Centro: I.E.S Leonardo Da Vinci Curso 2021/22 Título: Desarrollo de Aplicaciones Web



APLICACIÓN DE GESTION DE RESERVAS

Tutor: Alumnos:

Consuelo Moya Gutiérrez Sergio Muñoz Villullas

Daniel Ferrera Cubero

ÍNDICE

[DESCRIPCIÓN 2](#_TOC_250017)

[HERRAMIENTAS 3](#_TOC_250016)

[WORKSPACE 3](#_TOC_250015)

[BASE DE DATOS 4](#_TOC_250014)

[DIAGRAMAS 5](#_TOC_250012)

[MODELO ENTIDAD RELACIÓN 7](#_TOC_250010)

[DESARROLLO 9](#_TOC_250008)

[DATOS TÉCNICOS 9](#_TOC_250007)

[ANIMACIÓN DE CARGA 9](#_TOC_250006)

[MENÚ 10](#_TOC_250005)

[SOLICITUD DE CITA 12](#_TOC_250004)

[¿DÓNDE ESTAMOS? 26](#_TOC_250003)

[CONTACTANOS 28](#_TOC_250002)

[CONCLUSIONES Y FUTURAS MEJORAS 31](#_TOC_250001)

[FUENTES 32](#_TOC_250000)

# DESCRIPCIÓN

#### Esta aplicación surgió de la necesidad por la situación excepcional que estamos viviendo con el covid-19, las limitaciones de movimiento, el miedo a poder contagiarse y el riesgo de contagio nos han forzado a desarrollar nuevas herramientas para poder vivir el día a día. Esta aplicación es para ello, para poder solicitar citas en el local sin tener que asistir al mismo previamente por miedo al contagio o para pedir comida a domicilio (delivery).

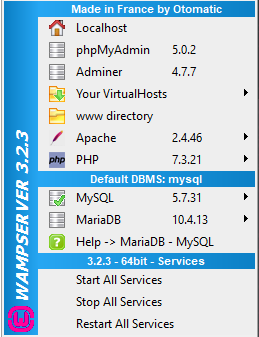
#### Con esta app podemos no solo hacer una reserva o pedir a domicilio, si no que además tenemos a nuestra disposición la página web donde tendremos nuestra cuenta, podremos ver nuestros pedidos, los gastos de los mismos, cambio de contraseña, recuperación de la cuenta y gestión del saldo de la misma.

#### Lo que pretendemos con esta aplicación web es ofrecer una herramienta a los usuarios para que su relación con el local sea lo mas cercana y eficaz posible.

# HERRAMIENTAS

## WORKSPACE

Tenemos diferentes plataformas de desarrollo web, como BitNami,MAMP,EasyPHP y XAMPP, nosotros en nuestro caso hemos elegido WAMPP, usada anteriormente en clase tanto en Desarrollo Web Entorno Servidor como en Desarrollo Web Entorno Cliente. Es un programa que establece los servicios necesarios para simular el lanzamiento de una aplicación web de manera real, estableciendo una base de datos, en nuestro caso MYSQL y el servicio de APACHE y PHP para ello.



Versión de WampServer 3.2.3

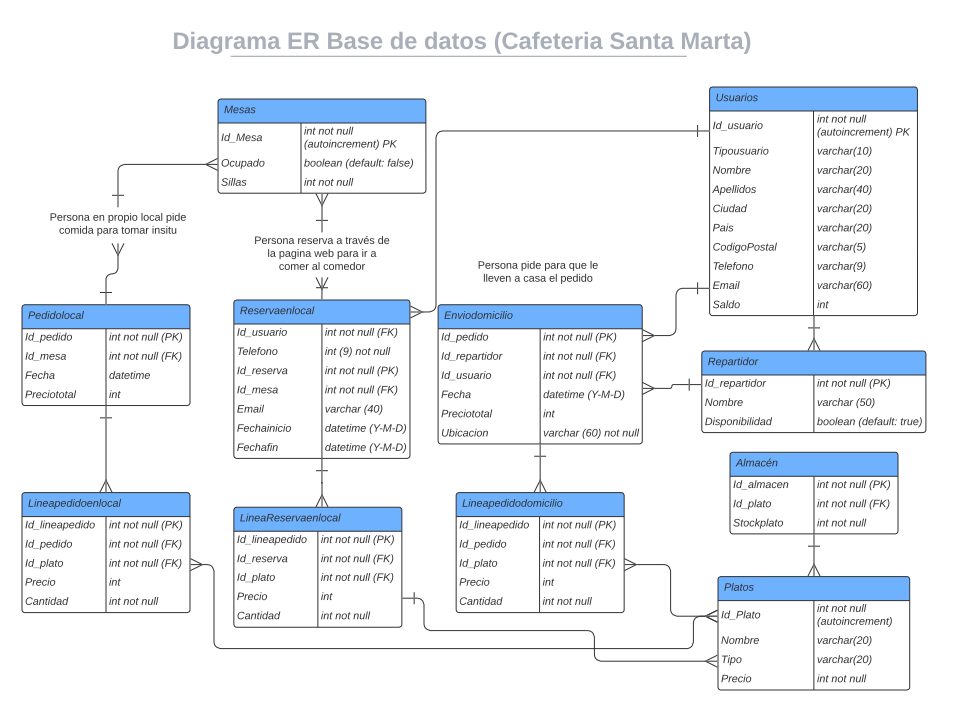
Versión de PHP 7.3.21

Versión de MySQL 5.7.31

Versión de Apache 2.4.46

# DIAGRAMAS

## MODELO ENTIDAD RELACIÓN DE LA BBDD



# DESARROLLO

## DATOS TÉCNICOS

* Nombre de la aplicación: Cafeteria Santa Marta
* Plataforma: Web
* Lenguaje de programación: PHP, JavaScript, CSS, HTML y MySQL.
* Requisitos mínimos del Sistema: Conexión a internet
* Manual de instalación: No tiene opciones de instalación, sólo ejecutar la app en internet.

## MENÚ

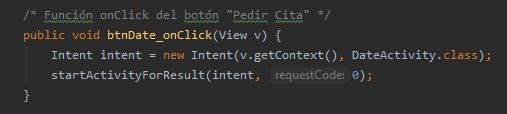
**- Breve descripción**

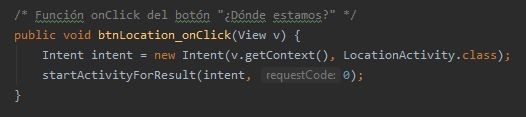
Una vez concluida la carga de la animación se abre el activity del menú este activity "activity\_main.xml" se estructura sobre un Constraint Layout. con las 3 secciones que más adelante describiremos y nos pedirá que le demos permisos a la aplicación para acceder al teléfono. Poco hay que decir de esa parte, pero podemos hablar del botón que tenemos abajo que es un botón que hace una solicitud Call() para llamar a un teléfono que sería el supuesto teléfono de emergencias (de aquí que nos pidiera permisos para el teléfono al ejecutar la app).

