Aplicación Presupuestos

Ismael Millán Pérez
David Gómez Rovira
M12 Proyecto

Índice de contenidos

Presentación

0. Presentación

Esta aplicación está pensada para una empresa de construcciones, de nombre **Euroconstrucciones**, para que los usuarios cuando naveguen por ella, puedan elaborar sus propios presupuestos para posteriormente ejecutarlos o no, en función del cliente. Esta será una web que se utilizará de forma real, es decir, tendrá todos los requisitos necesarios para poder darle salida a Internet.

Esta empresa, como su nombre indica, se dedica a la construcción y lleva en funcionamiento desde 1999. Es una empresa familiar con una dilatada experiencia en el sector. Este proyecto va a versar en la creación de una página web para elaborar presupuestos de los distintos tipos de obra que ejecuta la empresa.

Estos son:

- **Obra nueva**: construcción de chalets, casas pareadas, oficinas, naves industriales, piscinas, etc.
- **Reformas**: pisos completos, baños, cocinas, instalaciones de luz y agua, reformas en general.

Para esta aplicación, están contemplados 3 tipos de obra:
> Obra nueva
> Reformas:
    > Baño
    > Cocina

Esta aplicación no solo no estará cerrada a estos tipos, sino que quedará abierta a cualquier tipo de reforma, a parte de los que ya hay contemplados.

Breves especificaciones:

- Los usuarios dispondrán de partes que serán visibles para todos, mientras que habrá partes que serán expresamente para usuarios registrados en la misma. A su vez, dependiendo del tipo de usuario, se podrán visualizar o no, algunos contenidos.

- La creación de presupuestos, así como su guardado y visualización en listas está limitada a los usuarios que decidan registrarse en la aplicación.

- Un cliente podrá elaborar varios presupuestos de diferentes precios, en función de la calidad de los materiales a contratar. La calidad puede ser baja, media o alta. Puede realizar tantos presupuestos como desee en función de sus necesidades y su poder adquisitivo.

- Los presupuestos se guardarán en una base de datos, para que el administrador de la página, en una sección, pueda ver los presupuestos que han elaborado los distintos usuarios registrados.

- También dispondrán en la parte inferior de la pantalla la dirección y el teléfono de contacto de la empresa, para así facilitar la comunicación entre empresario y clientes.

Tecnologías utilizadas

1.Tecnologías utilizadas

Para este proyecto, las tecnologías que se han utilizado han sido varias.

- **Eclipse**, para programar el webservice.
- **Android Studio**, para programar el cliente web.
- **Spring Boot**, que es una herramienta para desarrollar el web.
- **Spring Security**, integrado en Spring Boot.
- **Lenguajes de programación**: Java de forma íntegra.

Spring Boot

Spring es un framework de Java que se utiliza para crear aplicaciones que posteriormente serán utilizadas en una máquina virtual Java.

Spring Boot es una herramienta creada en 2012 que reduce la complejidad de las aplicaciones basadas en Spring, construyéndolas para una mayor facilidad. Las configuraciones de esta herramienta son automáticas, además de utilizar un servidor tomcat, por el cual desplegaremos la aplicación web.

> A la hora de construir las vistas se ha utilizado Thymeleaf.
> Las configuraciones se encuentran en un único fichero .java. Es donde se configuran los filtros de seguridad que incorpora Spring Security. Aquí es donde otorgamos acceso a las vistas, indicamos los métodos para iniciar/cerrar sesión, indicamos quién tiene acceso y quién no a las páginas.
>
> Todo esto lo manejará Spring bajo la anotación @Bean.

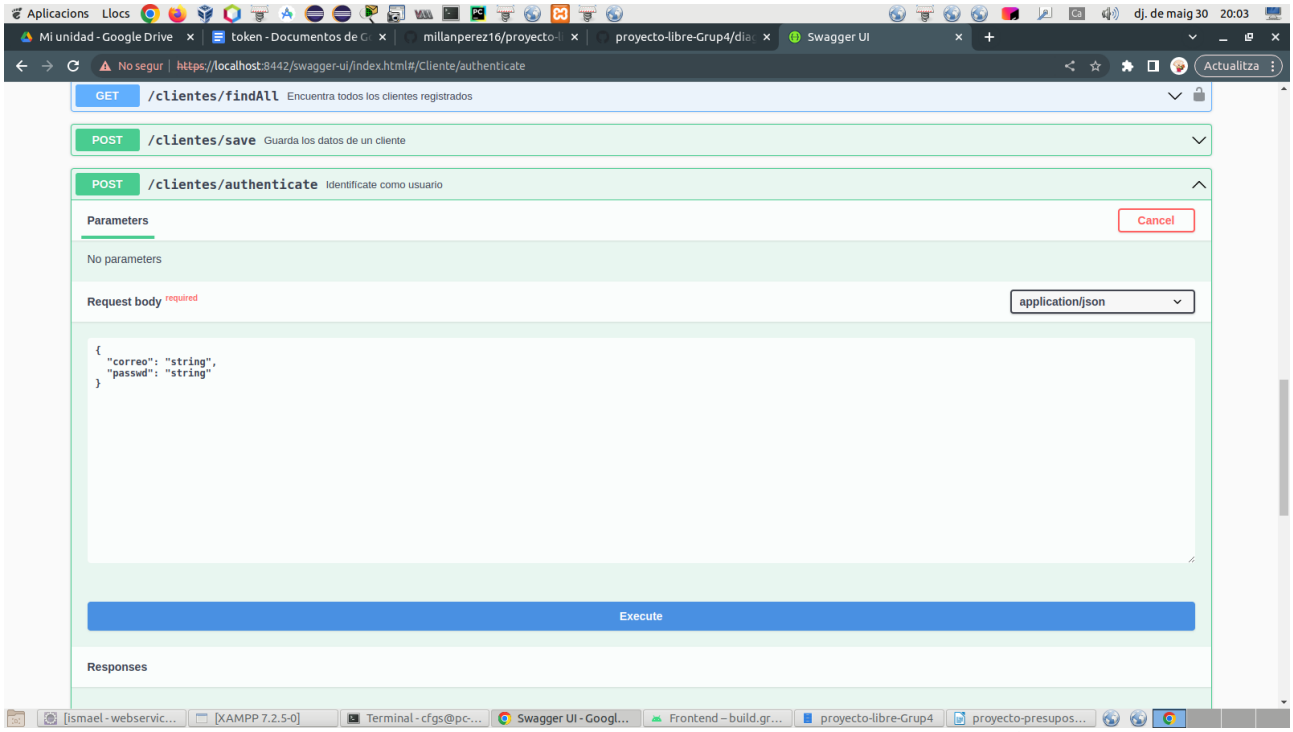
La estructura del webservice está realizada en forma de paquetes, donde estarán los siguientes:

- **Main**: Contiene dos ficheros, uno que arranca la aplicación y otro que serializa las contraseñas.
- **Controller**: Contiene los controladores que hay. Estos sirven para enviar información de la vista al modelo y viceversa.
- **Service**: Contiene las clases que agrupan las funcionalidades que tienen los métodos que están guardados en los repositorios.
- **Repository**: Contiene las clases que disponen de métodos y funcionalidades que comunican a su vez con la base de datos para gestionar la información con la que trabaja nuestra aplicación web, ya sea desde guardarla hasta eliminarla, pasando por actualizarla, etc...
- **Domain**: Aquí se guardan las clases de las que beberá nuestra aplicación, es donde se encuentra el modelo. Con estas clases, interactuamos con las vistas para enviar datos a través de los controladores.
- **Security**: Aquí es donde daremos acceso a las vistas, es decir, podremos crear listas blancas y negras. Esta aplicación dispone de ambas, lista blanca para una serie de vistas que todo el mundo podrá ver, y lista negra para que solo los usuarios registrados puedan ver. También aquí se configura el acceso a la base de datos para cuando hagamos login, el logout, etc.

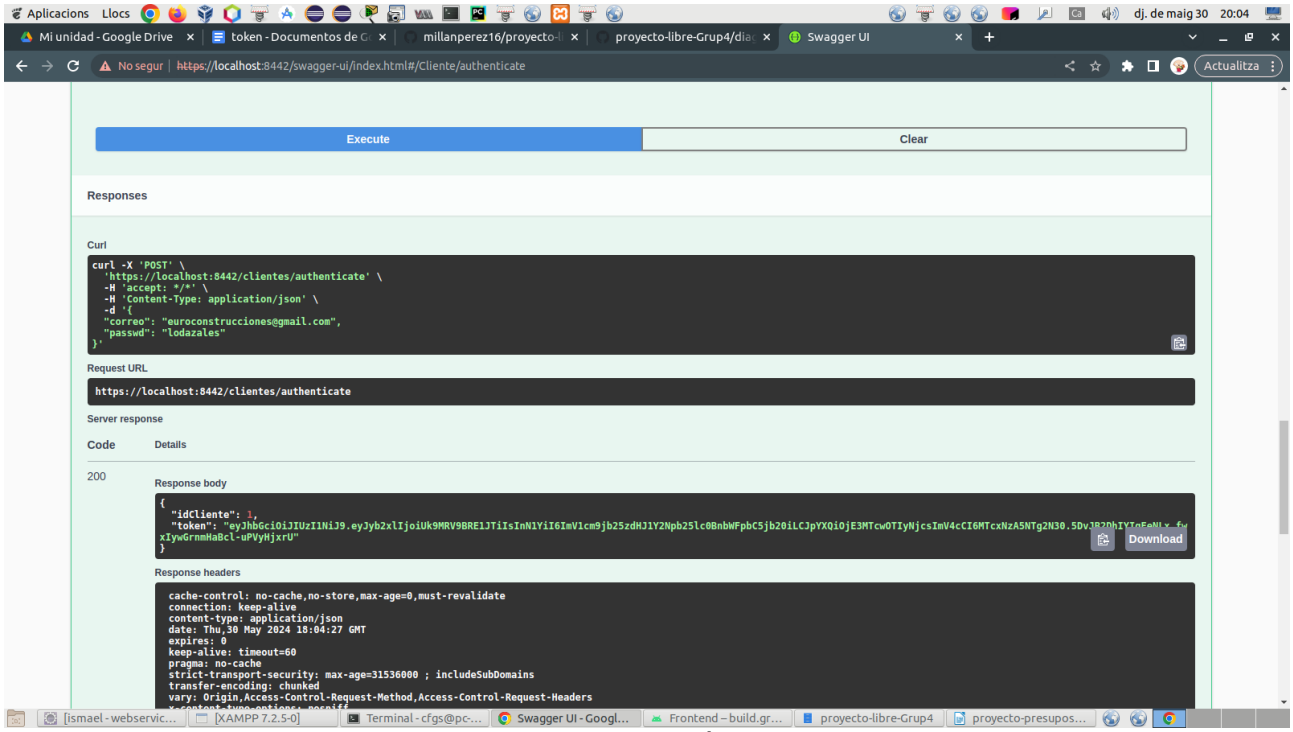
- **Token**: Almacena la respuesta del servidor al cliente con el token del usuario con el que se autentica, el filtro de autenticación del token y el servicio del mismo.
- **Details**: Contiene un fichero el cuál comunicará el repositorio de cliente con la base de datos, para cuando al hacer login, Spring encuentre el usuario que le estamos enviando en nuestra base de datos.
- **Validators.groups**: Almacena las validaciones del

