UNIVERSIDAD MIGUEL HERNANDEZ DE ELCHE

ESCUELA POLITÉCNICA SUPERIOR DE ELCHE

GRADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

"DESARROLLO DE UN SISTEMA DE DETECCIÓN Y AVISO DE FATIGA FÍSICA"

TRABAJO FIN DE GRADO

Octubre - 2023

AUTOR: Alejandro Mañogil Torregrosa

DIRECTOR/ES: Pedro Pablo Garrido Abenza

Agradecimientos

Quiero expresar mi sincero agradecimiento a las siguientes personas:

A mis padres, por su amor, apoyo y confianza constante en mí.

A mi tutor/a de proyecto, Pedro Pablo Garrido Abenza, por su orientación y apoyo invaluables durante la realización de este trabajo.

Al departamento de Ingeniería de Computadores de la Universidad Miguel Hernández (UMH), por hacer posible que podamos disponer de un dispositivo Garmin para el proyecto.

A todas las personas que contribuyeron directa o indirectamente a este proyecto. Su colaboración fue fundamental para su éxito.

Su ayuda y apoyo fueron esenciales en este viaje académico.

Índice general

Índice de figuras

Capítulo 1

Introducción

En el mundo actual, la tecnología de las telecomunicaciones y el desarrollo de aplicaciones móviles han revolucionado la forma en que interactuamos con el entorno que nos rodea. La creciente adopción de dispositivos inteligentes, como relojes deportivos y smartphones, ha dado lugar a un escenario en el que las aplicaciones desempeñan un papel fundamental en la monitorización de la salud y el rendimiento físico. En este contexto, el presente proyecto se adentra en el diseño y desarrollo de dos aplicaciones interconectadas que aprovechan este tipo de dispositivos y la tecnología Bluetooth para llevar a cabo la recolección y gestión de datos de ritmo cardíaco, con objeto de detectar situaciones de fatiga extrema del usuario.

1.1. Motivación

El desarrollo y la adopción de dispositivos "wearables" como por ejemplo, Fit Bit, Garmin, Smartwatches de Samsung, o Apple Watch, han desencadenado un cambio significativo en la forma en que abordamos el monitoreo de la salud y el rendimiento físico. La capacidad de estos dispositivos para recopilar y analizar datos fisiológicos en tiempo real ofrece un potencial notable en la detección temprana de ataques cardíacos [28] [11], enfermedades neurodegenerativas como el Parkinson [11] [29], estrés y salud mental [10] [24], y estudio del sueño [20]. También pueden ser aplicados a los deportes, para medir la carga de entrenamiento de un deportista [14] y mejorar su rendimiento, o la detección de fatiga extrema durante una actividad física [26].

El uso de dispositivos "wearables" como herramientas para la detección temprana de enfermedades y la identificación de problemas de salud ha atraído la atención de la comunidad médica y científica. Estos dispositivos tienen el potencial de monitorear de manera continua y no invasiva parámetros vitales como el ritmo cardíaco, lo que podría permitir la identificación temprana de anomalías y la toma de decisiones informadas en etapas iniciales de la enfermedad.

Además, la evaluación de la fatiga extrema durante la actividad física es crucial para prevenir lesiones y riesgos para la salud. Los dispositivos "wearables" pueden alertar a los usuarios cuando alcanzan niveles de fatiga que podrían resultar peligrosos, evitando así posibles daños [26].

La precisión en la medición de parámetros vitales es fundamental para garantizar la confiabilidad de los datos recopilados y, en última instancia, la utilidad de estos dispositivos en la detección temprana de enfermedades y la gestión de la fatiga. Tras una revisión bibliográfica sobre el estudio de la precisión de dispositivos "wearables" para la medición del ritmo cardíaco [12] [25] [31] se ha encontrado que estos dispositivos son generalmente fiables en sus mediciones. Sin embargo, es importante destacar que su precisión puede verse afectada durante la realización de ejercicios físicos que requieran movimientos más bruscos o amplios.

1.2. Objetivos

El objetivo principal de este proyecto es diseñar y desarrollar un sistema para la detección de fatiga y aviso a una serie de contactos predefinidos y al propio usuario, para ello se van a implementar dos aplicaciones interconectadas: una destinada a dispositivos Garmin y otra para dispositivos Android. Creando así un sistema asequible para el usuario debido a que la gran mayoría de la población posee un smarthpone, y el coste de un dispositivo Garmin no es muy elevado. Estas aplicaciones permitirán:

1. Aplicación Garmin
    - Recopilar y almacenar datos de ritmo cardíaco para su posterior envío a la aplicación Android.

- Implementar un algoritmo de detección de rangos de ritmo cardíaco, generando alertas en caso de fatiga extrema.

2. Aplicación Android

   - Recibir y almacenar los datos de ritmo cardíaco transmitidos por la aplicación Garmin.
   - Almacenar los datos recibidos de la aplicación Garmin en la base de datos.
   - Visualizar los datos de la base de datos en forma de tablas y gráficos para el análisis del usuario.
   - Enviar alertas de fatiga extrema a contactos predefinidos junto con la ubicación exacta en caso de que se alcance el rango de fatiga extrema.

1.3. Estructura de la memoria

En el presente capítulo 1 se presenta la motivación detrás del proyecto, los objetivos establecidos y una descripción general de la estructura de la memoria. En el capítulo 2 se describe detalladamente cómo se llevó a cabo el proyecto, incluyendo los materiales utilizados, las tecnologías empleadas y la metodología de desarrollo de las aplicaciones interconectadas. Después, en el capítulo 3 se proporciona una visión completa de los resultados obtenidos a lo largo del proyecto. Incluye detalles sobre el código fuente desarrollado, diagramas de flujo, fórmulas empleadas, datos recopilados, interfaz gráfica creada y funcionalidad de las aplicaciones. Por último, en el capítulo 4 presentamos las conclusiones y las posibles direcciones futuras para mejorar y expandir las aplicaciones desarrolladas.

Capítulo 2

Material y métodos

En este capítulo, se describirán los materiales, herramientas y métodos empleados en el desarrollo de las aplicaciones, detallando los pasos tomados para lograr los objetivos planteados.

2.1. Dispositivos utilizados

Para el desarrollo de este proyecto se han utilizado únicamente 2 dispositivos:

- Garmin Forerunner 245
- Xiaomi Mi Lite 11

2.1.1. Garmin Forerunner 245

Para este proyecto se ha utilizado el dispositivo Garmin Forerunner 245, pero cualquier dispositivo Garmin con un API mínimo de 1.2.0 habría sido válido.

Una investigación previa, detallada en el artículo [31], había validado la precisión del Forerunner 235 en la medición del ritmo cardíaco en reposo y su uso general en aplicaciones de monitoreo de la salud. Sin embargo, se observó que su precisión podía variar durante actividades físicas más intensas.

Gracias al departamento de Ingeniería de Computadores de la Universidad Miguel Hernández (UMH), hemos podido contar para este proyecto con el dispositivo Garmin Forerunner 245, una versión más reciente y avanzada en comparación con el Forerunner 235. Agradecemos enormemente la colaboración y el apoyo del departamento.

2.1.2. Xiaomi Mi 11 Lite

Para el desarrollo de la aplicación Android, se seleccionó el smartphone Xiaomi Mi 11 Lite. Sin embargo, cualquier dispositivo Android con una versión SDK 26 o superior es válido para el proyecto. Esto es equivalente a Android 8.0 (Oreo) o versiones posteriores.

La intención principal era que la aplicación de Android fuera para versiones SDK 21 o superior, pero debido a que en el desarrollo de la aplicación se ha utilizado el paquete de funciones java.time, este mínimo se tuvo que subir hasta el SDK 26.

2.2. Herramientas utilizadas

Para el desarrollo de este proyecto se han utilizado las siguientes tecnologías:

- Visual Studio Code
- Android Studio
- Connect IQ SDK
- Connect IQ Mobile SDK
- SQLite

2.2.1. Visual Studio Code

Visual Studio Code [22] es entorno de desarrollo integrado (IDE) que se ejecuta en el escritorio y está disponible para Windows, macOS y Linux. Viene con soporte

integrado para JavaScript, TypeScript y Node.js y tiene un rico ecosistema de extensiones para otros lenguajes y tiempos de ejecución (como C++, C, Java, Python, PHP, Go, .NET).

Para el desarrollo de la aplicación Garmin, la extensión de Monkey C [23] en Visual Studio Code resultó vital para trabajar en este lenguaje.

Monkey C

Monkey C [7] es un lenguaje orientado a objetos creado desde cero, diseñado para facilitar el desarrollo de aplicaciones en dispositivos "wearables". Monkey C es un lenguaje muy familiar a los siguientes lenguajes: Java, PHP, Ruby o Python.

2.2.2. Android Studio

Android Studio [9] es el entorno de desarrollo integrado oficial para la plataforma Android. Fue anunciado el 16 de mayo de 2013 en la conferencia Google I/O, y reemplazó a Eclipse como el IDE oficial para el desarrollo de aplicaciones para Android. La primera versión estable fue publicada en diciembre de 2014.

Para el desarrollo de la aplicación de Android en Android Studio se ha utilizado el lenguaje de programación Java.

Java

Java [17] es un lenguaje de programación de código abierto y orientado a objetos, y una plataforma informática que fue comercializada por primera vez en 1995 por Sun Microsystems.

Java es uno de los dos lenguajes de programación que se utilizan para desarrollar aplicaciones en Android junto a Kotlin [19]. En nuestro caso hemos decidido utilizar Java ya que tenemos un mayor control sobre este lenguaje que sobre Kotlin.

2.2.3. Connect IQ SDK

El Connect IQ SDK [4] es un conjunto de herramientas de desarrollo proporcionado por Garmin, diseñado para permitir a los desarrolladores crear aplicaciones, widgets y campos de datos personalizados que se ejecutan en dispositivos Garmin, como relojes inteligentes y dispositivos de navegación GPS (Global Positioning System). Este kit de desarrollo ofrece acceso a las funciones y sensores de los dispositivos Garmin, permitiendo a los desarrolladores crear experiencias personalizadas para usuarios, como aplicaciones de fitness, relojes de carátula personalizados y widgets que muestran información útil en la pantalla de un dispositivo Garmin. El Connect IQ SDK brinda a los desarrolladores la capacidad de ampliar y mejorar la funcionalidad de los dispositivos Garmin, lo que enriquece la experiencia de los usuarios y abre oportunidades para crear aplicaciones innovadoras en el ecosistema de Garmin. Además, este conjunto de herramientas también incluye un emulador que permite probar las aplicaciones en los distintos dispositivos Garmin, facilitando así el proceso de desarrollo y asegurando la compatibilidad con una variedad de plataformas.

2.2.4. Connect IQ Mobile SDK

El Connect IQ Mobile SDK [3] es una extensión del Connect IQ SDK de Garmin que permite a los desarrolladores crear aplicaciones móviles que se integran de manera estrecha con dispositivos Garmin compatibles. Con esta herramienta, los desarrolladores pueden diseñar aplicaciones móviles que se comunican con relojes inteligentes, dispositivos de navegación GPS y otros productos de Garmin. Esto posibilita la sincronización de datos, la configuración de dispositivos, el control remoto y otras interacciones entre la aplicación móvil y el dispositivo Garmin. El Connect IQ Mobile SDK es esencial para crear experiencias de usuario completas y conectadas en el ecosistema Garmin, permitiendo a los desarrolladores aprovechar al máximo las capacidades de los dispositivos y ofrecer soluciones más completas a los usuarios.

