

CuidAR++

Gnecco Cristian, Miño Maximiliano, Otalora Griselda
40024360, 41783275, 39213045
Miércoles, Grupo 7

Universidad Nacional de La Matanza,
Departamento de Ingeniería e Investigaciones Tecnológicas,
Florencio Varela 1903 - San Justo, Argentina

Resumen. Debido a los acontecimientos que acechan al mundo en la actualidad se necesita que cada uno desde su lugar aporte a la sociedad su conocimiento para lograr salir adelante. Para lograr este cometido hemos desarrollado una aplicación Android para que cualquier persona conozca las probabilidades de haber contraído Covid. En este informe se explica el funcionamiento de la aplicación, sus procesos, cómo se sincronizan y comunican. Además se explicará cómo se realiza la comunicación externa con el Web Service y cómo se logró que la aplicación sea tolerante a fallos y que algunos datos persistan por más que se cierre la aplicación. Además se proveerá de un manual de usuario para el entendimiento de la aplicación.

Palabras claves: Android, Covid, Aplicación.

1 Introducción

En el presente informe hablaremos sobre la aplicación desarrollada en Android Studio para dispositivos Android. La utilidad de esta aplicación es permitirle a cualquier ciudadano conocer las probabilidades de haber contraído Covid contestando un formulario simple con preguntas cerradas, seleccionando las opciones: SI o NO. Para poder utilizar esta aplicación es necesario ser usuario e ingresar con los datos solicitados, en caso de no serlo debe registrarse, ingresando los datos: Apellido, Dni, E-Mail y Contraseña. Luego el usuario deberá contestar el formulario e ingresar su temperatura u optar por tomar su temperatura con el celular gracias al sensor de temperatura¹. Luego de que el usuario confirme los datos, la aplicación los procesa y le indicará datos como la temperatura y los síntomas, mostrando la probabilidad de haber contraído Covid, además de información valiosa para el usuario de cómo proceder en cada caso. La aplicación además posee funcionalidades

¹ En el presente informe se expresara que se utiliza el sensor de temperatura pero cabe aclarar que el mismo se simula usando el sensor de luminosidad.

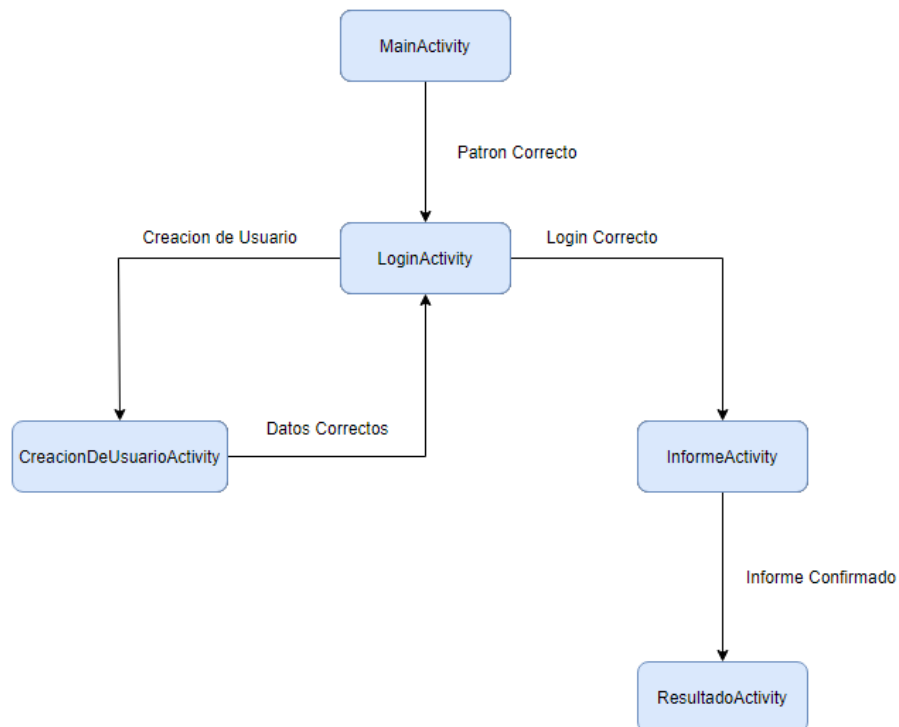
extras como la persistencia del valor de la temperatura utilizando Shared Preference y la detección del celular en posición horizontal utilizando el sensor de Proximidad.

