UNIVERSIDAD CARLOS III DE MADRID

MEMORIA DEL TRABAJO DE FIN DE GRADO

INTERFAZ DE PROGRAMACIÓN DE SOLUCIONES BIOMÉTRICAS BAJO BIOAPI 3.0 "FRAMEWORK LESS" EN PLATAFORMA ANDROID

Rubén Romero García

Director: Raúl Sánchez Reíllo

Co-director: Raúl Alonso Moreno

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE ACRÓNIMOS

API: Application Programming Interface

BDB: Biometric Data Block

BFP: BioAPI Function Provider

BIR: Biometric Information Record

BSP: Biometric Service Provider

CBEFF: Common Biometric Exchange Formats Framework

CER: Cross-over Error Rate

FAR: False Acceptance Rate

FER: Failure-to-enroll Rate

FMR: False Match Rate

FRR: False Rejection Rate

FPI: Function Provider Interface

GUI: Graphical User Interface

ID: Identity/Identification/Identifier

IDE: Integrated Development Environment

IRI: Internationalized Resource Identifier

MAC: Message Authentication Code

NDK: Native Development Kit

SB: Security Block or Signature Block

SBH: Standard Biometric Header

SO: Sistema Operativo

SPI: Service Provider Interface

UUID: Universally Unique Identifier

AGRADECIMIENTOS

Quiero agradecerle este trabajo a la gente sin la cual sería imposible estar escribiéndolo hoy: a mis padres, Javier y Yolanda, por su apoyo constante, y a mis hermanos Alberto y Mario.

También a todos mis amigos, sin nombres, porque seguro que me dejo alguno, por haber estado ahí siempre que lo he necesitado.

RESUMEN

La identificación biométrica es la identificación basada en las características biológicas, físicas y de comportamiento de una persona. Desde hace ya varios años es uno de los métodos más usados para confirmar la identidad de una persona. Su excelente tasa de reconocimiento, sumado a su riesgo prácticamente nulo de falsificación (si el sensor dispone de mecanismos de detección de sujeto vivo), ha hecho que importantes organismos como la policía o la banca lo consideren cada vez más esencial a la hora de reconocer personas.

Por otro lado, los smartphones o "teléfonos inteligentes" han obtenido en el último lustro una penetración altísima en la sociedad, convirtiéndose en herramientas indispensables para la gran mayoría de la población. Sus potentes sistemas operativos los convierten en poco menos que ordenadores de bolsillo, permitiéndoles realizar acciones con un coste computacional inasumible hace poco tiempo. De entre ellos, destacan aquellos que tienen Android como Sistema Operativo (SO), al estar dominando actualmente el mercado.

En este Trabajo de Fin de Grado (TFG) se ha portado el código de BioAPI, uno de los estándares para aplicaciones biométricas más importantes, desde Java hacia Android. Además se han realizado los cambios necesarios en el mismo para que cumpla las especificaciones de Framework Free, con el fin de poder usarlo sin problemas en dispositivos con poca memoria como pueden ser los smartphones de baja gama. Por último se ha creado una aplicación Android que utiliza el código creado, sin otro propósito que el de comprobar su funcionamiento.

El presente documento introduce al lector en la tecnología empleada y describe el proceso de desarrollo de la plataforma BioAPI Framework Free y de la aplicación.

ABSTRACT

Biometric identification, i.e, the identification that is based in biophysics and behavioral characteristics of a person, has been, for several years, one of the most used methods to verify the identity of someone. It has an excellent ratio of correct recognition and it is very hard to forge, and both points are making this science increasingly important for police or banking to recognize people.

Furthermore, smartphones have become essential for most people since last five years. They have powerful operating systems (like Android, the dominant at the moment) that allow them to make actions which computational cost was unacceptable a couple years ago, becoming them in a sort of pocket computers.

In this paper it has been done the transcription of the BioAPI's code, one of the most important standards for biometric applications, from Java to Android. Also, it has been applied the pertinent changes in that code to carry out the demanded requirements of the Framework Free version, for its use in low cost devices without large memory capacity. At last, an Android application has been created which implements that code, with the purpose of checking it out.

This document introduces the reader in the used technology and describes the development process of the transcription and the creation process of the application.

1. INTRODUCCIÓN

A lo largo de esta memoria se va a presentar un TFG que tiene como finalidad la implementación del estándar BioAPI (ISO/IEC 19784-1) en su modalidad de Framework Free, para la plataforma Android. Como método de prueba de dicha implementación, se ha creado una aplicación de identificación mediante usuario y contraseña que utiliza su código. La utilidad de este trabajo es dar un paso más en la implantación de la identificación biométrica en dispositivos móviles.

1.1. Motivación

Desde la explosiva aparición que protagonizaron los smartphones a mediados de la década pasada, han ido cobrando importancia paulatinamente hasta convertirse actualmente en la herramienta más usada por una gran cantidad de personas, por delante de otras más potentes como pueden ser los ordenadores. Una de las posibles explicaciones para este fenómeno es la flexibilidad que proporcionan los dispositivos móviles: ya sea para conectarse a internet, usar una aplicación concreta, comunicarse verbal o textualmente con alguien o simplemente para jugar, leer o escuchar música, estos instrumentos eliminan la necesidad de estar anclado a un lugar fijo. La idea no es nueva: existen desde hace tiempo e-books, consolas portátiles, reproductores MP3 y móviles; sin embargo, los smartphones aúnan todas las funciones de esos dispositivos en uno solo, y los usuarios utilizan al máximo todas ellas, como puede verse en la Figura 1[1]. Desde un punto de vista tecnológico, un smartphone no supera a un MP3 si se habla de reproducción de música, ni tiene las cualidades de pantalla que posee un e-book. Pero puede realizar las funciones de ambos de una forma más que correcta, lo que elimina la necesidad de tener que llevar un dispositivo para cada función.

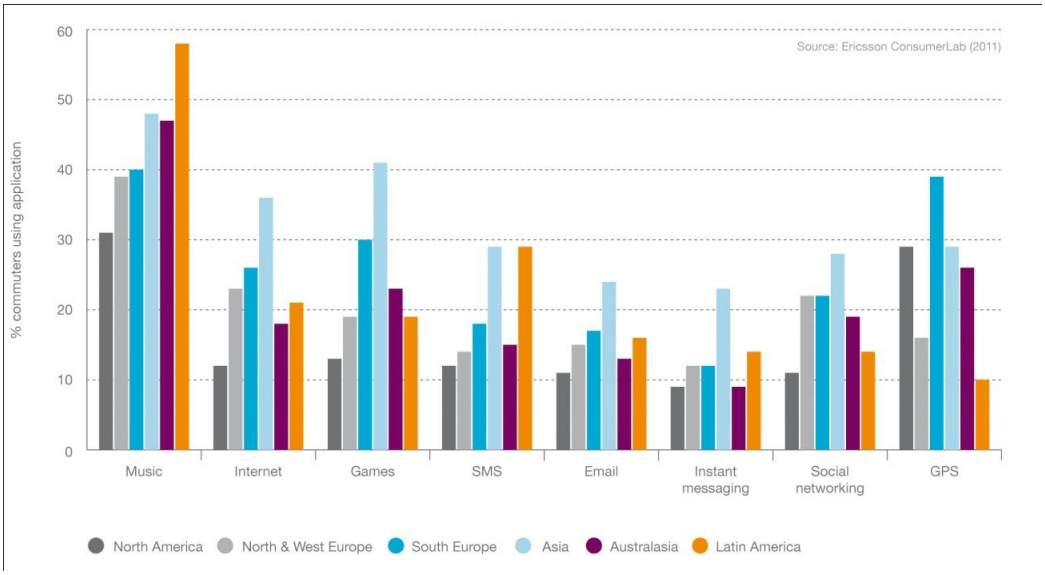


Figura 1. Uso diario del smartphone según sus aplicaciones (estudio por países)

Un ordenador portátil sería capaz de realizar todas esas funciones con mayor eficacia, pero los smartphones tienen una enorme ventaja sobre ellos: el tamaño. Caben en un bolsillo, frente a las quince pulgadas de pantalla y tres kilogramos de peso que puede tener un portátil, siendo de esta manera mucho más cómodos de usar. Esta unión de funcionalidad, comodidad y flexibilidad que proporcionan los smartphones (y de forma más reciente las tablets) los ha convertido en una tecnología imprescindible para la sociedad actual.