2.2.5. SQLite

SQLite [30] es un sistema de gestión de bases de datos relacional de código abierto que se destaca por ser ligero, eficiente y autónomo, lo que significa que no requiere un servidor independiente para funcionar. Este motor de bases de datos se utiliza comúnmente en aplicaciones móviles y de escritorio para almacenar y administrar datos de manera local. SQLite es conocido por su facilidad de uso y su capacidad para gestionar bases de datos pequeñas o medianas de manera eficiente, lo que lo hace ideal para aplicaciones que necesitan un almacenamiento de datos rápido y confiable sin la complejidad de un servidor de bases de datos completo.

Capítulo 3

Resultados

En este capítulo comenzaremos con un análisis del funcionamiento de algunas partes de las aplicaciones explorando los diagramas de flujo que ilustran los procesos internos y la interacción entre las distintas aplicaciones en la sección 3.1. A continuación, en la sección 3.2 tendremos un análisis de las partes más fundamentales del código fuente que contribuyen a la funcionalidad y el rendimiento de las aplicaciones. En la sección 3.3 revisaremos la base de datos con la que hemos trabajado en este proyecto, analizaremos las distintas tablas creadas y su creación en el código fuente. Además, proporcionaremos una visión general del algoritmo implementado para la detección de fatiga extrema y las fórmulas usadas en la sección 3.4.

Posteriormente, en la sección 3.5 presentaremos la interfaz gráfica desarrollada para ambas aplicaciones, destacando su diseño y usabilidad. Finalmente, en la sección 3.6 proporcionaremos un manual de usuario que guiará a los usuarios a través de las funcionalidades y la navegación de las aplicaciones. A medida que avanzamos en este capítulo, se proporcionarán ejemplos visuales y detalles técnicos que respalden los logros alcanzados en la implementación de las aplicaciones interconectadas.

3.1. Diagramas de flujo

En esta sección, se presentarán tres diagramas de flujo destinados a ofrecer una visión simplificada pero clara del funcionamiento de las aplicaciones desarrolladas. Si bien

estos diagramas no entrarán en detalles minuciosos, proporcionarán una representación visual que ayudará a comprender los procesos esenciales de las aplicaciones.

Dos de los diagramas de flujo representan diferentes modos de funcionamiento de la aplicación Garmin. El tercer diagrama de flujo mostrará el caso en el que la aplicación Android recibe un aviso de fatiga extrema por parte de la aplicación Garmin.

3.1.1. App Garmin - Detección de fatiga extrema

Aquí estamos en el caso en el que la aplicación Garmin está trabajando con el algoritmo implementado para la detección de fatiga extrema durante una actividad física.

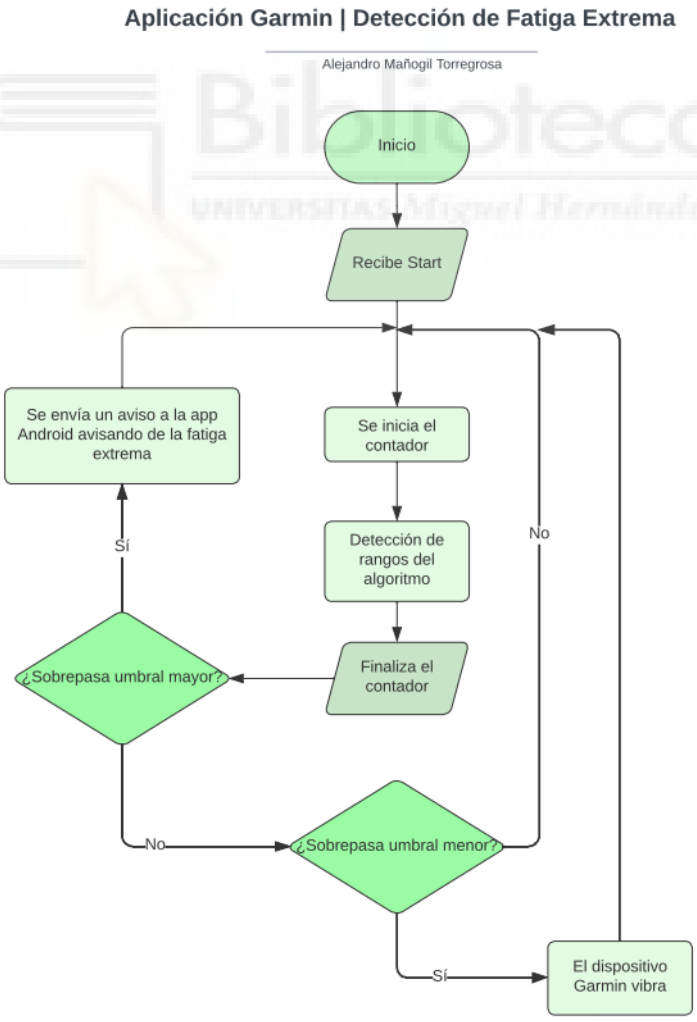

Figura 3.1: Detección de fatiga extrema

En el diagrama de flujo de la figura 3.1 se puede observar el funcionamiento de este modo de detección de fatiga extrema. Para empezar, cuando accedemos a este modo en la aplicación Garmin, hay que seguir una serie de instrucciones que aparecen en pantalla:

Figura 3.2: Instrucciones de conexión

Como indican las instrucciones de la figura 3.2, hay que conectar ambas aplicaciones y pulsar el botón `"Start"` en el dispositivo Garmin, este internamente mandará un mensaje a la aplicación Android. Si esta está activa y conectada correctamente al dispositivo Garmin, devolverá el mensaje al Garmin, una vez que reciba este mensaje se iniciará el contador de 60 segundos. Esto se hace para comprobar que ambas aplicaciones están funcionando correctamente y están bien conectadas.

Una vez se activa el contador, la aplicación con la información que obtiene del sensor del ritmo cardíaco detecta en que rango del algoritmo nos encontramos. Una vez finalizado el contador, si el rango en el que nos encontramos es el de fatiga extrema, se envía un aviso a la aplicación Garmin y se reinicia el contador, si no es así, comprueba si se ha superado el umbral menor, si este se ha superado vibrará el dispositivo Garmin y se reiniciará el contador. En el caso contrario, se reinicia igualmente el contador y se sigue implementado el algoritmo.

En el diagrama de flujo no aparece reflejado pero cabe destacar que si en algún momento se cambia al otro modo de funcionamiento, la aplicación pararía el contador. En caso de que no se cambie de modo, el contador seguiría activo siempre.

3.1.2. App Garmin - Envío de datos a la aplicación Android

Cuando la aplicación Garmin está trabajando en el modo de monitorización y recibe un input de envío de datos, el funcionamiento se muestra en el diagrama de flujo de la figura 3.3.

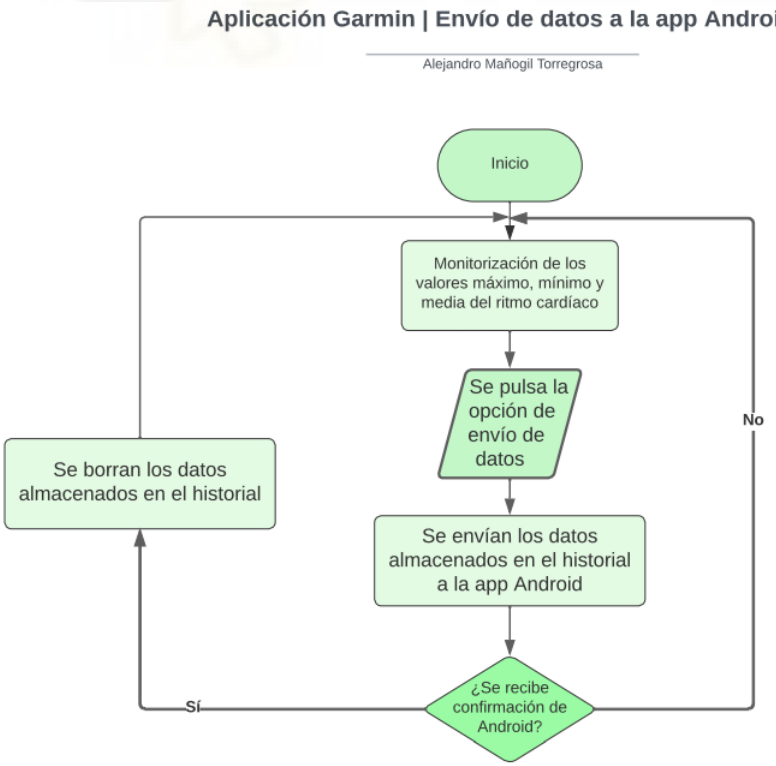

Figura 3.3: Envío de datos a la app Android

La aplicación Garmin cuando recibe dicho input envía los datos registrados en su historial a la aplicación Android y espera una confirmación antes de borrar los datos. Si la aplicación Garmin recibe la confirmación de la otra parte, borra los datos registrados en el historial y sigue el funcionamiento normal de la aplicación. En caso contrario, la aplicación Garmin no borraría ningún dato y volvería al funcionamiento normal. Esto evita la pérdida de datos por una conexión fallida o por un envío fallido.

3.1.3. App Android - Caso de aviso de fatiga extrema

Finalmente, el último diagrama de flujo que vamos a analizar es el caso en el que la aplicación de Android recibe un aviso de fatiga extrema.

Figura 3.4: Detección de fatiga extrema

Como podemos observar en la figura 3.4, es un diagrama de flujo sencillo. No están incluidas cada una de las acciones que la aplicación Android realizaría en caso de que el mensaje que recibe no fuera de fatiga extrema, pero es importante analizar esta parte.

La aplicación Garmin cuando está en el modo de detección de fatiga envía avisos a la aplicación Android cada cierto tiempo para indicarle en qué rango del algoritmo nos encontramos. Si uno de estos avisos es de fatiga extrema, la aplicación Android obtendría la latitud y longitud exacta del dispositivo móvil y mandaría un mensaje personalizado a cada uno de los contactos predefinidos por el usuario mediante correo electrónico.

3.2. Código fuente

En esta sección se van a tratar los fragmentos de código más relevantes. A continuación se abordarán los siguientes aspectos:

- Conexión entre aplicaciones
- Comunicación entre aplicaciones
- Obtención de latitud y longitud
- Envío de correo
- Monitorización del ritmo cardíaco

3.2.1. Conexión entre aplicaciones

La conexión entre aplicaciones la gestiona en su totalidad la aplicación de Android, concretamente la primera pantalla de la aplicación. Podemos observar esta pantalla en la figura 3.5.