AUTENTICACIÓN BASADA EN TOKENS

Este webservice, a la hora de autenticarse, se basa en tokens. Esto es, tenemos este método:



A la hora de autenticar, le pasaremos un email y una contraseña.



Cuando hacemos la llamada, este devolverá el id del cliente y su token, que es esa larga cadena de la captura. Ese token será el que nuestro cliente (Android en el caso de nuestra app) maneje para las llamadas al webservice. Contiene información como por ejemplo, el usuario al que pertenece, y nos servirá para validar la conexión del cliente con el webservice.

Nuestro webservice podrá hacer esto:
→ **Registrar, actualizar, eliminar y obtener información de los usuarios.**
→ **Guardar datos relativos a todos los tipos de presupuesto.** Podremos guardar Obras, Reformas de aseo y Reformas de cocina, así como **actualizar los datos de un presupuesto ya existente**.
→ **Obtener información de los presupuestos, actualizarlos** e incluso **eliminarlos**.

Todo esto se hará desde los controladores, que siguiendo la lógica de una aplicación Spring Boot, llamará a los servicios, a los repositorios y a su vez a la base de datos para ejecutar todas esas acciones a través de las llamadas desde el cliente.

WEBSERVICE

Pero antes de nada, ¿qué es un webservice?
Es una herramienta que permite que aplicaciones diferentes entre sí se comuniquen, intercambiando datos.
Esos datos dependiendo del tipo de nuestro servicio, pueden ser:

- **REST**: JSON más comúnmente, aunque puede usar XML.
- **SOAP**: XML.

El intercambio de datos se produce en base a una petición que recibe y el webservice genera una respuesta para posteriormente enviarla al cliente.

Es similar a una API, que realiza las mismas acciones, aunque no se limita a la web, sino que es multiplataforma. Por ejemplo, imaginemos que queremos usar la api de Gmail. Esta permite acceder a sus funcionalidades sin necesidad de usar la web y desde una aplicación diferente.

Como nuestra app está haciendo llamadas al webservice para devolverlas al cliente, estas son las llamadas que hace a cada uno de los controladores:

Métodos para Cliente

Método	Operación	Parámetros	Acción
/clientes/ save	POST		Guardará un cliente.
/clientes/ authenticate	POST	Correo, password String	Haremos el login desde aquí. Necesitaremos pasarle un **correo** y una **password**. **Nos devolverá un token y un id de cliente.**
/clientes/ update	PUT		Actualizará los datos del cliente. Funciona de forma **similar al save**.
/clientes/ findAll	GET	Pagina. Número entero	**Mostrará una lista con todos los clientes**. **Tendremos que pasarle un número como parámetro que será la página**. (Cada 5 clientes, carga una página)
/clientes/ findById/{id}	GET	Id. Número entero	Mostrará un cliente concreto por su id. Tendremos que pasarle su id de base de datos como parámetro
/clientes/ findByCorreo /{correo}	GET	Correo. String	Mostrará un cliente concreto por su email. Tendremos que pasarle su email como parámetro.
/clientes/ deleteById/ {id}	DELETE	Id. Número entero	**Eliminará un cliente de la base de datos**. Tendremos que pasarle su id de base de datos como parámetro

Métodos para Presupuesto

Método	Operación	Parámetros	Acción
/presupuestos/ init	GET		Se encarga de inicializar los parámetros que utilizará el cliente para calcular los presupuestos
/presupuestos/ findById/{id}	GET	Id. Número entero	Mostrará un presupuesto concreto por su id. Tendremos que pasarle su id de base de datos como parámetro
/presupuestos/ findByCliente/ {correo}	GET	Correo. String	Mostrará todos los presupuestos pertenecientes al usuario que le pasemos como parámetro. Tendremos que pasarle su email como parámetro.
/presupuestos/ findAll	GET	Pagina. Número entero	**Mostrará una lista con todos los presupuestos**. **Tendremos que pasarle un número como parámetro que será la página**. (Cada 5 presupuestos, carga una página)
/presupuestos/ deleteById/{id}	DELETE	Id. Número entero	Eliminará un presupuesto de la base de datos. Tendremos que pasarle su id de base de datos como parámetro

Métodos para Aseos

Método	Operación	Parámetros	Acción
/presupuestos/ aseos/ findAllAseos	GET	Pagina. Número entero	**Mostrará una lista con todos los presupuestos de reformas de aseo**. **Tendremos que pasarle un número como parámetro que será la página**. (Cada 5 presupuestos, carga una página)
/presupuestos/ aseos/ saveRefAseo	POST		Guardará un presupuesto de reforma de aseo.
/presupuestos/ aseos/ updateRefAseo	PUT		Actualizará un presupuesto de reforma de aseo.

Métodos para Cocina

Método	Operación	Parámetros	Acción
/presupuestos/ cocina/ findAllCocina	GET	Pagina. Número entero	**Mostrará una lista con todos los presupuestos de reformas de cocina**. **Tendremos que pasarle un número como parámetro que será la página**. (Cada 5 presupuestos, carga una página)
/presupuestos/ cocina/ saveRefCocina	POST		Guardará un presupuesto de reforma de cocina.
/presupuestos/ cocina/ updateRefCocina	PUT		Actualizará un presupuesto de reforma de cocina.

Métodos para ObraNueva

Método	Operación	Parámetros	Acción
/presupuestos/ obranueva/ findAllObra	GET	Pagina. Número entero	**Mostrará una lista con todos los presupuestos de reformas de cocina**. **Tendremos que pasarle un número como parámetro que será la página**. (Cada 5 presupuestos, carga una página)
/presupuestos/ obranueva/ saveObra	POST		Guardará un presupuesto de obra nueva.
/presupuestos/ cocina/ updateObra	PUT		Actualizará un presupuesto de obra nueva.

Todas estas operaciones forman parte del webservice.

APIS EN EL LADO DEL CLIENTE

Para facilitar la comunicación entre la aplicación y los diferentes servicios, se ha utilizado la librería conocida como Retrofit. Ha sido desarrollada por Square, el mismo grupo de desarrolladores de OkHttp, y su propósito es agilizar la implementación de clientes HTTP. Es esencialmente un cliente REST para Android y Java que transforma un cliente HTTP en una interfaz Java, y se sirve de la librería OkHttp para ello.

Podemos poner un ejemplo de su implementación conectándonos a la API de la Diputación de Barcelona para obtener los nombres de sus municipios. Es una API abierta que para pedir datos no requiere autenticación ni registro.

Comenzamos importando las dependencias necesarias, que de entrada son la propia de Retrofit y la de Gson Converter. Gson es una librería que facilita la serialización y deserialización de objetos JSON.

```
dependencies {
    implementation 'androidx.appcompat:appcompat:1.6.1'
    implementation 'com.google.android.material:material:1.5.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.4'
    testImplementation 'junit:junit:4.13.2'
    androidTestImplementation 'androidx.test.ext:junit:1.1.5'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.5.1'
    implementation 'com.squareup.retrofit2:retrofit:2.11.0'
    implementation 'com.squareup.retrofit2:converter-gson:2.11.0'
    implementation 'com.squareup.okhttp3:logging-interceptor:4.12.0'
    implementation 'com.squareup.okhttp3:okhttp-tls:4.12.0'
    implementation 'androidx.preference:preference:1.2.1'
    implementation 'com.android.volley:volley:1.2.1'
    implementation 'org.bouncycastle:bcprov-jdk15to18:1.78'
```

Con esto podemos crear una interfaz que contendrá todos los métodos que consideremos necesarios.

```
public interface ApiService {

    1 usage   dgr-85
    @GET("municipis")
    Call<Municipi> getMunicipi();
```

La clase Call es propia de Retrofit y se utiliza para realizar las llamadas a WebServices. La anotación GET (junto a POST, PUT, etc.) indican el tipo de llamada se hará con este método, y el parámetro que la acompaña es la rama de la URL base a la que acudir. La clase Municipi se ha creado para recoger los datos que nos interesan del objeto JSON que será devuelto por la API.