Dentro de los diferentes sistemas operativos que existen para dispositivos móviles, hay que destacar Android, que a día de hoy domina el mercado. Esta plataforma ha ganado mucho peso en los últimos años y posee un potencial enorme con vistas al futuro. Tanto en el ámbito laboral como en el personal, dominar los conceptos básicos de este sistema operativo representa una gran ventaja, y observando el índice de crecimiento que posee es lógico suponer que cada vez será más esencial para cualquier desarrollador. Por tanto, introducirse en el mundo de la programación para dispositivos móviles en el sistema operativo que más presencia tiene actualmente es una de las motivaciones para este trabajo.

Por otro lado, la biometría es un campo que, si bien es cierto que lleva más de un siglo desarrollándose, ha experimentado un gran salto en las últimas décadas. Métodos biométricos como el reconocimiento facial, de geometría de mano, de iris o de voz han dejado de ser parte de la literatura de ciencia ficción para convertirse en una realidad. A medida que el interés industrial por esta ciencia fue creciendo, surgió un grave problema en lo referente al uso de técnicas biométricas en ambientes informáticos. Al no existir un estándar relativo a ello, cada proveedor suministraba su propia interfaz de software para sus productos, lo que dificultaba o incluso imposibilitaba a las empresas el cambio de producto o de vendedor. Para tratar de solucionar este problema se fundó BioAPI Consortium, un consorcio que desarrolló un estándar para la conexión entre dispositivos biométricos y sus aplicaciones. Más adelante se hablará con detalle de dicho estándar; de momento basta con decir que portarlo hacia una plataforma nueva, ayudando al paso de las tecnologías biométricas a los dispositivos móviles, es otra de las motivaciones principales del presente trabajo.

1.2. Objetivos

Al hablar de los objetivos es conveniente separarlos en dos partes: los que forman el trabajo en sí, es decir, los requisitos y especificaciones que ha de cumplir la aplicación final, y los que se persiguen en cuanto a lo que la realización del trabajo le aporta al alumno.

En el terreno tecnológico, el primer objetivo es portar el código de BioAPI desde Java hasta Android. La primera versión del estándar BioAPI que se hizo fue creada en C

(ISO/IEC 19784-1). Actualmente, es la única versión comercializada y con una norma publicada. Existen otras dos versiones: una en C# (ISO/IEC 30106-3, en desarrollo) y otra, inacabada, en Java (ISO/IEC 30106-2), como las únicas versiones en programación orientada a objetos. En este trabajo se pide crear una versión nueva escrita para Android, a partir de la versión de Java. Además, el código debe cumplir los requisitos que exige la norma de la versión Framework Free del BioAPI C. Por tanto, el segundo y principal objetivo es adaptar esos requisitos a la programación orientada a objetos (ya que la versión Java actual no dispone de versión Framework Free) y aplicarlos a la nueva versión creada en Android.

Por último, el tercer objetivo es crear una aplicación que utilice la versión de BioAPI Framework Free que se habrá creado en Android, con el fin de comprobar el funcionamiento de la misma.

En cuanto al aprendizaje del alumno, el primer objetivo es la familiarización con una nueva plataforma, Android, y un nuevo entorno de desarrollo, Eclipse, completamente distintos a los aprendidos durante los estudios del Grado. La obligación de aprender a trabajar con estas herramientas desde cero es una de las mejores formas de asegurar que el estudiante domina las bases de las mismas, ya que sin ellas no se puede avanzar en un proyecto nuevo.

Un segundo objetivo, menos evidente que el primero, es la introducción del alumno en un tipo de trabajo similar al que se encuentra luego en el mundo laboral: los proyectos. Salvo pequeñas imitaciones en ciertas asignaturas, no se realizan proyectos de envergadura a lo largo de la carrera, y ésta es, por tanto, la primera toma de contacto del alumno con lo que se le exigirá cuando comience a trabajar.

1.3. Estructura

En el presente documento se realizará una breve introducción a la tecnología con el análisis del estado del arte y las tecnologías asociadas. Este apartado incluye los lenguajes de programación empleados: Java y su adaptación a la plataforma Android; el entorno de desarrollo usado: Eclipse; y una breve exposición de la historia del estándar BioAPI y de su versión Framework Free. A continuación se explicará cómo se ha llegado a la solución final entre los distintos planteamientos y alternativas que existían. Tras esto se expondrá de forma precisa el desarrollo de dicha solución, los problemas encontrados y la forma de resolverlos, así como las pruebas realizadas para asegurar su correcto funcionamiento. Por último se puede encontrar una breve conclusión que incluye el peso que puede tener este proyecto en líneas de trabajo futuras.

2. ESTADO DEL ARTE

En este capítulo se introducirá al lector en la tecnología. Se realizará un breve resumen de la historia de la telefonía móvil y se hablará de los diversos sistemas operativos para dispositivos móviles existentes. A continuación, se realizará una introducción a la biometría, desde sus inicios hasta la actualidad, y se explicarán los diversos elementos y técnicas que existen en este campo.

2.1. Historia y evolución de la telefonía móvil

La búsqueda de un sistema de comunicación a distancia que funcionase de forma móvil ha sido una constante desde los inicios de la Segunda Guerra Mundial. El considerado por muchos como el primer teléfono móvil de la historia, el Handie Talkie H12-16, que puede verse en la Figura 2[2], fue lanzado por Motorola durante esta época. Funcionaba mediante ondas de radio y se usaba para mantener la información entre las tropas constantemente actualizada.

Figura 2. Handie Talkie H12-16

Tras este prototipo, la comunicación inalámbrica comenzó un lento pero imparable ascenso. Martín Cooper fabricó el primer radio teléfono entre 1970 y 1973 en EEUU, siendo imitado por la compañía NTT en el mercado japonés (1979). En los años sucesivos se fueron introduciendo importantes modificaciones y mejoras, con el fin de perfeccionar el sistema, y finalmente, en la década de los 80 nace la primera generación de teléfonos móviles con la creación del Motorola DynaTAC 8000X (ver Figura 3[3]).

Figura 3. Motorola DynaTAC 8000X

Era un teléfono caro, grande, pesado y antiestético, pero también era el primer teléfono móvil destinado al uso civil y supuso una revolución para su época. Aunque fue concebido como un método para mantener una comunicación constante entre los distintos sectores de una empresa, la idea caló en el público y pronto empezaron a comercializarse modelos destinados al público en general.