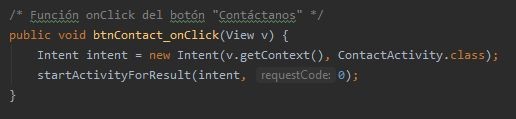
**-Componentes**

Los componentes a mostrar son 2 textview (tvTitle y tvName), 1 imageview (imgLogo), 1 divider, 3 Button (btnDate, btnWhere y btnContact) y 1 Extended Floating Action Button (extended\_fab)

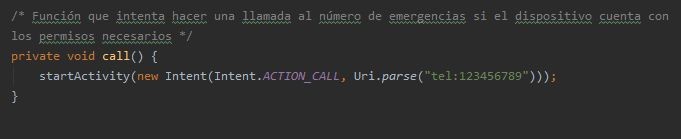
**-Funcionalidad**

1. **btnDate:** Al hacer clic en él accedemos a la activity "activity\_date.xml"
2. **btnWhere:** Al hacer clic en él accedemos a la activity "activity\_location.xml"

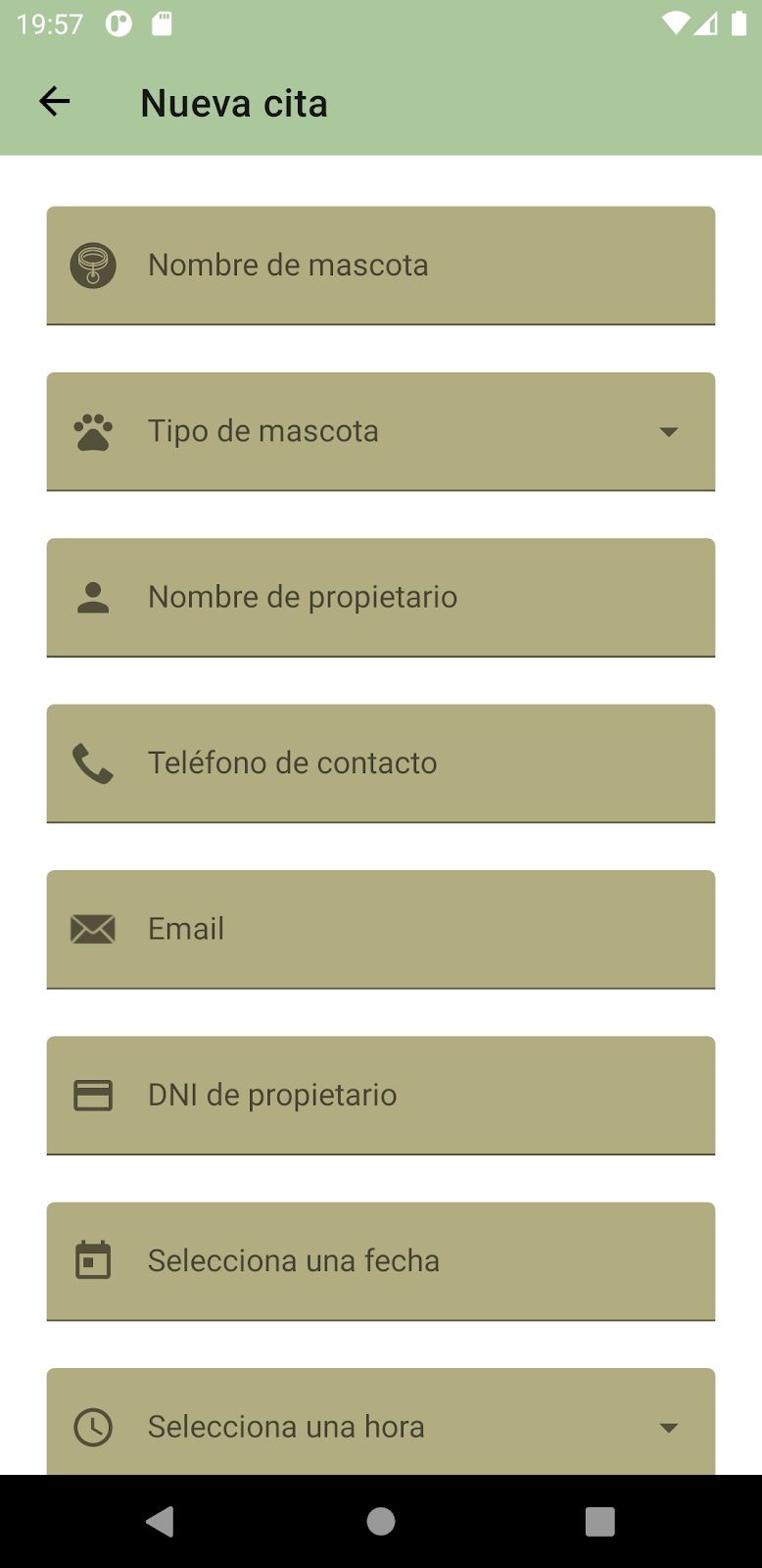


1. **btnContact:** Al hacer clic en él accedemos a la activity "activity\_contact.xml"
2. **emergency\_fab:** Al hacer clic en él aparecerá la aplicación predeterminada de llamadas y se producirá una llamada al teléfono que aparecerá en pantalla.



Esto ocurrirá sí y sólo si el usuarios ha dado permiso a la aplicación de realizar llamadas al iniciarla por primera vez.

## SOLICITUD DE CITA

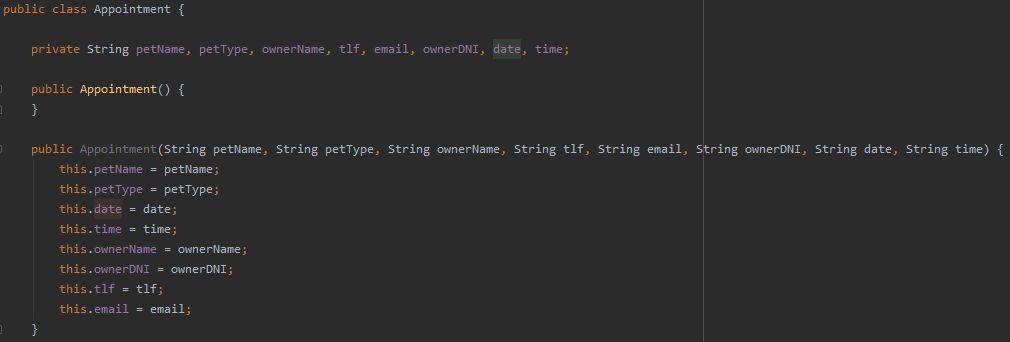
**- Breve descripción**

Aquí recogemos los datos necesarios para la cita que el usuario va a introducir, para ello hemos desarrollado un formulario con scroll, validación de datos, comprobación de consultas libres, cuadros de diálogo de advertencias y mail con la información de la cita en caso de que la cita esté disponible.

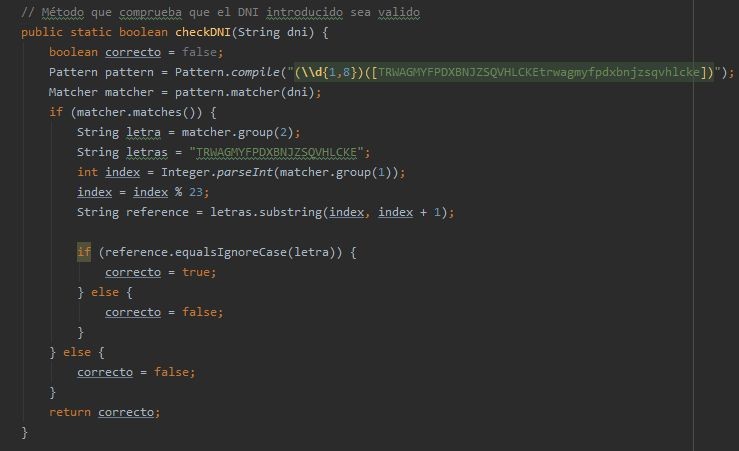
**-Componentes**

Como hemos explicado disponemos un formulario compuesto por TextInputLayout, TextInputEditText, AutoCompleteTextView(DropDownMenu), DatePickerDialog, AppBarLayout y Button. Y a parte del formulario dispondremos de un Objeto el cual contendrá el constructor por defecto y el parametrizado además de los métodos que cambian o devuelven las propiedades del objeto, y dos métodos para comprobar que tanto el DNI como el correo electrónico introducidos sean válidos.