Sabemos gracias a la documentación de la API cuál es la URL base a la que hay que llamar y también qué objeto JSON devuelve.

En este caso, podemos ver que devuelve un único objeto JSON con una serie de atributos, entre los cuales destaca un array llamado “elements” que representa la totalidad de municipios de Barcelona. Dentro de la cada elemento de este array destaca el atributo “municipi_nom”, que es el que nos interesa recoger.

Creamos una clase para los métodos que llamarán a las APIs.

```
public class ApiServiceImpl {
    18 usages
    private static ApiService apiService;
    1 usage
    private static final String URL_MUNICIPIS = "https://do.diba.cat/api/dataset/municipis/";
```

Definimos el método.

```
public static ApiService getApiServiceMunicipi(String like) {
    // Creamos un interceptor y le indicamos el log level a usar
    final HttpLoggingInterceptor logging = new HttpLoggingInterceptor();
    logging.setLevel(HttpLoggingInterceptor.Level.BODY);

    // Asociamos el interceptor a las peticiones
    final OkHttpClient.Builder httpClient = new OkHttpClient.Builder();
    httpClient.addInterceptor(logging);

    if (apiService == null) {
        Retrofit retrofitSingleton = new retrofit2.Retrofit.Builder()
                .baseUrl(URL_MUNICIPIS + "camp-municipi_transliterat-like/" + like + "/")
                .addConverterFactory(GsonConverterFactory.create())
                .client(httpClient.build())
                .build();
        apiService = retrofitSingleton.create(ApiService.class);
    }
    return apiService;
}
```

Atributos destacados:

- HttpLoggingInterceptor: es un objeto de OkHttp que registra información de peticiones y respuestas.
- OkHttpClient: objeto que realiza las llamadas propiamente dichas y lee la información de las respuestas recibidas.
- Retrofit: hace de puente entre el método abstracto que hemos declarado en la interfaz y la llamada HTTP en la que se convertirá. Las anotaciones declaradas en la interfaz son usadas para definir las peticiones.

Los parámetros del objeto Retrofit son:

- baseUrl: la URL a la que llamar. En el caso de esta API podemos añadir filtros para que sólo traiga los elementos que necesitemos. El string “like”, traído como argumento de la función, recoge el texto introducido por el usuario en el campo del municipio y lo usa para filtrar los municipios según el campo “municipi_transliterat” definido en la API.
- addConverterFactory: aquí se define un objeto capaz de serializar y deserializar objetos JSON.
- client: el cliente HTTP que se usará para realizar las peticiones.

Definimos las clases Java necesarias que representen las partes de los objetos JSON que nos interesa recoger. En este caso hemos visto que vienen muchos elementos, pero sólo nos interesa uno. Dado que el atributo que queremos forma parte de un atributo del objeto raíz, podemos optar por definir cada elemento en clases separadas.

```
public class Municipi {
    1 usage
    @SerializedName("elements")
    private ArrayList<Element> elements;
    1 usage   dgr-85
    public ArrayList<Element> getElements() { return elements; }
}
```

```
public class Element {
    1 usage
    @SerializedName("municipi_nom")
    private String municipiNom;

    1 usage   dgr-85
    public String getMunicipiNom() { return municipiNom; }
}
```

La anotación “SerializedName” informa del nombre que hay que buscar entre los atributos del objeto JSON.

Teniendo las clases preparadas, definimos la llamada en la actividad que llamará a la API.

```
    AutoCompleteTextView actv = findViewById(R.id.actvMunicipality);
    String like = actv.getText().toString();
    ApiService service = ApiServiceImpl.getApiServiceMunicipi(like);
    Call<Municipi> call = service.getMunicipi();
    dgr-85
    call.enqueue(new Callback<Municipi>() {
        dgr-85
        @Override
        public void onResponse(Call<Municipi> call, Response<Municipi> response) {
            if (response.body() != null) {
                loadDataList(response.body());
            }
        }

        dgr-85
        @Override
        public void onFailure(Call<Municipi> call, Throwable throwable) {
        }
    });
}
```

El método “enqueue” de la clase Call ejecuta el método “onResponse” si la llamada ha tenido éxito y “onFailure” si algo ha salido mal durante el proceso.

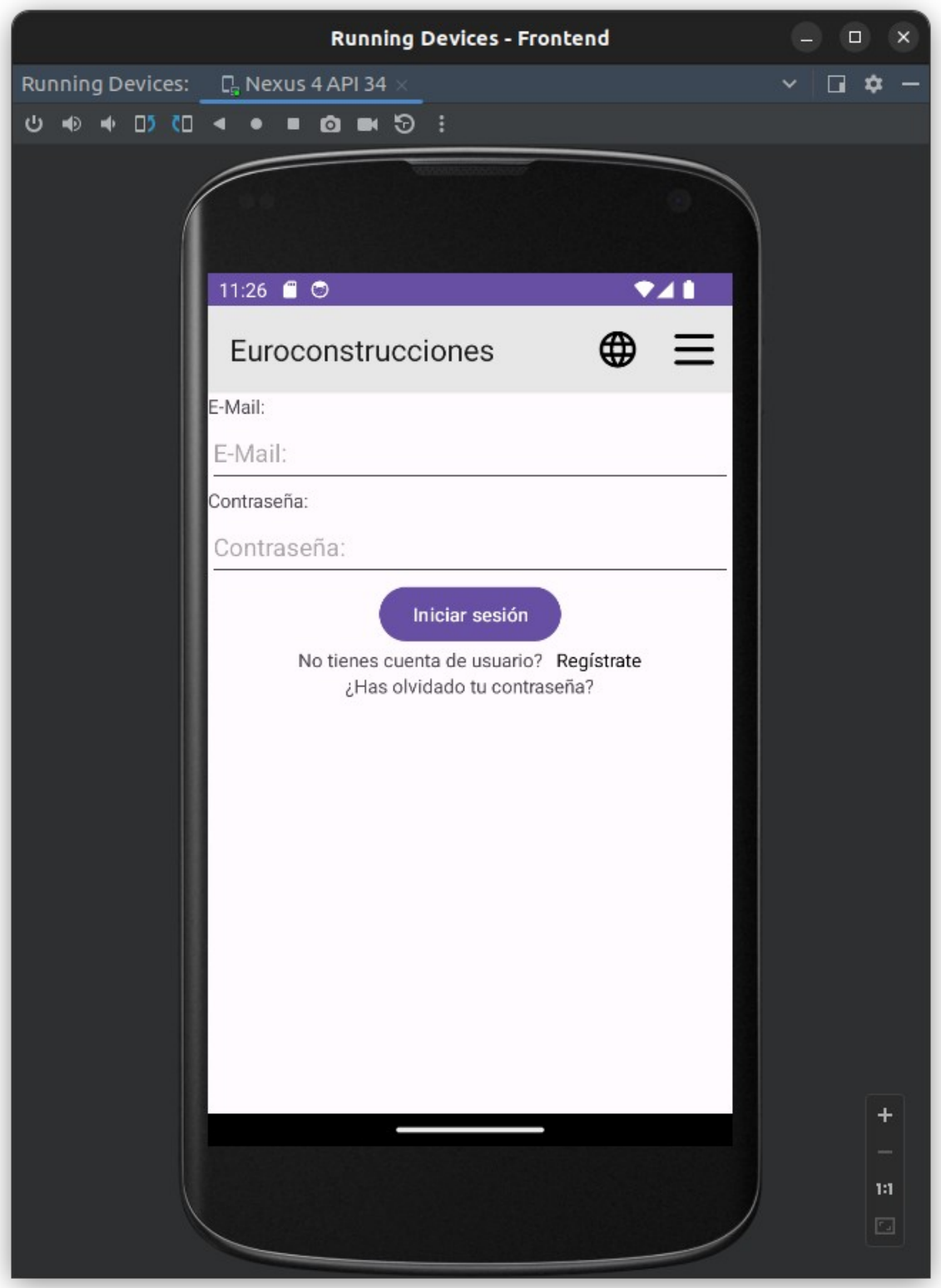
En este caso en particular, el método “loadDataList” carga los nombres de los municipios en un desplegable que se mostrará junto al campo de texto del municipio. Esto ya es propio de la aplicación y no guarda relación con Retrofit.

```
1 usage   dgr-85
private void loadDataList(Municipi municipi) {
    AutoCompleteTextView actvMunicipality = findViewById(R.id.actvMunicipality);
    actvMunicipality.setThreshold(3);
    List<Element> elements = municipi.getElements();
    List<String> names = new ArrayList<>();
    for (Element e : elements) {
        names.add(e.getMunicipiNom());
    }
    ArrayAdapter<String> adapter = new ArrayAdapter<>( context: this, android.R.layout.simple_list_item_1, names);
    actvMunicipality.setAdapter(adapter);
}
```

