21 de noviembre de 2018

Diplomado de Tecnología de Información y Comunicaciones

Modulo: Desarrollo de Aplicaciones Móviles Basadas en Frameworks

Profesores: Leonardo Guedez

Jesús Larez

Realizado por: Yrma Pina

Jusmari Belisario

Sergio Hernández

Proyecto

El presente trabajo describirá las modificaciones realizadas a la aplicación móvil del repositorio https://github.com/leolas95/curso-diplomado-android para incluir tres características adicionales a la aplicación original, la recepción y manejo de notificaciones, la inclusión en el menú de una barra de búsqueda y la funcionalidad para eliminar un elemento de la lista de contactos al deslizar el dedo a la izquierda sobre él, adicionalmente se creará una rama adicional a la rama borrar para implementar la funcionalidad de eliminar el elemento de la lista a nivel de la base de datos mediante la comunicación con la API del backend; se implementará también la unión de las tres características mencionadas anteriormente en una única aplicación.

Funcionalidad para la recepción y manejo de notificaciones.

Para la implementación de la funcionalidad de recepción y procesamiento de notificaciones por parte de la aplicación se creó el archivo *MyFirebaseInstanceService.java,* en este archivo se define la clase MyFirebaseInstanceService y se referencia en el archivo manifiesto AndroidManifest.xml de la forma siguiente

```
<service
    android:name=".MyFirebaseInstanceService"
    android:stopWithTask="false">

    <intent-filter>
        <action android:name="com.google.firebase.MESSAGING_EVENT" />
    </intent-filter>

</service>
```

en este archivo se referencia también el servicio de mensajería de Firebase.

En el archivo *MyFirebaseInstanceService.java* se incluyen las siguientes librerías para el uso del

componente Firebase:

```
import com.google.firebase.messaging.FirebaseMessagingService
import com.google.firebase.messaging.RemoteMessage
```

y estas librerías para el manejo de las notificaciones:

```
import android.app.NotificationManager;
import android.support.v4.app.NotificationCompat;
```

en la clase *MyFirebaseInstanceService* se definieron los métodos *onNewToken* que se utiliza para mostrar a través de la consola el Token del componente Firebase cuando es renovado y el método *onMessageReceived* que se ejecutara al recibir un mensaje mediante Firebase, en el método *onMessageReceived* en la línea 39 se construye la notificación y se genera en la aplicación, siempre y cuando la data recibida no se encuentre vacía, de la siguiente manera:

```
String idCanal = "1";
NotificationCompat.Builder builder =
        new NotificationCompat.Builder(this, idCanal)
                .setSmallIcon(R.mipmap.ic_launcher_round)
                .setContentTitle(titulo)
                .setContentText(contenido)
                .setAutoCancel(true)
                .setPriority(NotificationCompat.PRIORITY_HIGH)
                .setContentIntent(pendingIntent);

NotificationManager notificationManager =
        (NotificationManager)
getSystemService(Context.NOTIFICATION_SERVICE);

notificationManager.notify(0, builder.build());
```

En el archivo MainActivity.java se incluyen las siguientes librerías:

```
import com.google.android.gms.tasks.OnSuccessListener;
import com.google.firebase.iid.FirebaseInstanceId;
import com.google.firebase.iid.InstanceIdResult;
```

y dentro del método onCreate de la clase *MainActivity* en la línea 91 se incluye lo siguiente para la generación del token que permite recibir las notificaciones mediante el servicio de Firebase

```
FirebaseInstanceId.getInstance().getInstanceId().addOnSuccessListener(MainA
ctivity.this, new OnSuccessListener<InstanceIdResult>() {
    @Override
    public void onSuccess(InstanceIdResult instanceIdResult) {
        Log.w(TAG, "onSuccess: " + instanceIdResult.getToken());
    }
});
```

en el archivo MainActivity.java se incluyen también las librerías:

```
import android.app.NotificationChannel;
import android.app.NotificationManager;
```