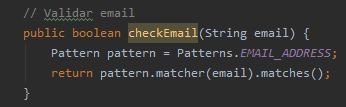
**-Desarrollo**

**Constructor Appointment**

Aquí tendremos la Clase objeto con el constructor vacío y parametrizado, acompañado de sus correspondientes Setter, Getters y su método ToString (No mostrado en la imagen por ser demasiado extenso).

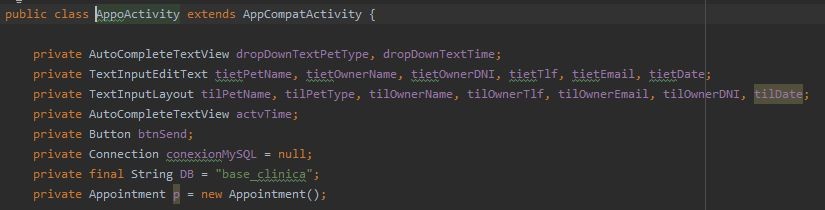


Disponemos de un método dentro del objeto para comprobar que el DNI introducido es válido, en caso de que sea correcto retornará true y en caso contrario devolverá false.

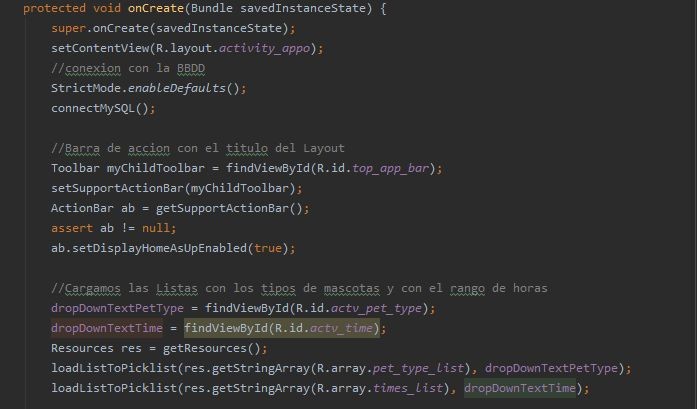


El método para comprobar la validez del correo es usando una utilidad de Android.util.

**Clase AppoActivity**



Lo primero que hacemos es declarar las variables de la aplicación, la conexión con la base de datos e inicializamos el constructor para poder trabajar con él durante todo el proceso.



Nada más crearse la actividad, lo primero que hace es llamar al método para establecer la conexión con la base de datos y después cargar la barra de acción del título y cargar las listas de las razas y las horas.



##### Metodo connectMySQL:

Para esta parte tenemos que implementar una librería en la app, que es

mysql-connector-java-5.1.6.jar, que nos permitirá hacer la conexión a la base de datos. En este método creamos 2 Strings, uno que tiene la ruta donde se encuentra el driver para realizar la conexión y otro con la dirección de la base de datos, indicando el

tipo de base de datos que es, la ip a la que nos conectaremos y el puerto que usaremos. Luego usamos la conexión creada al inicio de la clase AppoActivity y le pasamos los datos de la dirección y luego el nombre de la base de datos, más el usuario y la contraseña de este.

### Nombre de Mascota:

Captura el nombre de la mascota introducida, solo guarda en el constructor el valor del campo cuando haya algo escrito. Además hace comprobaciones cuando pierde el foco, cuando haces click para borrar el campo y cuando pasa de una caja a otra.

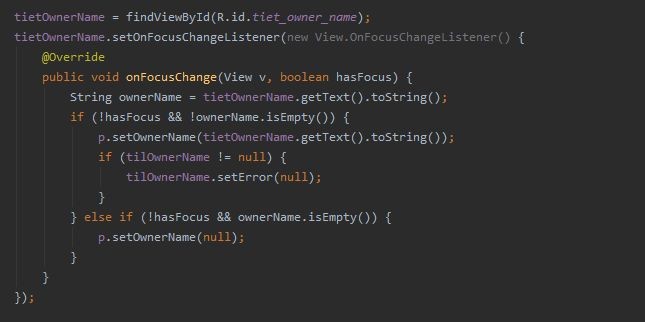


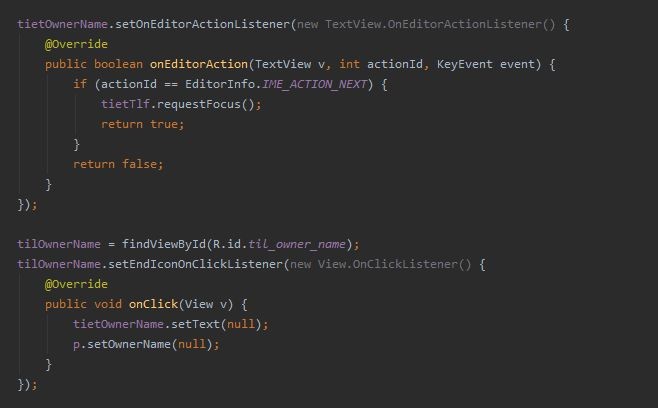
Además controlaremos los datos que introduzca para que solo pueda meter caracteres alfabéticos de tipo minúscula y mayúscula.

### Tipo de Mascota:

Aquí no inserta nada, lo que hace es mediante una lista desplegable elegir el tipo de animal y guarda el valor tanto en la caja como en la propiedad del objeto y pasa a la siguiente caja.

### Nombre de Dueño:

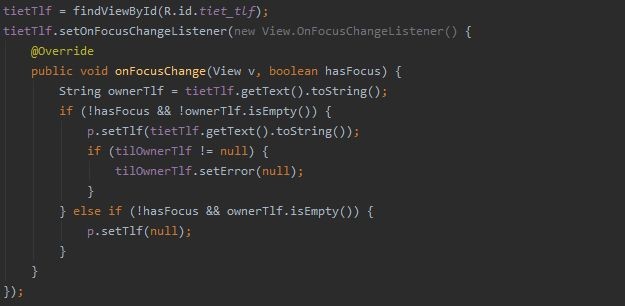


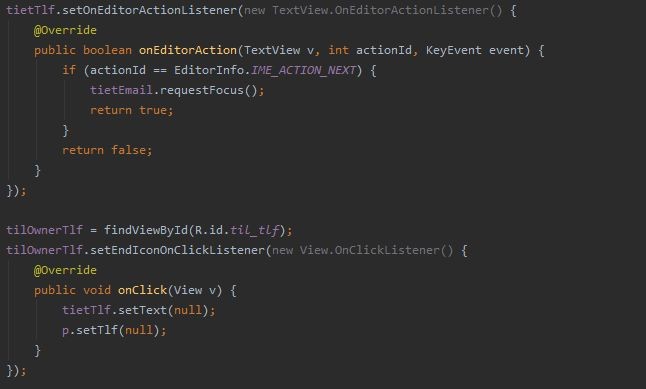


Aquí básicamente se implementa el mismo código que para el Nombre de mascota con la validez y todo para controlar la entrada de solo letras.

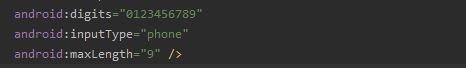
Solo que además le añadimos los espacios por los apellidos y los nombres compuestos y el guión para los apellidos compuestos.