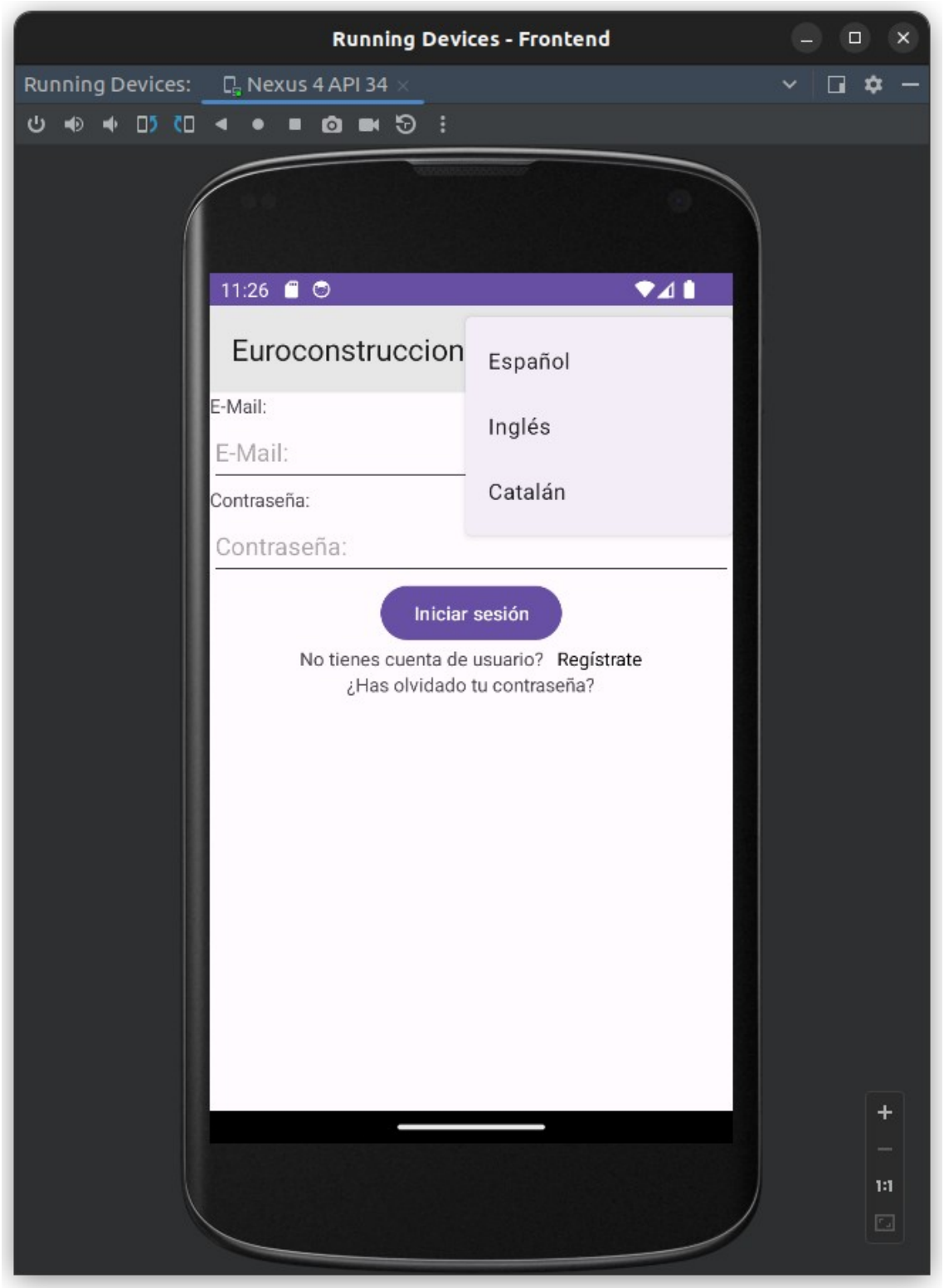
Diseño e implementación de la interfaz

2. Diseño e implementación de interfaz

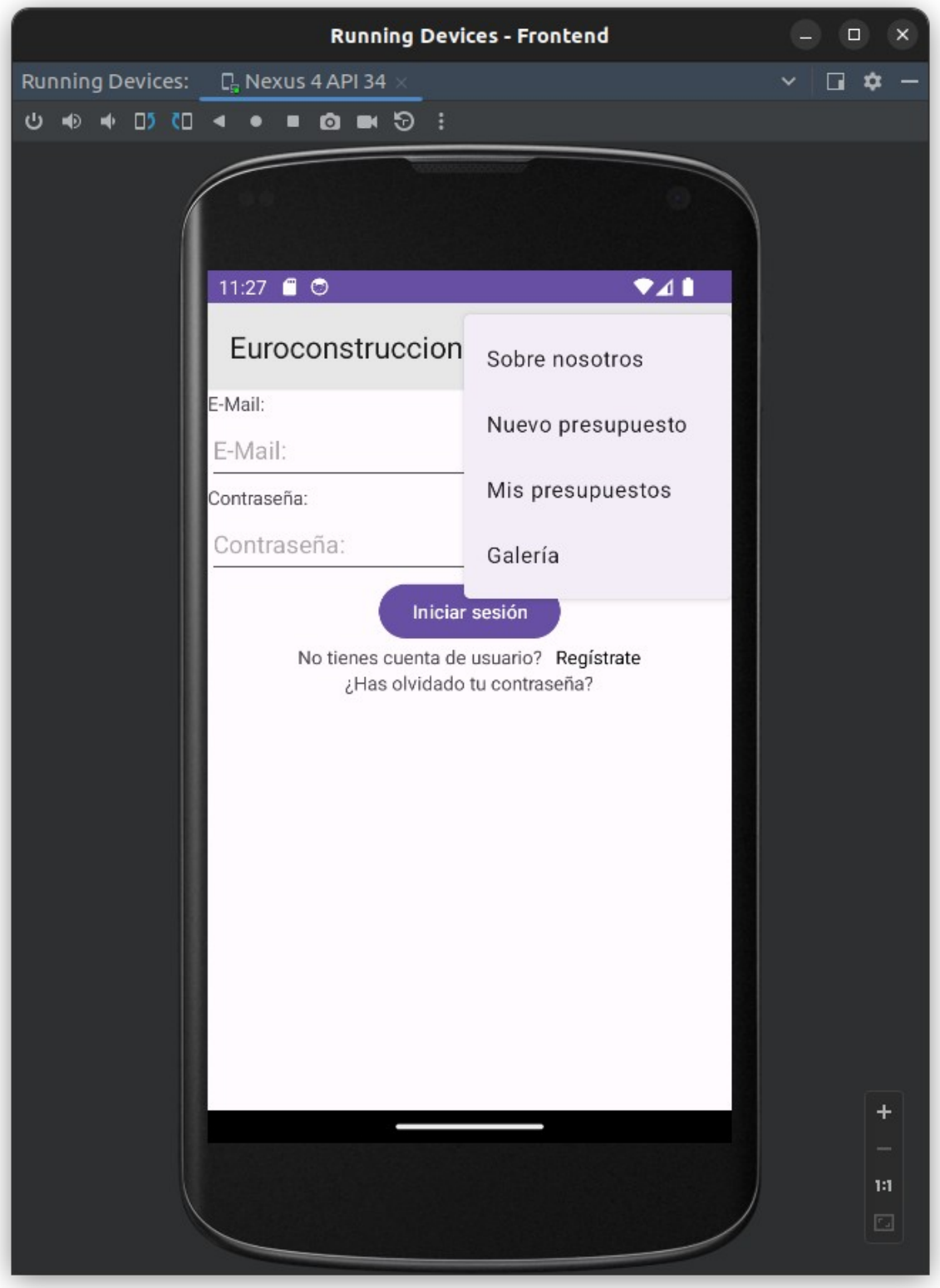
Dada la formalidad que suele rodear a la creación de documentos como son los presupuestos, se ha optado por una interfaz simple y funcional. La aplicación nos recibe con un formulario para iniciar sesión con correo y contraseña.



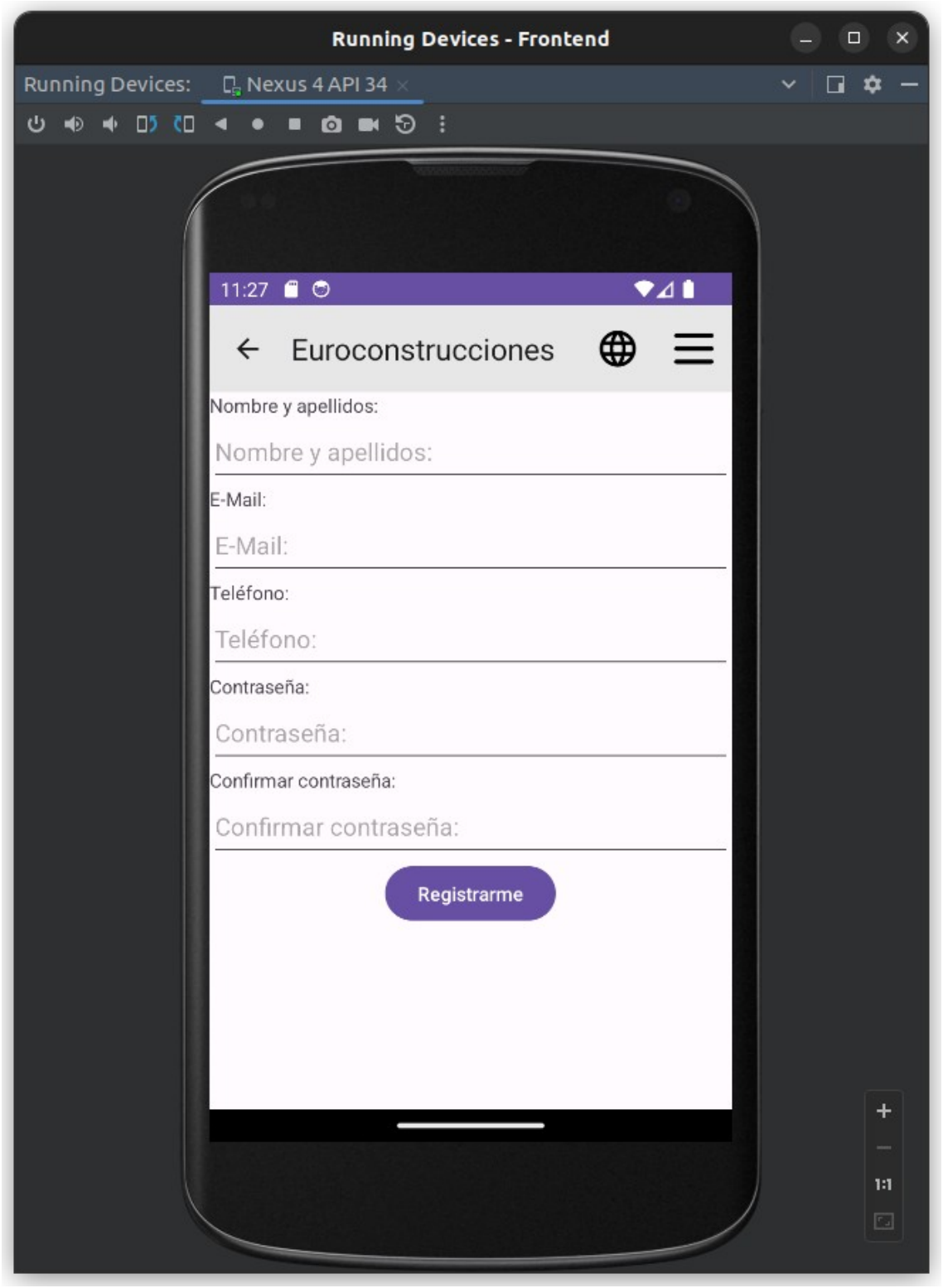
Desde cualquier sección de la aplicación, podemos cambiar el idioma pulsando el icono de la bola del mundo. Podemos elegir entre español, catalán o inglés.



