UNIVERSITAT POLITÈCNICA DE VALÈNCIA

CAMPUS D'ALCOI

Desarrollo de una APP para la mejora de la técnica de un ciclista mediante la monitorización de su postura a través de sensores inerciales.

MEMORIA PRESENTADA POR:

Oliver Nicholas Craven

GRADO DE INGENIERÍA INFORMÁTICA

TUTOR:

Rubén Pérez Llorens

Convocatoria de defensa: *Septiembre de 2017*

Contenido

1. Tipografía Utilizada

Para la siguiente memoria se utilizarán los siguientes tipos de letra:

- Calibri Light (Títulos)
- Calibri (Cuerpo)

La convención tipográfica que se utilizará para estas fuentes en la presente memoria viene determinada por la siguiente lista donde se indica que formato se seguirá y en que momento:

- Texto Normal: Calibri (Cuerpo) 11
- Apartado Principal: Calibri Light (Títulos) 16 Negrita
- Sub apartado: Calibri Light (Títulos) 13 Negrita
- Título de ilustración: *Calibri (Cuerpo) 9 Cursiva Azul Grisáceo*
- Código Fuente: *Calibri (Cuerpo) 11 Cursiva*
- Nombre de Archivo: *Calibri (Cuerpo) 11 Cursiva*

2. Introducción

Este proyecto trata del desarrollo de una APP[1] para la mejora de la técnica de un deportista mediante la monitorización de su postura a través de sensores inerciales durante la práctica de actividades deportivas con el fin de aportar un feedback para mejorar el rendimiento.

Es un proyecto de fondos públicos ofertado por AITEX[2], una asociación privada sin ánimo de lucro, desarrollado como parte de su proyecto global SPORT@FUTURE que pretende investigar soluciones deportivas innovadoras que garanticen la protección, aseguren el confort, maximicen el rendimiento y prevengan las lesiones de los deportistas. Para ello se utilizan las nuevas tecnologías, procesos y materiales para crear soluciones viables desde el punto de vista técnico, industrial y comercial.

2.1 La Aplicación

El principal objetivo de la APP es conectar con una solución textil, en forma de camiseta con sensores inerciales, y procesar los datos generados por ésta, y proporcionar un feedback de información al deportista a través de la pantalla del dispositivo móvil, de forma que éste pueda monitorizar y corregir su postura, consiguiendo así maximizar su rendimiento y prevenir cualquier tipo de daño o lesión causado por practicar deporte con una postura incorrecta.

La APP también incluirá otras funciones, siendo estas:

- Pantalla de configuración de alertas.
- Pantalla de soporte.

Estas funciones adicionales proporcionan al usuario una experiencia completa al usar la APP.

Se ofrece la opción de configurar alertas según el nivel de monitorización requerido, adaptando así la APP a las necesidades del usuario. Con esto se pretende aumentar al máximo el número de posibles usuarios, ya que, configurando la APP, el usuario puede tener un control adecuado y adaptado al deporte que esté desarrollando, por ejemplo, si practica ciclismo sobre un suelo plano podrá configurar una monitorización más estricta de su postura mientras que, si lo realiza sobre un suelo no uniforme como puede ser una montaña podrá configurar la APP para que solo se le alerte cuando está en una postura extremadamente incorrecta.

[1] Aplicación.
[2] Asociación de Investigación de la Industria Textil.

2.2 La Camiseta

La solución diseñada para corregir la postura incorrecta de los deportistas durante la práctica de deporte es una camiseta deportiva con 7 sensores inerciales integrados, localizados de la siguiente manera:

- 5 sensores situados encima de la columna vertebral.
- 2 sensores situados en el lateral izquierdo y derecho de la espalda.

Ilustración 1 Camiseta deportiva con sensores y caja de control.

Cada sensor contiene un acelerómetro de 3 ejes y un giroscopio de 3 ejes.

Situado en un lateral de la camiseta se encuentra el dispositivo electrónico de control escondido en un bolsillo. Este dispositivo se comunica de forma inalámbrica, utilizando el protocolo bLE[3], con la APP.

3. Plataforma de Desarrollo

3.1 ¿Qué es Android?

Android es un sistema operativo basado en el núcleo Linux. Fue diseñado principalmente para dispositivos móviles con pantalla táctil, como teléfonos inteligentes y tablets, pero también se ha aumentado su uso con la incorporación de soporte para relojes inteligentes, televisores y automóviles. Incluye de forma nativa una GUI[4], un navegador web y la posibilidad de instalar más aplicaciones. Inicialmente fue desarrollado por Android Inc, y en 2005 fue comprado por Google. Actualmente es desarrollado por Google y la OHA[5].

Las aplicaciones para Android se programan oficialmente en Java, aunque existe la posibilidad de desarrollar aplicaciones en C y C++ a través del Android NDK[6], en LUA con el Corona SDK y HTML5, JavaScript y CSS con el proyecto Cordova de Apache.

[3] Bluetooth Low Energy – Bluetooth de baja energía.
[4] Graphical User Interface – Una interfaz de usuario gráfico.
[5] Open Handset Alliance – una coalición de empresas del sector de telecomunicaciones.
[6] Native Development Kit.

3.2 ¿Porqué Android?

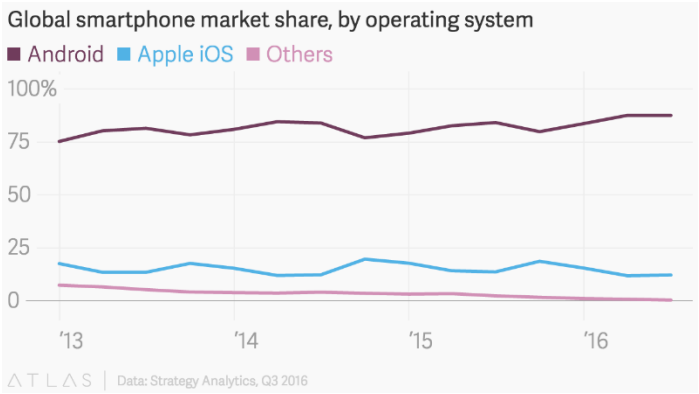

Ilustración 2. Distribución del Mercado Móvil [8]

La principal razón para elegir Android como plataforma de desarrollo de esta APP es su cuota de mercado. Desde su incorporación al mundo de dispositivos móviles en 2005, la plataforma Android ha ido creciendo hasta alcanzar una cuota mayoritaria del mercado. En el Q3[7] de 2016, Android constituía el 88%[8] del mercado global de teléfonos móviles.

Esta APP está diseñada principalmente para funcionar en teléfonos móviles, aunque se puede adaptar a otros dispositivos, como relojes inteligentes, en el futuro. Esta es una decisión de practicidad ya que los deportistas hoy en día suelen llevar encima su dispositivo móvil mientras practican deporte, aprovechando así las funciones varias que estos ofrecen como pueden ser el seguimiento de ruta, velocidad, inclinación, etc. por GPS o la monitorización del ritmo cardíaco. Por lo tanto, al usuario no le resultará molesto usar la camiseta deportiva y la APP correspondiente para tener una monitorización al instante de su postura. Con esto se puede aprovechar el 88% del mercado de dispositivos móviles que incorporan el sistema operativo Android, lo que corresponde a unos 328 millones[8] de dispositivos potenciales para la APP.

Otra razón para elegir Android como plataforma de desarrollo de la APP es su implementación como plataforma de código abierto y gratuito, que resulta fácil de aprender y manejar con una base programadora. No hay ningún coste monetario para desarrollar aplicaciones Android, simplemente desde cualquier ordenador con cualquier sistema operativo se descarga el SDK[9] y se incorpora al IDE[10].

La naturaleza de Android como sistema operativo de código libre y abierto facilita el proceso de desarrollo a la hora de resolver problemas, ya que existe abundante documentación y ejemplos, por parte de Google y otras empresas, organizaciones y personas.

