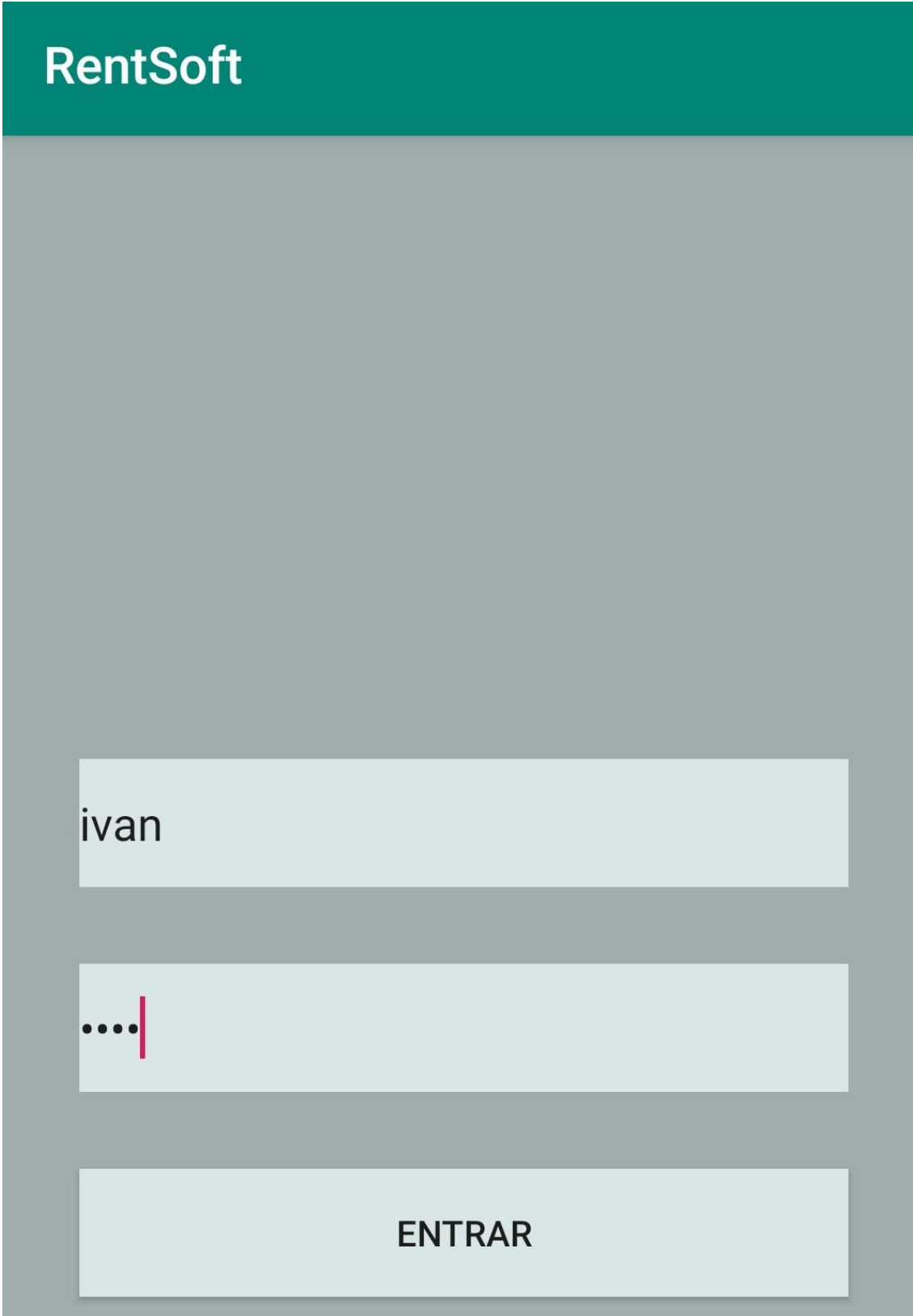
---

#### 4.4.ANDROID

Para empezar, tendremos un botón que conectará con el servidor. Si al pulsar el botón el programa no avanza, será porque no se está teniendo acceso con el servidor.



Después se abrirá una ventana para validar usuario y contraseña. Si son correctos seguirá para adelante.

