NinjaMessaging

Rubén Agudo Santos

16 de abril de 2014

Resumen

En este proyecto individual se ha decidido realizar una aplicación de mensajería. Obviamente no permite ningún tipo de mensajería real, solo la simulación con dos usuarios predefinidos. Todo el código fuente está disponible en `https://github.com/RubenAgudo/DAS/tree/master/NinjaMessaging` licenciado bajo licencia GPLv3.

Índice

1. Funcionalidades obligatorias

1.1. Localización

Este código está contenido en la clase *ChatActivity.java* y empieza cuando el usuario pincha sobre el icono de localización en el Action Bar.

Entonces se recoge el LocationManager y todos los providers. Nos da igual de donde provenga la última localización, siempre que la haya.

Entonces obtenemos la dirección física, si tenemos conexión a internet, si no, nos devuelve la latitud y la longitud. Finalmente se guarda el mensaje y se refresca el ListFragment que contiene los mensajes de ese usuario.

1.2. Bases de datos SQLite

Para gestionar las bases de datos lo mejor es crear un gestor de bases de datos, que en este caso será una MAE (Máquina abstracta de ejecución) que solo tiene una única instancia. De este modo, todas las clases acceden a la misma instancia y realizan las operaciones necesarias.

Se han creado los métodos necesarios para obtener los mensajes de un usuario, todos los usuarios, los chats recientes, para añadir un mensaje a la conversación de un usuario.

También se han creado unas funciones ``miscelaneas´´ menos relacionadas con los propios chats y usuarios. Esas son las de exportar el chat de un usuario a la SD, obtener un mensaje al azar de un usuario basándose en unos Strings predefinidos en la propia clase para simular una conversación y la posibilidad de actualizar la información de los usuarios.

Como curiosidad, decir que he utilizado los métodos insert, update, select propios de Android, ya que al utilizar expresiones del tipo:

```
1 db.update("Usuarios", values, "nombreUsuario=?", new String[]{detallesDe});
```

Evitamos que haya inyecciones SQL, es menos propenso a fallos que escribir una sentencia SQL pura con todas las comillas simples, dobles, concatenaciones de Strings etc. Y además, al ser algo propio de Android, está considerado como un buen hábito de programación.

1.3. Notificaciones

Cada vez que abres la aplicación, cada 15 segundos (no configurables), la aplicación envía una notificación con un mensaje. Cuando pinchas sobre la notificación, te lleva al chat de ese usuario, y la notificación desaparece.

El código puede consultarse en *NotificationService.java*.

1.4. Fragments

Se han utilizado principalmente ListFragments, que muestran o bien los chats recientes, o los propios mensajes.

Si estamos en la ventana principal, y ponemos en horizontal el dispositivo, en la misma pantalla se mostrarán dos fragments: El que contiene la lista de chats recientes y el fragment del propio chat.

Se ha controlado que si no hay ningún chat seleccionado, que muestre un aviso de que hay que seleccionar una conversación.

1.5. Menús y Action Bar

A lo largo de la aplicación, se utilizan en distintos sitios menús y acciones de la Action Bar.

Por ejemplo, cuando estas en la vista de Chat en modo retrato, es decir, hablando con un usuario, tenemos distintas acciones y opciones.

- **Action Bar, icono de localización:** Como he descrito con anterioridad, envía la última posición conocida transformada en dirección física si se dispone de conexión a internet, si no la latitud y la longitud.
- **Action Bar, icono de disquete:** Exporta la conversación actual a la SD, a un fichero de texto plano.
- **Menú, Ver detalles de usuario:** Abre una nueva actividad donde se muestran los datos de ese usuario.

Para consultar ese código, esta disponible en el archivo *ChatActivity.java* para ver las llamadas y como se tratan los clicks en el menú, y en */menu/-chat_activity.xml* para ver como están hechos los menús.

1.6. Comentarios en el código

Siguiendo los principios de clean code de Robert C. Martin no he puesto demasiados comentarios, solo los necesarios en lo que he considerado oportuno porque no lo hemos dado en clase o porque el código no podría entenderse todo lo bien que debería.

Aun así, prácticamente todos los métodos disponen de *javadoc*, para poder consultar que hace cada función.

2. Funcionalidades extra

2.1. Servicio

He creado un servicio que dispone de un *Handler* y de un *Runnable* para que cada 15 segundos cree una notificación.

Para crear la notificación, obtiene de la Base de Datos un chat al azar, que devuelve un usuario y un texto de mensaje y posteriormente se notifica.

El servicio también se lanza cada vez que se lanza la aplicación, para que no sea necesario apagar y encender el móvil. Esta opción en un futuro será configurable.

Lo que se hace en la notificación ha sido previamente explicado, en la sección de notificaciones.

En código relativo al servicio puede consultarse en *NotificationService.java*.

2.2. Receiver

El Receiver, está estrechamente relacionado con el servicio y con las notificaciones, ya que lo he creado para que al completar el arranque se lance el servicio que comienza las notificaciones cada 15 segundos.

Para ello he creado el propio Receiver extendiendo de BroadCastReceiver y en el manifest.xml le he dado permisos para ejecutar cosas en el BOOT COMPLETED mediante el código:

```
1 <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
```

2.3. Internacionalización y globalización

Para la globalización e internacionalización, he decidido utilizar el método recomendado por Android, que es replicar el fichero *strings.xml* dentro de otras carpetas que contienen el código de idioma correspondiente, es decir, *values-en, values-de* etc.

De este modo, el sistema obtendrá el fichero adecuado para el idioma en el que esté el dispositivo. También es verdad que hay algunos idiomas que Android no soporta por defecto, y que sería preferible dar al usuario la opción de forzar el idioma, pero eso será para futuras actualizaciones.

2.4. Uso de ficheros

la única manera de dar sentido al uso de ficheros dentro de mi aplicación era dar al usuario la oportunidad de exportar los mensajes que ha mantenido con un usuario a la SD en texto plano.

Esta funcionalidad está disponible desde la pantalla de chat en el Action Bar, que tiene un icono de disquete.

2.5. GeoCoder

Como no tenía mucho sentido utilizar un LocationListener para ``seguir´´ a un usuario lo que se me ocurrió, fue que utilizando la última posición conocida del móvil, obtener la dirección física.

Para conseguir esto se necesitan dos cosas: Conexión a internet y la clase GeoCoder de Android. Como el servicio al que consulta el GeoCoder no está en el núcleo de Android, requiere de conexión a internet.

Realmente lo que se hace es lo conocido como ``Reverse Geocoding´´. La precisión de este sistema según los desarrolladores de Android es variable, ya que a veces puede darte una dirección muy precisa, y otras veces simplemente la ciudad.

El código para el Reverse GeoCoding puede consultarse en *ChatActivity.java*.

2.6. Alert Dialog con layout personalizado

También he implementado la posibilidad de actualizar los datos personales de los usuarios mediante un Alert Dialog con un layout personalizado.

Para ello, he creado un layout, y cuando se pincha en el botón del Action Bar dentro de la actividad DetallesUsuario.java, llama al método editarUsuario, y ahí, se obtiene el LayoutInflater, se obtiene una View después de inflar el layout y solo se pone un botón de aceptar.

Nótese que al pinchar no carga los datos actuales, ya que eso no está programado, y sería lógico que lo hiciera, pero eso se realizará en futuras actualizaciones.