También facilita la publicación y distribución de la APP una vez esté finalizada, ya que no existe ninguna obligación de publicarla en tiendas online de aplicaciones para su distribución al usuario final. Para las aplicaciones desarrolladas para Android existen varios métodos de distribución. A continuación, se detallan los más comunes:

---

[7] Julio, Agosto y Septiembre.
[8] https://qz.com/826672/android-goog-just-hit-a-record-88-market-share-of-all-smartphones/
[9] Software Development Kit - conjunto de herramientas de desarrollo de software.
[10] Integrated Development Environment - entorno de desarrollo integrado. Una aplicación informática que proporciona servicios integrales para facilitarle al programador el desarrollo de software.

- La liberación del código fuente, normalmente a través de herramientas como GitHub, que permite al usuario final descargar, compilar e instalar él mismo la APP. Esta liberación puede ser pública o privada según las necesidades del software.
- La publicación en tiendas online de software abierto como F.Droid, sin necesidad de realizar ningún tipo de pago monetario.
- La publicación en la tienda online PlayStore de Google.
- La distribución del archivo APK[11] ya compilado de forma pública o privada.

La APP se desarrollará en lenguaje Java, el lenguaje de programación soportado oficialmente por Android, lo cual permitirá usar los APIs[12] nativos proporcionados por la plataforma para programar funciones claves de la APP como la comunicación con el dispositivo de control a través de bLE, la navegación entre pantallas y la creación de gráficos y dibujos para mostrar información al usuario.

3.3 Bluetooth Low Energy

Bluetooth low energy es una tecnología Wireless diseñado para dispositivos de bajo coste y baja potencia y ofrece una gran compatibilidad con dispositivos móviles, principalmente enfocados a los mercados de salud, deporte y casa. Presenta muchas ventajas sobre una conexión Bluetooth tradicional como baja latencia, mayor seguridad y poca consumición de batería, pero consigue mantener la misma distancia máxima teórica que una conexión Bluetooth convencional.

Todo dispositivo bLE usa GATT[13]. Una especificación genérica para el envío y recepción de tramos cortos de datos a través de una conexión bLE.

Para la creación de conexiones bLE se necesita:

- Un dispositivo central, por ejemplo, un teléfono móvil, que escanea buscando el anuncio de otros dispositivos bLE.
- Otro dispositivo, por ejemplo, una camiseta inteligente, que anuncia su existencia.

Estos dispositivos implicados en la creación de una conexión bLE se configurarán como servidor y cliente según la necesidad y el protocolo GATT especifica varios comandos para que el cliente pueda descubrir información acerca del dispositivo servidor. Esta información incluye:

- Los servicios que ofrece.
- Las características de los servicios ofrecidos.
- Los descriptores de las características

Ofrece comandos para la escritura y lectura de los valores de las características, y también para la notificación de cambios de dichos valores. Lo que significa que el cliente no tendrá que estar constantemente pidiendo actualizaciones al servidor, sino que el servidor cuando se produce

[11] Android Application Package - paquete del Sistema Operativo Android utilizado para instalar componentes en la plataforma.
[12] Application Program Interface.
[13] Generic Attribute Profile – Perfil Genérico de Atributos.

algún cambio lo notifica al cliente, produciendo así un ahorro importante de batería en los dos dispositivos.

4. Planificación

4.1 Planificación del Proyecto

El desarrollo de esta APP está previsto que tenga una duración de tres meses. Lo cual significa que se finalizará y se entregará a inicios de septiembre de 2017. La distribución del tiempo del desarrollo de la APP seguirá el siguiente diagrama de Gantt, que detalla el tiempo que se empleará en cada fase.

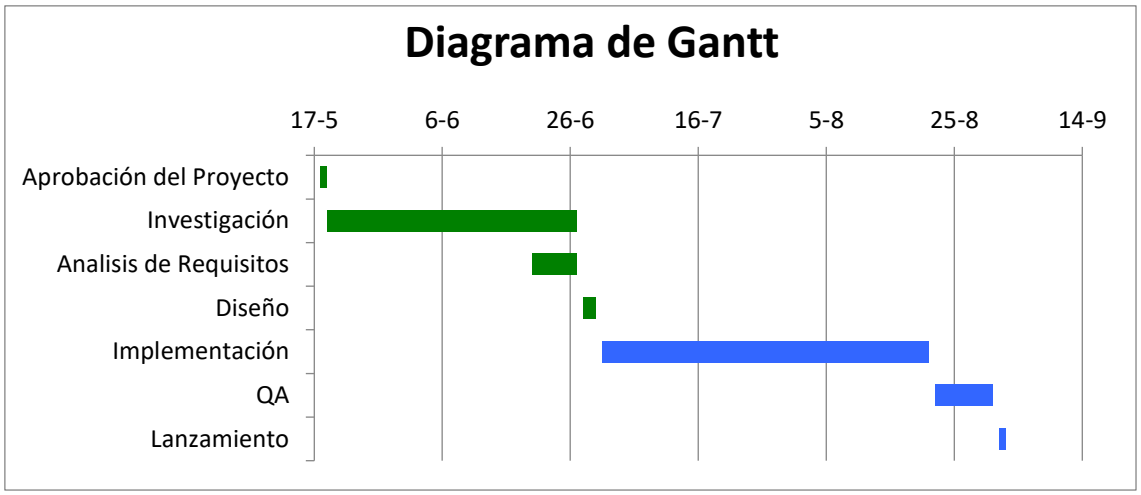

Ilustración 3 Diagrama de Gantt

Se puede observar que el proceso de investigación, análisis de requisitos y diseño de la APP tendrá una duración de poco menos de la mitad del tiempo previsto para el desarrollo del proyecto, lo cual permitirá desarrollar de forma eficaz la APP, ya que antes de comenzar con cualquier tipo de implementación estará definido todo lo que se requiere de la APP.

Para que esto se cumpla, se desarrollará la APP siguiendo el modelo de proceso de software en cascada.

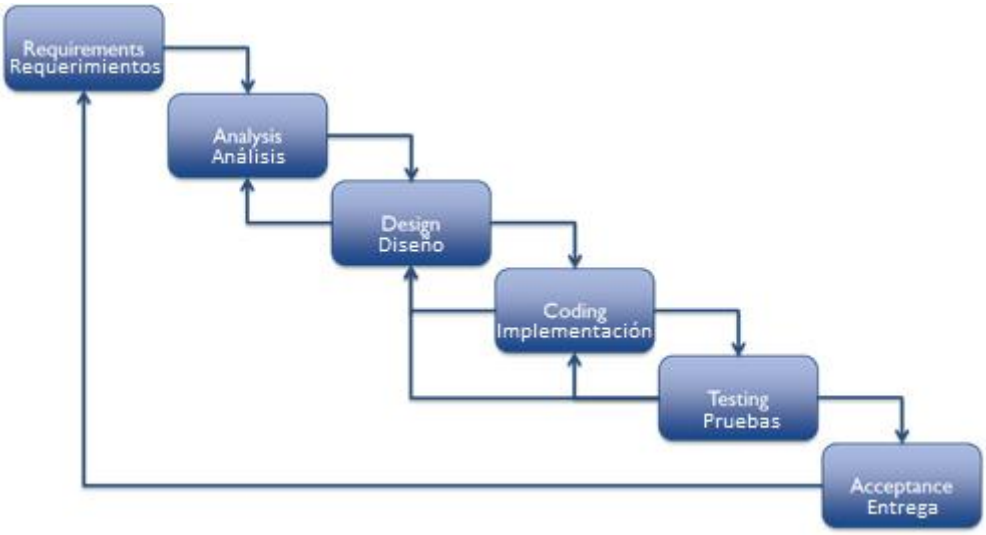

Ilustración 4 Modelo de desarrollo en cascada

Este modelo, también conocido como el modelo lineal secuencial, es un modelo con un enfoque metodológico que ordena las etapas del proceso de desarrollo de software, de forma que una etapa no puede comenzar hasta que no se finalice la etapa anterior.

También incluye la retroalimentación de información de una etapa posterior a una anterior para que en caso de que sea necesario, permitir la modificación de algún factor de la etapa anterior antes de continuar con el proceso de desarrollo.

Este modelo resulta fácil de implementar y entender y promueve un trabajo efectivo, lo que permitirá cumplir con lo planificado.