y dentro del método *onCreate* de la clase *MainActivity* en la línea 98 se incluye el siguiente código para la configuración del canal para la recepción de notificaciones necesario para las versiones de android mayores a Android Oreo:

```
// A partir de Android Oreo, las notificaciones deben pertenecer a un canal
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
    String idCanal = "1";
    String nombreCanal = "Mi canal";
    NotificationManager                notificationManager                =
getSystemService(NotificationManager.class);
    notificationManager.createNotificationChannel(new
NotificationChannel(idCanal,
            nombreCanal, NotificationManager.IMPORTANCE_HIGH));
}
```

Funcionalidad de barra de búsqueda.

Para la implementación de la barra de búsqueda se incluye en el layout del menú *res\menú\menu_main.xml* la opción de búsqueda:

```
        <item
    android:id="@+id/action_search"
    android:icon="@drawable/ic_search_white_24dp"
    android:title="@string/action_search"
    app:actionViewClass="android.support.v7.widget.SearchView"
    app:showAsAction="always|collapseActionView" />
```

En el archivo MainActivity.java se incluyen las siguientes librerías:

```
import android.support.transition.TransitionManager;
import android.support.v7.widget.SearchView;
import android.view.ViewGroup;
import android.widget.TextView;
```

En la línea 45 del archivo se crea un objeto TexView para el manejo del mensaje a mostrar en caso de que la búsqueda no arroje ningún resultado:

```
private TextView tvResultadoFiltroVacio;
```

En el layout *activity_main.xml* se define el campo de texto *tv_resultado_filtro_vacio* este campo de texto esta oculto por defecto:

```
<TextView
    android:id="@+id/tv_resultado_filtro_vacio"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:gravity="center"
    android:text="@string/busqueda_resultado_vacio"
    android:visibility="gone"/>
```

en el método *onCreate* de la clase *MainActivity* en la línea 78 se agrega al objeto

tvResultadoFiltroVacio el objeto de texto *tv_resultado_filtro_vacio*.
En el método *onCreateOptionsMenu* de la clase *MainActivity* que es donde se crea el menú se agrega el siguiente código en la línea 97 para el manejo del botón de búsqueda:

```
// Obtiene el elemento del menú referente a la accion de buscar
final MenuItem item = menu.findItem(R.id.action_search);

// Obtiene el view de la barra de búsqueda
SearchView searchView = (SearchView) item.getActionView();

// Establece el texto que funciona como pista en la barra
searchView.setQueryHint(getResources().getString(R.string.action_search));

// Establece el listener para las acciones de la barra de búsqueda
searchView.setOnQueryTextListener(new SearchView.OnQueryTextListener() {
    // Escucha cuando el usuario presiona el boton de buscar
    @Override
    public boolean onQueryTextSubmit(String query) {
        return false;
    }

    // Escucha a medida que el usuario escribe texto en la barra
    @Override
    public boolean onQueryTextChange(String newText) {
        adapter.filtrar(newText);

        // Si no hubo resultados, muestra el TextView que lo indica
        if (adapter.getItemCount() == 0) {
            recyclerView.setVisibility(View.GONE);
            tvResultadoFiltroVacio.setVisibility(View.VISIBLE);
        } else if (recyclerView.getVisibility() == View.GONE) {
            // Si el RecyclerView estaba oculto, lo hacemos visible otra vez
            recyclerView.setVisibility(View.VISIBLE);
            tvResultadoFiltroVacio.setVisibility(View.GONE);
        }
        return true;
    }
});

return super.onCreateOptionsMenu(menú);
```

En este segmento de código además de inicializar los parámetros de la barra de búsqueda se establece el *Listener* para la misma, y dentro de este se define el método *onQueryTextChange* que se ejecutara a medida que el usuario escriba luego de presionar el icono de búsqueda. Los resultados se van mostrando mediante la invocación del método *adapter.filtrar(newText)*. Si la búsqueda no arroja resultados se muestra el mensaje de lista vacía.