Años más tarde, en la década de los 90 y en plena evolución desde la tecnología móvil analógica hacia la digital, se creó el estándar GSM (Sistema Global para las Comunicaciones Móviles, del francés *Groupe Spécial Mobile*) que, al ser digital, permitía más enlaces simultáneos en un mismo ancho de banda. Además, integraba otros servicios en la misma señal, como el envío y recepción de mensajes de texto (SMS) o una elevada capacidad de envío y recepción de datos desde dispositivos de fax y módem: mediante una conexión con un ordenador era posible la navegación por internet o la recepción y envío de correos electrónicos. Por si esto fuera poco, daba al usuario la posibilidad de estar comunicado fuera del alcance de su red, utilizando la de otra compañía, mediante la *itinerancia*. Los operadores de red también acogieron con agrado este sistema, ya que con él podían elegir entre múltiples proveedores de sistemas GSM, al ser un estándar abierto que no necesitaba pago de licencias.

Este estándar fue el primer paso hacia la popularización actual de los teléfonos móviles. Tras él, continuaron apareciendo mejoras, dirigidas principalmente hacia una mayor capacidad de transmisión de datos, cuyo culmen llegó con el nacimiento de la tercera generación, basada en el protocolo UMTS (Servicio Universal de Telecomunicaciones Móviles, del inglés Universal Mobile Telecommunications System). Los puntos fuertes de este sistema eran sus capacidades multimedia, una transmisión de voz con calidad equiparable a la de las redes fijas y una elevada velocidad de acceso a Internet que permitía el envío de voz y datos simultáneamente en forma de videollamada.

Sin embargo, a pesar de las continuas mejoras, la verdadera revolución en el campo de la telefonía móvil llegó de la mano de los teléfonos inteligentes o smartphones. El primer teléfono catalogado como "*smartphone"* fue el GS88 de Ericsson de 1997. A partir de ese momento, diversas compañías fueron desarrollando modelos similares: Microsoft en 2001 con su Windows CE Pocket PC OS, RIM en 2002 con BlackBerry, Nokia en 2007 con el N95, Apple, también en 2007, con el iPhone, y finalmente Google, en 2010, con el Nexus One, el primer smartphone basado en el sistema operativo Android.[4]

Estos dispositivos se construyen sobre una plataforma informática móvil, aunando las capacidades de un teléfono convencional con otras propias de un ordenador personal. Entre las diferencias que existen con los móviles anteriores están: la posibilidad de instalar aplicaciones -incluso desde terceros- la función multitarea, el GPS o el acceso a internet vía WiFi o 3G. Aunque algunos teléfonos 3G ya disponían de esta última capacidad, la velocidad y la calidad de la transmisión de datos mejoran notablemente en

los smartphones. Por estos y otros motivos, desde la irrupción de estos dispositivos, la cantidad de líneas telefónicas en activo ha aumentado notablemente, como puede verse en la Figura 4[5].

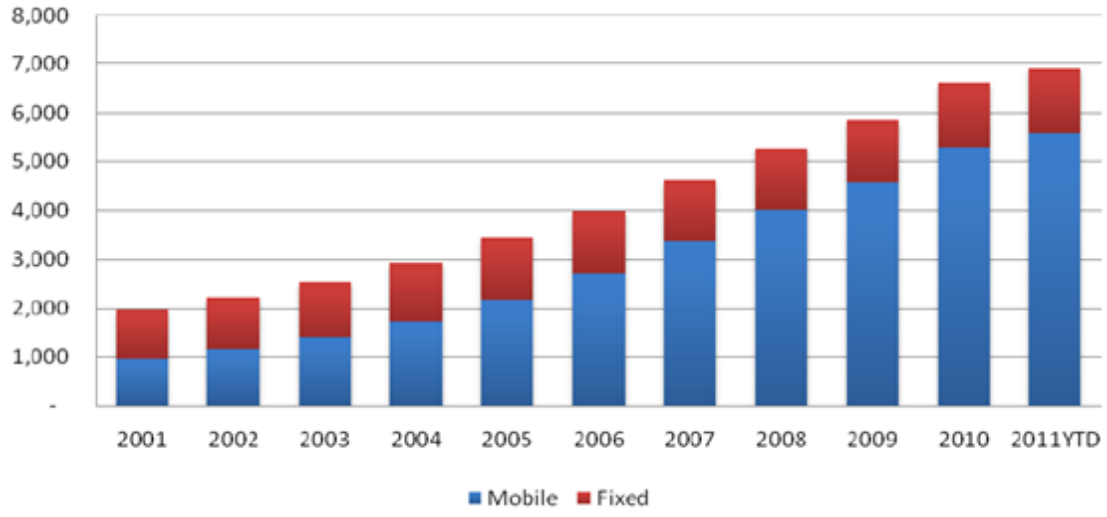


Figura 4. Conexiones telefónicas en activo a nivel mundial

2.2. Sistemas Operativos

Los sistemas operativos de los smartphones son el equivalente en dispositivos móviles a sus homónimos en ordenadores personales. Al estar diseñados para controlar una tecnología menos potente son más simples y sus funciones principales son la conectividad inalámbrica y la gestión del flujo de información y datos que entran y salen del teléfono.

Su programación interna se estructura en capas. La básica, el núcleo o kernel, es la que gestiona el acceso a los distintos elementos del hardware mediante drivers. También se ocupa de la gestión de procesos y de memoria y del sistema de archivos. Suele estar basado en un SO para ordenadores, como puede ser Windows, Linux o Unix. El siguiente nivel es el middleware, un conjunto de módulos que mueven las aplicaciones. En esta capa se encuentran los códecs multimedia, los intérpretes de páginas web o el motor de mensajería y comunicaciones, entre otros. Un nivel más arriba se encuentra el entorno de ejecución de aplicaciones, que se ocupa de gestionarlas y consta de un conjunto de interfaces que pueden ser usadas por los desarrolladores para crear nuevo software. Por último, en el nivel superior se encuentran las interfaces de usuario, que permiten la interacción persona-máquina por medio de componentes gráficos, como botones y campos de texto, y realizan la presentación visual de las aplicaciones.

Además de estas capas también existen una serie de aplicaciones nativas del teléfono no eliminables, como los menús o el marcador de números[6].

Hasta la llegada de Android, el primer SO de código abierto, cada empresa tenía su propio SO implantado en sus terminales. Por ello hay una gran variedad, siendo los más importantes Windows Mobile, Palm OS, Symbian, iPhone OS, BlackBerry y Android. Cada uno tiene sus ventajas y sus inconvenientes, como se puede ver en la comparativa realizada en las Figura 5, Figura 6,Figura 7 y Figura 8[7].