A login form for 'RentSoft'. The header is a teal bar with the text 'RentSoft' in white. The main area is a light gray rectangle. Inside, there are three light blue input fields. The first field contains the text 'ivan'. The second field contains four black dots and a red vertical cursor line. The third field is a wide button with the text 'ENTRAR' in black capital letters.

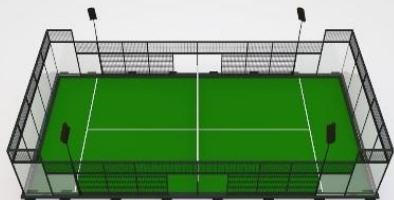
RentSoft

ivan

....

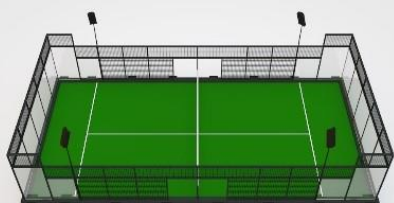
ENTRAR

Después aparecerá una lista con todas las pistas disponibles en el club.



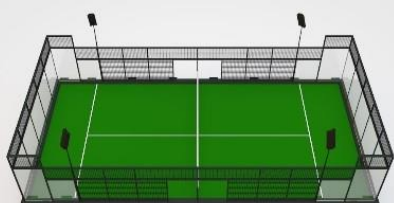
número de pista: 1

tipo de pista: padel



número de pista: 2

tipo de pista: padel



número de pista: 3

tipo de pista: padel

Podemos seleccionar cualquiera y esto nos llevará a otra ventana: las de las horas.

Fecha: 2019-12-01

Hora: 09:00

---

Fecha: 2019-12-01

Hora: 10:00

---

Fecha: 2019-12-01

Hora: 11:00

---

Fecha: 2019-12-01

Hora: 12:00

---

Fecha: 2019-12-01

Hora: 13:00

---

Fecha: 2019-12-01

Hora: 14:00

---

Fecha: 2019-12-01

Hora: 15:00

---

Fecha: 2019-12-01

Hora: 16:00

---

Fecha: 2019-12-01

Hora: 17:00

---

Fecha: 2019-12-01

Hora: 18:00

---

Fecha: 2019-12-01



Esta será la última ventana de la app y al pulsar en cualquiera de las horas disponibles saldrá una ventana de confirmación, así en caso de que el usuario le dé sin querer, no se alquilará la hora.

También me hubiera gustado añadirle otro menú para borrar las reservas que tiene disponible, pero necesito más tiempo para dominar la materia.

## 5. CONCLUSIONES

Según las pruebas de campo y las investigaciones realizadas con el presente trabajo se obtienen las siguientes conclusiones:

- Este sistema es el más indicado para sacarle rendimiento a un club o entidad deportiva, ya que permite optimizar las horas del día y obtener los datos de los que alquilan las pistas y, si fallan, tomar las medidas oportunas.
- Comparándolo con los sistemas actuales en el mercado, este programa podría llegar a representar un proyecto comercial rentable y viable.
- Este programa es un adelanto dentro de la gestión clásica de recopilación de documentos impresos: es más fiable, es más fácil de usar, de fácil aprendizaje, a largo plazo es más barato y es sostenible.
- Permite el alquiler de las pistas desde un terminal Android y así ahorrar tiempo en tele operaciones.

## 6. PROPUESTA DE AMPLIACIÓN.

### 6.1.INTERACTIVIDAD

Para ampliar este proyecto propongo darle el poder para añadir, modificar o borrar las pistas y diferenciarlas por tipos en el cliente de la administración, así se podría hacer totalmente interactivo y no dependería tanto del programador para cambiar las cosas.

También pondría algo para elegir las horas a las que se quiere abrir y cerrar y algún método para que el administrador pudiera poner las horas a la que se quieren reservar las pistas.

### 6.2.AMBICIÓN

También sería recomendable acceder a terminales IOS.

Sería óptimo llegar a subir todo esto a un programa web y reservar desde ahí, pero sería también muy poco productivo, ya que perdería esa confidencialidad de un club. Habría que estudiarlo detenidamente con la empresa a la que se venda.

---

### 6.3.CONTROL

En este programa añadiría un sistema de control del personal para que el sistema llevara las horas del personal que está trabajando en esa empresa.