### Teléfono:



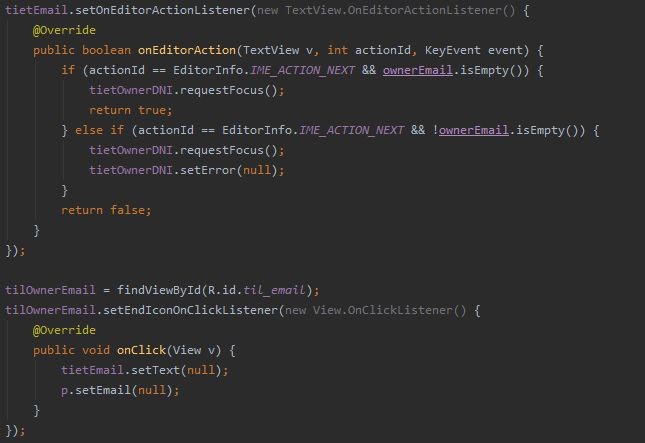


A la hora de coger el Teléfono implementamos prácticamente el mismo código que en Nombre de Mascota y Nombre de Dueño.



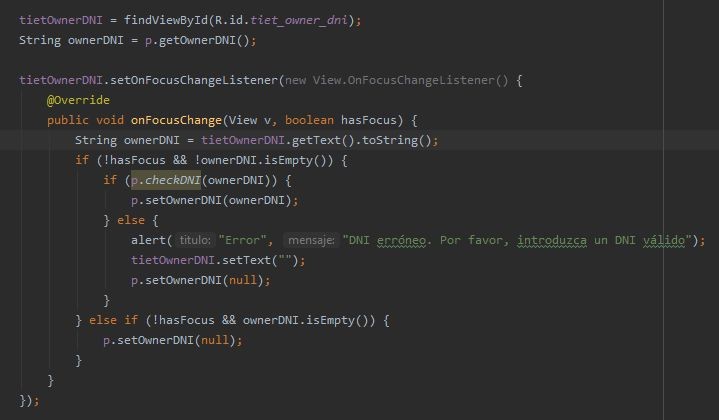
Con la diferencia que controlamos que solo puedan poner números hasta un máximo de 9 dígitos y además le indicamos que en vez de sacar un teclado alfanumérico saque un teclado únicamente numérico.

### Email:



Cuando introducimos el email lo primero que hace es comprobar que no esté vacío y si no lo está llama al método comprobar email del objeto, si el email no es válido, saltara un alertDialog informandonos del problema y borrara el mail que hayamos introducido, por el contrario, si es correcto seguirá y guardará en la caja y en el constructor.

### DNI:



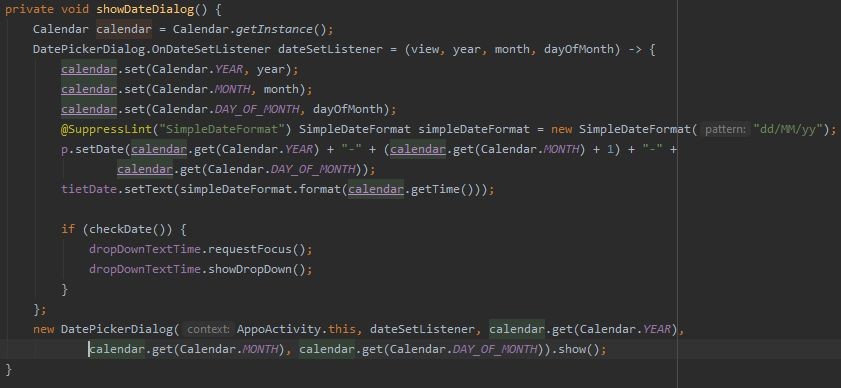


Al introducir el DNI lo primero que haces es comprobar que no esté vacío y si no lo está llama al método comprobar dni del Objeto, si es valido se guardará el valor en el constructor y en el campo, si no, saltará un alertDialog para avisarnos de que el DNI introducido no es valido, ademas lo tenemos controlado tanto si pierde el foco como si hace click en el botón siguiente.

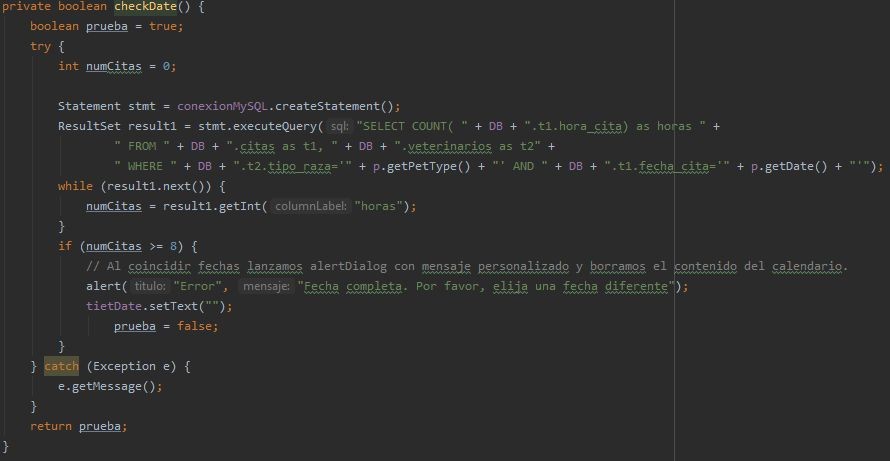


Además controlamos que solo pueda meter caracteres alfanuméricos, en mayúsculas las letras y como máximo 9.

### FECHA:



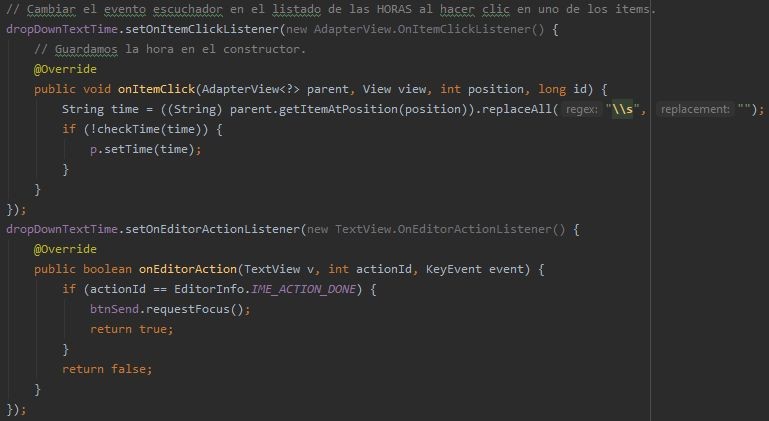
Guarda la instancia del calendario en una variable la cual vamos a utilizar para establecer la fecha por dia, mes y año. Para ello se usa DatePickerDialog, una interfaz de calendario para poder seleccionar la fecha. Una vez seleccionada llama al método CheckDate.