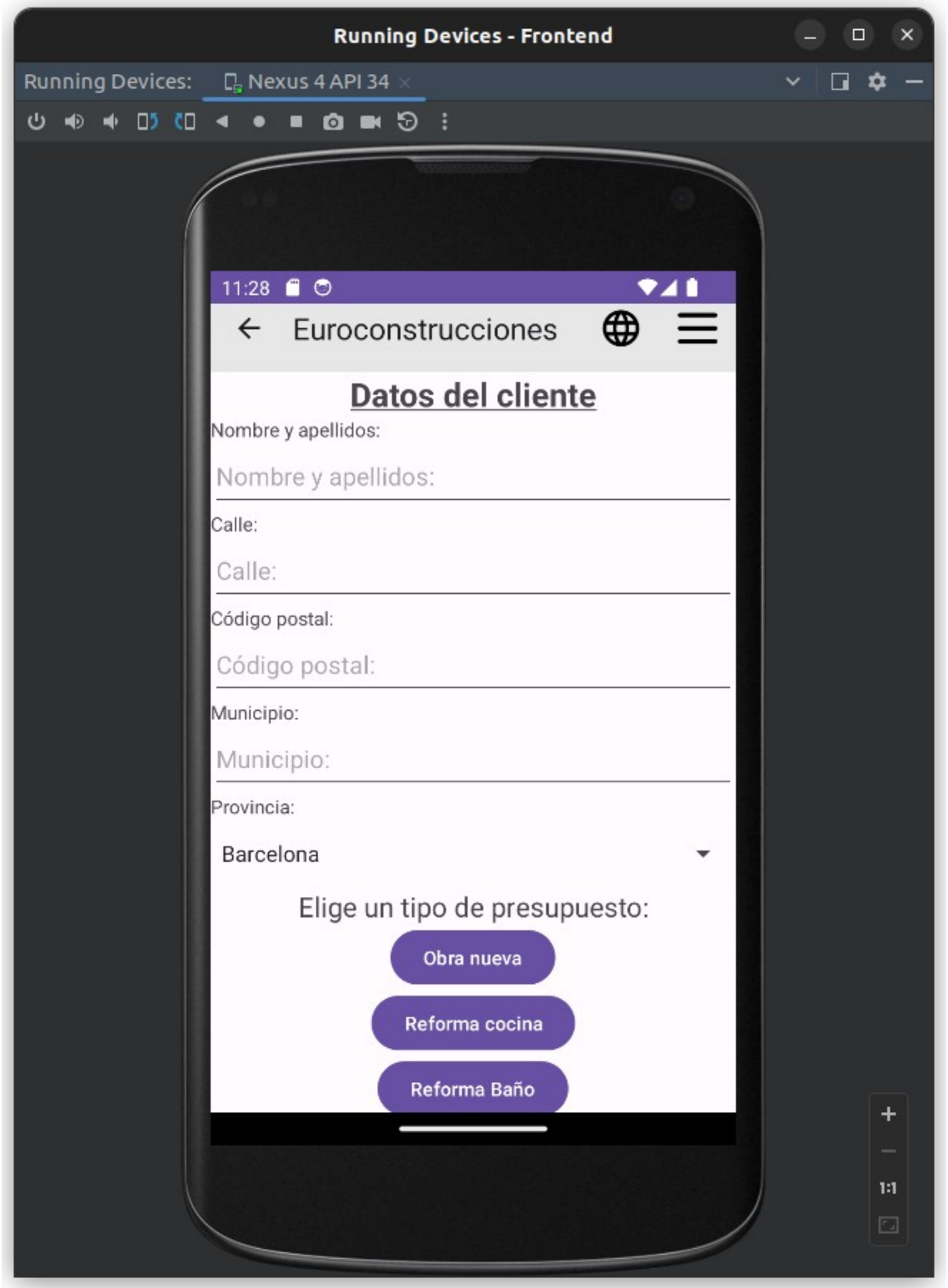
También disponemos de un pequeño menú que muestra una galería de imágenes, informa sobre la empresa, o en caso de haber iniciado sesión, permite crear nuevos presupuestos o ver los ya creados.



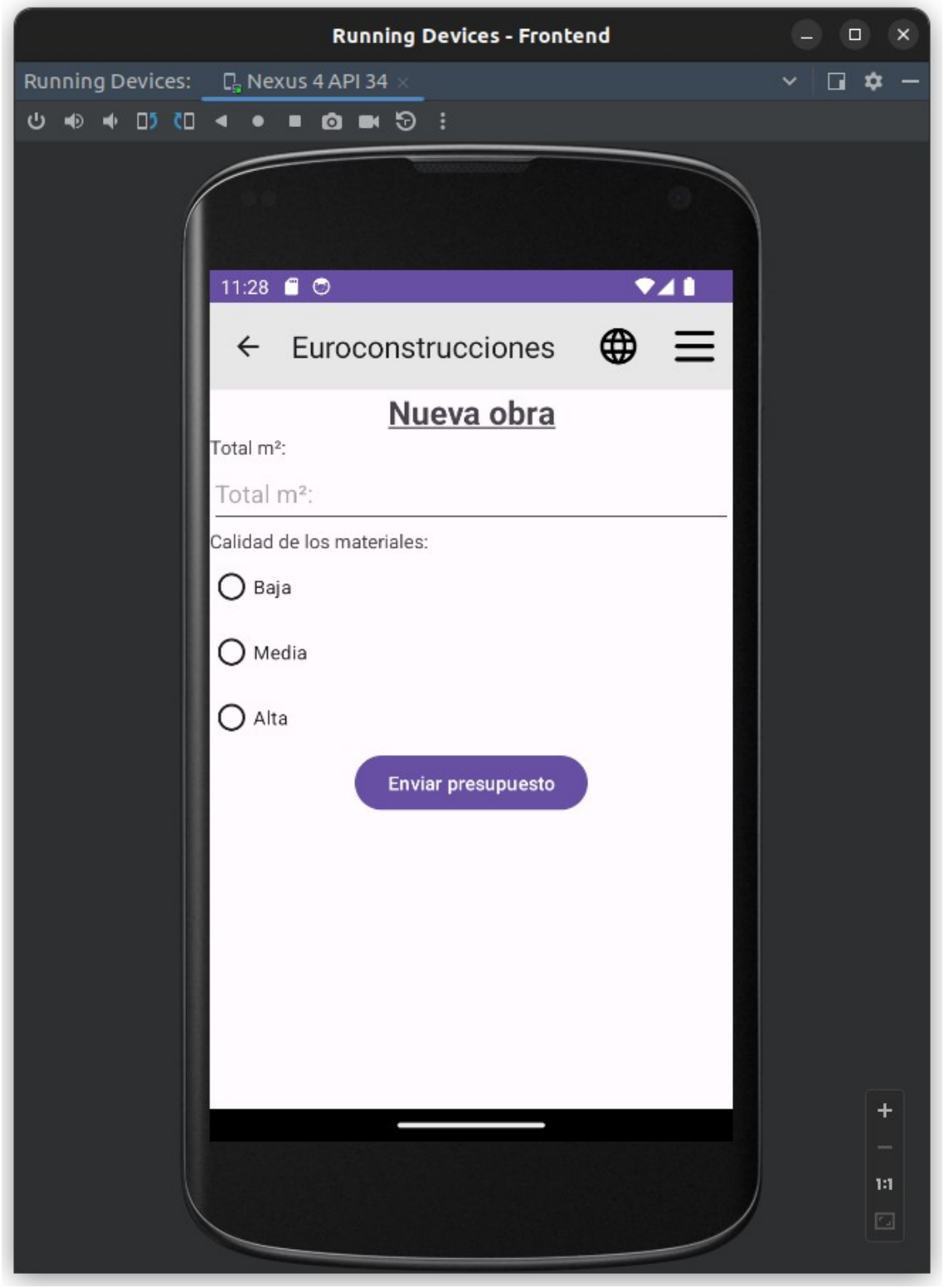
Pulsando la palabra “Regístrate” bajo el botón de iniciar sesión, se nos muestra un formulario para registrarnos.