Detalles básicos	 Android Cupcake	 BlackBerry OS 4.7	 iPhone OS 3.0	 S60 5th Edition	 Palm WebOS	 Windows Mobile 6.5
Tipo de núcleo	Linux	Propietario	OS X	Symbian	Linux	Windows CE
Adaptabilidad	Excelente	Buena	Mala	Excelente	Excelente	Excelente
Edad de la plataforma	Joven	Madura	Adolescente	Madura	Joven	Madura
Soporte para empresas	Nada	BlackBerry	Exchange	Exchange, Domino, BlackBerry	Exchange	Exchange, Domino, BlackBerry
Tecnologías inalámbricas	GSM, WiFi	GSM, CDMA, WiFi	GSM, WiFi	GSM, WiFi	GSM, CDMA, WiFi	GSM, CDMA, WiFi

Figura 5. Comparativa entre sistemas operativos móviles (I)

Interfaz de usuario	Android Cupcake	BlackBerry OS 4.7	iPhone OS 3.0	S60 5th Edition	Palm WebOS	Windows Mobile 6.5
Gestos	Sí	Sí	Sí	Limitado	Sí	Limitado
Tecnología de la pantalla	Capacitiva	Capacitiva	Capacitiva	Resistiva / Capacitiva	Capacitiva	Resistiva
Multitáctil	Sí (no oficial)	Sí	Sí	No	Sí	No
Cambios de temas	Sí	Sí	No	Sí	No	Sí
Obtención de información	Teclado virtual, teclado físico	Teclado virtual	Teclado virtual	Teclado virtual, T9, y triple clic; reconoce caracteres; teclado físico	Teclado físico	Teclado virtual, reconoce caracteres, teclado físico

Figura 6. Comparativa entre sistemas operativos móviles (II)

Funcionamiento						
	Android Cupcake	BlackBerry OS 4.7	iPhone OS 3.0	S60 5th Edition	Palm WebOS	Windows Mobile 6.5
Notificación	Bandeja	Pop-up, fondo	Pop-up	Pop-up	Bandeja	Bandeja, pop-up
Administración de contactos	Google	BES, BIS	Exchange, ActiveSync, Mac OS Address Book	Exchange, Domino, BlackBerry, iSync	Synergy	Exchange, Domino, BlackBerry, ActiveSync
Multitasking	Sí	Sí	No	Sí	Sí	Sí
Copiar / pegar	Sí	Sí	Sí	Sí	Sí	Sí
Ecosistema / Soporte multimedia	Amazon	iTunes sin DRM	iTunes	Ovi	Amazon	Windows Media Player
Búsqueda global	No	No	Sí	Sí	Sí	No
Actualización de firmware	OTA	Tethered, OTA	Tethered	Tethered, OTA	Desconocido	Tethered, OTA
Motor del navegador	WebKit	Propietario	WebKit	WebKit	WebKit	Internet Explorer
Tethering (módem)	Sí (no oficial)	Sí	Sí	Sí	Sí	Sí
Bluetooth estéreo	Sí	Sí	Sí	Sí	Sí	Sí

Figura 7. Comparativa entre sistemas operativos móviles (III)

Desarrollo de terceros						
	Android Cupcake	BlackBerry OS 4.7	iPhone OS 3.0	S60 5th Edition	Palm WebOS	Windows Mobile 6.5
Disponibilidad de SDK / Soporte	Sí	Sí	Sí	Sí	Sí	Sí
Tienda de aplicaciones	Sí	Próximamente	Sí	Próximamente	Sí	Sí
Disponibilidad de aplicaciones	Mediana	Mediana	Alta	Mediana	Baja	Alta
Aplicaciones nativas	No	No	Sí	Sí	No	Sí
Administración local de aplicaciones	Excelente	Buena	Excelente	Buena	Excelente	Buena

Figura 8. Comparativa entre sistemas operativos móviles (IV)

Como se puede ver en la comparativa, todos tienen sus puntos fuertes, por lo que no resulta apropiado hablar de un SO mejor que otro de forma general. Sin embargo, para el tema que se trata en este trabajo, Android es el SO más apropiado, ya que proporciona muchas facilidades a los desarrolladores y además, al ser el que mayor cuota de mercado tiene, posibilita que más personas tengan acceso a las aplicaciones creadas.

2.3. Historia de la biometría

La biometría es la ciencia que clasifica, registra e identifica a las personas mediante sus características biológicas, físicas y/o de comportamiento. Se cree que los antiguos egipcios ya usaban métodos biométricos, concretamente la firma, para confirmar la identidad de las personas, y se sabe con seguridad que los comerciantes chinos distinguían a los niños mediante impresiones en papel de la huella de la palma de la mano ya desde el siglo XIV.

En Europa, la identificación biométrica comenzó a utilizarse en el siglo XIX con fines policiales. Originalmente la comparación se hacía de forma tosca, confiando en la memoria visual; básicamente consistía en reconocer o no al sospechoso, basándose en

las características físicas que el policía encargado consiguiera recordar, hasta que en 1883 un miembro de la policía parisina, Alphonse Bertillon, desarrolló el sistema antropométrico de clasificación. Este método funcionaba midiendo de forma precisa ciertas longitudes y anchuras del cuerpo y la cabeza del individuo, y registrando marcas personales indelebles como tatuajes, manchas o cicatrices. Fue usado ampliamente hasta que se descubrieron dos personas diferentes con el mismo conjunto de medidas. Este descubrimiento relegó la antropometría a la categoría de pseudociencia, y obligó a buscar una alternativa para la identificación policial de sospechosos[8].

Fue en el año 1892 cuando Francis Galton, un antropólogo inglés, realizó las primeras pruebas satisfactorias de identificación de criminales mediante huella dactilar basándose en los estudios de Henry Faulds sobre la unicidad y estabilidad de las mismas durante la vida de un individuo. Este nuevo método se impuso rápidamente, convirtiéndose en poco tiempo en la forma oficial de identificación en diversos países. Es tal su eficacia que en la actualidad, aunque los métodos de archivo y comparación han mejorado notablemente, aún se usan los cuatro rasgos seleccionados por Galton para diferenciar dos huellas entre sí: arcos, presillas internas, presillas externas y verticilos.[9]

Aunque la huella dactilar sigue siendo el más empleado, a lo largo del siglo XX se han propuesto y estudiado una gran cantidad de métodos biométricos nuevos. En 1936 el patrón de iris fue sugerido como método de identificación por Frank Burch, idea que fue desarrollada finalmente entre 1985 y 1994 por Leonard Flom y Aran Safir, usando los algoritmos de reconocimiento de John Daugman. En el 2006 se evaluaron los mejores algoritmos de reconocimiento facial, resultando algunos tan exactos que distinguían entre gemelos idénticos. También se han propuesto y desarrollado métodos de identificación que emplean la vascularización, la firma, la voz, el modo de andar… Prácticamente cualquier característica biofísica tiene su propio estudio biométrico, con mayor o menor eficacia.

La eficacia de un método biométrico depende de su tasa de falso positivo o FAR, de su tasa de falso negativo o FRR y de la de fallo de alistamiento o FMR. En los dispositivos reales se cruzan ambas tasas y el sistema se ajusta según el punto de cruce, conocido como CER, como puede verse en la Figura 9[10]. Cuanto más bajo es el CER, más exactitud proporciona el sistema.[11]

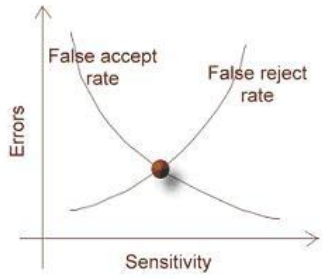


Figura 9. Cruce de FAR y FRR. Localización del CER

2.4. Elementos de un sistema biométrico

En todos estos sistemas se sigue un esquema predefinido de obtención de datos que es independiente del método biométrico utilizado y puede verse en la Figura 10. Este esquema consta de dos fases: reclutamiento y verificación. Ninguna de ellas tiene sentido sin la otra, y ningún sistema biométrico está completo si le falta alguna.