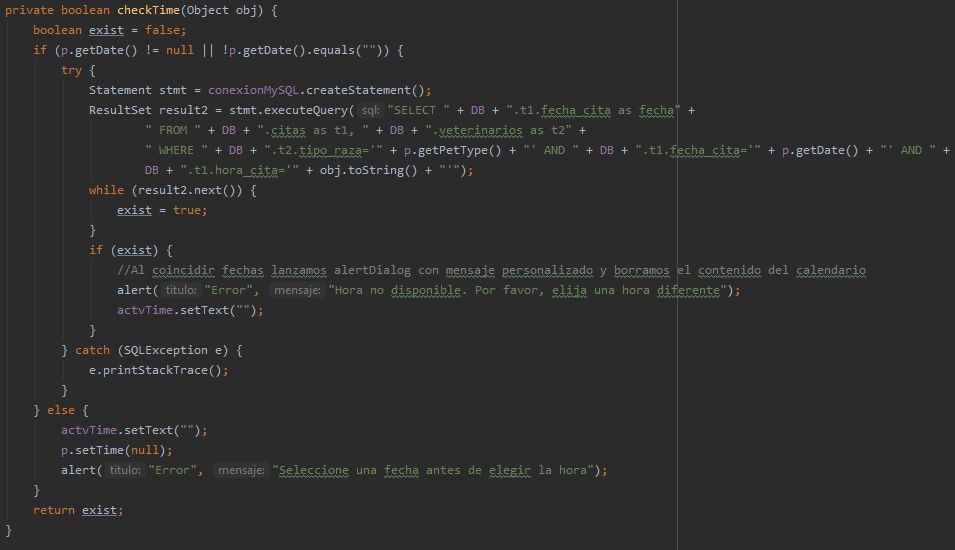
##### CheckDate

Este método comprobará que el día seleccionado tenga espacio para almacenar otra cita, para lo cual conecta con la base de datos, y hace un select count en la tabla citas de las horas basado en el tipo de raza del animal y en la fecha seleccionada. Entonces como al día tenemos establecido que solo puede haber 8 citas por doctor si el count es exactamente 8 te dice que la fecha está cogida y te hace volver a elegir fecha.

### HORA:



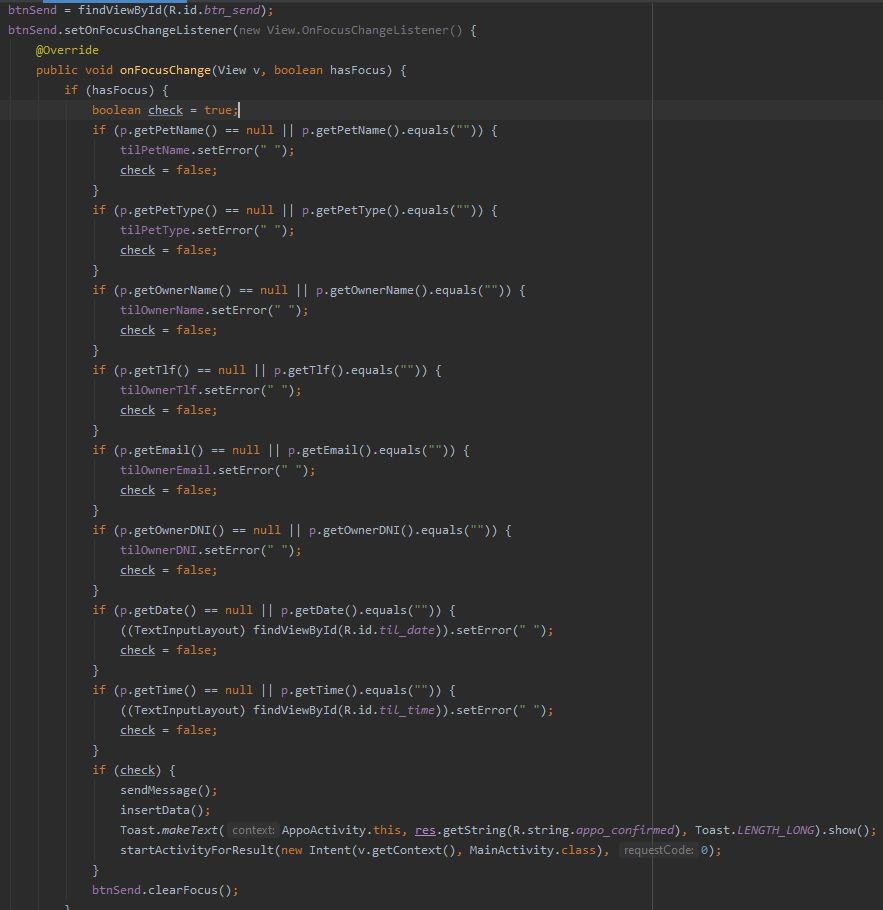
Selecciona un rango de horas del mostrado, cuando haga una selección, comprobará si esa hora en la fecha ya escogida con el tipo de animal está disponible y si es así, guardará el valor en el constructor y en la caja.



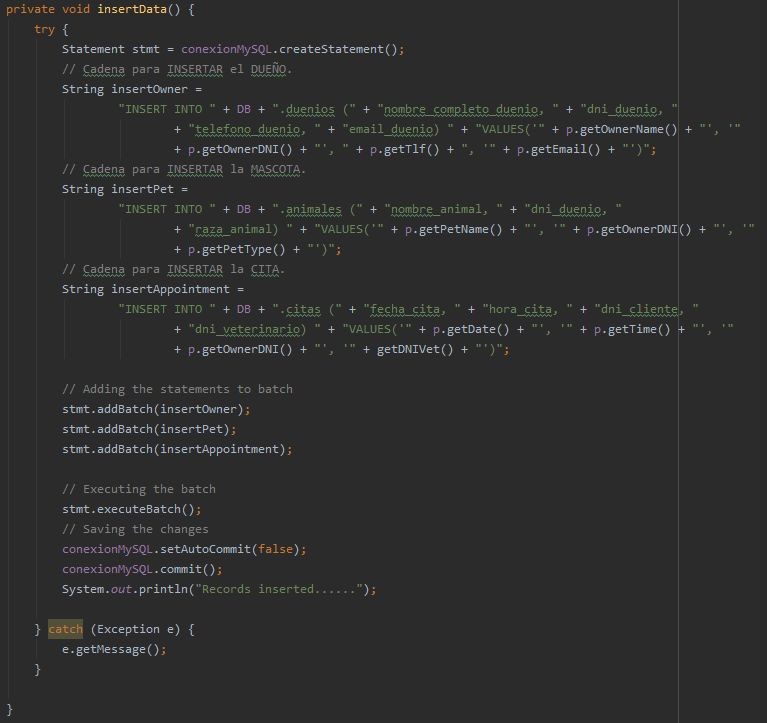
##### checkTime

Este método comprueba si la hora seleccionada es válida, por lo tanto primero comprueba que haya una fecha seleccionada, y después hace la sentencia select que comprueba si la hora seleccionada del tipo de mascota específico en el día indicado anteriormente está disponible. Si existe te salta un mensaje diciendo que ya tienen esa hora ocupada.

### ENVIAR FORMULARIO:

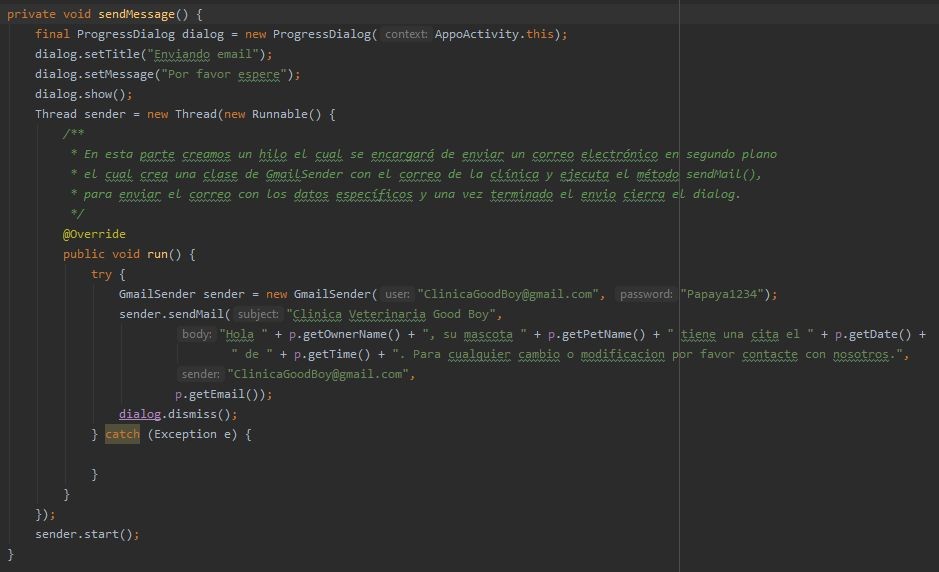


Una vez rellenado todas las casillas y validado que esté libre la cita que se desea, al pulsar el botón de enviar comprueba que en todas las casillas haya datos. Si falta algún valor aparecerán exclamaciones de emergencias en las cajas correspondientes y si por el contrario está todo correcto llamaremos a los métodos sendMessage e insertData.