2 Desarrollo

2.1 Repositorio en GitHub

Enlace repositorio GitHub: <https://github.com/crisgnecco/TP2-android>

2.2 Diagrama Funcional



2.3 Ejecución concurrente y mecanismos de sincronización

Las actividades se ejecutaron en el hilo principal y los servicios en background, de esta manera, el hilo principal no posee problemas de bloqueo. Cuando se inicia sesión, es a través de la función "agregarEvento" que la actividad interactúa con el service, pasando a través de dicha función la descripción y el

tipo de evento a registrar. Una vez logueado, nos encontramos con la activity Informe, que interactúa con el service de la misma manera que lo hace Login, pero a través de la función “iniciar”, en este caso, este service es el encargado de actualizar el token, por lo que se encuentra ejecutándose constantemente o hasta que vuelva a la activity de iniciar sesión.

Por otro lado la activity InformeActivity registra como máximo 2 sensores usando el método “registerListener” los cuales ejecutan en hilos independientes de forma predeterminada. Para asegurar la mutua exclusión en el método onSensorChanged se utiliza “synchronized” en toda la porción de código correspondiente al método. Cuando no se necesita registrar más los sensores se los quita con el método “unregisterListener” [1].

Las clases “serviceRegistrarEvento” y “serviceActualizacionToken” extienden de la clase IntentService [2], que maneja solicitudes asincrónicas (expresadas como Intent) bajo demanda, gracias a esto no hace falta la sincronización, solo basta con el onHandleIntent(Intent intentServicio) para recibir las solicitudes del usuario.

2.4 Comunicación entre los componentes

La comunicación entre las actividades se realizó mediante Intents, a través del método “startActivity”. Se instancia el intent con la activity contexto y la activity receptora del mensaje y se utiliza el método “putExtra” para agregar los datos que se quieren transferir mediante el intent. Luego se inicia la activity receptora mediante el método startActivity al cual se pone como parámetro el intent. Del lado de la activity receptora se instancia un intent con el método “getIntent”, luego se utiliza el método getExtras en el intent y se lo asigna a un objeto del tipo “Bundle” para luego recuperar los datos utilizando los distintos métodos get (getString, getInt, getBoolean, etc).

Además, se realizaron comunicaciones entre la activity y el service, donde se utilizó un Intent para poder iniciar el service desde la activity, a través de la función “startService”. Para que esta activity se comunique con el service, primeramente, se llama a una función de dicho service, que se encarga de modificar una variable, de esta manera, el service solo actúa cuando se lo habilita.

2.5 Técnica utilizada para la comunicación con el servidor

Para la comunicación con el servidor usamos retrofit de modo asincrónico. Lo que primero que se hizo para lograr esta comunicación fue declarar en el build.gradle de la app unas dependencias: implementation 'com.squareup.retrofit2:retrofit:2.9.0' y implementation 'com.squareup.retrofit2:converter-gson:2.9.0'. Estas dependencias son para poder incorporar retrofit y su convertidor de formato de gson. En el manifest agregamos el permiso de internet:

uses-permission android:name="android.permission.INTERNET"

Luego definimos los objetos planos Java (POJOs) que representan el cuerpo de las peticiones, los cuales se transforman a objetos Json que corresponde al formato que usan las peticiones, que en este caso están definidos en las clases SoaRequest, SoaRequestEvento y SoaRequestLogin, y las respuestas recibidas en formato Json las vamos a procesar convirtiéndolas en objetos planos Java [3], tales están definidas en las clases SoaResponseLogin, SoaResponse y SoaResponseEvent. Se creó la interfaz que represente al servicio REST, donde están todas las peticiones a realizar, esta se llama SoaService. Cada una de las peticiones se realiza a una ruta específica de la API (register , login, event o refresh completando de esta manera la URI).