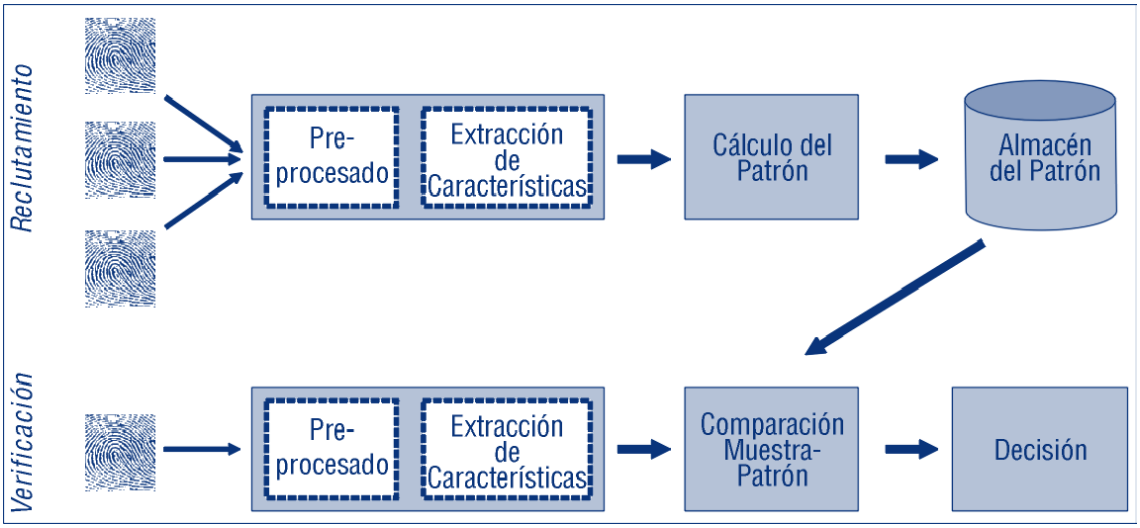


Figura 10. Etapas de un sistema de identificación biométrica

En el reclutamiento se toman una o varias muestras del usuario, se procesan (rotación, ampliación o reducción, limpieza de ruido, etc.) y se extraen sus características para extraer un patrón individual que se almacenará. Este proceso se lleva a cabo de forma supervisada por una segunda persona que compruebe en cada momento que el usuario que está siendo reclutado es realmente quien dice ser.

Según el tipo de método biométrico que se emplee en el sistema, las muestras pueden ser tomadas en un mismo día (iris, huella dactilar) o en varios (voz, cara) para asegurar que el patrón toma en cuenta la variabilidad existente. Las características extraídas dependen tanto del método biométrico como del algoritmo empleado, ya que existen distintas formas de comparar muestras incluso dentro del mismo método (por ejemplo, comparar huellas mediante las cuatro marcas de Galton, o mediante un conjunto diferente de características individuales).

En la verificación, proceso en el que no es necesaria supervisión, se extrae una muestra biométrica del usuario. Se repiten los pasos del reclutamiento: se procesa la muestra y se extraen las características. A continuación, se comparan con las que el sistema contiene almacenadas, y se devuelve un resultado positivo o negativo.

La verificación de la identidad de un usuario no es exacta. Las pequeñas variaciones que existen entre muestras del mismo usuario o el error que pueden tener los instrumentos de captura empleados hacen que sea inviable reducir la elección a un simple

verdadero/falso. Lo que se hace en su lugar es emplear un porcentaje umbral de semejanza, suponiendo que todo aquel individuo que lo supere en la comparación entre su muestra y la almacenada está verificado. La elección de un umbral adecuado es básica para evaluar la fortaleza de un sistema biométrico, ya que uno demasiado elevado dispararía la tasa de falsos negativos, mientras que un umbral muy bajo haría que la tasa de falsos positivos fuese inaceptablemente alta.

Por último hay que indicar que la identificación puede ser de dos tipos. Con el primer tipo, llamado reconocimiento, se compara al usuario introducido con todos los existentes en el sistema, y en el caso de encontrar una coincidencia se comunica que la identidad ha sido confirmada. El otro tipo es conocido como autenticación, y se diferencia del reconocimiento en que el usuario introduce la muestra biométrica y su supuesta identidad; tras ello, el sistema únicamente compara los datos introducidos con los que posee correspondientes a la identidad introducida.

2.5. **Métodos biométricos**

Prácticamente cualquier característica biológica o de comportamiento de una persona puede ser usada para su identificación. Normalmente las biológicas se consideran más exactas, ya que cualquier característica de comportamiento es potencialmente imitable, y por tanto, podría no garantizar la identificación con un nivel de error lo suficientemente bajo. Algunos de los parámetros a tener en cuenta al analizar la eficacia de una técnica biométrica son la universalidad, es decir, si las características se pueden extraer de cualquier usuario, la unicidad, que es la probabilidad de que no existan dos personas con las mismas características, la facilidad de captura, el rendimiento, la aceptación por los usuarios, la estabilidad de la muestra, es decir, si permanece inalterable durante la vida del usuario, la robustez frente al fraude o el coste.

A continuación se presentan algunos de los métodos que se estudian actualmente para la identificación biométrica, incluyendo algunos cuya implantación es ya una realidad, haciendo un breve repaso de sus pros y sus contras. Es conveniente empezar esta presentación haciendo hincapié en el hecho de que no existe la técnica biométrica perfecta: una que proporcione una fiabilidad casi absoluta pero que exija un tiempo de proceso grande puede no ser la más adecuada si el tiempo es un factor limitante.

Cada técnica tiene sus ambientes de aplicación destacados, y es necesario un estudio previo del entorno antes de elegir una de ellas.

- Huella dactilar: Es la que tiene una aceptación mayor. Se usa desde el siglo XIX y existen numerosos estudios que prueban tanto la estabilidad de la huella a lo largo de la vida como su unicidad. La facilidad de extracción y el escaso coste de los dispositivos que emplean esta técnica ha permitido que siga compitiendo

con otras biométricamente más exactas pero más caras, como el iris, o de extracción más complicada, como el ADN.
Como punto en contra, su uso por parte de la policía para identificar criminales ha hecho que la población asocie este método con actos delictivos, por lo que algunas empresas prefieren invertir un poco más en una técnica más cara que no tenga implícita esa imagen policial.

- ADN: Es la única técnica conocida que ha conseguido garantizar un cien por cien de éxito. Mediante comparación de ADN la identidad de una persona queda confirmada sin posibilidad de fallo, ya que no hay dos personas que compartan el mismo. Como contrapartida, existe una gran dificultad a la hora de procesar los resultados en tiempo real, necesitándose mucho más tiempo para realizar la comparación que con otros sistemas. Además, las técnicas de extracción (saliva, sangre, sudor) son muy invasivas y el usuario tiende a rechazarlas.

- Iris: Los sistemas que emplean esta técnica parten de los algoritmos creados por John Daugman en 1993. Tiene unos resultados excelentes y de gran fortaleza, y hay estudios que confirman la inalterabilidad del iris a lo largo de la vida, así como su unicidad, siendo esta última muy superior a la de la huella dactilar. Además, la técnica de extracción no es invasiva ni tiene connotaciones policiales negativas. La única razón por la que esta tecnología no es la más usada a nivel mundial es el elevado coste de los equipos.

- Voz: Es una de las técnicas más estudiadas, ya que la creación de un sistema fiable de reconocimiento biométrico basado en la voz haría posible el reconocimiento de una persona de forma remota mediante algo tan sencillo, accesible y barato como un teléfono. Sin embargo, y aunque existen algunos sistemas que responden de forma bastante correcta, la voz está sujeta a múltiples variaciones, tales como la edad, las enfermedades o simples cambios en el estado de ánimo. El umbral del sistema debe rebajarse de forma notable para poder asimilar estos cambios, lo que lo convierte en menos fiable de lo preciso. Además, es muy propenso al fraude, ya que al no necesitar presencia física basta con una grabación para burlarlo. Para solucionar este problema, algunos sistemas obligan al usuario a repetir una locución aleatoria determinada para identificarse.