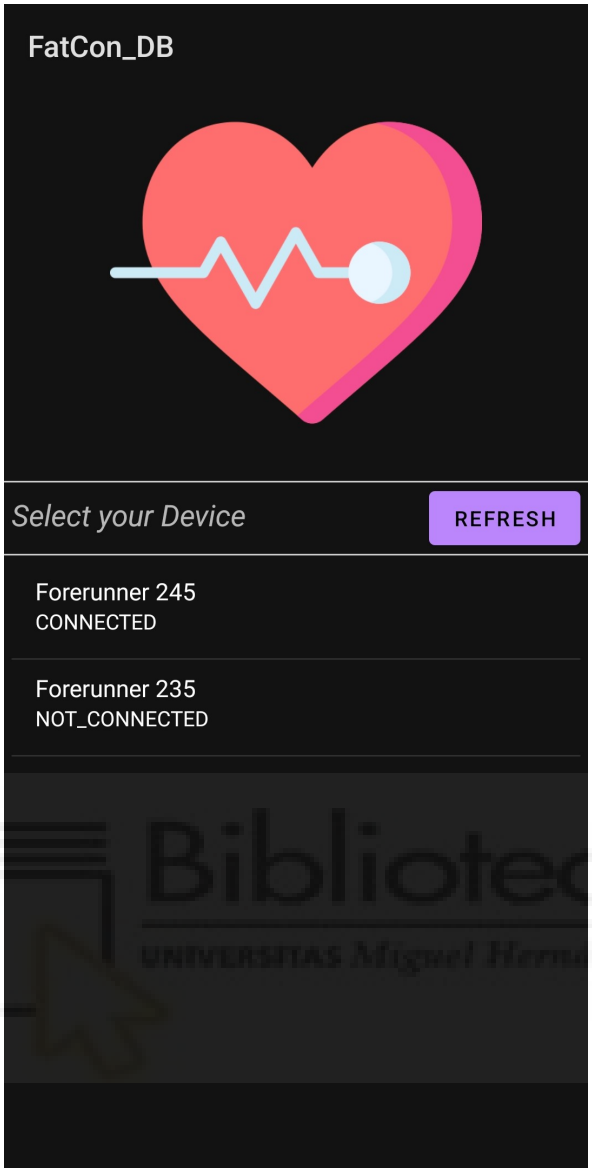

Figura 3.5: Pantalla de conexión

Para que esta conexión sea posible, la aplicación de Android debe conocer el ID de la aplicación de Garmin, por lo que en el código se declara este identificador como una constante:

```
1 public static final String CIQ_APP_ID = "76792b07-58a0-4b98-
    ac96-a75135f55421";
```

A este identificador podemos acceder en el archivo `"manifest.xml"` del proyecto de la aplicación Garmin.

A parte del ID, las constantes y variables que vamos a necesitar para la conexión

entre aplicaciones son las siguientes:

```
1  public static final String IQ_DEVICE = "IQDevice";
2  private ConnectIQ connectIQ;
3  private CIQDeviceAdapter deviceAdapter;
4  private boolean sdkIsReady = false;
```

También necesitaremos un gestor de eventos o "listener" para manejar los distintos eventos de la conexión:

```
1  private ConnectIQ.ConnectIQListener connectIQListener = new
       ConnectIQ.ConnectIQListener() {
2    @Override
3    public void onSdkReady() {
4      Log.i("MYLOG", "sdk preparado");
5      loadDevices();
6      sdkIsReady = true;
7    }
8    @Override
9    public void onInitializeError(ConnectIQ.IQSdkErrorStatus
         iqSdkErrorStatus) {
10     Log.i("MYLOG", "sdk falló iniciando");
11     String msg = "No se pudo inicializar el SDK. Error: " +
           iqSdkErrorStatus.name();
12     Toast.makeText(MainActivity2.this, msg, Toast.LENGTH_LONG
           ).show();
13     sdkIsReady = false;
14   }
15   @Override
16   public void onSdkShutDown() {
17     Log.i("MYLOG", "sdk apagado");
18     sdkIsReady = false;
19   }
20 };
```

En el "listener" hemos sobreescrito los siguientes métodos: "onInitializeError", "onSdkReady" y "onSdkShutDown", para que se ajusten a nuestra aplicación.

En el método "onCreate", que se ejecuta cuando se inicia la aplicación, tenemos el siguiente código:

```
1  ListView deviceListView = (ListView) findViewById(R.id.
       deviceListView);
2  deviceAdapter = new CIQDeviceAdapter(this, android.R.layout.
       simple_list_item_2);
3  deviceListView.setAdapter(deviceAdapter);
4  deviceListView.setOnItemClickListener(new AdapterView.
       OnItemClickListener() {
5    @Override
6    public void onItemClick(AdapterView<?> parent, View view,
         int position, long id) {
7      IQDevice device = deviceAdapter.getItem(position);
8      forwardToCounter(device);
9    }
10 });
11 Button buttonRefresh = (Button) findViewById(R.id.
       buttonRefresh);
12 buttonRefresh.setOnClickListener(new View.OnClickListener() {
13   @Override
14   public void onClick(View v) {
15     loadDevices();
16   }
17 });
18 connectIQ = ConnectIQ.getInstance(this, ConnectIQ.
       IQConnectType.WIRELESS);
19 connectIQ.initialize(this, true, connectIQListener);
```

Lo primero que hacemos es obtener una referencia del ListView del layout, es decir, obtenemos un objeto de la lista de la pantalla de la aplicación. A continuación creamos un objeto de la clase "CIQDeviceAdapter" y se establece dicho adaptador al

ListView. Establecemos la función de "onClickListener" del ListView para cuando se pulse una opción de la lista se guarde en "device" el dispositivo el cual se ha seleccionado y se llamará a la función "forwardToCounter". Después, obtenemos una referencia del botón de refrescar y ponemos que al pulsarlo se llame a la función "loadDevices". Finalmente, creamos una instancia de la clase "ConnectIQ" y la inicializamos.

Cuando se selecciona un dispositivo de los que salen en el ListView de la pantalla principal, se llama a la función "forwardToCounter", esta función hace lo siguiente:

```
1  public void forwardToCounter(IQDevice device) {
2    IQDevice.IQDeviceStatus status = device.getStatus();
3    if (status == IQDevice.IQDeviceStatus.CONNECTED) {
4      try {
5        connectIQ.getApplicationInfo(CIQ_APP_ID, device, new
             ConnectIQ.IQApplicationInfoListener() {
6
7          @Override
8          public void onApplicationInfoReceived(IQApp iqApp) {
9            Intent intent = new Intent(MainActivity2.this,
                 MainActivity.class);
10           intent.putExtra(MainActivity2.IQ_DEVICE, device);
11           startActivity(intent);
12         }
13
14         @Override
15         public void onApplicationNotInstalled(String s) {
16           Toast.makeText(MainActivity2.this, "Instala la
                 aplicación de CIQ!", Toast.LENGTH_LONG).show();
17         }
18       });
19     } catch (InvalidStateException |
           ServiceUnavailableException e) {
20       e.printStackTrace();
21     }
```

```
22    } else {
23      Toast.makeText(MainActivity2.this, "Conecta tu
           dispositivo primero!", Toast.LENGTH_SHORT).show();
24    }
25  }
```

Primero comprobamos el *status* del dispositivo, si está desconectado aparece el siguiente mensaje: `"Conecta tu dispositivo primero!"`. En cambio, si está conectado se utiliza la función `"getApplicationInfo"` para obtener información sobre la aplicación Garmin. Si la aplicación Garmin no está instalada en el dispositivo seleccionado se mostrará el mensaje `"Instala la aplicación de CIQ!"`, y si está instalada se crea un Intent (objeto que usaremos para cambiar de actividad) para abrir la actividad de la siguiente pantalla pasando como dato la constante `"IQ_DEVICE"` y la variable `"device"`.

Finalmente, cuando el SDK está listo y cuando se pulsa el botón de refrescar se llama a la función `"loadDevices"`, esta función hace lo siguiente:

```
1   public void loadDevices() {
2     Log.i("MYLOG", "Ejecutando loadDevices");
3     try {
4       List<IQDevice> devices = connectIQ.getKnownDevices();
5       if (devices != null) {
6         deviceAdapter.setDevices(devices);
7         for(IQDevice device : devices) {
8           connectIQ.unregisterForDeviceEvents(device);
9           connectIQ.registerForDeviceEvents(device,
               deviceEventListener);
10            }
11          }
12    } catch (InvalidStateException |
         ServiceUnavailableException e) {
13          e.printStackTrace();
14    }
15  }
```

Esta función guarda dentro de la lista `"devices"` los dispositivos conocidos. Si la variable no es nula se añaden estos dispositivos al adaptador de la ListView y se recorre la lista de dispositivos uno a uno para desregistrar y registrar los eventos de cada dispositivo (hace falta desregistrarlos antes para que no haya dúplicas).

3.2.2. Comunicación entre aplicaciones

A diferencia de la conexión entre aplicaciones, de la cual solo se encarga la aplicación de Android, de la comunicación entre aplicaciones se encarga la aplicación de Garmin. Esta es la encargada de enviar los datos almacenados de la monitorización del ritmo cardíaco, o bien la encargada de informar a la aplicación de Android en que rango nos encontramos del algoritmo implementado.

Lo primero que hay que comentar es la función de la aplicación de Android que se encarga de los mensajes recibidos provenientes de la aplicación Garmin. Esta función la vamos a ver en 3 partes, la primera es la siguiente:

```
1 connectIQ.registerForAppEvents(device, ciqApp, new ConnectIQ.
     IQApplicationEventListener() {
2    @Override
3    public void onMessageReceived(IQDevice iqDevice, IQApp
        iqApp, List<Object> list, ConnectIQ.IQMessageStatus
        iqMessageStatus) {
4
5       if (iqMessageStatus == iqMessageStatus.SUCCESS) {
6          List<String> msg = new ArrayList<>();
7          String dta = list.get(0).toString();
8          msg = Arrays.asList(dta.split(","));
9          ...
```

Podemos observar que la función recibe como argumento una lista de objetos. Los mensajes que se mandan entre ambas aplicaciones siempre van a ser Strings, por lo que obtenemos los datos de la posición `'0'` de la lista de objetos, y los guardamos en

otra lista pero separando los valores por ’,’. Esta es la primera parte de la función que recibe los datos, las siguientes partes se encargan de distinguir que tipo de datos se ha mandado.

Desde el dispositivo Garmin, para enviar los datos de la monitorización del ritmo cardíaco a la aplicación de Android, existe una opción en el menú de la aplicación. Cuando esta opción se ejecuta se llama a la siguiente función:

```
function enviarHistorialAndroid(){
  data = history;
  Communications.transmit(data,null,listener);
  data = "";
  ResetHistory();
}
```

Esta función llama a la función "transmit" de la librería "Communications" y envía la variable "data". Esta variable se observará detalladamente más adelante, pero básicamente es un String en el que se guarda la fecha, hora, ritmo cardíaco máximo, mínimo y el valor medio.

La parte de la función de Android que se encarga de los datos provenientes a dicha monitorización es la siguiente:

```
if (!msg.get(0).equals("[-- | --") ) {
  sendData("OK");
  Toast.makeText(getApplicationContext(), "Mensaje recibido",
     Toast.LENGTH_SHORT).show();
  bddManager dbHelper = new bddManager(MainActivity.this);
  dbHelper.insertarRegistrosHR(msg);
  try{
    NavHostFragment navHostFragment = (NavHostFragment)
       getSupportFragmentManager().findFragmentById(R.id.
       nav_host_fragment_content_main);
    boardFragment = (BoardFragment) navHostFragment.
       getChildFragmentManager().getFragments().get(0);
```

```
9       if (boardFragment != null) {
10        boardFragment.destroyTable();
11        boardFragment.createTable();
12      }
13    }catch (ClassCastException e){
14      e.printStackTrace();
15    }
16  } else {
17    return;
18  }
```

Primero comprueba si el mensaje recibido es nulo, si es así, se acaba la función. En caso contrario, manda la confirmación de que han llegado los datos, aparece el siguiente mensaje: `"Mensaje recibido"` y se crea un objeto de la clase bddManager, que es la que se encarga de trabajar con la base de datos, y se llama a una función que inserta los distintos valores recibidos en la base de datos. Posteriormente, dentro del try se destruye y se vuelve a crear la tabla de la base de datos. Esto lo estudiaremos en la sección de interfaz gráfica.