En la clase UsuarioAdapter declarada en el archivo UsuarioAdapter.js en la línea 31 se define una lista de respaldo para conservar la lista de usuarios cargada por la aplicación.

```
// Lista de respaldo para la barra de búsqueda
```

```
private ArrayList<Usuario> copiaUsuarios;
```

en la línea 102 se define el método *setCopiaUsuarios* que guarda el respaldo de la lista de usuarios en la lista *copiaUsuarios* este método será llamado dentro del método *refrescarUsuarios* de la clase MainActivity. Se define también en la línea 106 el método *filtrar* que toma el texto que va introduciendo el usuario en la barra de búsqueda y va actualizando la lista de usuario cuando el texto coincida con parte del nombre, el apellido o la empresa de los usuarios de la lista:

```
public void filtrar(String consulta) {
    // Al principio de la búsqueda, no hemos encontrado ningun usuario
    usuarios.clear();

    // Si la consulta esta vacia, devolvemos todos los usuarios
    if (consulta.isEmpty())  {
        usuarios.addAll(copiaUsuarios);
        return;
    }

    // Normaliza el texto de búsqueda, convirtiendolo a minuscula
    consulta = consulta.toLowerCase();
    for (Usuario usuario : copiaUsuarios) {
        if (usuario.getNombre().toLowerCase().contains(consulta) ||
                usuario.getApellido().toLowerCase().contains(consulta) ||
                usuario.getEmpresa().toLowerCase().contains(consulta)) {
            usuarios.add(usuario);
        }
    }
    notifyDataSetChanged();
```

Funcionalidad para la eliminación de usuarios de la lista

Para implementar la opción de eliminar un usuario de la lista con el gesto de arrastre a la derecha se agregaron las siguientes librerías al archivo MainAcivity.java:

```
import android.support.design.widget.CoordinatorLayout;
import android.support.design.widget.Snackbar;
import android.support.v7.widget.helper.ItemTouchHelper;
import com.ucab.leonardo.cursodiplomado.RecyclerItemTouchHelper;
```

Se agregó el archivo *RecyclerItemTouchHelper.java* donde se define el *listener* y donde se definen métodos de la clase *RecyclerItemTouchHelper,* que interactúan con el objeto *clFilaUsuario*, dentro de estos métodos está el método *onSwiped* que se utilizara en la clase MainActivity para definir las acciones a ejecutar al realizar el desplazamiento del dedo sobre el usuario a eliminar.

En la declaración de la clase *MainActivity* se indica que esta implementa la interfaz *RecyclerItemTouchHelperListener*

Dentro del método *onCreate* de la clase *MainActivity* se incluye el siguiente código para instanciar la clase *ItemTouchHelper.SimpleCallback* y luego se instancia la clase *ItemTouchHelper* y se llama al

método *attachToRecyclerView* para relacionar el listener a la vista y definir la dirección en la que será considerado el evento Swipe.

```
ItemTouchHelper.SimpleCallback itemTouchHelperCallback = new
RecyclerItemTouchHelper(0, ItemTouchHelper.LEFT, this);

new
ItemTouchHelper(itemTouchHelperCallback).attachToRecyclerView(recyclerView)
;
```

En la clase *MainActivity* se crea el objeto *coordinatorLayout* y dentro del método onCreate se relaciona *coordinatorLayout* con la vista principal de *activity_main.xml* donde se cargara el Snackbar que permitirá al usuario revertir la acción luego de eliminar el usuario.