- Retina: La geometría vascular de la retina es uno de los métodos más seguros de identificación. La unicidad presentada supera incluso la del iris, y no necesita métodos añadidos para detectar si el usuario está vivo. Sin embargo, para poder tomar la muestra es necesario usar láser y la mayoría de gente siente rechazo hacia ello.

- Vascular: Consiste en usar la geometría de las venas de la mano o del dedo como método de identificación. Su uso está en estudio para ambientes donde la identificación digital externa (huella) sea difícil de extraer, ya sea por erosión de la misma o por suciedad.

- Cara: Es una de las tecnologías que más rápidamente está progresando. A mediados de 2006 se celebró la Face Recognition Grand Challenge, un congreso donde se presentaron nuevos algoritmos de reconocimiento facial. Se descubrió que proporcionaban una fiabilidad 10 veces superior a los mejores de 2002, y 100 veces superior a los de 1995. Como ventaja, el método de captura es muy sencillo y barato, y métodos como la hiperresolución del rostro hacen que las fotos en baja calidad no sean ya un problema. La captura es tan sencilla que en ocasiones el usuario no es consciente de que está siendo verificado, lo cual puede llegar a ser potencialmente peligroso si se usa con fines poco éticos. Como contra, es un método muy sensible a variaciones muy habituales como gafas, peinado o vello facial.

- Geometría de la mano: Es el método biométrico más rápido. En un segundo es capaz de confirmar la identidad de una persona. Además, los modelos más modernos van actualizando su base de datos con cada introducción, variando algunos detalles del usuario que pueden cambiar de una autenticación a otra (por ejemplo, adelgazamiento o cicatrización de heridas). Es un método que está especialmente indicado para ambientes con una gran afluencia de individuos, donde la rapidez es un factor crítico y la seguridad puede ser más laxa, ya que la tasa de error es muy superior a otros sistemas.

- Firma: Se utiliza desde antes que la huella dactilar y siempre se ha cuestionado su fiabilidad, ya que la falsificación es relativamente fácil. Desde hace unos años los métodos no solo comprueban la firma sino también la dinámica de la misma: rapidez, paradas, presión sobre el papel… De este modo se alcanza una fiabilidad similar a la de un método biométrico, pero incluso de este modo sus críticos afirman que es potencialmente falsificable con el entrenamiento adecuado.

- Forma de andar: Su estudio se encuentra en desarrollo. De momento su fiabilidad es mínima, y presenta el mismo problema que el reconocimiento facial, el usuario podría no saber que está siendo analizado. De cualquier modo, como método basado en el comportamiento, no hay expectativas de que supere a la firma a corto o medio plazo.[12]

2.6. El problema de la estandarización

Un estándar es algo que sirve de patrón o referencia. Aplicado al campo de la informática, es el conjunto de normas que debe cumplir un producto o procedimiento para ser compatible con otro. Su finalidad principal es reducir las diferencias entre productos y hacer más sencilla la colaboración y el crecimiento de las tecnologías, pero además debe proporcionar un ambiente de estabilidad y madurez que repercuta en un mayor beneficio tanto de los consumidores como de los desarrolladores.

Históricamente, al ser la biometría un campo tan disperso, no han existido estándares en el ámbito industrial. Cada fabricante desarrollaba sus propios sistemas y procedimientos según sus normas, y cuando esta ciencia empezó a expandirse la dispersión se reveló como un importante problema. Al ser los sistemas tan diferentes entre sí, los usuarios se veían obligados a rediseñar toda su estructura si querían cambiar de proveedor o, en ocasiones, incluso de modelo. Invertir en tecnología biométrica era un negocio cuanto menos arriesgado, algo a lo que no todos los empresarios podían optar. Y por el lado de los desarrolladores, innovar resultaba una tarea casi imposible. Esto dificultaba tanto el crecimiento del sector biométrico como el desarrollo de numerosos sistemas biométricos que ofrecían resultados prometedores pero a precios elevados.

La industria biométrica decidió solucionar el problema mediante el desarrollo de varios estándares que sirvieran de base para las futuras investigaciones reduciendo los riesgos económicos antes mencionados. En 2001 el ANSI (American National Standards Institute) desarrolló el estándar ANSI X.9.84 para definir las condiciones de los sistemas biométricos para la industria de servicios financieros en todo lo referente a la seguridad en torno a la transmisión y almacenamiento de datos. En 2002, ANSI y el consorcio BioAPI presentan el estándar ANSI / INCITS 358 para garantizar la interoperabilidad entre sistemas. En 2006, el NIST, en cooperación con el FBI, crean el estándar PIV-071006 que establece los criterios de calidad que deben cumplir los dispositivos de captura de huellas dactilares para poder ser usados en agencias federales estadounidenses.

Internacionalmente se creó, a partir de 2003, el ISO/IEC JTC1/SC37, encargado de normalizar los distintos aspectos del Reconocimiento Biométrico: desde la terminología, hasta los aspectos sociales y jurídicos, pasando por los formatos de datos, las metodologías de evaluación y los interfaces de programación de aplicaciones (APIs). Estructurado en 6 grupos de trabajos (working groups – WG), es precisamente el WG3 el que se encargó de internacionalizar el estándar BioAPI bajo el número ISO/IEC 19784-1, el cual estuvo basado en la primera norma de ANSI, pero que evolucionó en lo que coloquialmente se conoció como BioAPI 2.0. Posteriormente esa norma se fue evolucionando añadiendo más contenido a la norma, así como nuevas partes de la misma. En la actualidad se está procediendo a unificar todo el trabajo previo, para crear el que en el futuro cercano sea conocido coloquialmente como BioAPI 3.0.

Dentro del ámbito de dicho WG3, se planteó la necesidad de tener una implementación de BioAPI en lenguajes de orientación a objetos, e incluso que pudiese admitir varios lenguajes. De ahí surgió el proyecto de norma ISO/IEC 30106, que en la actualidad se encuentra en pleno desarrollo. Dicha norma está dividida en 3 partes, siendo la primera una especificación en lenguaje UML, la segunda la implementación en lenguaje Java, y la tercera la implementación en lenguaje C#. En el próximo capítulo se profundizará más en esta temática.

En definitiva, a lo largo de los años surgen numerosos estándares con diversas finalidades, buscando facilitar el crecimiento de la industria biométrica y el desarrollo de nuevos sistemas, métodos y aplicaciones. Sin embargo, y a pesar de la buena intención, a día de hoy la estandarización biométrica sigue siendo deficiente comparada con la de otras tecnologías y es un campo de trabajo dentro del sector de la biometría en el que aún queda mucho por avanzar.

3. TECNOLOGÍAS ASOCIADAS

En este apartado se va a realizar una introducción a las tecnologías que han sido empleadas en la realización de este trabajo. Primero se hablará del lenguaje de programación Java y su adaptación para el entorno Android y del entorno de desarrollo Eclipse. Tras esto, y usando de base los conceptos explicados en el apartado anterior sobre biometría, sus técnicas y el problema de la estandarización, se explicarán al lector la importancia y las especificaciones técnicas del estándar BioAPI, tanto en su versión completa como en la de Framework Free.

3.1. Java

Java es un lenguaje de programación desarrollado en 1995 por James Gosling. Es del tipo de programación orientada a objetos, es decir, aquella que usa interacciones entre entidades con un determinado estado, comportamiento e identidad, llamadas objetos, para crear programas informáticos.