##### InsertData

Método encargado de añadir las sentencias de inserción de dueño, mascota (animal) y cita a un lote (batch) el cual se ejecutará al recibir dichas sentencias y registrará la operación al hacer un commit de esta.



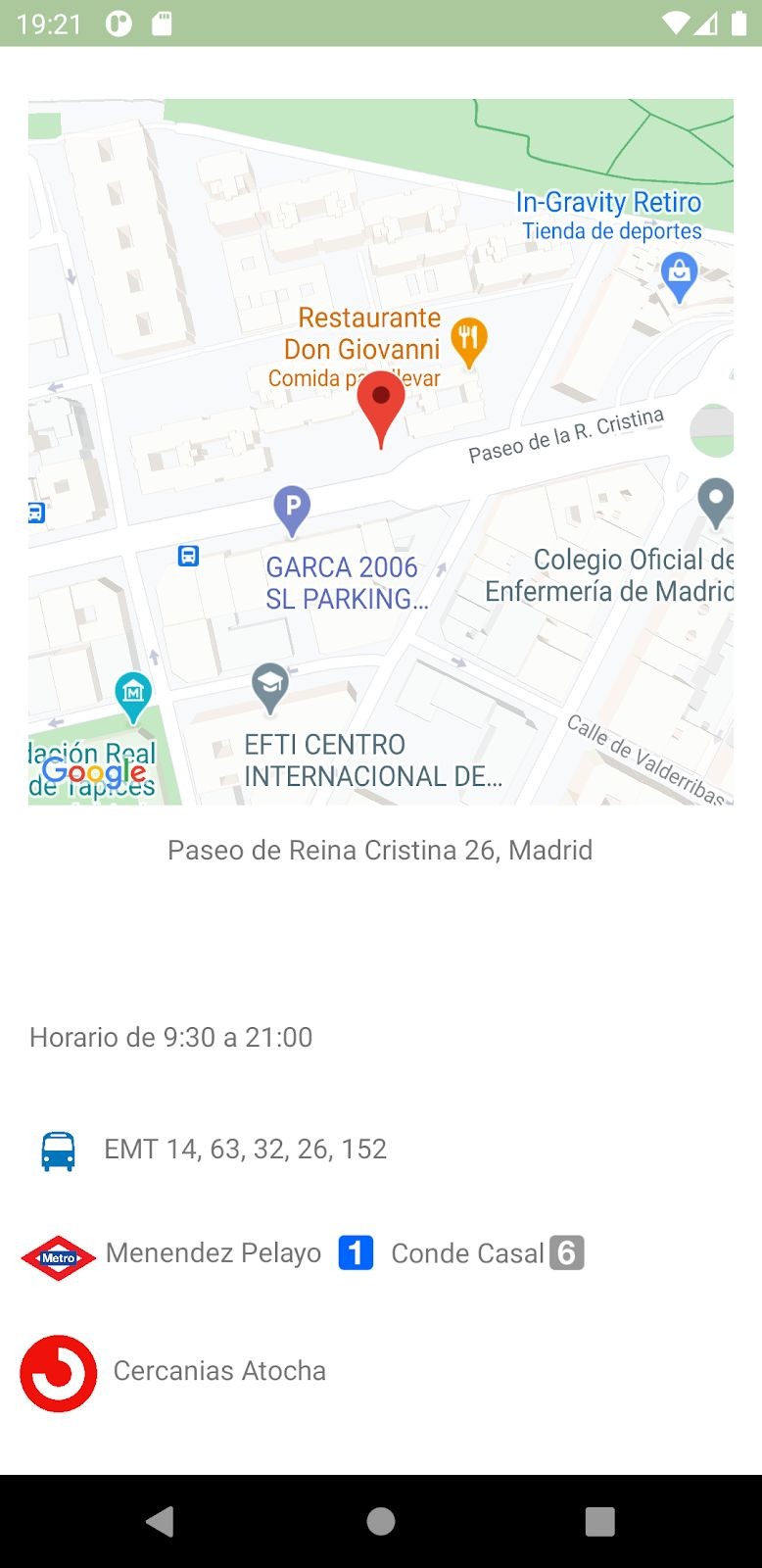
##### sendMessage

Este método lo utilizamos para que envíe un correo electrónico al usuario que pide el alta de la cita con los datos específicos de este y de la cita solicitada. Primero creamos un ProgressDialog el cual saldrá en pantalla indicando que se está enviando el correo, y a través de un hilo enviamos el correo para que no detenga el proceso del Dialog y para que una vez termine y envíe el correo el hilo cerrará el Dialog indicando que el correo ha sido enviado.

##### getDNIVet

Este método lo usamos para obtener el DNI del veterinario, el cual usaremos más adelante a la hora de hacer el resto de consultas. Usamos la conexión a la base de datos y con un ResultSet seleccionamos el DNI del veterinario basado en el tipo de raza elegida en la app, para almacenarlo en un String.

## ¿DÓNDE ESTAMOS?

* **Breve descripción**

En este Activity lo que queremos mostrar es la situación de la clínica y las formas alternativas para poder llegar a ella, para ello implementamos un fragment con google maps, donde señala la dirección de la clínica veterinaria y unos textviews con la dirección, el horario y los distintos tipos de transporte público que se encuentran cerca de la clínica.

* **Elementos**

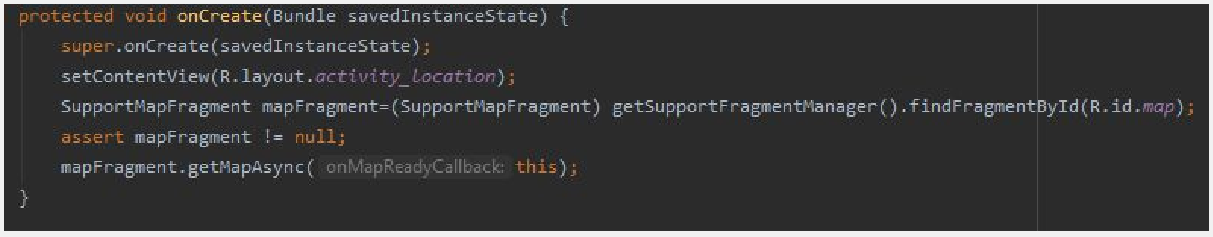
Lo primero que hacer es en nuestro archivo app/build.gradle añadimos las siguientes líneas en el bloque dependencies:



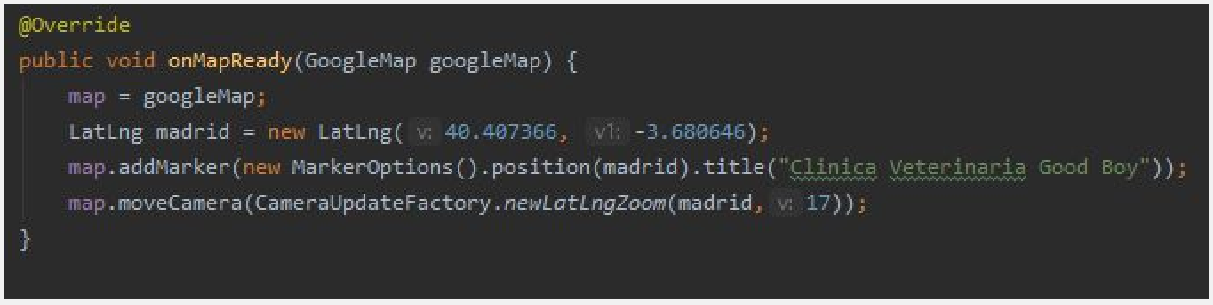
Luego creamos el fragment donde se verá el mapa:



Seguidamente iremos a la clase del activity y en el método onCreate pondremos lo siguiente para implementarlo:



Y en el siguiente método configuramos el marcador de google con las coordenadas de la clínica, el nombre y el zoom con el que veremos el mapa:

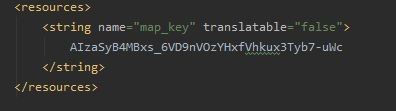


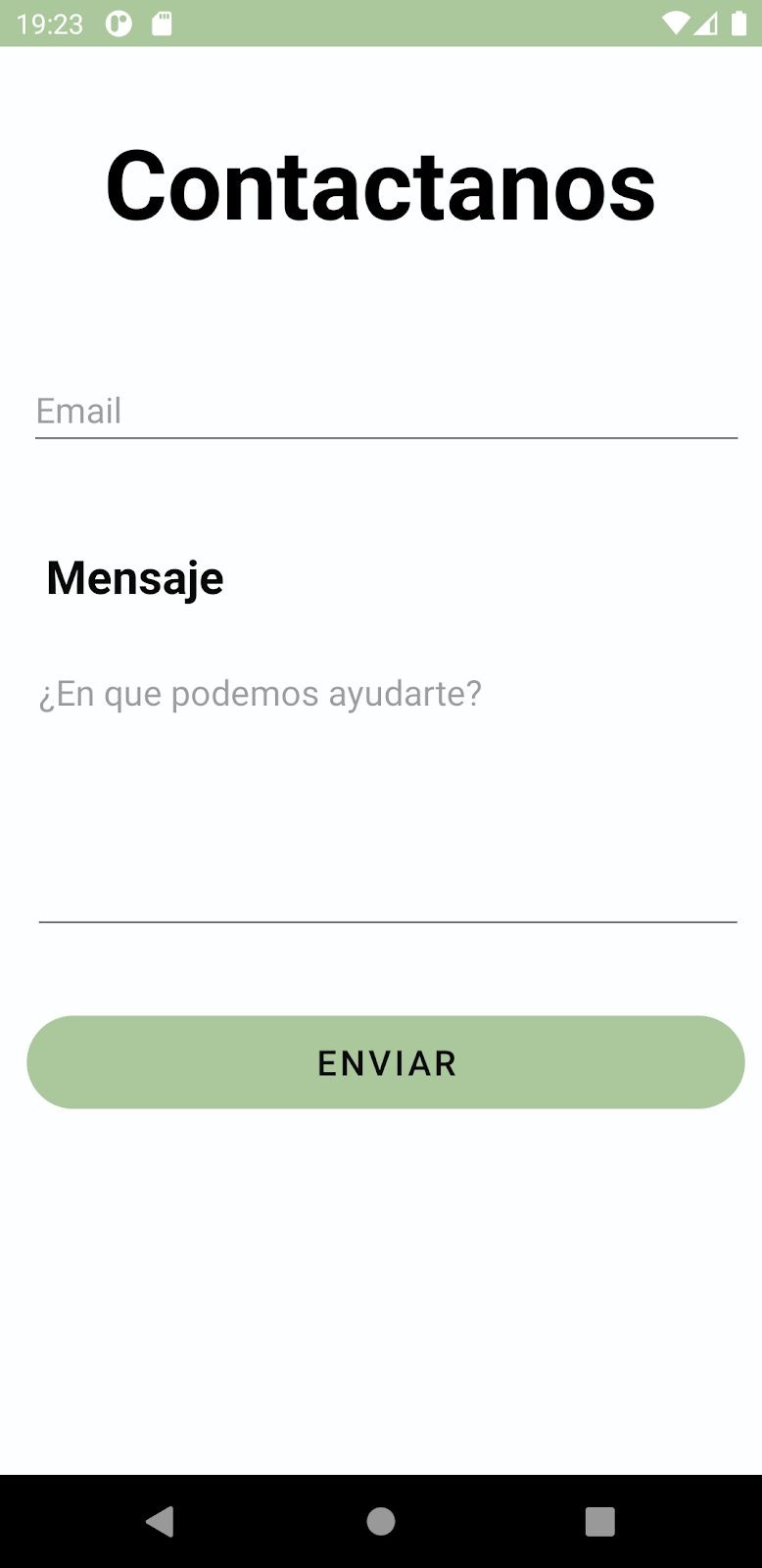
Y por supuesto lo declaramos en el manifest:



Por último debemos generar un API\_KEY para nuestro proyecto, para ello necesitamos ingresar a la [consola de desarrolladores de google](https://console.developers.google.com/), crear una aplicación, habilitar el [API de](https://console.developers.google.com/apis/library/maps-android-backend.googleapis.com) [Maps para Android](https://console.developers.google.com/apis/library/maps-android-backend.googleapis.com), [crear una Credencial](https://console.developers.google.com/apis/credentials) tipo API Key y añadir las restricciones para la Aplicación de Android. Cabe decir que esta Key sólo puede usarse con un móvil que tenga la cuenta de google con la que se ha creado, por lo tanto, solo podremos ver el mapa en dispositivos donde este esa cuenta, para poder generar una key para usarse en cualquier dispositivo, necesitaremos registrar la aplicación en la Google Play Store.

Crearemos un recurso llamado map\_api.xml y ahí pegamos la clave:





**- Clases necesarias**

## CONTACTANOS

* **Breve descripción**

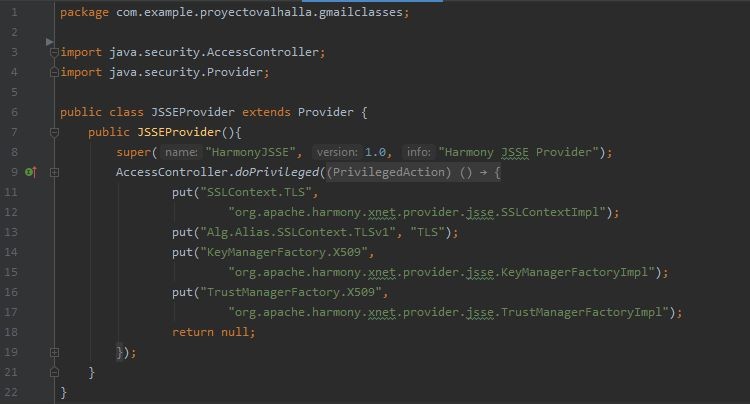
Este apartado de la aplicación se encarga de gestionar las dudas o preguntas de los posibles clientes hacia la clínica. En esta activity se muestra un EditText para recoger el email al que quieres que se te envíen las respuestas a tus dudas, y debajo de este un Multiline Text para escribir la duda que tengas. Una vez rellenados los campos pulsas el botón de "SEND" para que mediante un hilo se envie al Mail especificado anteriormente un correo por defecto desde una cuenta de Gmail que creamos para la app con un mensaje por defecto diciendo que ya le contactaremos y después de dos saltos de línea la duda que escribió en el activity.