4.2 Investigación

Durante esta etapa se investigarán los pasos necesarios para desarrollar una APP para el sistema operativo Android que implementa:

- Conexiones bLE.
- Elementos graficos mostrados por pantalla.
- Varias pantallas en una misma aplicación.
- Un diseño adecuado a los guías de estilo de Google.

Para ello se utilizarán varias fuentes de información que se pueden encontrar en la bibliografía.

4.3 Análisis de Requisitos

En la etapa de análisis de requisitos se analizan las necesidades de los usuarios finales de la APP para determinar que objetivos ésta debe cubrir.

La APP que se ha planteado desarrollar proporcionará al usuario una forma sencilla y completa de monitorizar su postura mientras practica deporte, consiguiendo así aumentar y maximizar su

rendimiento y prevenir cualquier tipo de daño o lesión que pudiese ser causado por una postura incorrecta. Para ello se detallan a continuación las necesidades básicas que tiene la APP:

- Interfaz simple e intuitiva: El deportista usará la APP mientras practica deporte, por lo tanto, tendrá que tener una interfaz simple e intuitiva para que, con una simple mirada, pueda saber en un instante si lleva una postura correcta o incorrecta y si es necesario los pasos necesarios para corregirla.
- Fiabilidad: La APP debe, en todo momento, ser capaz de mostrar al usuario la monitorización de su postura, por lo tanto, tendrá que implementar las APIs de bLE de forma adecuada para poder conectarse con el dispositivo de control de la camiseta.

También se detallan los siguientes requisitos básicos que tiene la APP:

- Conexión con el dispositivo de control: La APP debe ser capaz de establecer una conexión bLE con el dispositivo de control de la camiseta deportiva, para poder recibir los datos que genera.
- Interpretación de datos: La APP debe ser capaz de procesar los datos generados por los sensores inerciales y mostrarlos al usuario.
- Escalabilidad: Hoy en día el tamaño de los dispositivos móviles en el mercado es enormemente variado, por lo tanto, la APP debe implementar una interfaz escalable para incluir cualquier tamaño de dispositivo móvil y así aumentar disponibilidad y cuota del mercado.

Implementando todos estos requisitos y necesidades se obtendrá una APP de monitorización de postura con un alto nivel de usabilidad.

4.4 Diagrama Casos de Uso

A continuación, se detalla el diagrama de caso de uso de la APP

Ilustración 5 Diagrama Casos de Uso

Como se puede observar, se trata de una APP muy simple sin funciones extras innecesarias. El usuario abrirá la APP y ésta, de forma automática, se conectará con el dispositivo de control de la camiseta y empezará un seguimiento de su postura.

A partir de ahí, el usuario podrá: modificar el nivel de monitorización que requiere o solicitar ayuda.

5. Diseño

El desarrollo del diseño de la aplicación se realizará basándose en la información recopilada en la fase anterior y en los guías de estilo de Google para Android.

Estas guías se denominan Material Design, una impulsión por parte de Google de implementar en las aplicaciones para Android interfaces sencillas, limpias, escalables y universales que se aproximan a la realidad, usando tipografías claras, colores e imágenes llamativos, luces y sombras reales y animaciones lógicas.

También hay que tener en cuenta los principios básicos del desarrollo de interfaces basadas en la interacción con el usuario, estos son, la usabilidad, la utilidad y la accesibilidad.

Para desarrollar un diseño usable, hay que analizar varios componentes, que son:

- Facilidad de aprendizaje: La APP realizará todas las funciones básicas de forma automática cuando se inicia. Lo que permitirá un proceso corto y fácil de aprendizaje. Para las funciones más avanzadas se seguirán los guías de estilo de Google para que su uso le resulte familiar al usuario.
- Eficiencia: Debido a la conexión automática con el dispositivo de control de la camiseta, el usuario no debe tener ningún tipo de problema en realizar una monitorización de su postura una vez domina el funcionamiento de la APP.
- Eficacia: Se conseguirá un diseño eficaz automatizando todo el proceso de conexión con el dispositivo de control para la monitorización de la postura del usuario. Esto hará que se pueda implementar una interfaz básica para que el usuario no pueda cometer errores.
- Satisfacción: Con el diseño propuesto, se conseguirá un alto nivel de satisfacción en el usuario al usar la APP ya que, debido a que todas las funciones básicas se realizan automáticamente, se podrá implementar una interfaz simple.

La utilidad de la APP depende de su usabilidad. A partir de una interfaz usable, se consigue una utilidad alta y esto permite ofrecer al usuario un mayor beneficio. Automatizando todo el proceso de conexión con el dispositivo de control de la camiseta se permite implementar un diseño de interfaz simple, lo que reduce el tiempo de aprendizaje, aumenta la satisfacción y permite al usuario una mayor explotación de la APP.

La accesibilidad de la APP consiste en ofrecer al mayor número de usuarios posible la posibilidad de acceder a la APP y usarla. Para conseguir esto, varios factores del diseño se ven implicados.

- Colores: Los colores elegidos para el diseño de la APP deben facilitar su uso. Para ello se escogen colores simples que facilitan la lectura.
- Escalabilidad: La APP debe adaptarse al tamaño del dispositivo móvil y permitir su lectura con facilidad. Para ello los elementos gráficos se escalarán a través de Density-independent pixels (dp)[14], una unidad, usada en la programación de aplicaciones para

[14] Píxeles independientes de la densidad.

Android, de medida flexible que se escala de forma uniforma en cualquier pantalla, y para los textos se usarán Scaleable pixels (sp)[15], una unidad de medida con las mismas cualidades de escalabilidad que los dp, pero con la adición de la integración del factor de escala de textos del dispositivo móvil del usuario.

- Bidireccionalidad: Para idiomas que se leen de derecha a izquierda, los elementos de la interfaz deben reordenarse de forma adecuada para su uso. Para ello se usan los mismos valores de relleno al principio que al final de cada elemento.

5.1 Prototipos

Una vez estén definidos los requisitos y necesidades de la APP, se puede proceder a diseñar prototipos iniciales que muestran de forma básica su diseño. Estos prototipos no funcionales de baja fidelidad aportan una visualización de cómo será el diseño de la APP, sin necesidad de entrar en el funcionamiento. A continuación, se pueden observar los primeros diseños de la APP aportados por la empresa AITEX.

Ilustración 6 Diseños Iniciales

En la imagen de la izquierda se puede observar un primer diseño de la pantalla inicial que se muestra al usuario una vez abre la APP. Es un diseño simple que ofrece la posibilidad de empezar la monitorización de su postura o configurar el nivel de monitorización que requiere. Este diseño fue revisado en los siguientes prototipos, sustituyéndose por un diseño más ajustado a las especificaciones de Material Design, facilitando así su uso al usuario.

[15] Píxeles escalables.

A continuación, en la imagen central se puede observar el primer diseño provisional para la pantalla de monitorización de la postura del usuario. La pantalla se compone de una imagen de una bicicleta cuyas ruedas cambian de color, entre rojo, naranja y verde conforme varía el ángulo enviado por los sensores y el nivel de monitorización elegido por el usuario. Al lado se muestra al usuario el ángulo de dichos sensores. El diseño de esta pantalla fue revisado en los siguientes prototipos por no ser demasiado útil a la hora de indicar al usuario su postura.

Finalmente, en la imagen de la derecha se puede observar la pantalla de configuración del nivel de monitorización. Al usuario se le ofrece varias opciones y cada una con un nivel de monitorización e información respecto a los rangos que se emplearán en el cálculo de los ángulos de los sensores para informar sobre su postura. También se revisó esta pantalla, ajustándola a un diseño más simple para el usuario.

A continuación, se pueden observar más prototipos de diseños posibles de la APP.