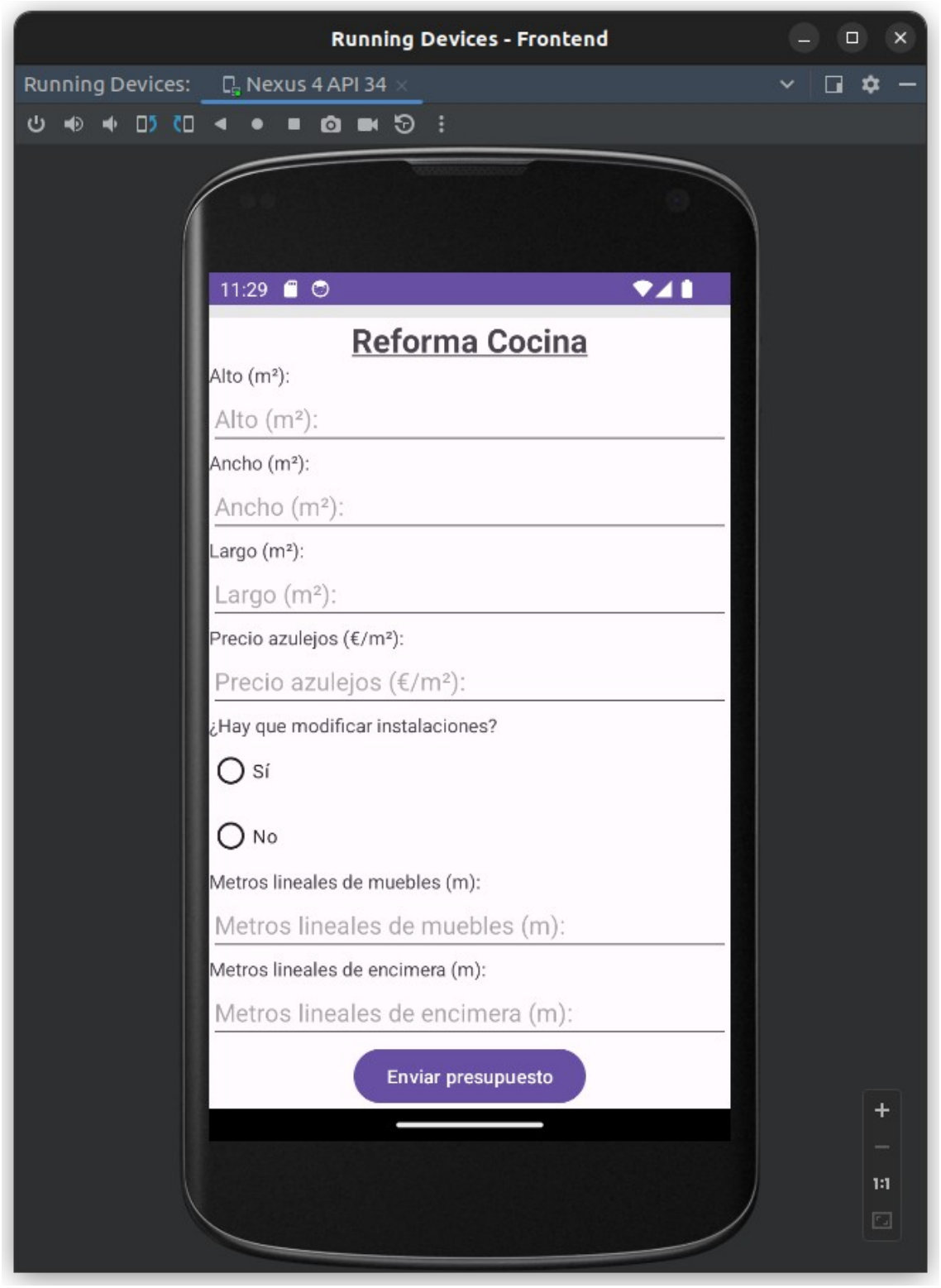
Cuando creamos un nuevo presupuesto, el formulario tiene varios pasos. En primer lugar rellenamos los datos del lugar donde se hará el trabajo y los datos de contacto de la persona que lo encarga.


Running Devices - Frontend
Running Devices:
Nexus 4 API 34
11:28
Euroconstrucciones
Datos del cliente
Nombre y apellidos:
Nombre y apellidos:
Calle:
Calle:
Código postal:
Código postal:
Municipio:
Municipio:
Provincia:
Barcelona
Elige un tipo de presupuesto:
Obra nueva
Reforma cocina
Reforma Baño
1:1

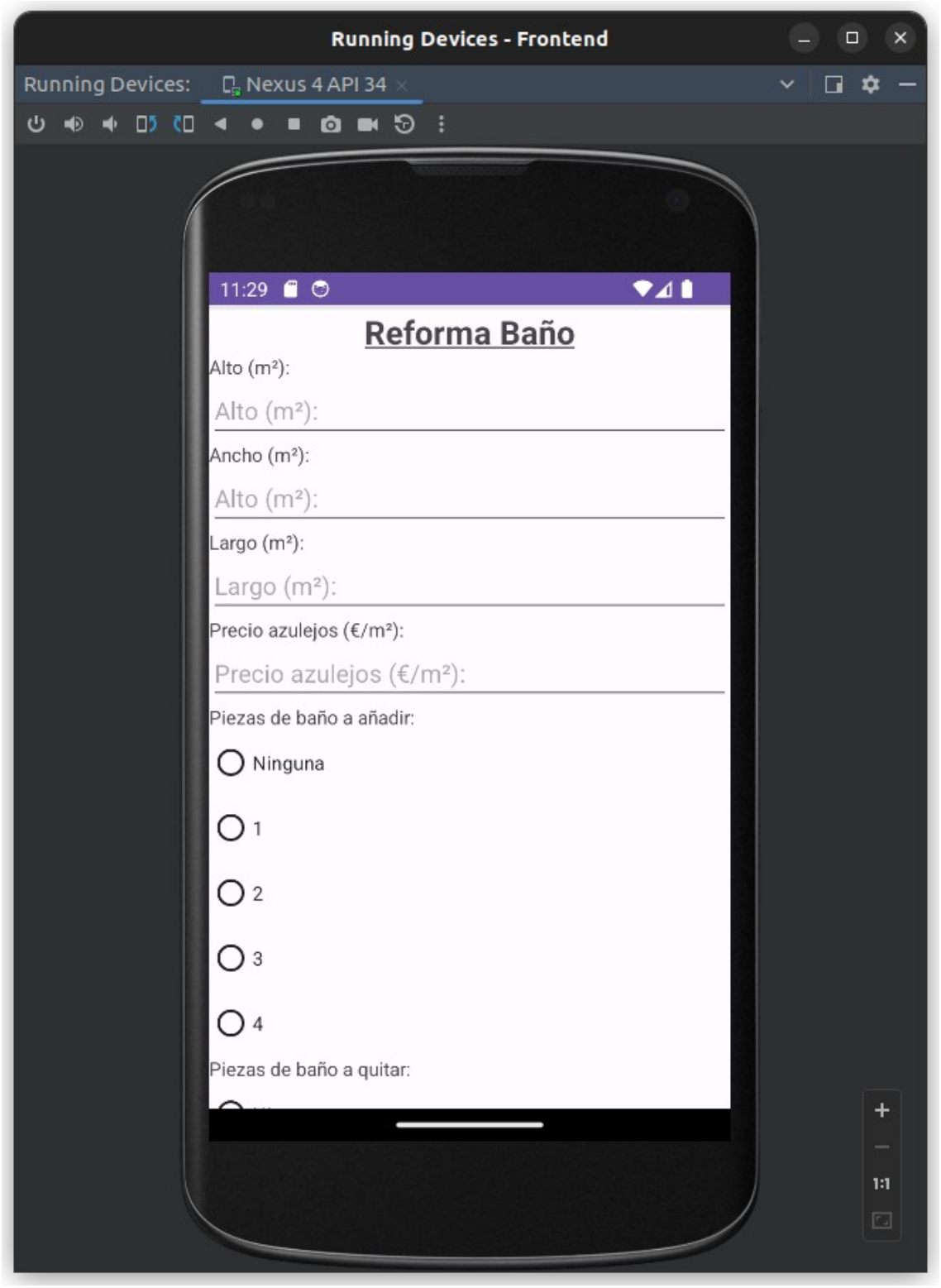
Una vez rellenado este formulario, podemos elegir si queremos hacer una obra nueva, reformar una cocina o reformar un baño. El formulario para realizar una obra nueva es el siguiente:



Formulario para reformar una cocina:



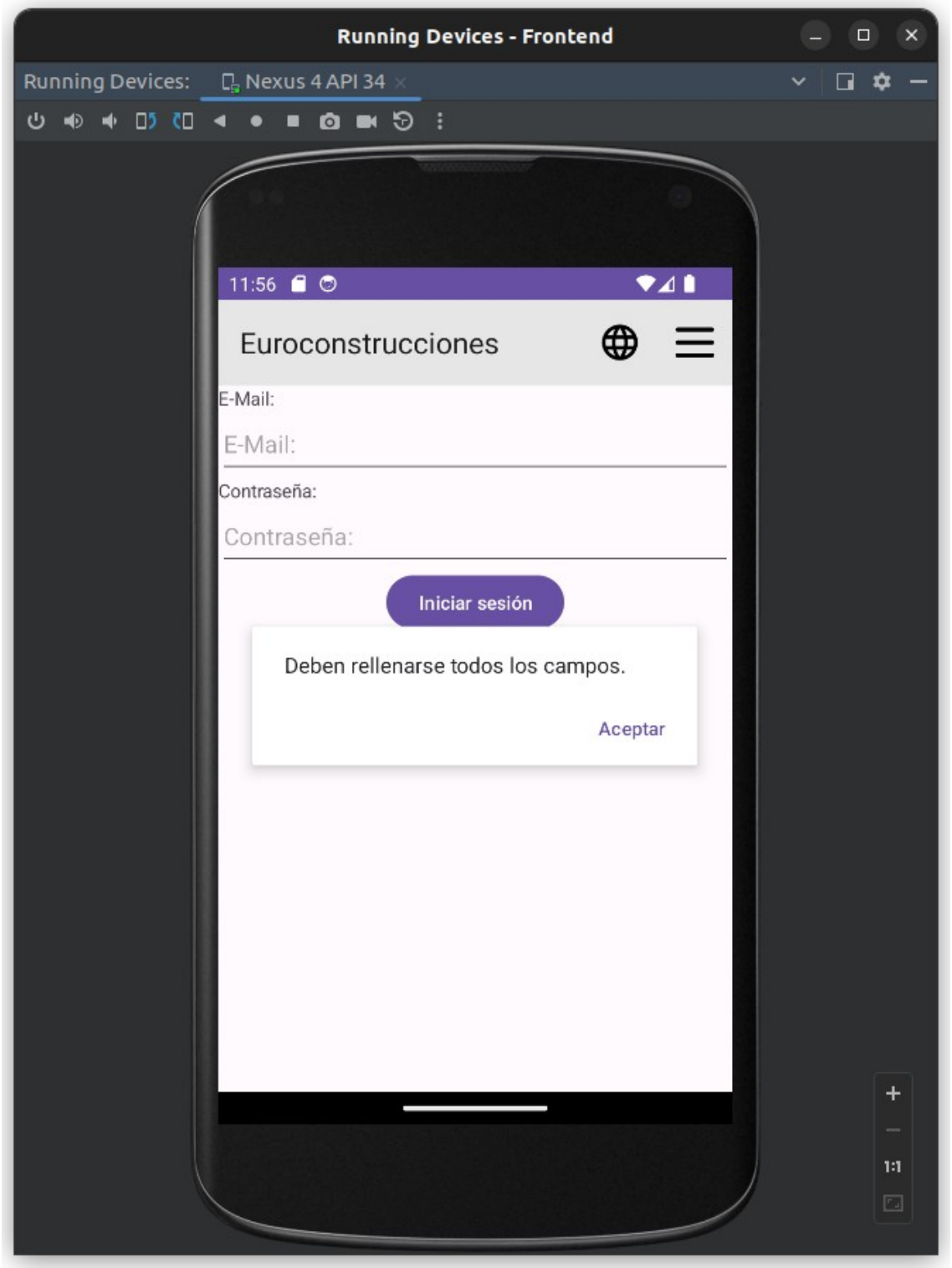
Formulario para reformar un baño:



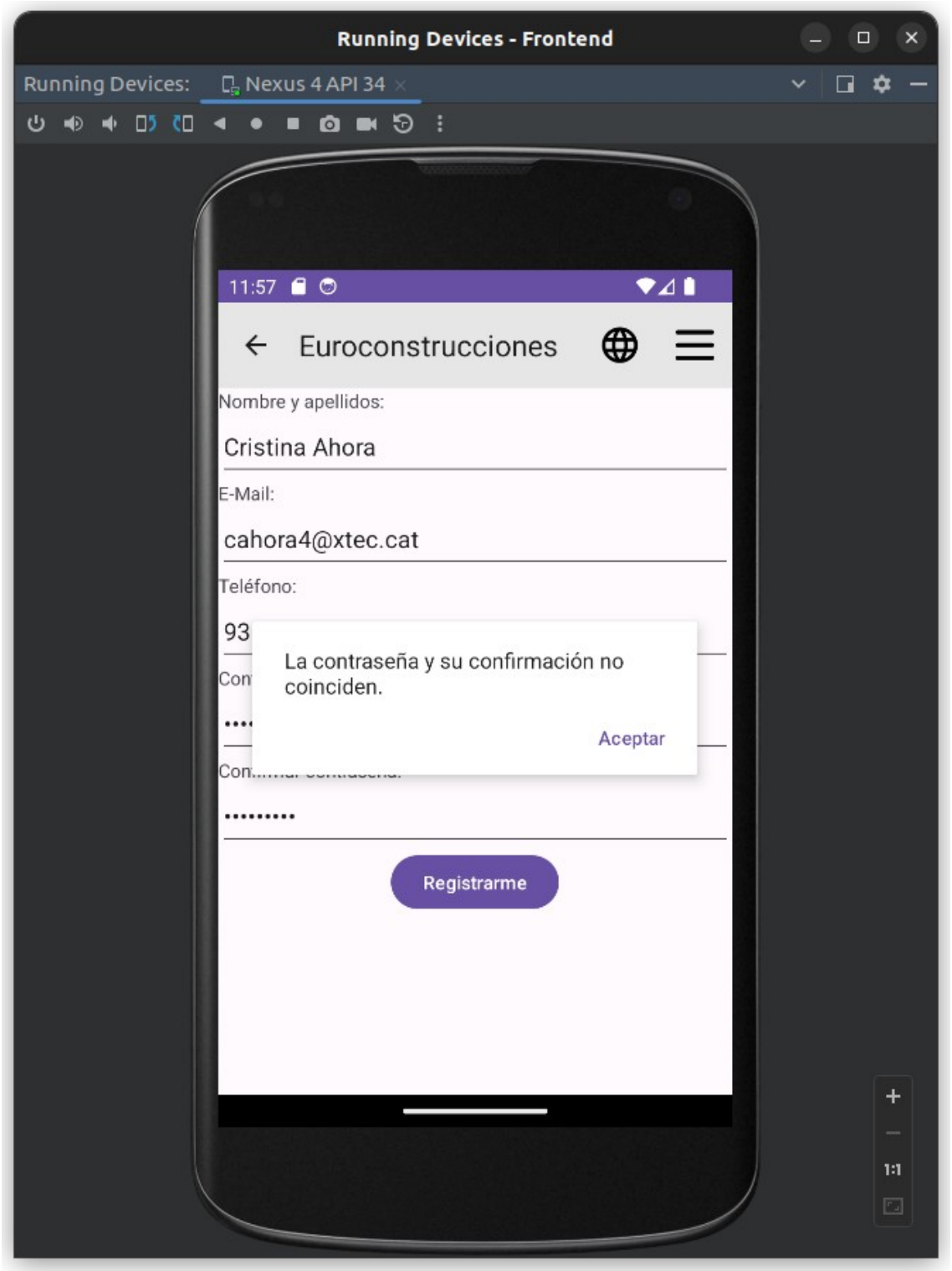
Manual/Guía de usuario

3. Manual de usuario

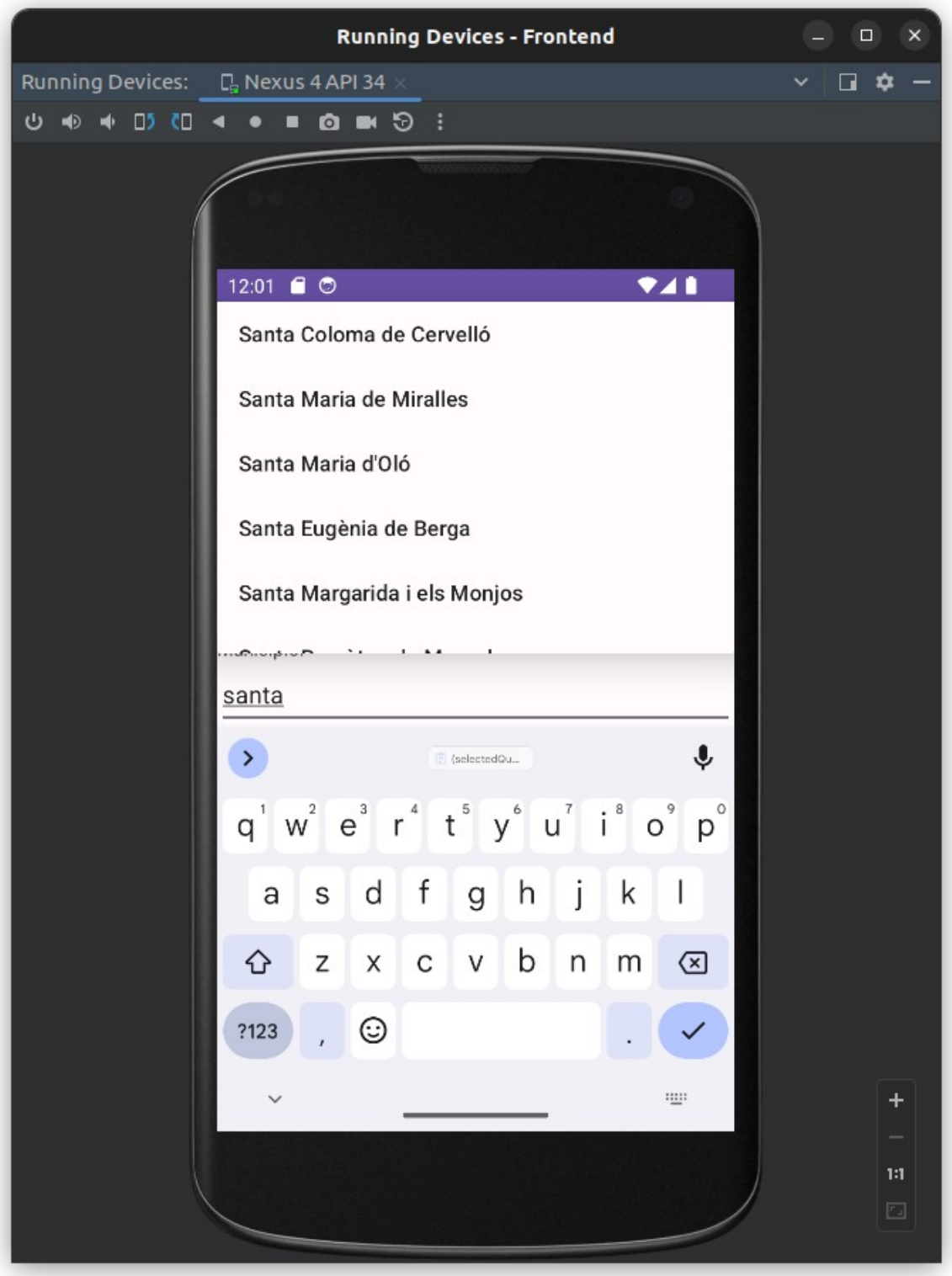
Todos los campos de los formularios deben rellenarse para ser aceptados. En todos ellos se nos muestra la misma alerta si no es el caso.