La otra opción de envío de datos desde el dispositivo Garmin viene relacionado con la implementación del algoritmo de detección de fatiga extrema [26]. El dispositivo Garmin avisa al dispositivo Android en que región del algoritmo se encuentra el usuario, a menos que la última región enviada fuera la misma que la actual. Vamos a verlo en código:

```
1  var rango = new [4];
2  var ult = new [4];
3
4  function contador() as Void{
5    if(_contador == false){
6      myTimer.start(method(:onEventTriggered), 60000, true);
7      _contador = true;
8    }
9  }
10
```

```
11 function onEventTriggered() as Void {
12   for(var i=0; i<rango.size();i++){
13     if(rango[i]==true && ult[i]!=rango[i]){
14       rangosAlgoritmo(i);
15     }
16   }
17   ult = rango;
18 }
19
20 function rangosAlgoritmo(num){
21   if(num==2){
22     Attention.vibrate(vibrateData2);
23   }else if(num==3){
24     Attention.playTone({:toneProfile=>toneProfile});
25     Attention.vibrate(vibrateData2);
26     Communications.transmit("Mail",null,listener);
27   }
28 }
```

Esta parte del código solo se pone en funcionamiento cuando en la aplicación de Garmin activamos el modo de detección de fatiga extrema. Básicamente, se activa un contador de 60 segundos y le ponemos como método para cuando se cumpla el tiempo la función `"onEventTriggered"`, el true del tercer argumento de la función significa que cuando termine el contador se vuelva a iniciar. Cuando el contador llega a 0 se llama a la función `"onEventTriggered"`, esta función comprueba que el rango en el que nos encontramos actualmente no es el mismo que el rango anterior (la variable rango representa el rango actual y ult representa el último), si el rango es el mismo que el anterior no se envía ningún mensaje, si es distinto se llamará a la función `"rangosAlgoritmo"`. Esta función dependiendo del rango en el que estemos (viene indicado en la variable que recibe como argumento), realizará una acción u otra.

Finalmente, la parte de la función `"onMenssageReceived"` de la aplicación Android que nos falta por analizar es la siguiente:

```
...
if(msg.get(0).equals("Mail")){
  getCoordenada();
} else if(msg.get(0).equals("FatCon")){
  sendData("Algoritmo");
}
...
```

Los puntos suspensivos que tenemos al principio representan la primera parte analizada de esta función, los que se encuentran al final representan la segunda parte analizada. Si recibimos un `"Mail"` significa que estamos en el rango de fatiga extrema, por lo que se llama a la función `"getCoordenada"` que obtiene las coordenadas exactas en las que se encuentra el dispositivo Android y envía un correo a los contactos predefinidos con dicha ubicación y un mensaje de aviso. Si llega un mensaje `"FatCon"` la aplicación enviará un mensaje al Garmin con el mensaje `"Algoritmo"`, este es el intercambio de mensajes que hacen las aplicaciones para comenzar el funcionamiento de detección de fatiga extrema.

3.2.3. Obtención de latitud y longitud

Antes de nada, para la obtención de la latitud y longitud del dispositivo móvil, la aplicación requiere que el usuario le de permisos de ubicación precisa. Para esto, en la función `"onCreate"` tendremos el siguiente código:

```
if(permisos==false){
  ActivityCompat.requestPermissions(MainActivity.this,new
     String[]{Manifest.permission.ACCESS_FINE_LOCATION},
     REQUEST_CODE);
}
```

Este trozo de código comprueba si el booleano `"permisos"` es falso, este booleano lo he inicializado como falso, ya que al iniciarse la aplicación por primera vez, no tendremos el permiso que necesitamos. En caso de que este booleano sea falso, se

ejecutará la función "requestPermissions" que lo que hace es solicitarle al usuario el permiso de ubicación precisa. La función que maneja la respuesta de dicha solicitud es la siguiente:

```
@Override
public void onRequestPermissionsResult(int requestCode,
    @NonNull String[] permissions, @NonNull int[] grantResults
    ) {
  super.onRequestPermissionsResult(requestCode, permissions,
      grantResults);
  if (requestCode==REQUEST_CODE && grantResults.length>0) {
    if (grantResults[0]==PackageManager.PERMISSION_GRANTED) {
      permisos = true;
    } else {
      Toast.makeText(this, "La app requiere permisos de
          ubicación", Toast.LENGTH_LONG).show();
    }
  }
}
```

Si el usuario nos concede el permiso solicitado, el booleano se pondrá en true. En caso contrario, la aplicación mostrará el mensaje: "La app requiere permisos de ubicación", y cada vez que inicie de nuevo la aplicación, se le volverá a solicitar dicho permiso. Una vez explicado todo esto, podemos empezar a analizar cómo se obtienen la latitud y la longitud.

Cuando el dispositivo Garmin detecta que el usuario se encuentra en el rango de fatiga extrema del algoritmo implementado, se comunica con la aplicación Android para que ésta pueda obtener la ubicación exacta del dispositivo y avise a los contactos predefinidos en la aplicación por el usuario. Cuando la aplicación Android recibe este aviso, llama a la función "getCoordenada", como hemos visto anteriormente. Vamos a analizar esta función en 2 partes, la primera parte es:

```
1  private void getCoordenada() {
2    try {
3      LocationRequest locationRequest = new LocationRequest();
4      locationRequest.setInterval(10000);
5      locationRequest.setFastestInterval(3000);
6      locationRequest.setPriority(LocationRequest.
           PRIORITY_HIGH_ACCURACY);
7      if (ActivityCompat.checkSelfPermission(this, Manifest.
           permission.ACCESS_FINE_LOCATION) != PackageManager.
           PERMISSION_GRANTED) {
8        return;
9      }
10     LocationServices.getFusedLocationProviderClient(this).
           requestLocationUpdates(locationRequest, new
           LocationCallback() {
11       ...
12     }, Looper.myLooper());
13   }catch (Exception ex){
14     System.out.println("Error es :" + ex);
15   }
16 }
```

Esta función crea una instancia de la clase `"LocationRequest"` que se utiliza para solicitar actualizaciones de ubicación al sistema, se configura el intervalo de actualizaciones de ubicación a 10 segundos, y el más rápido a 3 segundos, y se establece la prioridad `"PRIORITY_HIGH_ACCURACY"`, para obtener la ubicación más precisa posible. A continuación, se vuelve a comprobar si la aplicación tiene el permiso de ubicación precisa, y en caso contrario, se interrumpe la función. Finalmente, si la aplicación tiene dicho permiso se solicitan actualizaciones de ubicación al cliente de `"FusedLocationProvider"` utilizando el método `"requestLocationUpdates"` y se proporciona el objeto `"LocationCallback"` para que maneje las actualizaciones de ubicación. A continuación, vamos a analizar la segunda parte de la función, esta corresponde a los puntos suspensivos que están dentro de los corchetes al crear el

"LocationCallback":

```
@Override
public void onLocationResult(LocationResult locationResult) {
  super.onLocationResult(locationResult);
  LocationServices.getFusedLocationProviderClient(
     MainActivity.this).removeLocationUpdates(this);
  if (locationResult != null && locationResult.getLocations()
     .size() > 0) {
    int latestLocationIndex = locationResult.getLocations().
       size() - 1;
    latitud = locationResult.getLocations().get(
       latestLocationIndex).getLatitude();
    longitud = locationResult.getLocations().get(
       latestLocationIndex).getLongitude();
    MailSender mail = new MailSender();
    String mailOrigen = getEmailOrigen();
    String contrasenaOrigen = getContrasenaOrigen();
    if(mailOrigen!=null && contrasenaOrigen!=null){
      if( !mailOrigen.equals("") && !contrasenaOrigen.equals(
         "")){
        mail.SendEmailWithEmail(latitud,longitud,getNombres()
           ,getEmails(),getNombrePropietario(),mailOrigen,
           contrasenaOrigen);
      } else{
        mail.SendEmail(latitud,longitud,getNombres(),
           getEmails(),getNombrePropietario());
      }
    } else{
      mail.SendEmail(latitud,longitud,getNombres(),getEmails
         (),getNombrePropietario());
    }
  }
}
```

En esta función se llama al método "removeLocationUpdates" para dejar de recibir actualizaciones de ubicación. Posteriormente, se comprueba que hemos obtenido la ubicación correctamente, en su caso, obtenemos la latitud y la longitud de la ubicación recibida más reciente. Una vez que tenemos los valores de latitud y longitud creamos una instancia de la clase "MailSender" que es la que se encarga del envío de correos.

Llegados a este punto, cabe destacar que hay 2 formas de efectuar el envío de correos:

1. Envío con correo remitente personal
2. Envío con correo remitente por defecto

Si el usuario ha añadido su correo personal a la aplicación, el correo de aviso a sus contactos se enviará desde dicho correo, en caso contrario, se enviará desde el correo de la aplicación. Para distinguir en que opción estamos trabajando, se llama a las funciones "getEmailOrigen" y "getContrasenaOrigen". En caso de que estas funciones devuelvan un valor nulo o vacío, se llamará a la función "SendEmail" de la clase "MailSender", que es la que se encarga de enviar correos utilizando la cuenta por defecto. En caso contrario, estas funciones devolverán los valores de usuario y contraseña guardados en la base de datos de la aplicación, se llamará a la función "SendEmailWithEmail" y se añadirán como argumentos el usuario y la contraseña del correo que se va a utilizar como remitente. Las funciones de "getEmails" y "getNombres" devuelven una lista con los nombres y correos de los contactos predefinidos por el usuario en la aplicación Android, y la función "getNombrePropietario" devuelve el nombre del usuario.

3.2.4. Envío de correo

Una vez obtenidas la longitud y la latitud, se llama a la función de envío de correo. Como hemos comentado antes, existen dos modos de enviar el correo, la única diferencia entre ambos es la cuenta de correo desde la cual se envían los correos, pero ambas funciones trabajan igual. Vamos a analizar la función "SendEmailWithEmail" en dos partes, la primera:

```
1  public void SendEmailWithEmail(Double latit, Double longit,
       List<String> nombres, List<String> emails, String
       nombrePropietario, String _origen, String _contrasena){
2    this.nombrePropietario = nombrePropietario;
3    int size = nombres.size();
4    for(int i=0;i<size;i++){
5      destino = emails.get(i);
6      nombre = nombres.get(i);
7      mensaje = "Hola "+nombre+"!\n\nLe informamos desde FatCon
           de un aviso de fatiga extrema de "+nombrePropietario+
         " en la siguiente ubicación:\n\n Latitud: "+ latit + "
         , Longitud: "+longit;
8      try {
9        ...
10         } catch (AddressException e) {
11       e.printStackTrace();
12     } catch (MessagingException e) {
13       e.printStackTrace();
14     }
15   }
16 }
```

Esta primera parte de la función es sencilla, se inicia un bucle que recorre cada entrada en las listas de nombres y emails y se personaliza un mensaje para cada uno. La segunda parte de la función es la que se encuentra dentro del try, vamos a analizarla:

```
1 String stringHost = "smtp.gmail.com";
2 Properties properties = System.getProperties();
3 properties.put("mail.smtp.host", stringHost);
4 properties.put("mail.smtp.port", "465");
5 properties.put("mail.smtp.ssl.enable", "true");
6 properties.put("mail.smtp.auth", "true");
7 javax.mail.Session session = Session.getInstance(properties,
```

```
    new Authenticator() {
  @Override
  protected PasswordAuthentication getPasswordAuthentication
      () {
    return new PasswordAuthentication(_origen, _contrasena);
  }
});
MimeMessage mimeMessage = new MimeMessage(session);
mimeMessage.addRecipient(Message.RecipientType.TO, new
    InternetAddress(destino));
mimeMessage.setSubject(asunto);
mimeMessage.setText(mensaje);
Thread thread = new Thread(new Runnable() {
  @Override
  public void run() {
    try {
      Transport.send(mimeMessage);
    } catch (MessagingException e) {
      e.printStackTrace();
    }
  }
});
thread.start();
```

Esta segunda parte de la función es la encargada de configurar y enviar el correo electrónico. Primero se crea un objeto `"Properties"` y configuramos sus distintas propiedades para la conexión SMTP. Posteriormente, creamos una instancia de la clase `"Session"` utilizando las propiedades configuradas, y un objeto `"Authenticator"` que se utiliza para proporcionar las credenciales de autenticación al servidor SMTP. Creamos una instancia de la clase `"MimeMessage"` con la sesión previamente creada, y le proporcionamos el destino, el asunto y el mensaje del correo electrónico. Finalmente, creamos un hilo para enviar dicho correo y lo ponemos en funcionamiento llamando a la función `"start"` del objeto `"Thread"`.