* **Imports y librerías**

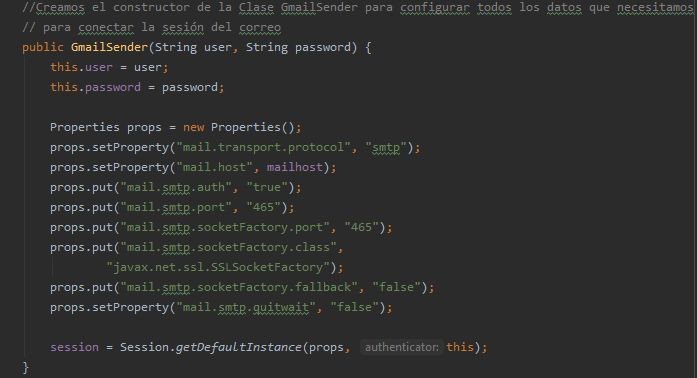
Para esta parte del proyecto se requieren 3 librerías y sus implementaciones en el Gradle de la app. Son activation.jar , addicional.jar y mail.jar. Aparte necesitaremos añadir al Manifest las líneas de ACCESS\_NETWORK\_STATE e INTERNET.

Aparte de la clase MainActivity, necesitaremos crear 2 clases extra para la funcionalidad del correo electrónico, son JSSEProvider y GmailSender (Son los nombres que les he dado a las clases).

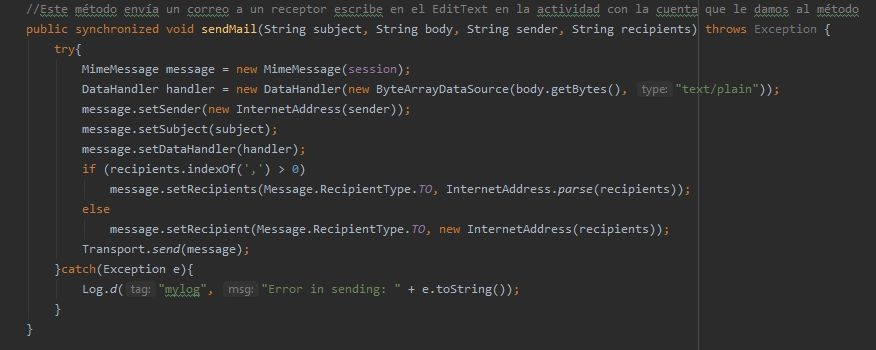
**JSSEProvider :** Esta clase la creamos para generar una conexión segura a través del cifrado y encriptación de los datos. Esta clase se extiende de la clase Provider.



**GmailSender :** Esta clase consta de un constructor que utilizaremos para establecerlos datos necesarios para iniciar sesión con el Gmail que utilizaremos para enviar el correo, estableciendo los datos necesarios como puertos y protocolos.

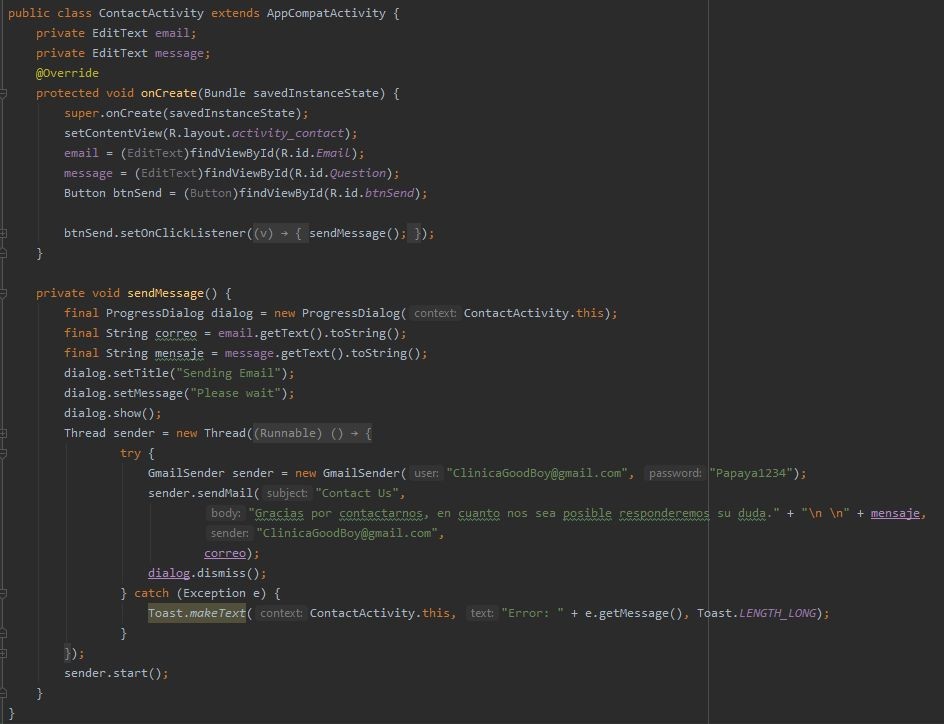


Y un método que utilizaremos para enviar al usuario el mail con el correo que proporciona en el EditText y la duda escrita en el Multiline Text.



**Clase ContactActivity**

En el ContactActivity instanciamos el EditText, el Multiline Text y el Button y crearemos el método sendMessage(), que almacenará los datos obtenidos en los EditText en Strings y creamos el método sendMessage(), el cual se encargará de crear un ProgressDialog que aparecerá al dar al botón de enviar y además creará un hilo que ejecutará el método sendMail() de la clase GmailSender recogiendo los datos del EditText y del Multiline Text para añadirlos al método y así enviar el correo para después cerrar el ProgressDialog.



# CONCLUSIONES Y FUTURAS MEJORAS

Debido a la situación actual que estamos pasando con la pandemia, es necesario más que nunca informatizar ciertas prácticas tan cotidianas y normales como ir a hacer la compra semanal, una cita médica o como en el caso de nuestra aplicación, una cita para una clínica veterinaria. A la larga, cada vez será más necesaria este tipo de aplicaciones para cualquier cosa y creemos que hacer una interfaz sencilla e intuitiva es esencial para poder llegar a cualquier rango de edad, porque a día de hoy hasta las personas mayores utilizan smartphones y a veces mucho mejor que las personas jóvenes.

Como futuras mejoras para la aplicación, a parte de un desarrollo para dispositivos iphone, sería posible añadirle un apartado de inicio de sesión para poder acceder a la ficha clínica de sus mascotas y poder ver sus informes, tratamientos, facturas, etc… además de la posibilidad de gestionar las citas ya realizadas para poder tanto consultarlas como modificarlas o eliminarlas y un chat en vivo para poder consultar con el veterinario cualquier problema o consulta que tengas.

# FUENTES

* + Apuntes Recibidos durante el curso
  + Logos diseñados por Alba Navarro
  + https://stackoverflow.com/
  + https://es.stackoverflow.com/
  + https://commons.wikimedia.org/wiki/Category:Madrid\_Metro\_icons
  + https://blog.ensalza.com/api-google-maps
  + https://console.cloud.google.com/
  + https://[www.sgoliver.net/blog](http://www.sgoliver.net/blog)
  + https://mscdroidlabs.es/
  + https://developer.android.com
  + https://material-ui.com/es/
  + [http://www.aprendeandroid.com](http://www.aprendeandroid.com/)
  + https://developers.google.com
  + [https://medium.com/@juancadh](https://medium.com/%40juancadh)
  + https://[www.w3schools.com](http://www.w3schools.com/)
  + https://stackoverrun.com/es/q/1511359
  + Canales de Youtube:
    - Sociedad Androide
    - pildorasinformaticas
    - Mario Ezquerro
    - Coding in Flow
    - Stevdza-San