Además se creó la clase ConexionApi con las funciones para ejecutar las peticiones (registrarEvento y actualizarToken). La función registrarEvento es utilizada por primera vez, después que dio exitoso el login, lo que se hace es una inicialización del servicio a través de la función agregarEvento, mandando por parámetros el tipo de evento y una descripción del mismo, luego se instancia el ServiceRegistroEvento y se lo inicializa. Esta función posee una variable de la clase SoaRequestEvent, que es completada por los parámetros que recibe y lo primero que hace es crear un objeto retrofit, definiendo la URL base, que en nuestro caso es <http://so-unlam.net.ar/api> y dejándolo encargado de la conversión a formato json. Luego se inicializa una instancia de la interfaz SoaService y esta se encargará de llamar a la función para registrar el evento pasándole el Token del usuario y un SoaRequestEvent cuyo trabajo es ejecutar la petición POST. Esta función devolverá una instancia Call<SoaResponseEvent> que luego ejecuta la función enqueue, que es el encargado de enviar de manera asíncrona la petición y notifica a la aplicación con un callback cuando regresa una respuesta. Como la petición es asíncrona, Retrofit maneja la ejecución de peticiones en un hilo en segundo plano para que el hilo de la UI principal no sea bloqueado o interfiera con este. Con el método enqueue se tiene que usar dos métodos de Callback onResponse y onFailure pero solo se llama a uno de ellos en respuesta de la petición de registrar evento. El método onResponse es invocado cuando una respuesta pudo ser correctamente manejada, aunque el servidor responda con un mensaje de error a la petición. Cuando la respuesta del servidor tiene el body vacío, significa que hubo un error por lo cual, el errorbody que está en json lo convertimos a un objeto plano de java con el modelo de ErrorResponse donde podemos fácilmente recuperar el mensaje que notifica cual es el error y este se muestra por pantalla, pero si la respuesta es correcta el body tendrá el success en true. Las demás peticiones al Webservice se llevan a cabo de manera similar, con distintos parámetros y en los lugares que corresponde. Solo en el caso del login y el refresh, es donde se necesita el token que viene en la respuesta del Json, y este es guardado en una constante.

2.6 Persistencia de datos en la aplicación

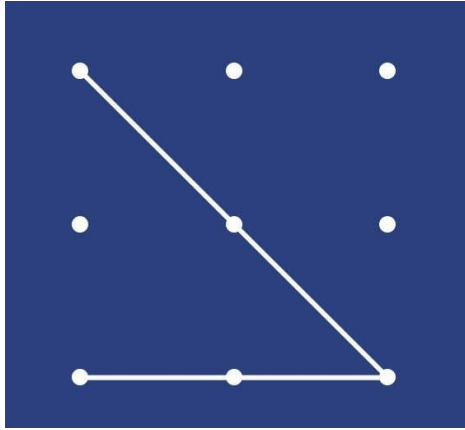
En esta aplicación el dato que guardamos es el valor de la temperatura, ya sea ingresado por el usuario u obtenido gracias al sensor de temperatura. Para guardar este valor se utiliza la clase Shared Preference [4] proporcionada por Android Studio. Cuando se define la preferencia se graba un archivo XML con un nombre identificador y un modo de acceso, el cual se compone de clave-valor donde clave es un identificador único y valor es el valor asociado a dicho identificador. En nuestro caso la clave será la cadena “temperatura” y el valor será el que se obtenga del campo temperatura (TextView) ingresado por el usuario o por el valor obtenido del sensor de temperatura una vez confirmado el formulario. Por otro lado el nombre de la preferencia será “temperatura” y el modo de acceso será “Modo Privado”. Este dato se recuperará cuando el usuario ingrese a la pantalla del formulario una vez iniciado sesión y se guarda cada vez que se confirma el formulario. Esto permite que cuando se cierre la aplicación y se pierdan todos los datos de la misma, el valor de la temperatura persista gracias al archivo temperatura.XML definido.

2.7 Manual de Usuario

La aplicación comienza con una pantalla la cual informa el valor de la batería [5] y un patrón de bloqueo el cual se deberá colocar para poder ingresar a la pantalla de Login.



Tener en cuenta que el patrón de bloqueo [6] es fijo y es el siguiente:



En la pantalla de Login el usuario puede optar por iniciar sesión o registrarse para ingresar al formulario.