3.2.5. Monitorización del ritmo cardíaco

El dispositivo Garmin es el encargado de la monitorización del ritmo cardíaco del usuario. Para que esto sea posible hacen falta los siguientes pasos:

```
Sensor.setEnabledSensors( [Sensor.SENSOR_HEARTRATE] );
Sensor.enableSensorEvents( method(:onSensor) as Null);
```

En la función `"initialize"` de la aplicación Garmin (es equivalente al `"onCreate"` de Android) se habilita el sensor de ritmo cardíaco y se establece como controlador de eventos del sensor a la función `"onSensor"`. Esta función la vamos a analizar en 2 partes, la primera de ellas:

```
function onSensor(sensorInfo){
  if( sensorInfo.heartRate != null ) {
    if (haveHR == false) {
      haveHR = true;
      Attention.vibrate(vibrateData2);
    }
    HR = sensorInfo.heartRate.toString();
    HRvalue = sensorInfo.heartRate;

    _avg = _avg + HRvalue;
        count++;

    _today = Gregorian.info(Time.now(), Time.FORMAT_MEDIUM);
    actualHour = _today.hour;
    if(currentHour != actualHour){
      record_date();
      currentHour = actualHour;
    }
    for(var i=HRlist.size()-1;i>0;i--){
      HRlist[i]=HRlist[i-1];
    }
    HRlist[0]=HRvalue;
```

```
23      if (sensorInfo.heartRate < RHRval) {
24        RHRval = sensorInfo.heartRate;
25        RHR = HR;
26      }
27      if (sensorInfo.heartRate < minHR) {
28        minHR = sensorInfo.heartRate;
29      }
30      if (sensorInfo.heartRate > maxHR) {
31        maxHR = sensorInfo.heartRate;
32      }
33                              ...
34    } else {
35      HR = "--";
36      haveHR = false;
37    }
38    Ui.requestUpdate();
39  }
```

Esta primera parte de la función es la que comprueba si el sensor nos está devolviendo un valor no nulo, en ese caso se encarga de ir actualizando el valor de la variable que se muestra por pantalla y los valores de ritmo cardíaco máximo, mínimo y la media, comprueba si se cambia la hora (en su caso almacenará los datos) y va actualizando los valores de la lista `"HRlist"` que es la que se usa para mostrar los valores actuales en una gráfica.

La segunda parte de la función se encuentra donde están los puntos suspensivos:

```
1  //SI HR<60
2  if(HRvalue<60){
3    rango[0]=true;
4    rango[1]=false;
5    rango[2]=false;
6    rango[3]=false;
7  }
8
```

```
9  //SI 60 < HR < UMBRALMENOR*MAXHR
10 else if(HRvalue>60 && HRvalue<_umbralMenor*_maxH){
11   rango[1]=true;
12   rango[0]=false;
13   rango[2]=false;
14   rango[3]=false;
15 }
16 //SI UMBRALMENOR*MAXHR < HR < UMBRALMAYOR*MAXHR
17 else if(HRvalue>_umbralMenor*_maxHR && HRvalue<_umbralMayor*
      _maxHR){
18   rango[2]=true;
19   rango[0]=false;
20   rango[1]=false;
21   rango[3]=false;
22 }
23 //SI ALCANZA EL UMBRAL MAYOR
24 else{
25   rango[3]=true;
26   rango[0]=false;
27   rango[1]=false;
28   rango[2]=false;
29 }
```

Esta segunda parte de la función es la que se encarga de comprobar en qué rango del algoritmo implementado estamos.

3.3. Base de datos

Una de las dos funciones principales de nuestras aplicaciones es la monitorización y almacenamiento de los datos de ritmo cardíaco del usuario. Este almacenamiento se lleva a cabo en la base de datos SQLite [30], que es el gestor de bases de datos relacional que incorpora Android Studio. Para el desarrollo de las aplicaciones hemos necesitado crear 3 tablas:

- Datos monitorizados
- Contactos
- Contraseña de aplicación

Cabe destacar que para la creación de dichas tablas hay que crear una clase de tipo `"SQLiteOpenHelper"` y ejecutar los comandos de creación de tabla en la función `"onCreate"`. Esta función en las clases de este tipo solo se ejecuta cuando se crea la base de datos. Nuestra función `"onCreate"` se vería de la siguiente manera:

```
1  private static final String COMMENTS_TABLE_CREATE1 = "CREATE
       TABLE heartRate(id INTEGER PRIMARY KEY AUTOINCREMENT, dia
       TEXT NOT NULL, fecha TEXT NOT NULL, hora TEXT NOT NULL,
       maxHR INTEGER NOT NULL, minHR INTEGER NOT NULL, AVG
       INTEGER NOT NULL)";
2
3  private static final String COMMENTS_TABLE_CREATE2 = "CREATE
       TABLE contactos(id INTEGER PRIMARY KEY AUTOINCREMENT,
       nombre TEXT NOT NULL, email TEXT NOT NULL)";
4
5  private static final String COMMENTS_TABLE_CREATE3 = "CREATE
       TABLE contrasenaApp(id INTEGER PRIMARY KEY, nombre TEXT
       NOT NULL, email TEXT, contrasena TEXT)";
6
7  @Override
8  public void onCreate(SQLiteDatabase db) {
9    db.execSQL(COMMENTS_TABLE_CREATE1);
10   db.execSQL(COMMENTS_TABLE_CREATE2);
11   db.execSQL(COMMENTS_TABLE_CREATE3);
12 }
```

En la tabla `"heartRate"` se almacenarán el día de la semana, la fecha, la hora, el ritmo cardíaco mínimo, máximo y la media. Esta tabla se utilizará posteriormente para dibujar las gráficas y mostrar los datos por tabla.

La tabla `"contactos"` almacena el nombre y la cuenta de correo electrónico de los contactos que ha añadido el usuario. Estos contactos son a los que se les enviará el correo electrónico en caso de que el usuario alcance el rango de fatiga extrema del algoritmo implementado.

Finalmente, la tabla `"contrasenaApp"` solo va a tener 1 entrada como máximo. El usuario tiene la opción de añadir su cuenta de correo electrónico o no. En caso de que la añada, se almacenará su nombre, su cuenta de correo electrónico y su contraseña de aplicación. Se requiere la contraseña de aplicación en vez de la contraseña normal ya que es la que se utiliza para acceder a una cuenta de Gmail desde aplicaciones o dispositivos que no admiten la autenticación de dos factores con un código de verificación.

3.4. Fórmulas

En esta sección vamos a analizar el algoritmo de detección de fatiga extrema implementado en la aplicación Garmin y la fórmula para calcular el ritmo cardíaco máximo según la edad.

3.4.1. Algoritmo de detección de fatiga extrema

Como ya hemos analizado anteriormente, la aplicación Garmin tiene dos funciones diferentes. Una de ellas es la continua monitorización del ritmo cardíaco para la detección de fatiga extrema y el envío de un correo electrónico en dicho caso.

Al inicio del proyecto, la intención era trabajar con un algoritmo de detección de fatiga extrema que encontramos en el artículo [26]. Este algoritmo utiliza los valores de ritmo cardíaco y el oxígeno en sangre. Aunque el dispositivo Garmin Forerunner 245 permite medir ambos valores, el problema que encontramos es que no puede medir el oxígeno en sangre mientras se realiza una actividad, ya que requiere que el usuario esté quieto. La única manera de trabajar con dicho algoritmo era que el usuario se midiera el oxígeno en sangre justo antes de empezar la actividad, para así tener el valor más próximo posible, pero este valor iba a ser constante durante

todo el monitoreo, por lo que descartamos dicha opción.

Posteriormente, encontramos otro artículo [16] que definía un algoritmo muy similar, pero éste no utilizaba los valores del oxígeno en sangre, sino que utilizaba el número de pulsaciones por minuto del usuario, por lo que es el algoritmo que hemos implementado en la aplicación.

El algoritmo en cuestión es el siguiente:

```
var max_HR;
if (heart_rate < 60 ){
        //RANGO 1
} else if (heart_rate > 60) && ((heart_rate < (70%*max_HR)){
        //RANGO 2
} else if (heart_rate > (70%*max_HR) && (heart_rate < (80%*
   max_HR)){
        //RANGO 3
} else {
        //RANGO 4
}
```

La variable "max_HR" se obtiene con la fórmula del máximo ritmo cardíaco que analizaremos en la siguiente subsección.

En dicho algoritmo, cuando se alcanza el rango 3, el dispositivo Garmin vibra para avisar al usuario. Cuando se alcanza el rango 4 es cuando se envía el correo electrónico a los contactos predefinidos por el usuario.

Debido a que nuestra intención es que nuestras aplicaciones sean lo más personalizables para cada usuario, decidimos que los umbrales que definen el rango 3 y el rango 4 debían ser configurables, por lo que cada usuario puede tener un algoritmo diferente, adaptándose a sus necesidades, utilizando la misma aplicación. En caso de que el usuario no configure los umbrales, los umbrales por defecto serán los siguientes:

```
var max_HR;
if (heart_rate < 60 ){
        //RANGO 1
```

```
} else if (heart_rate > 60) && ((heart_rate < (85%*max_HR)){
        //RANGO 2
} else if (heart_rate > (85%*max_HR) && (heart_rate < (100%*
   max_HR)){
        //RANGO 3
} else {
        //RANGO 4
}
```

Esto significa que cuando se alcance el `85%` del ritmo cardíaco máximo, el dispositivo Garmin vibrará, y si se alcanza el `100%` del ritmo cardíaco máximo se enviará el correo electrónico.

3.4.2. Fórmula del ritmo cardíaco máximo

Anteriormente, en el algoritmo, hemos utilizado la variable `"max_HR"` y hemos comentado que esta representaba el valor del máximo ritmo cardíaco para cada persona. Este valor depende de la persona en concreto, y puede obtenerse mediante una prueba de esfuerzo. Sin embargo, existen fórmulas que proporcionan un valor aproximado en función de la edad.

Lo más común es utilizar la típica fórmula de `max_HR = 220 - edad` para sacar dicho valor. Tras informarnos un poco al respecto, pudimos encontrar este artículo [27], que estudia las fórmulas que más se utilizan para obtener este valor. La conclusión a la que llega en el artículo es que ninguna de las fórmulas es precisa, ya que todas tienen un grado de error bastante importante, pero que en el caso de utilizar alguna de estas, la más precisa es la fórmula de Invar, que es la que se ha utilizado para este proyecto: `max_HR = 205,8 - 0,685*edad`.

3.5. Interfaz gráfico desarrollado

En esta sección vamos a analizar las interfaces gráficas desarrolladas para ambas aplicaciones.

3.5.1. Aplicación Android

Para explicar con claridad la interfaz gráfica de la aplicación Android vamos a dividirlo en dos *activities* (un *activity* es una clase que representa una pantalla en la aplicación), el *activity* de conexión y el *activity* de control.

Activity de conexión

Cuando abrimos nuestra aplicación, nos aparecerá la pantalla que vemos en la figura 3.6.