---

### 6.4.SEGURIDAD

Hice una pequeña prueba para intentar cotejar las contraseñas a un algoritmo llamado “mb5”, pero he estado observando en la empresa en la que estoy que cualquiera puede tener acceso a una contraseña hecha con ese algoritmo, así que mi proposición es hacer un nuevo algoritmo muy similar a ese u otro y así proteger el servidor y el programa en general.

También vería factible hacer más validaciones a la hora de las gestiones de los programas y mandar un mensaje en cada interacción para fortalecer la seguridad.

## 7. BIBLIOGRAFÍA.

Casi todo lo insertado en este proyecto fue aprendido con el profesor que instruía “Desarrollo de Interfaces” en el IES Donoso Cortés en Don Benito.

También hice un curso en Udemy de Android y manejo de Android Studio:  
<https://www.udemy.com/course/programacion-de-android-desde-cero/>

Para la descarga de JavaHelp: <https://jar-download.com/artifacts/javahelp/javahelp>

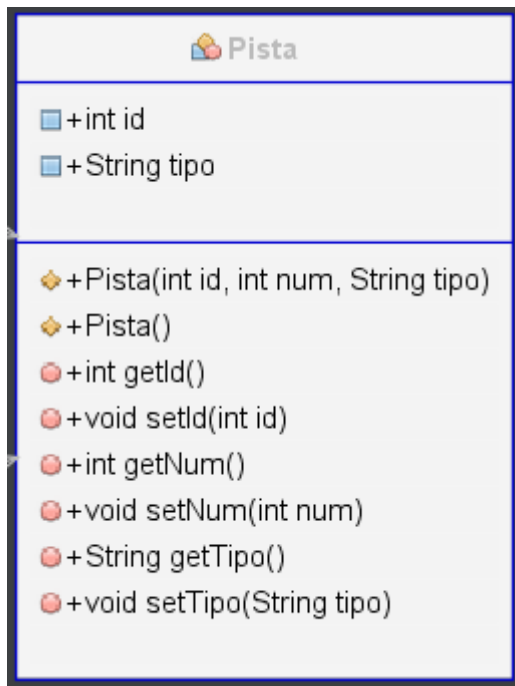
Para la descarga de EdisonCorX: <https://sourceforge.net/projects/edisoncorsx/>

## 8. ANEXOS.

### 8.1.MODELOS

El modelo de la aplicación básicamente son las clases utilizadas para la creación de objetos en Java. Al crear las clases para sacar los datos de la base de datos creamos objetos en lenguaje Java para así hacer una lista de registros de la base de datos.

#### 8.1.1. PISTAS



Aquí se guarda cada registro de la base de datos para ser almacenados en listas Java referente a las pistas.

Cada pista lleva un número que será el Id que tenga en la base de datos.