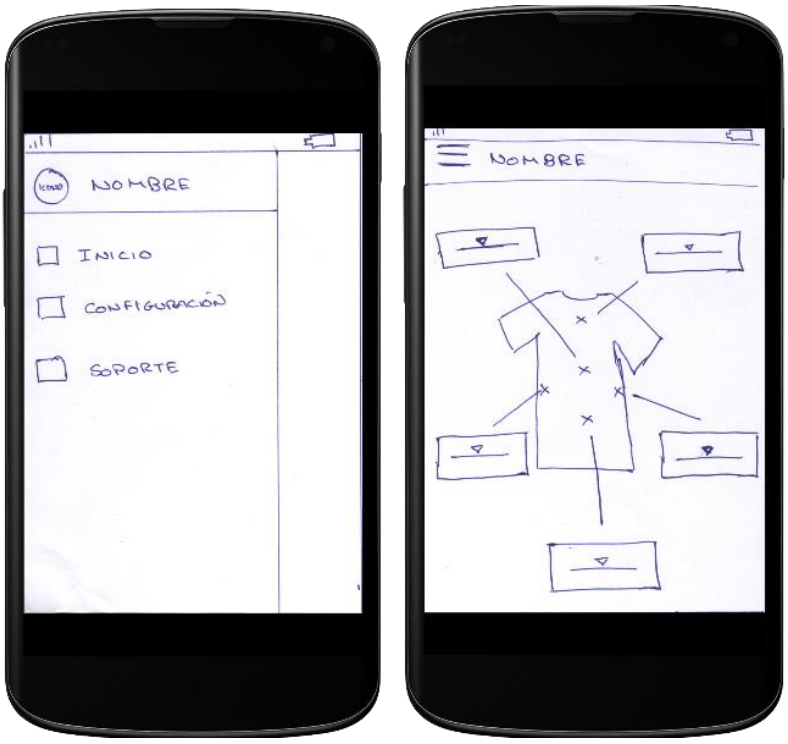

Ilustración 7 Diseños Revisados

En la imagen de la izquierda, se puede observar el diseño propuesto para la navegación entre pantallas. Se propone emplear una barra de navegación estilo Material Design con el logo y el nombre de la APP en la parte superior, seguido por una lista de las varias pantallas de la APP acompañadas por un incono indicativo de su función.

En la imagen de la derecha, se puede observar un diseño posible de la pantalla de monitorización de la APP donde el usuario puede obtener la información resultante de la monitorización de su postura. Se trata de una imagen de una camiseta con referencias en forma de cruz para indicar las posiciones de los sensores, y un balancín indicativo del ángulo de cada sensor. Este diseño se descartó por no ser demasiado útil a la hora de mostrar la información al usuario.

5.2 Colores

Los principios de Material Design especifican el uso de colores llamativos, con sombras oscuras y acentuaciones brillantes. Para incorporar esta idea en el diseño de la APP, se ha establecido como color principal el naranja, de acuerdo con el logo de AITEX.

Partiendo de esta elección, se ha empleado una herramienta proporcionado por Google para generar tonos claros y oscuros contrastantes del naranja principal para mostrar divisiones entre superficies. También se ha empleado para la elección de un color secundario, contrastante con el naranja principal, que se usa para acentuar partes de la interfaz como botones flotantes, enlaces, barras de progreso, etc.

Debido a que se trata de una aplicación para deportistas, se ha elegido usar un fondo blanco para aumentar su usabilidad, ya que un fondo blanco facilita la lectura en situaciones de sol directo en la pantalla.

A continuación, se pueden observar los varios colores a usar en la APP, con sus respectivos valores hexadecimales.

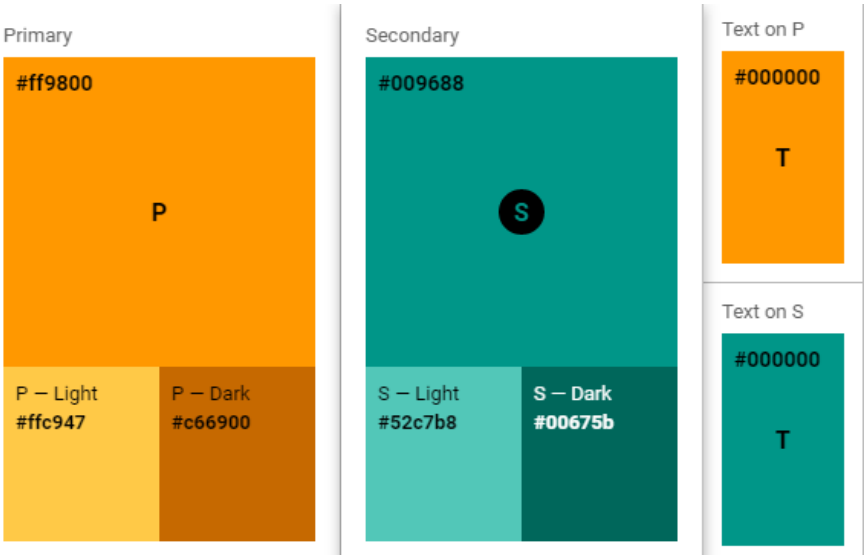

Ilustración 8 Colores de la APP

5.3 Diseño Final

A continuación, se muestra el diseño final de cada pantalla de la APP, incorporando todo lo establecido anteriormente.

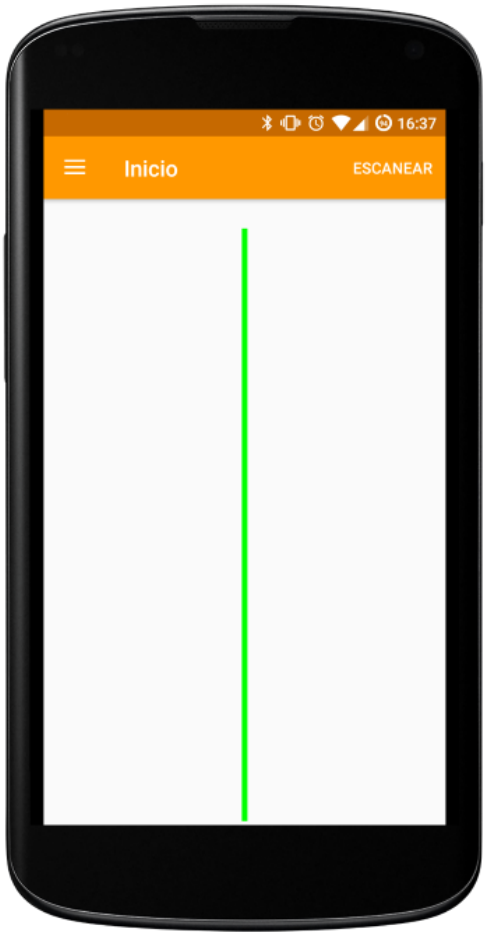

Ilustración 9 Pantalla Inicial

En la ilustración 9 se puede observar el diseño final de la pantalla inicial de la APP. Esta pantalla es la primera a la que se enfrentara el usuario cuando abre la APP.

Como elemento principal hay una representación de los estados de cada sensor de la camiseta, cuyo color variará según el ángulo de dichos sensores.

En la parte superior derecha se puede observar un botón que tiene dos estados: escanear y parar. Este botón sirve para iniciar una búsqueda de bLE o pararlo. Aunque la conexión con el dispositivo de control de la camiseta se realiza de forma automática, se ofrece este botón al usuario como medida de seguridad por si acaso este proceso automático falla.

En la parte superior izquierda se puede observar un icono que indica la presencia de una barra de navegación desplazable como indica Material Design. El uso de este elemento común a las aplicaciones para el sistema operativo Android, aumenta la usabilidad de la aplicación.