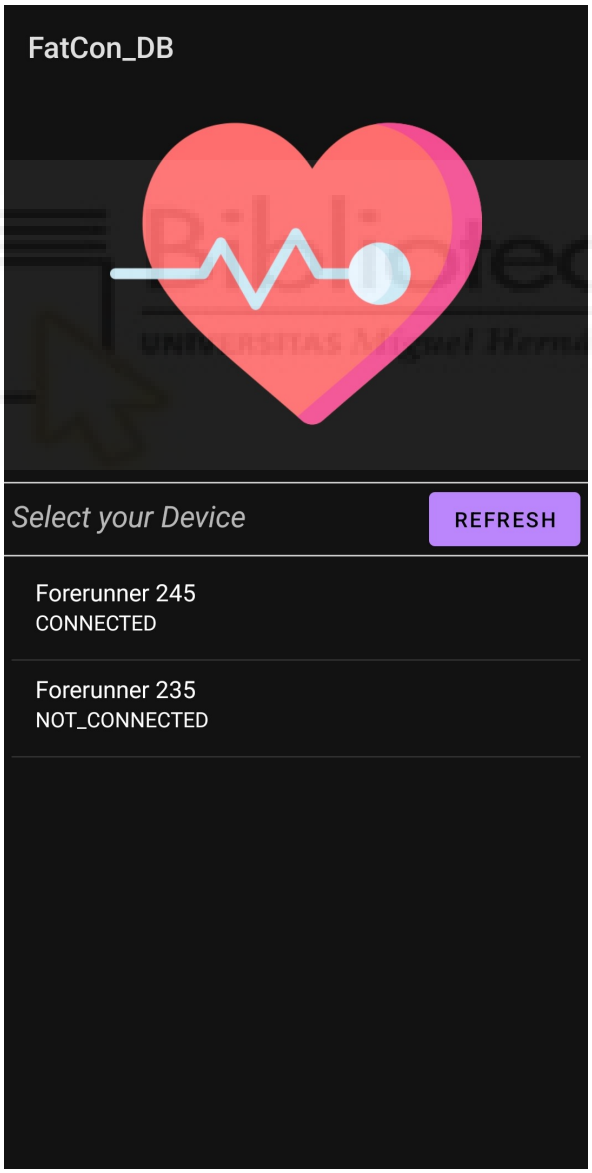

Figura 3.6: Activity de conexión

La función de esta *activity* es que el usuario escoja el dispositivo que va a usar. En esta pantalla tenemos un botón que sirve para refrescar la lista de dispositivos, y la misma lista de dispositivos, que muestra los dispositivos que han sido conectados alguna vez al teléfono móvil.

Cuando seleccionamos un dispositivo, la aplicación recoge la información que necesita y se la envía a la siguiente *activity*.

Activity de control

Una vez que hemos seleccionado el dispositivo Garmin, nos aparecerá la pantalla que vemos en la figura 3.7.

Base de Datos

ID	Dia	Fecha	Hora	Max
1	Lun	2023-09-18	11:00	127
2	Lun	2023-09-18	12:00	128
3	Lun	2023-09-18	13:00	132
4	Lun	2023-09-18	14:00	140
5	Lun	2023-09-18	15:00	145
6	Lun	2023-09-18	16:00	143
7	Lun	2023-09-18	17:00	148
8	Lun	2023-09-18	18:00	149
9	Lun	2023-09-18	19:00	146
10	Lun	2023-09-18	20:00	143
11	Lun	2023-09-18	21:00	137
12	Lun	2023-09-18	22:00	147
13	Lun	2023-09-18	23:00	138
14	Mar	2023-09-19	00:00	142
15	Mar	2023-09-19	01:00	139
16	Mar	2023-09-19	02:00	134
17	Mar	2023-09-19	03:00	141
18	Mar	2023-09-19	04:00	137
19	Mar	2023-09-19	05:00	146
20	Mar	2023-09-19	06:00	150

Base de Datos

Fecha	Hora	Max	Min	AVG
2023-09-18	11:00	127	74	63
2023-09-18	12:00	128	71	66
2023-09-18	13:00	132	75	70
2023-09-18	14:00	140	82	77
2023-09-18	15:00	145	86	79
2023-09-18	16:00	143	80	76
2023-09-18	17:00	148	84	81
2023-09-18	18:00	149	88	83
2023-09-18	19:00	146	89	86
2023-09-18	20:00	143	77	71
2023-09-18	21:00	137	73	60
2023-09-18	22:00	147	84	70
2023-09-18	23:00	138	72	58
2023-09-19	00:00	142	79	66
2023-09-19	01:00	139	80	68
2023-09-19	02:00	134	73	57
2023-09-19	03:00	141	84	69
2023-09-19	04:00	137	71	60
2023-09-19	05:00	146	78	72
2023-09-19	06:00	150	85	77

Figura 3.7: Tabla de la base de datos

En esta pantalla, como podemos observar, se muestran todas las entradas de la base de datos en forma de tabla.

Si pulsamos las tres líneas que aparecen arriba a la izquierda, se nos abrirá el menú lateral de la figura 3.8.

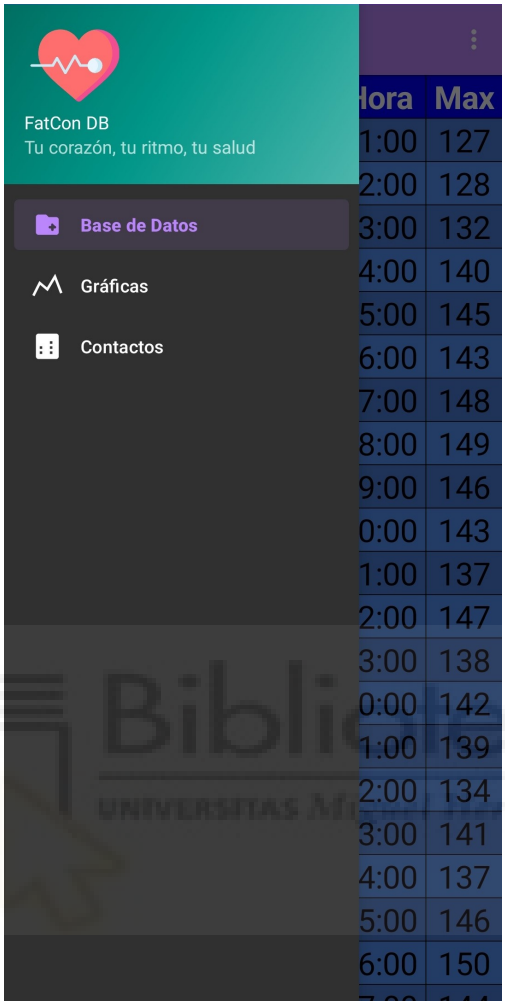

Figura 3.8: Menú lateral

Una vez en el menú, podemos elegir las siguientes opciones:

- Pantalla de tabla: muestra las entradas de la base de datos representadas en una tabla.
- Pantalla de gráficas: muestra las entradas de la base de datos representadas en gráficas.
- Pantalla de contactos: permite añadir y eliminar los contactos a los que se le enviará el correo electrónico en caso de fatiga extrema.

En la pantalla de gráficas podremos ver las entradas de la base de datos separadas por días, semanas y meses, pudiendo ver las entradas de los mínimos, de los máximos y de las medias.

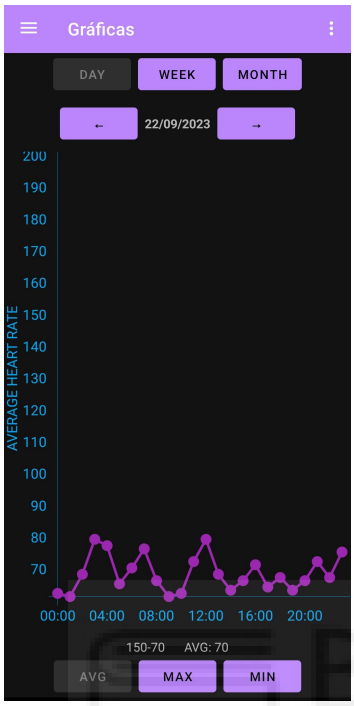

Figura 3.9: Gráfica día de las medias

Figura 3.10: Gráfica día de los máximos

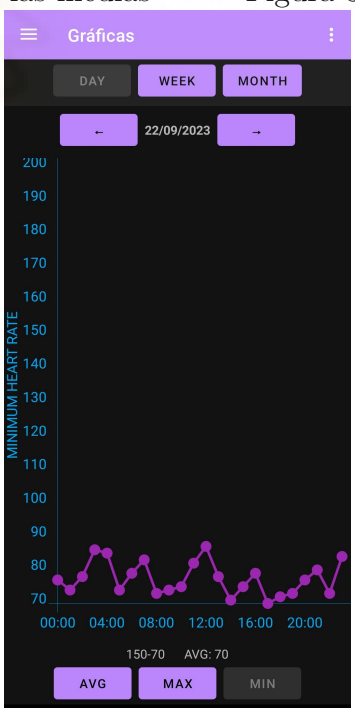

Figura 3.11: Gráfica día de los mínimos

En esta pantalla tenemos distintas series de botones. Los botones de abajo sirven para seleccionar el dato que queremos representar en las gráficas, los botones con flechas sirven para cambiar el día, la semana o el mes, depende de que estemos representando. Finalmente, los botones de arriba sirven para cambiar entre gráfica diaria, gráfica semanal y gráfica mensual.

A continuación, podemos observar una gráfica semanal y otra mensual en las figuras 3.13 y 3.12 respectivamente.

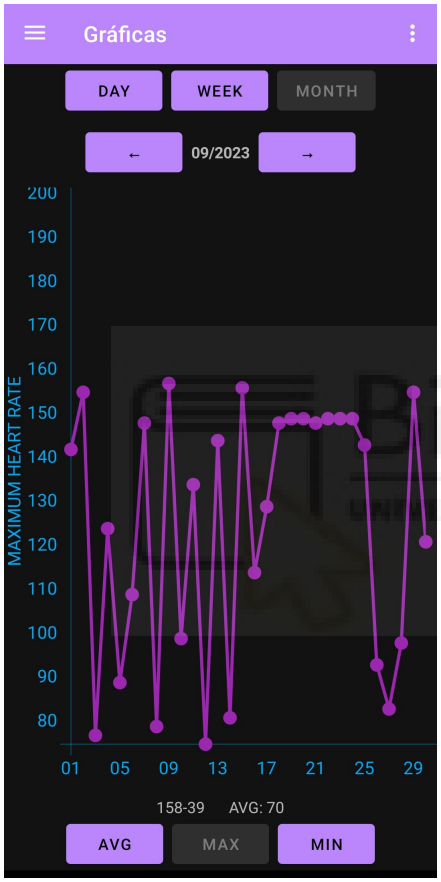

Figura 3.12: Gráfica mensual

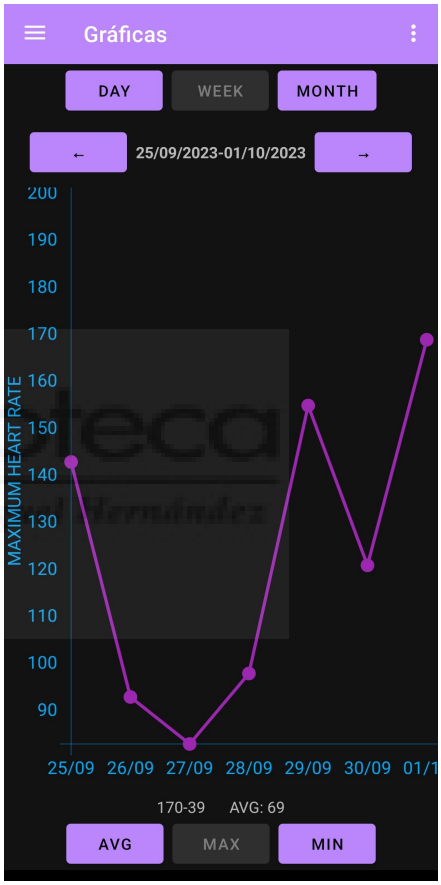

Figura 3.13: Gráfica semanal

Cabe destacar, que encima de los botones inferiores, aparece una línea que muestra el máximo, el mínimo y la media de todos los valores que aparecen en la gráfica representada.