En la clase *MainActivity* se define el método *onSwiped*, que se ejecutara cuando se ejecute el evento Swipe, se elimina la fila correspondiente al usuario sobre el que se ha registrado el evento. En la línea 175 de este método se define el Snackbar donde se le ofrece al usuario la posibilidad de deshacer la eliminación del registro.

```
Snackbar snackbar = Snackbar.make(coordinatorLayout,
        nombre + " eliminado", 5000);
snackbar.setAction("Deshacer", new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        adapter.restoreItem(usuario, deletedUserPosition);
        Log.w(TAG, "Restaura, adapter.getItemCount() " +
adapter.getItemCount());

        // Si era el primero o el ultimo de la lista, hace scroll hacia esa
posicion
        if (eraElPrimero(deletedUserPosition) ||
eraElUltimo(deletedUserPosition)) {
            layoutManager.scrollToPosition(deletedUserPosition);
        }
    }
});
snackbar.show();
```

Implementación de la eliminación en backend del usuario

Para ejecutar el borrado a través de la API se creó el archivo *RespuestaBorraUsuario.java* donde se definió la clase del mismo nombre, *RespuestaBorraUsuario.*

En el archivo *ApiServices.java* se incluyó la librería *retrofit2.http.DELETE* para habilitar el método delete de la librería. Dentro de la interfaz ApiServices se agregó el código asignando a la variable de entorno @DELETE el punto de entrada de la API y la función *borrarUsuario* donde se indica el nombre del parámetro a utilizar por el método.

```
@Headers("Content-Type: application/json")
@DELETE("/usuarios/{email}")
Call<RespuestaBorraUsuario> borrarUsuario(@Path("email") String email);
```

Dentro del método *onSwiped* de la clase *MainActivity* en la línea 174 se captura el email del usuario a eliminar

```
final String email = usuario.getEmail();
```

Y en la línea 195 se incluye el método *addCallback* del snackbar para hacer uso del método *onDimissed*, ya que el llamado a ese método indica que el usuario no hizo clic en la opción deshacer que le fue ofrecida y en consecuencia se puede proceder a la eliminación del registro.

```
snackbar.addCallback(new Snackbar.Callback() {

    @Override
    public void onDismissed(Snackbar snackbar, int event) {
        //see Snackbar.Callback docs for event details

        final ApiService apiService = retrofit.create(ApiService.class);
        //Retrofit retrofit = ClienteRetrofit.obtenerClienteRetrofit();
        PeticionBorrarUsuario peticion = new PeticionBorrarUsuario(email);
        Call<RespuestaBorraUsuario> call = apiService.borrarUsuario(email,
peticion);
        call.enqueue(new Callback<RespuestaBorraUsuario>() {
            @Override
            public void onResponse(Call<RespuestaBorraUsuario> call,
Response<RespuestaBorraUsuario> response) {
                if (response.isSuccessful()) {
                    Log.w(TAG, "Usuario borrado con exito");
                } else {
                    Toast.makeText(MainActivity.this,
                            "La peticion no fue exitosa. Intente nuevamente",
                            Toast.LENGTH_SHORT).show();
                }
            }

            @Override
            public void onFailure(Call<RespuestaBorraUsuario> call, Throwable
t) {
                Toast.makeText(MainActivity.this,
                        "Error de conexion en la red. Intente nuevamente",
                        Toast.LENGTH_SHORT).show();
            }
        });
    }
```

Inclusión de las tres funcionalidades implementadas en una única aplicación

Es perfectamente posible realizar el merge de la rama master con la de las diferentes modificaciones, se deben ejecutar los siguientes comandos git:

$ git clone https://github.com/leolas95/curso-diplomado-android
$ cd curso-diplomado-android
$ git checkout barra-busqueda
$ git checkout notificaciones
$ git checkout proy_SHYPJB
$ git merge --no-ff barra-busqueda
$ git merge --no-ff proy_SHYPJB
Se debe resolver el conflicto en los archivos indicados
$ git add MainActivity.java
$ git add UsuarioAdapter.java
$ git merge --continue
$ git merge --no-ff notificaciones
Se debe resolver el conflicto en el archivo indicado
$ git add MainActivity.java
$ git merge --continue

Se adjunta APK compilado con las tres funcionalidades integradas en la rama proy_SHYPJB con el nombre curso-diplomado-android-integrada.apk.