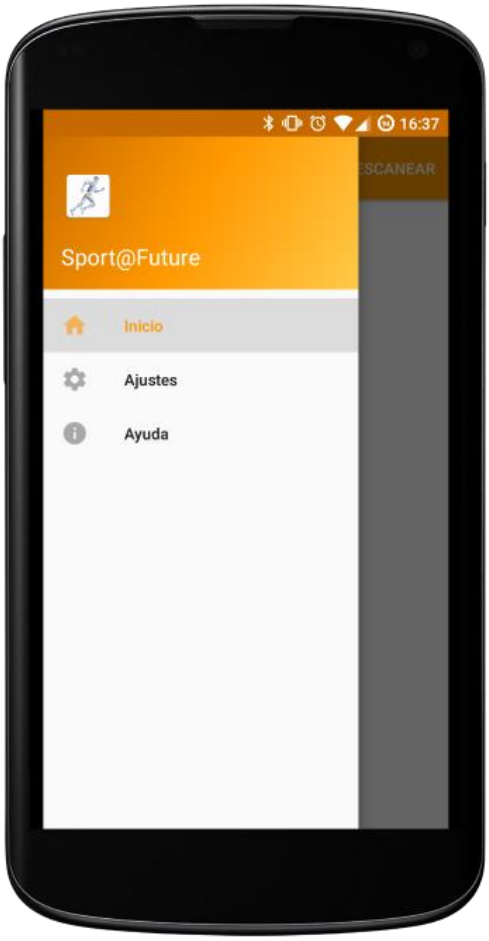

Ilustración 10 Barra de Navegación

En la ilustración 10 se puede observar el diseño final de la barra de navegación desplazable de la APP, a la cual se accede a través del icono situado en la esquina superior izquierda o deslizando de izquierda a derecha en el lado izquierda de la APP. Estas dos formas de acceso son comunes a las aplicaciones para el sistema operativo Android, lo que significa que el usuario no debe tener problemas para encontrar este elemento.

Ofrece acceso a las varias pantallas de la APP.

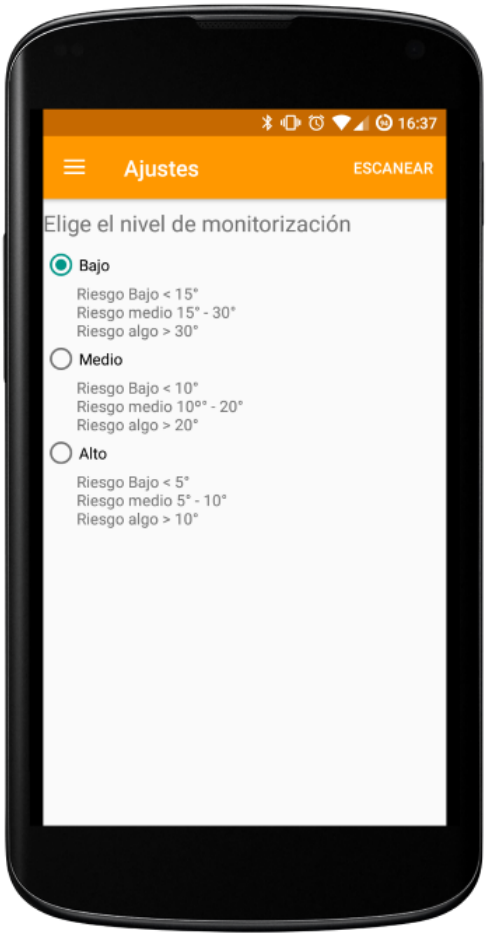

Ilustración 11 Pantalla de Ajustes

En la ilustración 11 se puede observar el diseño final de la pantalla de ajustes de la APP.

Como elemento principal, esta pantalla ofrece al usuario la opción de elegir el nivel de monitorización que requiere. Cada opción aporta una pequeña explicación de los niveles de monitorización que vigila. También, en la parte superior izquierda, se puede observar un icono que indica la presencia de una barra de navegación desplazable.

Ilustración 12 Pantalla de Ayuda

En la ilustración 12 se puede observar el diseño final de la pantalla de soporte de la APP.

Como elemento principal, ofrece la información de contacto de la empresa AITEX a través de varios campos de texto formateados. También, en la parte superior izquierda, se puede observar un icono que indica la presencia de una barra de navegación desplazable.

6. Implementación

6.1 Permisos

Como medida de seguridad, el sistema operativo Android se base en la separación de aplicaciones por niveles de privilegio. Las aplicaciones se ejecutan en un entorno de sandbox[16], un sistema de aislamiento de procesos para ejecutar aplicaciones con seguridad y de manera separada. Este proceso permite controlar de cerca los recursos proporcionados al software.

El sistema operativo Android, de esta manera, controla el acceso a recursos e información que existen fuera del sandbox de cada aplicación y, por lo tanto, cada aplicación debe solicitar permiso para acceder a estos elementos.

Para ello, implementa un sistema de niveles de protección. Este sistema se compone de dos niveles, normal y peligroso.

Los permisos de nivel normal no suponen gran riesgo a la información privada del usuario ni de otras aplicaciones, por lo que el sistema operativo automáticamente concede los permisos de este tipo a las aplicaciones que los soliciten.

Los permisos de nivel peligroso se forman por los que requieren información privada del usuario o de otras aplicaciones, y pueden afectar a la información guardada del usuario o el funcionamiento de otras aplicaciones. El usuario debe aprobar específicamente el uso de este tipo de permiso.

Esta APP, para ser funcional necesita acceder a los elementos Bluetooth del dispositivo. Para ello en el archivo *AndroidManifest.xml*, se solicitan los siguientes permisos:

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

Este permiso permite la conexión con dispositivos ya emparejados. Es un permiso de nivel de protección normal.

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

Este permiso permite encontrar y emparejarse con dispositivos Bluetooth. Es un permiso de nivel de protección normal.

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

Este permiso permite el acceso a la ubicación precisa del dispositivo móvil. Es un permiso de nivel de protección peligroso. Se requiere para generar una lista de dispositivos Bluetooth encontrados.

La APP, para ser funcional, también necesita una versión mínima (4.0) de Bluetooth en el dispositivo móvil. Este requerimiento se puede comprobar durante la ejecución de la APP, o se

[16] Caja de arena.

puede incorporar en el *AndroidManifest.xml* y así denegar la instalación de la APP en dispositivos que no lo cumplen.

```
<uses-feature android:name="android.hardware.bluetooth_le"
              android:required="true" />
```

6.2 Main Activity

Esta clase contiene la actividad principal de la aplicación, que se mostrará al usuario cuando abre la APP y de la cual todos los demás procesos partirán.

Esta actividad es la que se ocupa de interactuar con el usuario y generar la interfaz de la APP. Para ello se implementan los dos métodos fundamentales *onCreate()* y *onPause()*.

6.2.1 onCreate()

El sistema operativo realiza esta llamada cuando se crea la actividad.

Se inicializan los componentes fundamentales de la actividad y se llama a *setContentView()* para definir el diseño de la interfaz de usuario de la actividad.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);
```

Se intenta cargar el primer fragmento, denominado home, y se añade a la pila de transacciones de fragmentos.

```
if(savedInstanceState == null){
    Fragment fragment = null;
    Class fragmentClass;
    fragmentClass = FragmentHome.class;
    try {
        fragment = (Fragment) fragmentClass.newInstance();
    } catch (Exception e) {
        e.printStackTrace();
    }

    FragmentManager fragmentManager =
getSupportFragmentManager();

fragmentManager.beginTransaction().replace(R.id.flContent,
fragment).commit();
}
```

Se guardan los valores del tamaño de la pantalla.

```
DisplayMetrics displayMetrics = new DisplayMetrics();
getWindowManager().getDefaultDisplay().getMetrics(displayMetrics);
GlobalVariables.displayHeight = displayMetrics.heightPixels;
GlobalVariables.displayWidth = displayMetrics.widthPixels;
```

Se genera la barra de navegación entre fragmentos.

```
DrawerLayout drawer = (DrawerLayout)
findViewById(R.id.drawer_layout);
ActionBarDrawerToggle toggle = new ActionBarDrawerToggle(
        this, drawer, toolbar, R.string.navigation_drawer_open,
R.string.navigation_drawer_close);
drawer.setDrawerListener(toggle);
toggle.syncState();

NavigationView navigationView = (NavigationView)
findViewById(R.id.nav_view);
navigationView.setNavigationItemSelectedListener(this);
```

Se comprueba el estado de bLE y los permisos concedidos y se inicializa un adaptador de Bluetooth para poder trabajar con bLE.