Su sintaxis está basada en gran medida en C, Cobol y Visual Basic, pero elimina herramientas de bajo nivel, como la manipulación directa de punteros o la liberación de memoria. Esto reduce muchos de los errores que tienen lugar en los lenguajes de los que parte y lo hace más simple y sencillo de usar.

Pero, independientemente de su sencillez, lo que hace especial al lenguaje Java es su filosofía de independencia. Los creadores pretendían que un programa escrito en Java pudiera ejecutarse sin modificaciones en cualquier tipo de hardware, y así lo indicaron con el axioma *"Write once, run anywhere"*. Para conseguirlo, el código fuente se compila para obtener unas instrucciones simplificadas, conocidas como bytecode. Esta simplificación se ejecuta en una máquina virtual, escrita en código nativo de la plataforma destino, que interpreta y ejecuta el código. Como contrapartida, esta solución aumenta el tiempo de reacción, y aunque se ha mejorado mucho en este sentido desde la primera implementación, Java sigue siendo considerado un lenguaje más lento que otros.

3.2. Android

3.2.1. Historia

La historia de Android comienza en 2005, año en que Google compró la firma Android Inc, una empresa que desarrollaba software para dispositivos móviles. A finales de 2007 se anunció oficialmente la existencia de este SO, una plataforma para dispositivos móviles construida sobre la versión 2.6 del kernel de Linux. Este anuncio vino

acompañado de la creación de la Open Handset Alliance (OHA), una alianza de 78 compañías de software, hardware y telecomunicaciones, entre ellas auténticos gigantes en sus respectivos campos como Ebay o Google, que nació con el objetivo de desarrollar estándares abiertos para dispositivos móviles.

3.2.2. El código abierto

Desde antes de su primer lanzamiento Android levantó una gran expectación, debido a lo innovador que era su proyecto y a la inmensa presencia de sus desarrolladores en el campo de las nuevas tecnologías. Antes de profundizar en los aspectos técnicos de este SO, es recomendable explicar brevemente el motivo por el cual fue tan rompedor.

Los primeros teléfonos móviles tenían la función de permitir la comunicación entre cualquier punto del planeta, sin las limitaciones del cableado presentes en la telefonía fija. Con el tiempo, la evolución de estos dispositivos hizo necesario el desarrollo de sistemas operativos cada vez más complejos y eficientes. BlackBerry, desarrollado por RIM, fue uno de los primeros SO que fue más allá de las funciones típicas de SMS y llamadas de voz, revolucionando el mercado al implantar en un móvil funciones hasta entonces solo asociadas a las PDA's. Tras éste llegaron otros similares, como PalmOS o Windows, pero lo que realmente marcó el punto de inflexión fue la llegada del iPhone de Apple, que funcionaba con el SO iOS y popularizó, entre otras cosas, el uso de pantallas táctiles en móviles. Todos estos SO eran similares en el hecho de que las aplicaciones desarrolladas en ellos por personas ajenas a las compañías eran dependientes de la firma del software para su implantación. En este momento es cuando Android, un SO libre y de código abierto, irrumpe en el mercado de la mano de Google, permitiendo a terceros desarrollar sus propias aplicaciones y destruyendo con ello la barrera existente al respecto.

Es decir que, sin restar importancia a la fuerza que Google tenía en esos momentos y que transmitía a todos sus productos, el detalle que llamaba la atención de Android era la facilidad que daba a cualquier persona para aprender, idear y desarrollar sus propias aplicaciones, sin tener que salvar los múltiples inconvenientes que proporcionaban otras firmas. Aunque esta filosofía ya existía previamente, por ejemplo con el SO Linux, nunca había sido aplicada a dispositivos móviles.

Para responder a la inevitable pregunta de por qué es tan sencillo programar en Android, será necesario hacer una breve exposición de lo que representa el código abierto.

Como ya se ha comentado antes Android es un SO abierto desarrollado por una compañía que pretende crear estándares abiertos. Tras cada actualización, el código es liberado por la empresa desarrolladora. Dicha liberación se realiza bajo licencia Apache, lo que permite a los desarrolladores la libertad de usarlo para cualquier propósito, distribuirlo, modificarlo y lanzar versiones modificadas de ese software. Este hecho por

sí solo ya representó una revolución, pero hay que tener en cuenta que, además, Android tiene disponibles herramientas para la programación en los tres principales sistemas operativos para ordenadores -Windows, MacOS y Linux- que pueden descargarse gratuitamente desde la página oficial. Por otro lado, la creación de aplicaciones se hace utilizando un entorno de programación libre (Eclipse) y mediante el uso de un lenguaje común y extendido de programación (Java). Como comparación, su principal competidor, iOS, utiliza un lenguaje de programación y un entorno de desarrollo propio para sus aplicaciones, y solo facilita herramientas para su sistema operativo, MacOS, aunque actualmente existen máquinas virtuales que permiten el desarrollo en otras plataformas, pero que suponen una ralentización tan grande del proceso de trabajo, que lo hace inviable.

No es de extrañar que en poco tiempo Android se haya convertido en uno de los SO para dispositivos móviles preferidos por los desarrolladores, ni que haya conseguido una enorme cantidad de aplicaciones, muchas de las cuales son gratuitas.

3.2.3. Estructura

La estructura de Android se compone de aplicaciones ejecutadas en un framework Java, orientadas a objetos sobre el núcleo de las bibliotecas de Java en una máquina virtual Dalvik con compilación en tiempo de ejecución. El código, de más de 12 millones de líneas, contiene partes en Java, en XML, en C y en C++.

Los componentes principales de Android se pueden ver en la Figura 11[13].

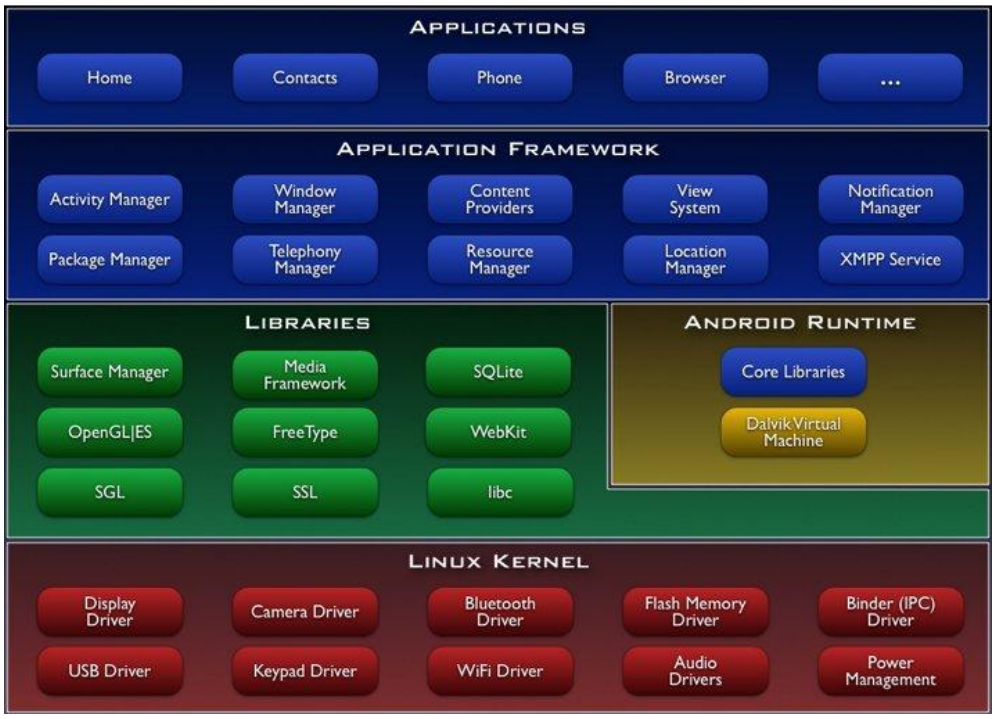


Figura 11. Arquitectura de Android

El nivel más alto lo forman las aplicaciones, escritas sin excepción en lenguaje Java. Incluyen, entre otras, un cliente de correo electrónico, SMS, agenda o calendario.