La última pantalla que falta por analizar es la de contactos que se muestra en la figura 3.14.

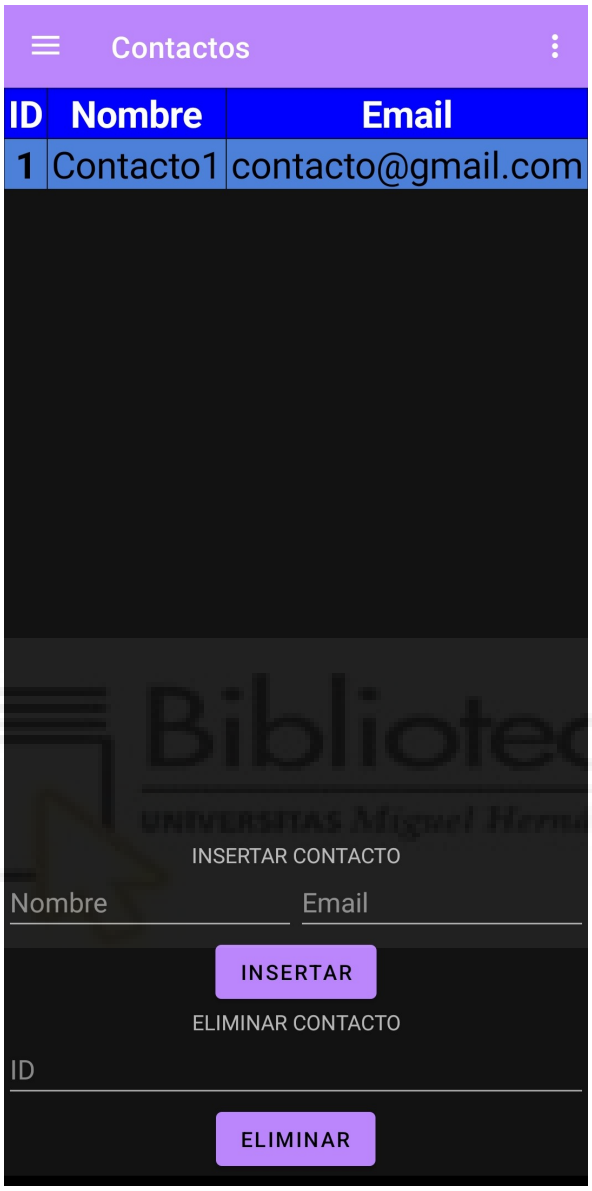

Figura 3.14: Pantalla de contactos

En esta pantalla podemos añadir los contactos a los que queremos que se les avise en caso de superar el umbral mayor. Nos aparecerá una tabla con todos los contactos que hemos añadido, y abajo tendremos la opción de añadir nuevos contactos, e incluso de eliminarlos.

Antes de pasar a la siguiente sección, falta por comentar los 3 puntos que podemos encontrar arriba a la derecha. Si pulsamos este botón nos aparecerán las siguientes opciones:

- Cambiar correo origen
- Cambiar parámetros algoritmo

Estas pantallas las veremos más adelante en la sección 3.6 de manual de usuario.

3.5.2. Aplicación Garmin

Como bien venimos comentando a lo largo del proyecto, la aplicación Garmin tiene dos funciones: monitorización y almacenamiento de valores de ritmo cardíaco y detección de fatiga extrema. Para analizar la interfaz gráfica de dicha aplicación, vamos a separarla en dichas funcionalidades.

Monitorización del Ritmo Cardíaco

Para la monitorización y almacenamiento de los valores del ritmo cardíaco, la aplicación se divide en varias pantallas:

- Gráfica de los 100 últimos valores actuales del ritmo cardíaco
- Gráfica de los máximos almacenados en la aplicación
- Gráfica de los mínimos almacenados en la aplicación
- 8 pantallas que muestran 4 variables del historial cada una

Vamos a comenzar por las pantallas que contienen gráficas. Podemos observar una de ellas en la figura 3.15.

Figura 3.15: Gráfica de mínimos

Lo primero que hay que destacar es la parte de arriba de la pantalla, que es similar para todas. El valor HR que se muestra a la izquierda va variando continuamente, y representa el valor actual del ritmo cardíaco del usuario. El valor RHR a la derecha representa el mínimo valor del ritmo cardíaco desde la última vez que se almacenaron datos. Una vez que los datos se almacenan, este valor del mínimo se reinicia.

Una vez explicada la parte de arriba, podemos observar la gráfica de todos los mínimos almacenados en la aplicación. Cada variable del historial almacenado tiene un valor de ritmo cardíaco mínimo, máximo y la media, por lo que esta gráfica tendrá tantos datos como variables almacenadas en el historial. La gráfica de máximos es igual pero con distintos datos, y la gráfica que va mostrando los últimos valores obtenidos por el sensor es un poco diferente, ya que varía continuamente.

En la figura 3.16 se puede observar una de las pantallas que muestran los datos almacenados en el historial.

Figura 3.16: Historial de valores

Como hemos comentado anteriormente, hay 8 pantallas como esta, cada una de ellas muestra 4 variables del historial hasta llegar a 32, que es el espacio que tiene esta lista. Con estos datos se dibujan las gráficas de máximos y mínimos.

Detección de Fatiga Extrema

Para la detección de fatiga extrema tenemos una única pantalla que irá variando en función del rango del algoritmo en el que nos encontremos.

Figura 3.17: Rango del Algoritmo 1

Figura 3.18: Rango del Algoritmo 2

Figura 3.19: Rango del Algoritmo 3

Figura 3.20: Rango del Algoritmo 4

En las figuras 3.17 3.18 3.19 y 3.20 se puede observar esta pantalla, que como bien comentamos anteriormente, la parte superior es idéntica, mientras que la parte inferior representa con colores el rango en el que nos encontramos.

Si alcanzamos el rango del algoritmo 3 el dispositivo Garmin vibraría, y si alcanzamos el rango 4, además de vibrar, emitiría un sonido y enviaría un correo electrónico a los contactos predefinidos.

3.6. Manual de usuario

En esta sección se van a tratar un par de aspectos para el completo entendimiento de las aplicaciones.

3.6.1. Contraseña de aplicación

Cuando iniciamos la aplicación de Android por primera vez nos aparecerá la pantalla que se observa en la figura 3.21.

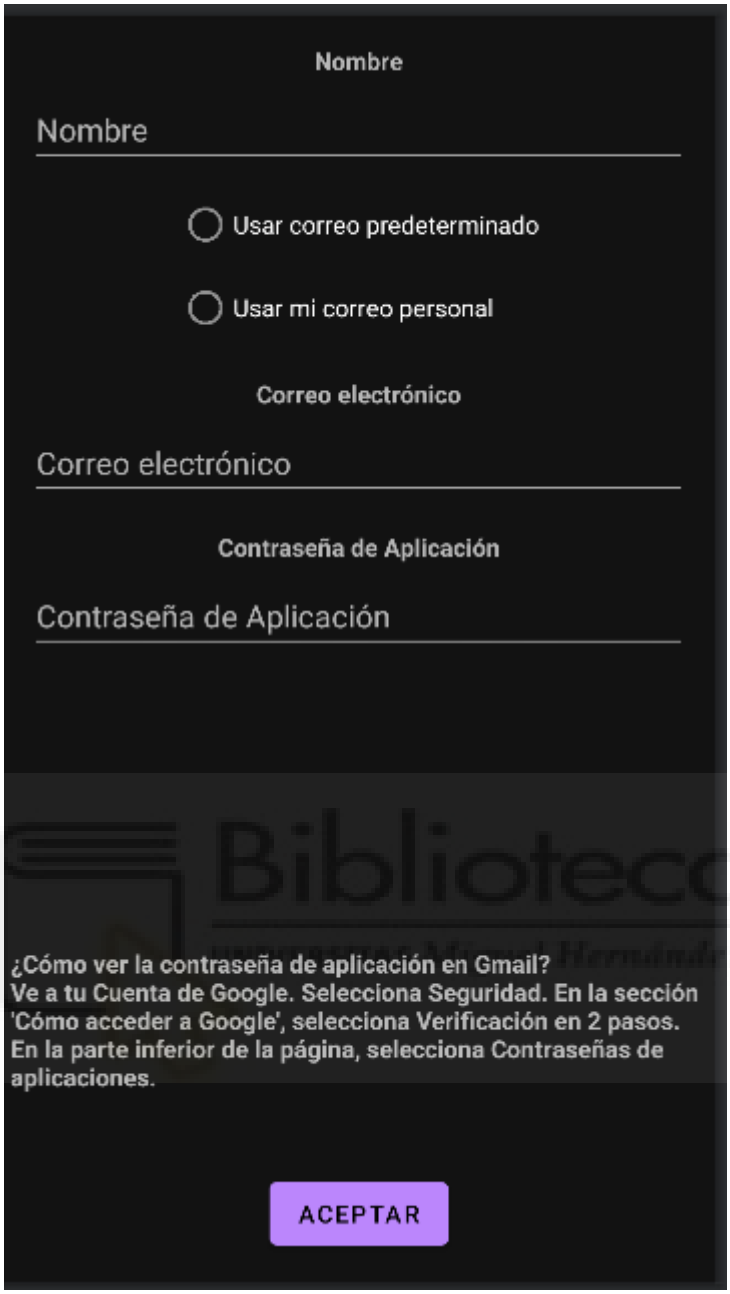

Figura 3.21: Pantalla contraseña de aplicación

En esta pantalla nos solicita los siguientes datos:

- Nombre del usuario
- Elegir entre utilizar correo predeterminado o utilizar correo personal
- Correo electrónico
- Contraseña de aplicación

Para que quede claro, la opción de utilizar un correo predeterminado o el correo personal es para el envío de correos electrónicos. Si seleccionamos la opción de usar correo predeterminado, cuando estemos en el modo de detección de fatiga extrema y se supere el umbral mayor, se enviará un correo electrónico a cada uno de los contactos predefinidos por el usuario desde la cuenta de correo de la aplicación. En caso contrario, se enviará desde la cuenta de correo del usuario.

Si seleccionamos la opción de usar correo predeterminado, las casillas para introducir nuestro correo electrónico y la contraseña se desactivarán.

Si seleccionamos la opción de usar mi correo personal, tendremos que introducir nuestra cuenta de correo electrónico y nuestra contraseña de aplicación.

Una contraseña de aplicación [13] es una contraseña de 16 dígitos que concede permiso a una aplicación o a un dispositivo menos seguros para acceder a tu cuenta de Google. Estas contraseñas solo se pueden utilizar con cuentas que tengan activada la verificación en dos pasos.

Para obtener dicha contraseña de aplicación habrá que seguir los siguientes pasos:

1. Ir a `"Gestionar tu cuenta de Google"`
2. Acceder a la sección `"Seguridad"`
3. En el apartado de `"Cómo inicias sesión en Google"` seleccionamos la opción `"Verificación en dos pasos"`
4. Si la verificación en dos pasos está desactivada tendremos que activarla, con los pasos que esto conlleva
5. En la parte inferior nos aparecerá `"Contraseñas de aplicaciones"`, seleccionamos dicha opción y generamos una nueva contraseña de aplicación
6. Por último, introducimos esta contraseña en la aplicación

3.6.2. Configuración de umbrales

Como hemos comentado anteriormente, para que el algoritmo sea lo más personalizable posible, hemos añadido la posibilidad de cambiar los umbrales para que

se adapte a las necesidades del usuario. Cuando queramos cambiar los umbrales accederemos a la pantalla que muestra la figura 3.22.