```
if
(!getPackageManager().hasSystemFeature(PackageManager.FEATURE_BLUETO
OTH_LE)) {
        Toast.makeText(this, "BLE No Soportado",
                Toast.LENGTH_SHORT).show();
        finish();
    }
    final BluetoothManager bluetoothManager =
            (BluetoothManager)
getSystemService(Context.BLUETOOTH_SERVICE);
    mBluetoothAdapter = bluetoothManager.getAdapter();

if (ContextCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_FINE_LOCATION) !=
PackageManager.PERMISSION_GRANTED) {
    ActivityCompat.requestPermissions(this, new
String[]{Manifest.permission.ACCESS_FINE_LOCATION},
MY_PERMISSIONS_REQUEST_FINE_LOCATION);
}
```

6.2.2 onPause()

El sistema operativo llama a este método como el primer indicador de que el usuario está abandonando la actividad, aunque no siempre significa que la actividad se esté destruyendo, por lo que aquí se desconecta el adaptador Bluetooth para liberar el recurso a otras aplicaciones o el sistema operativo.

```
@Override
protected void onPause() {
    super.onPause();
    if (mBluetoothAdapter != null && mBluetoothAdapter.isEnabled())
{
        scanLeDevice(false);
        try{
            mGatt.disconnect();
        }
        catch (NullPointerException e){
            Log.i("gatt empty",e.toString());
        }
    }
}
```

6.2.3 Interfaz de Usuario

La actividad principal genera la interfaz de usuario para toda la aplicación, que incluye el botón de escanear/parar y la barra de navegación entre fragmentos. Así pues, cuando se carguen los varios fragmentos de la aplicación, estos simplemente tendrán que reemplazar cierta parte de la interfaz para mostrar su información, creando así una interfaz común para toda la aplicación.

Para ello se aprovecha el menú de opciones básico de Material Design para crear un botón de escanear/parar de Bluetooth de la siguiente forma:

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.main, menu);

    if (!mScanning) {
        menu.findItem(R.id.menu_stop).setVisible(false);
        menu.findItem(R.id.menu_scan).setVisible(true);
    } else {
        menu.findItem(R.id.menu_stop).setVisible(true);
        menu.findItem(R.id.menu_scan).setVisible(false);
    }
    return true;
}
```

Según el valor booleano *mScanning* se muestra el elemento *menu_stop* o *menu_scan* y la funcionalidad del botón se implementa de la siguiente forma:

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_scan:
            scanLeDevice(true);
            break;
        case R.id.menu_stop:
            scanLeDevice(false);
            break;
    }
    return true;
}
```

Así, si está activo el elemento *menu_scan* se empezará la búsqueda de dispositivos Bluetooth y si está activo el elemento *menu_stop* se parará cualquier búsqueda activa.

La barra de navegación se implementa de la siguiente forma:

```
@Override
public boolean onNavigationItemSelected(MenuItem item) {
    Fragment fragment = null;
    Class fragmentClass = null;
    FragmentManager fragmentManager;
    DrawerLayout drawer;

    switch(item.getItemId()){
        case R.id.nav_home:
            fragmentClass = FragmentHome.class;
            break;
        case R.id.nav_settings:
            fragmentClass = FragmentSettings.class;
            break;
        case R.id.nav_help:
            fragmentClass = FragmentHelp.class;
            break;
        default:
            fragmentClass = FragmentHome.class;
            break;
    }

try {
        fragment = (Fragment) fragmentClass.newInstance();
    } catch (Exception e) {
        e.printStackTrace();
    }

    fragmentManager = getSupportFragmentManager();
    fragmentManager.beginTransaction().replace(R.id.flContent,
fragment).commit();
    item.setChecked(true);
    setTitle(item.getTitle());
    drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
    drawer.closeDrawer(GravityCompat.START);
    return true;
}
```

Así pues, se carga el *ItemId()* del elemento seleccionado y se genera un instancia de ese fragmento, lo cual se carga a través del *fragmentManager* y se muestra al usuario.

6.2.4 Conexión Bluetooth Low Energy

La conexión bLE se genera a través de métodos y sus respectivas callbacks.

En primer lugar, se inicia una búsqueda de dispositivos low energy.

```
private void scanLeDevice(final boolean enable) {
    if (enable) {
        mHandler.postDelayed(new Runnable() {
            @Override
            public void run() {
                if (Build.VERSION.SDK_INT < 21) {
                    mBluetoothAdapter.stopLeScan(mLeScanCallback);
                    invalidateOptionsMenu();
                } else {
                    mLEScanner.stopScan(mScanCallback);
                    mScanning = false;
                }
            }
        }, SCAN_PERIOD);

        if (Build.VERSION.SDK_INT < 21) {
            mBluetoothAdapter.startLeScan(mLeScanCallback);
        } else {
            mLEScanner.startScan(filters, settings, mScanCallback);
            mScanning = true;
        }
    } else {
        mScanning = false;
        if (Build.VERSION.SDK_INT < 21) {
            mBluetoothAdapter.stopLeScan(mLeScanCallback);
        } else {
            mLEScanner.stopScan(mScanCallback);
        }
    }
    invalidateOptionsMenu();
}
```

Se puede observar que este método recibe un valor booleano, el cual se fija a través del botón escanear/parar explicado anteriormente. Si este valor lo permite y se cumplen los requisitos de nivel mínimo de Bluetooth en el dispositivo, se empezará una búsqueda de dispositivos low energy y se genera una llamada a su método de callback *mScanCallback()*.

mScanCallback() se implementa de la siguiente forma.

```
private ScanCallback mScanCallback = new ScanCallback() {
    @Override
    public void onScanResult(int callbackType, ScanResult result) {
        Log.i("callbackType", String.valueOf(callbackType));
        Log.i("result", result.toString());
        BluetoothDevice bluetoothDevice = result.getDevice();
        if(bluetoothDevice.getAddress().equals(deviceMAC)){
            connectToDevice(bluetoothDevice);
            mScanning = false;
        }
    }
```

Genera un conexión con el dispositivo a través del método *connectToDevice()* si su dirección MAC[17] coincide con la dirección del dispositivo de control de la camiseta.

connectToDevice() se implementa de la siguiente forma.

[17] Media Access Control - Identificador de 48 bits individual y única de la tarjeta red del dispositivo.

```
public void connectToDevice(BluetoothDevice device) {
    if (mGatt == null) {

        mGatt = device.connectGatt(this, true, gattCallback);
        Toast.makeText(this, "Connected to Device",
Toast.LENGTH_SHORT).show();
        mScanning = false;
        scanLeDevice(false);

    }
}
```

Genera una llamada al callback *gattCallback* si no existe ninguna conexión ya abierta.

gattCallback recibe el estado de la conexión bLE, y según el valor recibido realiza una búsqueda de servicios o cierra la conexión.

```
private final BluetoothGattCallback gattCallback = new
BluetoothGattCallback() {
    @Override
    public void onConnectionStateChange(BluetoothGatt gatt, int
status, int newState) {
        Log.i("onConnectionStateChange", "Status: " + status);
        switch (newState) {
            case BluetoothProfile.STATE_CONNECTED:
                Log.i("gattCallback", "STATE_CONNECTED");
                mScanning = false;
                gatt.discoverServices();
                break;
            case BluetoothProfile.STATE_DISCONNECTED:
                Log.e("gattCallback", "STATE_DISCONNECTED");
                gatt.close();
                mGatt = null;
                mScanning = false;
                break;
            default:
                Log.e("gattCallback", "STATE_OTHER");
        }

    }
```

Si se recibe el estado *STATE_CONNECTED,* se genera una llamada al método *discoverServices(),* y si se recibe el estado *STATE_DISCONNECTED*, se cerrará cualquier conexión abierta.

El método *discoverServices()* se implementa de la siguiente forma.

```
@Override
public void onServicesDiscovered(BluetoothGatt gatt, int status) {
    List<BluetoothGattService> services = gatt.getServices();
    Log.i("onServicesDiscovered", services.toString());

    for(BluetoothGattService service : services){
        if(service.getUuid().equals(SERVICE_UUID)){
            characteristic_1 =
service.getCharacteristic(CHARACTERISTIC_UUID_1);
            characteristic_2 =
service.getCharacteristic(CHARACTERISTIC_UUID_2);
            ...
        }
    }

    for (BluetoothGattDescriptor descriptor :
characteristic_1.getDescriptors()) {
        descriptor.setValue(
BluetoothGattDescriptor.ENABLE_NOTIFICATION_VALUE);
        mGatt.writeDescriptor(descriptor);
    }
    gatt.readCharacteristic(characteristic_1);
    gatt.setCharacteristicNotification(characteristic_1,true);
...
}
```

Este método genera una lista de todos los servicios que ofrece el dispositivo bLE, y se compara el UUID[18] de cada uno de ellos con el UUID del servicio que se busca. Una vez se encuentra el servicio buscado, se guardan todas las características de éste, y se solicita la notificación en caso de cambio de cada una de ellas.

Este cambio de valor se procesa a través del callback *onCharacteristicChanged()*.