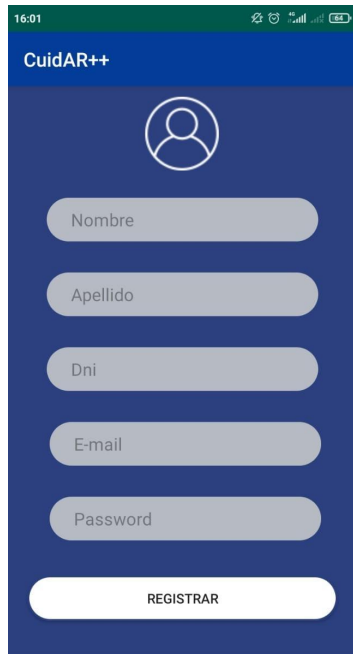
A screenshot of a mobile application interface for 'CuidAR++'. The screen has a dark blue background. At the top, there's a status bar with the time '16:01' and various icons. Below the status bar, the app name 'CuidAR++' is displayed in white. The main heading 'Bienvenido' is centered in a large, bold, white font. Below the heading, there are two input fields: 'Email' and 'Password', both with rounded rectangular borders. At the bottom, there are two buttons: 'INICIAR SESION' and 'CREAR CUENTA', both with rounded rectangular borders and white text on a dark blue background.

Si ya posee un usuario y desea iniciar sesión deberá ingresar su usuario y contraseña y presionar el botón “Iniciar Sesión”

Si los datos ingresados son incorrectos la aplicación avisará mediante un mensaje en la parte inferior de la pantalla. En caso de que los datos sean

correctos se mostrará la pantalla del formulario.

Por otro lado si la persona no posee un usuario debe registrarse presionando el botón “Crear Cuenta”, lo cual pasará a la pantalla “Creacion de Usuario”



En esta pantalla se deberá llenar los datos solicitados y tocar el botón “Registrar”.

Si los datos ingresados son incorrectos la aplicación avisará mediante un mensaje en la parte inferior de la pantalla. En caso de que los datos sean correctos se informará en la parte inferior de la pantalla el nombre del usuario creado y el mismo usuario deberá volver a la pantalla de inicio de sesión para iniciar sesión de la misma manera anteriormente explicada y se pasará a la pantalla del formulario

Una vez la persona inicie sesión se pasará a la pantalla del formulario

16:02

CuidAR++

Formulario

Temperatura 12.8 °C **TOMAR TEMPERATURA**

¿Percibiste una marcada pérdida del olfato de manera repentina?

☐ SI

☒ NO

¿Percibiste una marcada pérdida del gusto (sabor de los alimentos) de manera repentina?

☐ SI

☒ NO

¿Tenés tos?

☐ SI

☒ NO

¿Tenés dolor de garganta?

☐ SI

☒ NO

¿Tenés dificultad respiratoria o falta de aire?

☐ SI

☒ NO

¿Tenés dolor de cabeza?

☐ SI

☒ NO

¿Tenés dolor de garganta?

☐ SI

☒ NO

¿Tenés dificultad respiratoria o falta de aire?

☐ SI

☒ NO

¿Tenés dolor de cabeza?

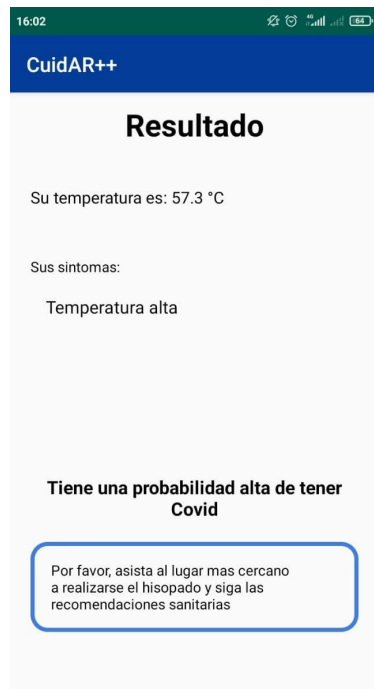
☐ SI

☒ NO

CONFIRMAR

En esta pantalla se deberá contestar a cada pregunta seleccionando la opción “SI” o “NO” e ingresar la temperatura o tocar el botón “tomar temperatura” para utilizar el sensor de temperatura. Una vez que se asegure de los datos ingresados se debe presionar el botón “Confirmar” el cual lo llevará a la pantalla “Resultado” el cual le informará de su condición.