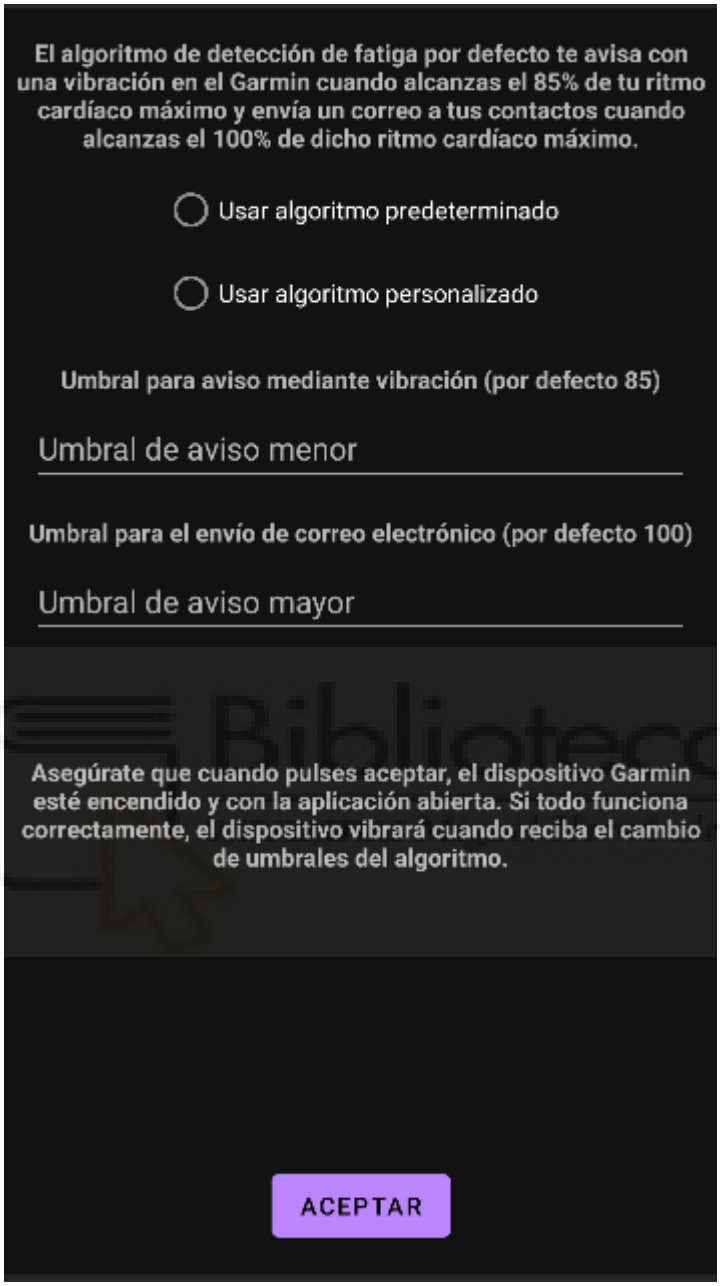

Figura 3.22: Pantalla configuración de umbrales

En esta pantalla podremos elegir entre usar el algoritmo por defecto o usar algoritmo personalizado.

En el caso de que usemos el algoritmo por defecto, se usarán los umbrales que ya se han explicado anteriormente y que se explican en la misma pantalla.

En el otro caso, tendremos que introducir manualmente los valores de los umbrales

que queremos que limiten los rangos del algoritmo. Estos valores no son el valor del ritmo cardíaco, si no el porcentaje del ritmo cardíaco máximo. Si introducimos un 70, estamos diciendo que ese umbral será el `"70%"` de nuestro ritmo cardíaco máximo.

Capítulo 4

Conclusiones

A la vista de los resultados presentados en el capítulo 3, se ha podido comprobar que se han conseguido los objetivos del proyecto, que eran desarrollar una aplicación para dispositivos Android y otra para dispositivos Garmin, que en conjunto, nos sirvan para la monitorización y almacenamiento de valores del ritmo cardíaco, y la detección de fatiga extrema durante el ejercicio físico. Además de ser configurables por el usuario, permitiendo configurar los límites de fatiga adaptados a las necesidades personales.

El desarrollo de este proyecto, me ha permitido seguir desarrollándome a nivel personal, ampliando mis conocimientos tanto en el lenguaje Java como en el mundo de las aplicaciones Android, también me ha permitido descubrir el lenguaje Monkey C, que nunca había trabajado con él.

Como hemos comentado, los objetivos de desarrollo se han cumplido en su totalidad, pero este proyecto tiene un objetivo futuro que aún está por cumplirse, y es que sirva de ayuda para todas esas personas que necesiten un monitoreo más estricto del ritmo cardíaco, o simplemente para esas personas que quieren tener un registro general de todos los valores de ritmo cardíaco.

4.1. Trabajos futuros

A pesar de haber cumplido con los objetivos generales, al principio de este proyecto, la idea era trabajar con un algoritmo que también utilizaba los valores de oxígeno en sangre. Como bien comentamos, esta opción era inviable ya que con el dispositivo Garmin utilizado, no nos permitía medir dicho valor mientras se estaba realizando una actividad física, teniendo que buscar otro algoritmo y adaptarlo a nuestras posibilidades.

De cara a versiones futuras se podrían estudiar distintas posibilidades para hacer este objetivo posible, experimentando con otros dispositivos como los Samsung Wear y su sistema operativo Wear OS, y conseguir poner en marcha el algoritmo que en un principio iba a ser el implementado.

También podrá añadirse una función de que recoja los instantes en los que el usuario sobrepasa los umbrales, para así almacenar estos datos junto a los del ritmo cardíaco y oxígeno en sangre.

Por último, podrá desarrollarse una aplicación que dé soporte a smartphones de tipo iOS de Apple, así como otra aplicación web, para poder consultar los datos desde el ordenador.

Bibliografía

[1] Garmin Connect IQ. [Online]. Available: https://apps.garmin.com/es-ES/.

[2] Garmin Connect IQ Developer site. [Online]. Available: https://developer.garmin.com.

[3] Garmin Connect IQ Mobile SDK. [Online]. Available: https://developer.garmin.com/connect-iq/core-topics/mobile-sdk-for-android/.

[4] Garmin Connect IQ SDK. [Online]. Available: https://developer.garmin.com/connect-iq/overview/.

[5] Garmin Developer Virtual Conference (GDVC. [Online]. Available: https://developer.garmin.com/gdvc/overview/.

[6] Garmin Flexible and Interoperable Data Transfer (FIT) SDK. [Online]. Available: https://developer.garmin.com/fit/overview/.

[7] Garmin Monkey C. [Online]. Available: https://developer.garmin.com/connect-iq/monkey-c/.

[8] Repositorio Garmin GitHub. [Online]. Available: https://github.com/garmin/connectiq-apps.

[9] Android. Android studio. [Online]. Available: https://developer.android.com/studio.

[10] J. Choi and R. Gutierrez-Osuna. Using heart rate monitors to detect mental stress. In *2009 Sixth International Workshop on Wearable and Implantable Body Sensor Networks*, pages 219–223, 2009.

[11] S. Das, R. Richard, M. N, S. Charan, and C. Ravindra. Wearable smart heart monitor using iot. In *2022 IEEE International Conference on Distributed Computing and Electrical Circuits and Electronics (ICDCECE)*, pages 1–6, 2022.

[12] Z. Ge, P. W. C. Prasad, N. Costadopoulos, A. Alsadoon, A. K. Singh, and A. Elchouemi. Evaluating the accuracy of wearable heart rate monitors. In *2016 2nd International Conference on Advances in Computing, Communication, & Automation (ICACCA) (Fall)*, pages 1–6, 2016.

[13] Google. Application password. [Online]. Available: https://support.google.com/mail/answer/185833?hl=es: :text=Una

[14] Y. Higuchi, N. Saijo, T. Ishihara, T. Usui, T. Murakami, M. Miyata, K. Ono, S. Usui, and H. Togo. New wearable heart rate monitor for contact sports and its potential to change training load management. In *2019 41st Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*, pages 4121–4124, 2019.

[15] D. F. Hsu and D. H.-P. Huang. Heart rate accuracy from wearable wristbands. 2022.

[16] D. Ichwana, R. Z. Ikhlas, and S. Ekariani. Heart rate monitoring system during physical exercise for fatigue warning using non-invasive wearable sensor. In *2018 International Conference on Information Technology Systems and Innovation (ICITSI)*, pages 497–502, 2018.

[17] Java. Java. [Online]. Available: https://www.java.com/es/.

[18] B. Jepson. *Wearable Programming for the Active Lifestyle*. O'Reilly Media, Inc., 2017.

[19] Kotlin. Kotlin. [Online]. Available: https://kotlinlang.org/.

[20] S. Lee, S. Nam, and H. Shin. The analysis of sleep stages with motion and heart rate signals from a handheld wearable device. In *2016 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 1135–1137, 2016.

[21] Microsoft. Microsoft .NET Core SDK. [Online]. Available: https://dotnet.microsoft.com/en-us/download/dotnet.

[22] Microsoft. Visual Studio Code. [Online]. Available: https://code.visualstudio.com.

[23] OmniSharp. Extensión Monkey C para Visual Studio Code. [Online]. Available: https://marketplace.visualstudio.com/items?itemName=garmin.monkey-c.

[24] M. Onuki, M. Sato, and J. Sese. Estimating physical/mental health condition using heart rate data from a wearable device. In *2022 44th Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*, pages 4465–4468, 2022.

[25] J. Parak and I. Korhonen. Evaluation of wearable consumer heart rate monitors based on photopletysmography. In *2014 36th Annual International Conference of the IEEE Engineering in Medicine & Biology Society*, pages 3670–3673, 2014.

[26] D. I. Putra, D. Suarmin, and S. Ekariani. Fatigue warning system during physical exercise based on heart rate and oxygen saturation using non-invasive wearable sensor. In *2022 International Symposium on Information Technology and Digital Innovation (ISITDI)*, pages 22–26, 2022.

[27] R. A. Robergs and R. Landwehr. The Surprising History of HRmax=220-age Equation. *Journal of exercise physiology (JEPonline)*, 5(2):1–10, 2002.

[28] P. Sasidharan, T. Rajalakshmi, and U. Snekhalatha. Wearable cardiorespiratory monitoring device for heart attack prediction. In *2019 International Conference on Communication and Signal Processing (ICCSP)*, pages 0054–0057, 2019.

[29] A.-K. Schalkamp, K. J. Peall, N. A. Harrison, and C. Sandor. Wearable movement-tracking data identify parkinson's disease years before clinical diagnosis. *Nature Medicine*, Jul 2023.

[30] SQlite. Sqlite. [Online]. Available: https://www.sqlite.org/index.html.

[31] St0ve, M. P. and Haucke, E. and Nymann, M. L. and Sigurdsson, T. and Larsen, B. T. Accuracy of the wearable activity tracker Garmin Forerunner 235 for the

assessment of heart rate during rest and activity. *Journal of Sports Sciences*, 37(8):895–901, Apr 2019.

Anexos

Anexo A

Acrónimos

Siglas	Significado
API	Application Programming Interface
GPS	Global Positioning System
FIT	Flexible and Interoperable Data Transfer
GUI	Graphical User Interface
IDE	Integrated Development Environment
SDK	Standard Development Kit