```
@Override
public void onCharacteristicChanged(BluetoothGatt gatt, final
BluetoothGattCharacteristic characteristic) {
    byte[] data = characteristic.getValue();
    UUID uuid = characteristic.getUuid();

    Utils.convertirArray(data,uuid);
}
```

De este método, se obtiene el UUID de la característica que ha generado el callback y un array de bytes con su valor. Estos valores posteriormente se pasan al método *convertirArray()* de la clase *Utils* para que puedan ser procesados y mostrados al usuario.

6.3 Fragments

Un fragmento representa un comportamiento o una parte de la interfaz de usuario en una actividad. Esta APP combina varios fragmentos dentro de una actividad para crear una interfaz flexible de varias pantallas.

[18] Universally Unique Identifier - Identificador Único Universal.

6.3.1 Fragment Home

Este fragmento muestra los datos procesados, recopilados del dispositivo de control de la camiseta, al usuario.

Para ello, en su método *onCreateView()* que genera su interfaz por primera vez, se genera un nuevo *drawView* donde se mostrará de forma gráfica los datos, y se sustituye el contenido de la actividad *MainActivity* por la vista generada.

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup
container,
                         Bundle savedInstanceState) {
    drawView = new DrawView(getContext());
    drawView.setBackgroundColor(Color.WHITE);
    return inflater.inflate(R.layout.fragment_fragment_home,
container, false);

}
```

6.3.2 Fragment Settings

Este fragmento muestra la pantalla de ajuste al usuario, donde podrá elegir el nivel de monitorización que requiere.

Para ello, en su método *onCreateView()* se genera la vista del fragmento, se actualizan los valores de selección de los niveles de monitorización y se inicializa un monitor de cambios a través del método *setOnCheckedChangeListener()* para que se guarden los nuevos valores cuando el usuario realiza un cambio de selección.

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup
container, Bundle savedInstanceState) {

    final View view = inflater.inflate(R.layout.fragment_settings,
container, false);
    radioGroup = (RadioGroup) view.findViewById(R.id.radioGroup);
    radioButtonLow = (RadioButton)
view.findViewById(R.id.radioButtonLow);
    radioButtonMed = (RadioButton)
view.findViewById(R.id.radioButtonMed);
    radioButtonHigh = (RadioButton)
view.findViewById(R.id.radioButtonHigh);

    switch(GlobalVariables.notificationValue){
        case 0:
            radioButtonLow.setChecked(true);
            break;
        case 1:
            radioButtonMed.setChecked(true);
            break;
        case 2:
            radioButtonHigh.setChecked(true);
            break;
        default:
            radioButtonLow.setChecked(true);
            break;
    }

    radioGroup.setOnCheckedChangeListener(new
RadioGroup.OnCheckedChangeListener() {
        @Override
        public void onCheckedChanged(RadioGroup group, @IdRes int
checkedId) {
            if(checkedId == radioButtonLow.getId()){
                GlobalVariables.notificationValue = 0;
            }
            if(checkedId == radioButtonMed.getId()){
                GlobalVariables.notificationValue = 1;
            }
            if(checkedId == radioButtonHigh.getId()){
                GlobalVariables.notificationValue = 2;
            }
        }
    });

    return view;

}
```

6.3.3 Fragment Help

Este fragmento muestra una pantalla de ayuda, con información de contacto, al usuario.

Para ello, en su método *onCreateView()* genera una interfaz que reemplaza la parte de la actividad correspondiente con su interfaz.

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup
container,
                         Bundle savedInstanceState) {
    // Inflate the layout for this fragment
    View view = inflater.inflate(R.layout.fragment_fragment_help,
container, false);
    view.setBackgroundColor(Color.WHITE);
    return view;
}
```

Esta interfaz se define en un archivo *xml* de la siguiente manera.

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.upv.olcra.sportfuture.FragmentHelp">

    <ImageView
        ... />

    <TextView
        .../>

    <TextView
        ... />
    ...
</RelativeLayout>
```

6.4 Utils

Esta clase Java se compone de varios métodos para procesar los datos recibidos por el dispositivo de control.

6.4.1 convertirArray()

Este método recibe el array de bytes generado por el callback *onCharacteristicChanged()* y el UUID de la característica que lo ha generado. Este array contiene los datos generados por el sensor inercial de la siguiente manera:

[Acx_low] [Acx_high] [Acy_low] [Acy_high] [Acz_low] [Acz_high] [Gyx_low] [Gyx_high] [Gyy_low] [Gyy_high] [Gyz_low] [Gyxz_high]

Siendo:

- Acx_low: byte bajo eje x acelerómetro.
- Acx_high: byte alto eje x acelerómetro.
- Acy_low: byte bajo eje y acelerómetro.
- Acy_high: byte alto eje y acelerómetro.

- Acz_low: byte bajo eje z acelerómetro.
- Acz_high: byte alto eje z acelerómetro.
- Gyx_low: byte bajo eje x giroscopio.
- Gyx_high: byte alto eje x giroscopio.
- Gyy_low: byte bajo eje y giroscopio.
- Gyy_high: byte alto eje y giroscopio.
- Gyz_low: byte bajo eje z giroscopio.
- Gyz_high: byte alto eje z giroscopio.

Para procesar este array de datos, se obtiene el dato correspondiente a cada dos bytes del array de la siguiente forma:

```
for(int i = 0; i <= 5; i++) {
    dataConv[i] = (data[2 * i + 1] & 0xFF << 8 | data[2 * i] &
0xFF);
}
```

Una vez se tienen los datos de los distintos ejes de aceleración hay que realizar la siguiente conversión, primero convirtiendo el dato desde complemento a 2, y multiplicando por la ratio de aceleración[19] para obtener el dato en g (9.8m/s^2):

```
for (int i = 0; i <= 2; i++) {
    if (dataConv[i] > 32767)
        dataConv[i] = (-(65534 - dataConv[i]));
    dataConv[i] = dataConv[i] * RATIO_ACC;
}
```

Una vez calculados los valores de aceleración, se procede a calcular el ángulo que forma cada sensor inercial llamando al método *calcularAngulo(valores,uuid)*.

6.4.2 calcularAngulo()

Este método recibe el array de valores calculados y la UUID de la característica que ha generado los valores.

Con estos datos se procede a calcular el ángulo que forma cada sensor.

[19] Ratio_acc = (4.0/32767)

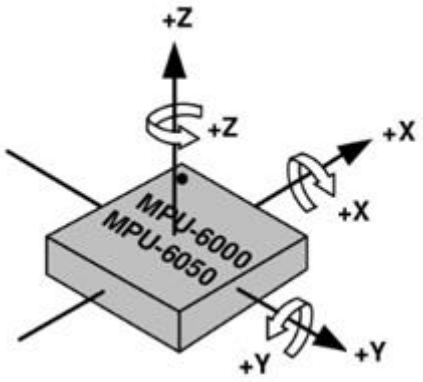

Ilustración 13 Orientación del Sensor Inercial

Los sensores inerciales están orientados en la camiseta de forma que el eje Z es perpendicular a la zona lumbar y el eje X está alineado con la columna vertebral.

Por lo tanto, para mostrar una vista de perfil del ángulo que forma cada sensor, se calculará la flexión, es decir, el ángulo formado por los ejes Z e X. Este cálculo se realiza de la siguiente forma:

$$flexion = atan2(valorX, valorZ) * 180/\pi$$

Convertido a código Java:

```
public static void calcularAngulo(double[] valores, UUID uuid){
    double flexion;
    flexion = (Math.atan2(valores[0],valores[2])) * 180/Math.PI;
    GlobalVariables.values.put(uuid,flexionFiltro);
}
```

El valor calculado se guarda en un HashMap global para su futuro uso.

6.5 Global Variables

Esta clase Java extiende *Application* para poder ser accedida desde cualquier clase y servir para guardar variables globales.

En ella se guarda un HashMap con los valores de cada ángulo y el UUID del sensor que lo genera, la altura y anchura de la pantalla del dispositivo móvil y un valor entero para indicar el nivel de monitorización elegido por el usuario.