En el formulario de registro, la contraseña y su confirmación deben ser idénticas; en caso contrario, no se puede proseguir.



Al rellenar el campo del municipio para un presupuesto nuevo, se despliega un menú con los municipios de Barcelona que ayuda a completarlo.



Por lo demás, navegar entre las diferentes pantallas de la aplicación es simple e intuitivo, y los casos en que la aplicación debe mostrar un error son pocos.

Ampliaciones/ mejoras para versiones futuras

4. Ampliaciones/mejoras para versiones futuras

- **Añadir campo de estado**
  Una mejora a proponer, sería que cuando un usuario guarda un presupuesto, por defecto aparezca un nuevo campo que sea ‘Estado’ y al guardarse, se guarde con el estado ‘En espera’ y el administrador lo pueda cambiar a ‘Aprobado’ o ‘Cancelado’ por ejemplo.

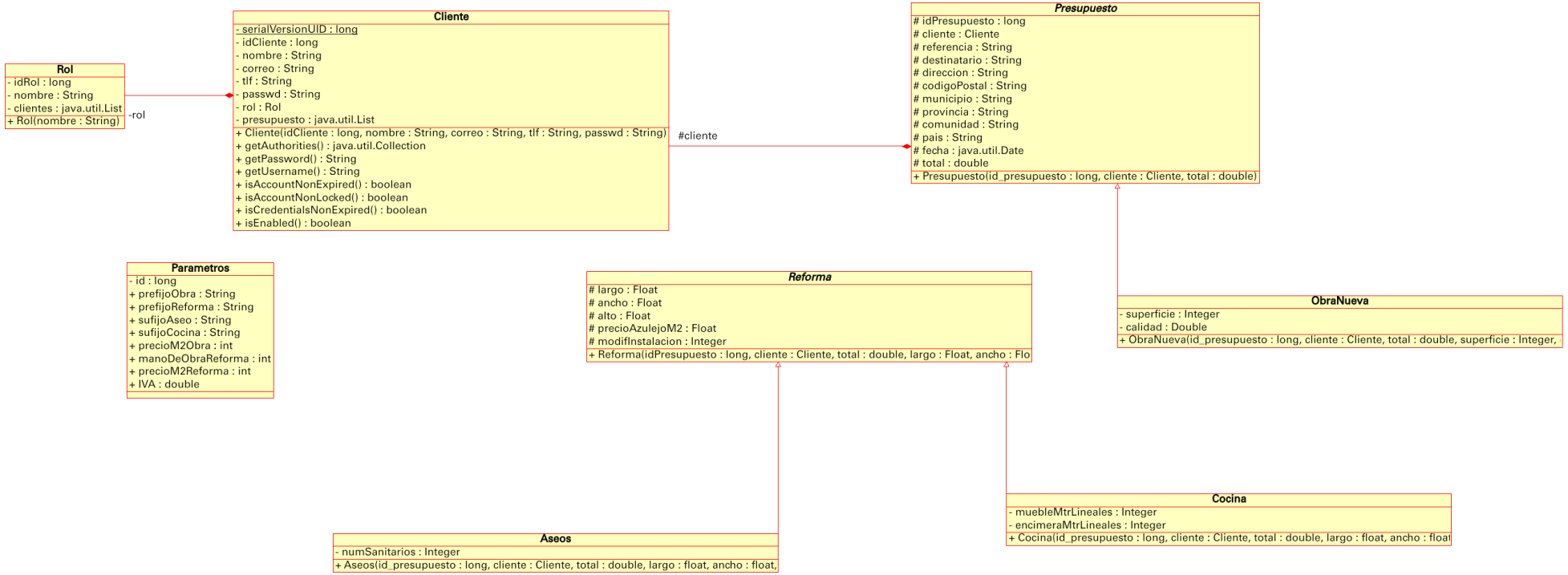
- **Añadir opción para exportar en PDF**
  También se puede añadir otra funcionalidad, que consiste en generar un documento de tipo PDF una vez el presupuesto ya está generado, donde se especifica el presupuesto o los presupuestos que el usuario en concreto desee.

- **API de auto-completado para formularios**
  La API que se utiliza actualmente para ayudar al usuario no ofrece toda la información que los formularios necesitan; filtra y muestra los nombres de los municipios de Barcelona, pero no hace lo mismo para las calles ni para los códigos postales. Debería buscarse otra API que se adapte mejor a las necesidades de la aplicación.

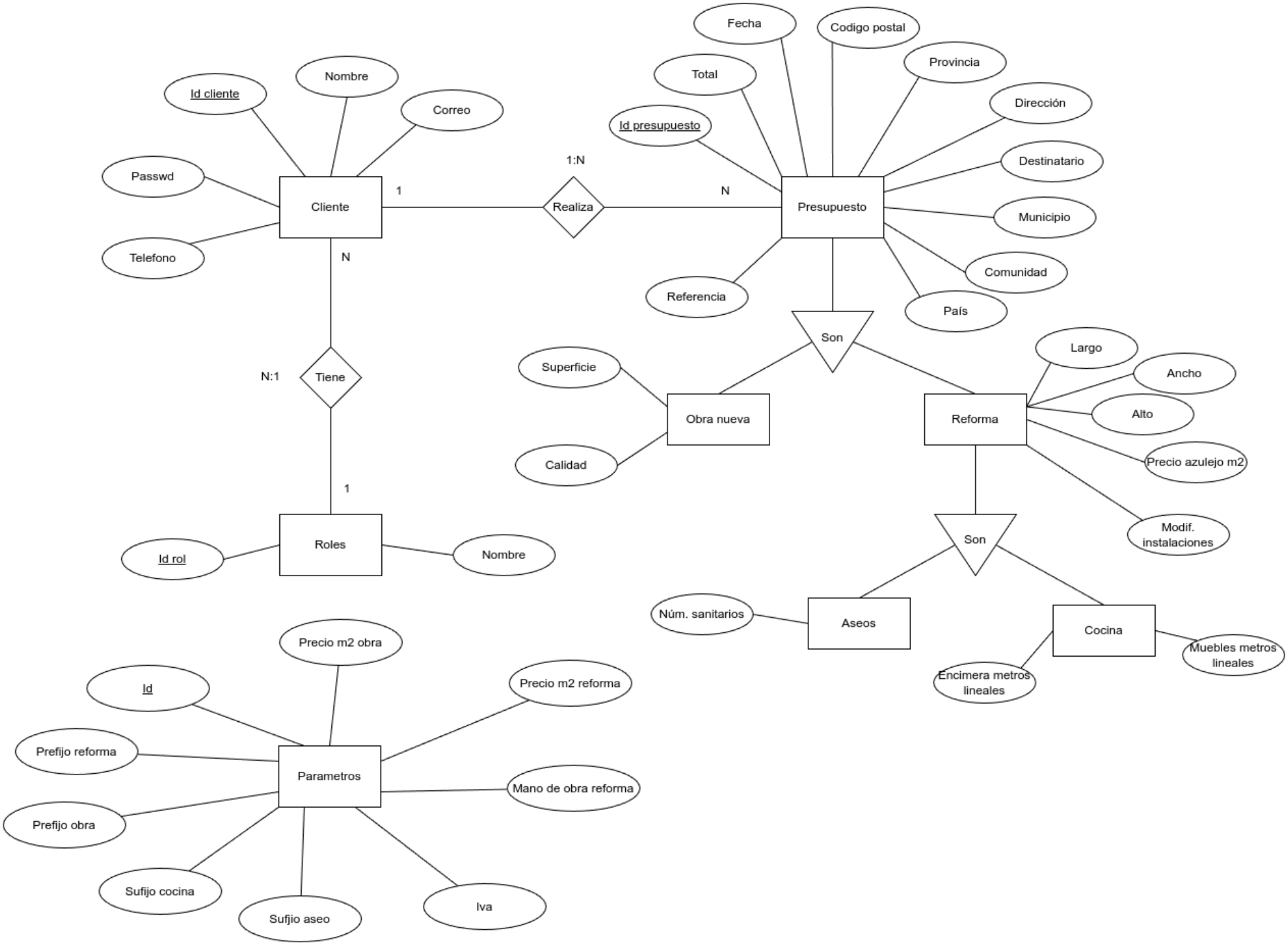
- **Enviar correo a los usuarios**
  Una mejora importante que no se ha implementado por falta de tiempo, es permitir que la aplicación envíe correos a los usuarios que lo soliciten. Serviría para recibir copias de sus presupuestos y para recuperar la contraseña en caso de haberla olvidado.

Anexo: Diagramas y modelos

5. Modelo de objetos – Diagrama de clases UML



6. Modelo Entidad-Relación



Modelo relacional