En la pantalla “Resultado” se le muestra su temperatura, sus síntomas, su condición e información importante.



3 Conclusiones

- **Recaudos para que la aplicación sea tolerante a fallos**

Se valida la conexión a internet (wi-fi o red móvil) cada vez que el usuario ingresa a la aplicación para prevenir que se intente conectar al servidor sin estar conectado a internet. [7]

En el registro de usuario, se hacen validaciones de formato de mail [8], cantidad de caracteres numéricos en DNI y longitud de la contraseña antes de enviar el request de registro al web service.

En caso de querer registrarnos con un DNI ya existente, el web service devuelve un mensaje de “El DNI del usuario ya fue registrado”. Este mensaje de error es tomado por la aplicación y mostrado al usuario.

- **Problemas encontrados y resueltos**

Durante el desarrollo de la aplicación, uno de los problemas que más nos surgió fue realizar la conexión con la api y la mayoría de las veces, fue por haber escrito mal el nombre de la variable en la request que mandabamos. Otro problema que se tuvo con la conexión con la api, fue al querer registrar un evento, cuando la request apuntaba a TEST, la conexión nos daba un código 200, por lo que se pensó que ya funcionaba, pero cuando cambiamos al ambiente de PROD, nos tiraba un error 401, esto se soluciono corrigiendo la manera de mandar el Header, ya que en principio habíamos puesto @Header("token") String token, y lo que le mandabamos era solamente el token, faltaba la palabra "Bearer" y el nombre que iba dentro de las comillas era "Authorization". Una vez hecho estos cambios, la aplicación hacía correctamente el registro de los eventos.

Otro problema que tuvimos fue al registrar el evento de tomar temperatura, si se tomaba la temperatura de manera seguida, la aplicación se colgaba, por lo que se eligió registrar el evento de confirmar formulario.

- **Lecciones aprendidas**

En las primeras veces intentando conectar al web service usando retrofit, obteniamos un error de "CLEARTEXT communication not permitted by network security policy", lo solucionamos agregando "usesCleartextTraffic" en true dentro del manifest ya que viene por defecto deshabilitado. Investigando un poco, leímos que esto significa habilitar que información almacenada o transmitida no esté encriptada y al usar HTTP, los datos siguen sin estar encriptados por lo que es un riesgo de seguridad. Si quisiéramos mejorar la seguridad de los datos de la aplicación, deberíamos optar por una comunicación con protocolo HTTPS que provee encriptación. [9]

Además este trabajo nos sirvió para llevar a la práctica la conexión con un web-service mediante API-REST y manejar temas como peticiones, respuestas, tokens y persistencia de datos.

4 Referencias

- [1] «Android, Sensores,» 2021. [En línea]. Available:https://developer.android.com/guide/topics/sensors/sensors_overview?hl=es-419
- [2] «Android, IntentService,» 2021. [En línea]. Available: <https://developer.android.com/reference/android/app/IntentService>.
- [3] «Qastack, obtener Respuesta deserializada,» 2021. [En línea]. Available: <https://qastack.mx/programming/32519618/retrofit-2-0-how-to-get-deserialised-error-response-body>

- [4] «Android, Shared Preference,» 2021. [En línea]. Available:<https://developer.android.com/reference/android/content/SharedPreferences>
- [5] «Android, battery-monitoring,» 2021. [En línea]. Available: <https://developer.android.com/training/monitoring-device-state/battery-monitoring.html>
- [6] «Sociedad Androide , Patron de desbloqueo,» 2021. [En línea]. Available: <https://www.youtube.com/watch?v=5g7dgzC-IDg>
- [7] «Stackoverflow, Validar conexion a internet,» 2021. [En línea]. Available: <https://stackoverflow.com/questions/32547006/connectivitymanager-getnetworkinfo-deprecated>
- [8] «Cesarg, validar email,» 2021. [En línea]. Available: <https://cesarg.cl/validacion-email-java/>.
- [9] «Usejournal, Evitar el error cleartext en comunicación,» 2021. [En línea]. Available:<https://blog.usejournal.com/6-daily-issues-in-android-cleartext-traffic-error-52ab31dd86c2>