```
public class GlobalVariables extends Application {

    public final static HashMap<UUID,Double> values = new
HashMap<>();

    public static int notificationValue = 0;

    public static int displayWidth = 0;
    public static int displayHeight = 0;
}
```

6.6 Draw View

Esta clase extiende la clase View para poder implementar una vista gráfica de los datos procesados obtenidos de los sensores. Para ello se sobrescribe el método *onDraw()* de la clase View. Este método recibe un objeto *canvas,* un objeto de la clase *Canvas* que define métodos para dibujar texto, líneas, figuras y muchas más formas gráficas primitivas.

En primer lugar, se crea un objeto tipo *Paint* que define cómo se dibujarán las formas gráficas. Se inicializa este objeto con un valor de color y tamaño de línea dentro del método *init()* de la clase DrawView.

```
private void init() {
    paint.setColor(Color.GREEN);
    paint.setStrokeWidth(10);
}
```

A continuación, se puede proceder a sobrescribir el método *onDraw().*

```
@Override
public void onDraw(Canvas canvas) {

    int
startX11,startY11,startX12,startY12,endX11,endY11,endX12,endY12;
    int
startX21,startY21,startX22,startY22,endX21,endY21,endX22,endY22;
    int
startX31,startY31,startX32,startY32,endX31,endY31,endX32,endY32;
    int
startX41,startY41,startX42,startY42,endX41,endY41,endX42,endY42;
    int
startX51,startY51,startX52,startY52,endX51,endY51,endX52,endY52;
    int valueGreen = 0, valueOrange = 0;

    int displayHeight = GlobalVariables.displayHeight;
    int displayWidth = GlobalVariables.displayWidth;
    double value;
    double anguloAjustado;

    int h = displayHeight / 12;
```

Se inicializan varios valores para las coordenadas de cada línea que formará la visualización gráfica de cada sensor y un valor para la altura que tendrá cada línea obtenido según el tamaño de la pantalla para que se visualice bien en cualquier tamaño de pantalla.

También se definen los valores de los rangos de cada nivel de monitorización, para así poder variar el color de la representación del sensor según el tamaño del ángulo que forma y el nivel de monitorización elegido.

```
switch(GlobalVariables.notificationValue){
    case 0:
        valueGreen = 15;
        valueOrange = 30;
        break;
    case 1:
        valueGreen = 10;
        valueOrange = 20;
        break;
    case 2:
        valueGreen = 5;
        valueOrange = 10;
        break;
    default:
        valueGreen = 0;
        valueOrange = 0;
        break;
}
```

Una vez inicializados estos valores, se puede proceder a generar la visualización gráfica de cada sensor, según los datos obtenidos y calculados.

Para ello, en primer lugar, se extrae el valor del ángulo de *values,* y a continuación se calculan las coordenadas (x,y) iniciales y finales de la línea.

```
value = GlobalVariables.values.get(UUID.fromString("0000FF63-0000-
1000-8000-00805f9b34fb"))+ 90;
startX31 = startX32 = displayWidth / 2;
startY31 = startY32 = displayHeight / 2;

endX31 = (int) (startX31 + h * Math.sin(value * Math.PI/180));
endY31 = (int) (startY31 + h * Math.cos(value * Math.PI/180));

endX32 = (int) (startX32 - h * Math.sin(value * Math.PI/180));
endY32 = (int) (startY32 - h * Math.cos(value * Math.PI/180));
```

Estas coordenadas se calculan de forma secuencial, y se usa como sensor pivote para la mostración por pantalla, el sensor situado en el centro de la camiseta. Así pues, cada conjunto de coordenadas calculado dependerá de las coordenadas del sensor anterior, creando así una visualización continúa y ajustada a la realidad de la posición de los sensores en cada instante.

Una vez obtenidos las coordenadas, hay que establecer un color para *paint* según la desviación del ángulo y el nivel de monitorización elegido. Esto se hace de la siguiente manera.

```
anguloAjustado = value - 180;
if(anguloAjustado < 0 ){
    anguloAjustado *= -1;
}

if (anguloAjustado <= valueGreen) {
    paint.setColor(Color.GREEN);
    canvas.drawLine(startX31,startY31,endX31,endY31,paint);
    canvas.drawLine(startX32,startY32,endX32,endY32,paint);
}
else if(anguloAjustado <= valueOrange) {
    paint.setColor(Color.rgb(255, 152, 0));
    canvas.drawLine(startX31,startY31,endX31,endY31,paint);
    canvas.drawLine(startX32,startY32,endX32,endY32,paint);
}else{
    paint.setColor(Color.RED);
    canvas.drawLine(startX31,startY31,endX31,endY31,paint);
    canvas.drawLine(startX32,startY32,endX32,endY32,paint);
}
```

La desviación del ángulo se calcula respecto a 180° debido a que el sistema de coordenadas de una pantalla en el sistema operativo Android empieza desde la coordenada (0,0) en la esquina superior izquierda, aumentando el eje X hacia la derecha y el eje Y hacia abajo. Por lo que el ángulo de 0° se considera recto hacia abajo siguiendo el eje Y positivo.

Una vez calculados todos los valores necesarios y dibujadas las representaciones de los sensores, se procede a invalidar el *canvas* y así llamar otra vez automáticamente al método *onDraw()* que calculará de nuevo los valores para representar los sensores. Esto se hace dentro de un nuevo *Runable()* para forzar una pequeña espera en el redibujado y crear una mejor experiencia de usuario.

```
new Handler().postDelayed(new Runnable(){
    public void run(){
        invalidate();
    }
}, 1000);
```

7. Futuras Mejoras

En futuras revisiones de la APP se pueden implementar mejores para aumentar el nivel de satisfacción del usuario y la usabilidad y utilidad de la APP.

Estas mejoras pueden incluir la incorporación de una pantalla de configuración a través de un PreferenceFragment. El uso de esta API Android permite crear una pantalla de configuración coherente con la experiencia de usuario de otras aplicaciones Android.

Con ello se podría permitir al usuario opciones como:

- Elegir el idioma de la APP.
- Elegir entre un tema día/noche para las interfaces de la APP.
- Sincronizar los ajustes con la cuenta personal de Google del usuario para futuras instalaciones.

También se podría aumentar la utilidad de la APP, incorporando a este fragmento el nivel de monitorización elegido, guardando así su estado para futuros usos de la APP.

8. Conclusión

Este proyecto impulsado por AITEX es uno de gran importancia a la hora de ayudar al deportista aumentar su rendimiento y prevenir lesiones debidas a practicar deporte con una postura incorrecta.

Ha sido un reto desarrollar esta APP de forma correcta, siguiendo todos los pasos necesarios para un desarrollo profesional, partiendo de una recopilación de información, y generación de diseños y acabar con la implementación.

Ha servido para aumentar y solidificar mis conocimientos sobre el desarrollo de proyectos informáticos profesionales y también para aumentar mis conocimientos de la programación de aplicaciones Android.

Dicho esto, solo queda destacar que esta invención es una tecnología simple, usada de forma innovadora, puesto al alcance de todo el mundo.

9. Bibliografía

A continuación, se puede encontrar una lista de enlaces que han servido para la recolecta de datos e información a lo largo de este proyecto.

- Android Market share: https://qz.com/826672/android-goog-just-hit-a-record-88-market-share-of-all-smartphones/
- Herramienta de creación de maquetas: https://mockuphone.com/
- Herramienta Color Tool de Google.: https://material.io/color/
- https://developer.android.com/develop/index.html
  - Bluetooth Low Energy: https://developer.android.com/guide/topics/connectivity/bluetooth-le.html
  - Fragments: https://developer.android.com/guide/components/fragments.html
- https://stackoverflow.com/