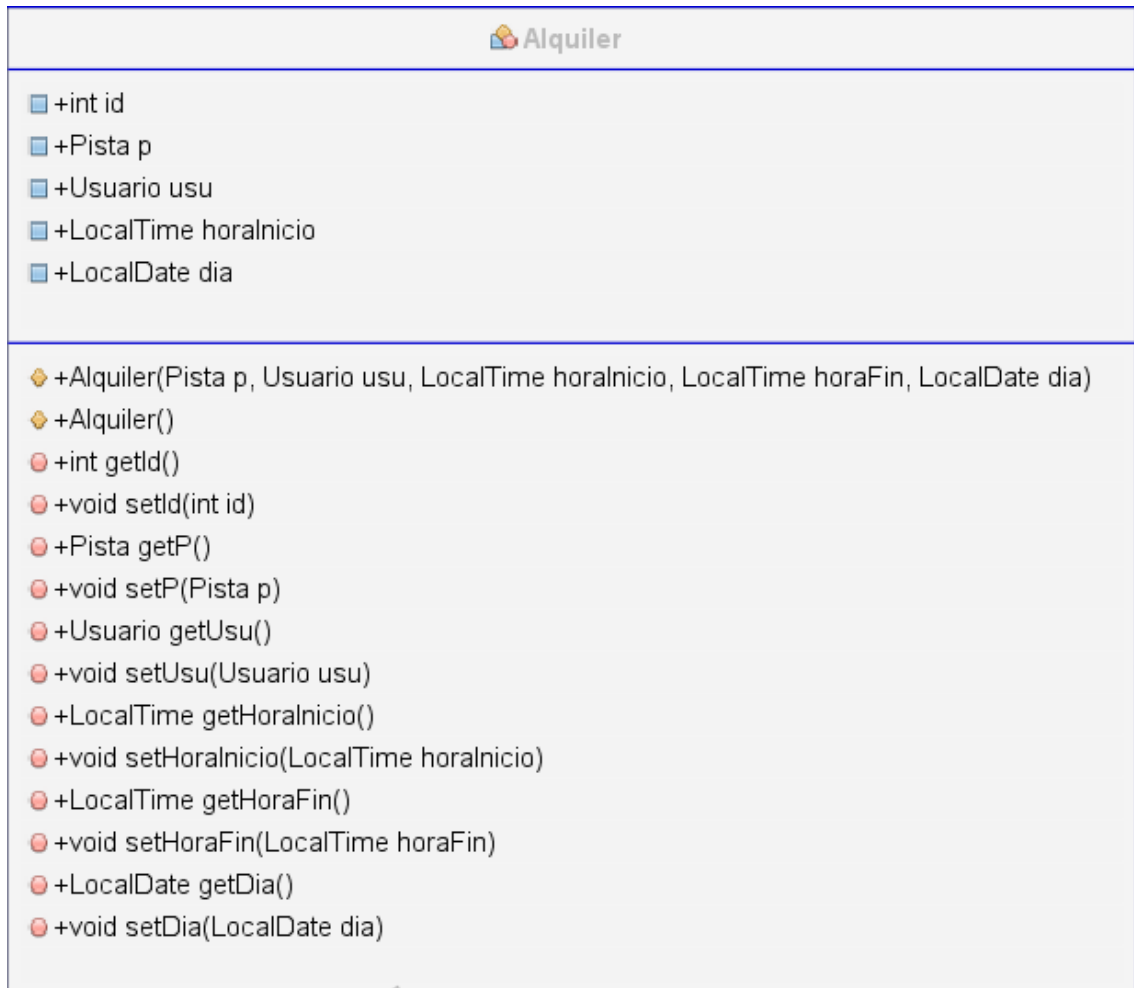
También llevará otro número que será el numero de que le tengan puesto a una pista que sea del mismo tipo (esta no aparece en la imagen).

Y por último tenemos el tipo, que será donde se guardarán el tipo de pista. Por ejemplo, si

es de baloncesto, guardará que es para este deporte.

Y más abajo tenemos los constructores y los métodos.

### 8.1.2. ALQUILER



Aquí se guarda cada registro de la base de datos para ser almacenados en listas Java referente a los alquileres. Aquí se incluyen otra serie de cosas.

Para empezar, tenemos el id que será el numero identificador de la base de datos, luego tenemos la Pista, que corresponderá con aquella donde se quiere realizar o se ha realizado el alquiler. Del mismo modo, tendremos el usuario que quiere realizar esa reserva.

Después tenemos dos objetos LocalTime (uno de ellos no aparece en la imagen) que servirán para identificar la hora exacta a la que se alquilaron las pistas, tanto la hora de inicio como la hora de fin.

Y por último tenemos un objeto `LocalDate` que servirá para saber la fecha exacta en la que se hizo la reserva.

---

### 8.1.3. USUARIOS



Aquí se guarda cada registro de la base de datos para ser almacenado en listas Java referente a los usuarios. Me he visto obligado a recortar tanto los constructores como los métodos, ya que no dispongo de la suficiente resolución como para verlo completo.

Aquí tenemos muchísimas cosas que comentare rápidamente. Para empezar, tenemos el identificador de cada usuario que diferenciará a unos de otros. Para continuar tenemos el usuario y contraseña para, a la hora de acceder a los programas, saber si se está insertado en la base de datos. Luego tenemos un booleano que servirá para diferenciar si el usuario es administrador del sistema y si puede acceder a la aplicación de escritorio. Después tenemos un correo de recuperación, ya que quiero probar en siguientes versiones si al olvidar la contraseña el usuario, puede enviársele un correo con una contraseña temporal para luego cambiarla. Después tenemos la información personal que llevará el nombre, los apellidos, el teléfono, el sexo, fecha de nacimiento, el país donde vive, la comunidad autónoma, la provincia, la ciudad y, por último, el domicilio.

---

### 8.1.4. DETALLES

Una vez creadas todas las clases de los modelos están se compilan en una clase .jar que serán importadas después a todos los programas.