El siguiente nivel es el marco de trabajo de aplicaciones o application framework. Proporciona acceso completo a todos los APIs usados por las aplicaciones base. La arquitectura está diseñada para simplificar la reutilización de componentes: las capacidades de una aplicación pueden ser reutilizadas por otra, permitiendo el reemplazo de los componentes por parte del usuario.

En el nivel inmediato inferior se encuentran las bibliotecas. En Android se incluyen una serie de bibliotecas de C/C++, accesibles a través del framework. Dichas bibliotecas incluyen, entre otras, un administrador de interfaz gráfica, un framework OpenCore para poder hacer uso de audio y vídeo, una base de datos relacional SQLite que nos permite tener una base de datos local con la que gestionar todos los datos de nuestra aplicación, una API gráfica OpenGL (pensada principalmente para el uso en juegos 3D), un motor gráfico SGL que puede usarse para manejar gráficos en 2D, tanto para juegos como para la interfaz, un motor de renderizado WebKit con el que podemos integrar el motor de renderización web que usan Safari, Chrome y el propio Android, una librería de renderizado de textos (FreeType) y una biblioteca estándar de C, Bionic.

Al mismo nivel que las bibliotecas está el runtime de Android. Es un set de bibliotecas núcleo que proporcionan prácticamente todas las funciones disponibles en las bibliotecas base de Java. Cada aplicación Android corre en su propio proceso y en su propia instancia de la máquina virtual Dalvik, que puede ejecutar eficientemente varias instancias de forma simultánea. La máquina virtual ejecuta clases compiladas desde Java que han sido transformadas a un formato .dex (Dalvik executable), formato optimizado para memoria mínima, y se apoya en el kernel de Linux para funciones a bajo nivel.

En la base de la arquitectura de Android se encuentra el núcleo Linux, del que depende para los servicios base del sistema como seguridad, gestión de memoria, gestión de procesos, pila de red o modelo de controladores y que actúa como una capa de abstracción entre el hardware y el resto de la pila de software[14].

3.3. **Eclipse**

Eclipse es un Entorno de Desarrollo Integrado (IDE) multiplataforma de código abierto. Fue desarrollado en 2001 por IBM, pero en 2003 se creó la Fundación Eclipse con el fin de garantizar el desarrollo futuro de Eclipse en código abierto y este consorcio ha sido el encargado de su desarrollo desde entonces. Los IDE son programas informáticos compuestos por un conjunto de herramientas de programación que permiten a los desarrolladores escribir, compilar, probar mediante una interfaz gráfica y depurar sus programas de una forma rápida y sencilla.

Eclipse, en su condición de IDE multiplataforma, proporciona soporte para diversos lenguajes de programación, como C, C++, Cobol, Fortran, PHP o Python, pero su uso más extendido es con Java. Una característica de este entorno es que las funcionalidades no están implantadas de forma fija, se van añadiendo mediante módulos (plugins) según las necesidades del usuario, método que previene una posible sobrecarga en exceso y sin necesidad del entorno de desarrollo, o de los recursos del ordenador

En el presente trabajo se ha utilizado Eclipse en su versión de IDE Android. Para poder hacer esto, ha sido necesario integrar el plugin ADT (Android Development Tools), que permite el uso de las herramientas y métodos básicos para el desarrollo de aplicaciones Android.

3.4. BioAPI

BioAPI es un estándar para aplicaciones biométricas que define una Interfaz de Programación de Aplicaciones o API (Application Programming Interface) y una Interfaz de Proveedores de Servicios o SPI (Service Provider Interface) a través de las cuales puedan comunicarse dispositivos de diferentes proveedores, permitiendo de este modo la creación de sistemas biométricos más complejos mediante la interconexión de diferentes componentes.

3.4.1. Historia

El consorcio BioAPI se crea en 1998 formado por las empresas Bioscrypt, Compaq, Iridiam, Infineon, NIST, Saflink y Unisis, y contando con el apoyo de algunas de las compañías informáticas de más peso como Hewlett-Packard o IBM. Su finalidad es crear un estándar biométrico que solucione el problema de la estandarización (ver apartado 2.6).

El propósito de cualquier estándar es crear una base común a partir de la cual se puedan crear distintas entidades que puedan operar entre sí y que sean intercambiables, sin sufrir problemas de conexiones defectuosas. Esto incrementa la competición entre proveedores a la vez que reduce el riesgo para el consumidor. En el caso de BioAPI, el objetivo principal es crear una API (Interfaz de Programación de Aplicaciones) y una SPI (Interfaz de Proveedores de Servicios) comunes que definan el modo en el que los programadores de aplicaciones y los proveedores de soluciones biométricas deben diseñar sus programas.

Su primera especificación vio la luz en el año 2000. Proporcionaba compatibilidad con otros estándares que estaban surgiendo en el momento, como Human Authentication API (HA-API), ya que todas las compañías trabajaban con el interés común de facilitar la implementación, adopción y compatibilidad de las tecnologías biométricas. Era

completamente independiente del sistema operativo usado y de los datos biométricos que se tomaban de muestra.

Entre sus diversos beneficios, BioAPI permite el rápido desarrollo de aplicaciones que usen la biometría, ya que para diseñarlas solo hay que seguir la estructura predeterminada, sin necesidad de idear una propia. Además, la posibilidad de implementación en todos los sistemas operativos y en múltiples plataformas biométricas permite una mayor variedad de alternativas y métodos. Proporciona interfaces sencillas para las aplicaciones y métodos estándar tanto de acceso a funciones, algoritmos biométricos y dispositivos, como de gestión de los datos obtenidos y de los tipos de tecnología. También da acceso a los desarrolladores a métodos sólidos y seguros de almacenamiento de datos y apoyo para la verificación e identificación biométricas en entornos de computación distribuida. Por su parte, el framework de BioAPI permite que las aplicaciones operen interconectadas con varios métodos biométricos, pudiendo elegir en cada momento cuál de ellos usar.

3.4.2. Arquitectura

Un sistema BioAPI completo está estructurado por capas siguiendo un modelo API/SPI, como puede verse en la Figura 12[15]. El nivel inferior está formado por los dispositivos biométricos. En el siguiente escalón están situados los Proveedores de Servicios Biométricos o BSPs (Biometric Service Providers), que proporcionan dichos servicios a las aplicaciones a través de unas determinadas interfaces (SPI y API), bien mediante la gestión directa de una Unidad de BioAPI o bien mediante Proveedores de Funciones de BioAPI o BFPs (BioAPI Function Providers). En el siguiente nivel se encuentra el Framework, que se encarga de la comunicación entre las aplicaciones biométricas y los BSPs. El Framework se comunica con los BSPs, situados un nivel más abajo, mediante la interfaz SPI y con las aplicaciones, situadas un nivel más arriba, mediante la interfaz API.

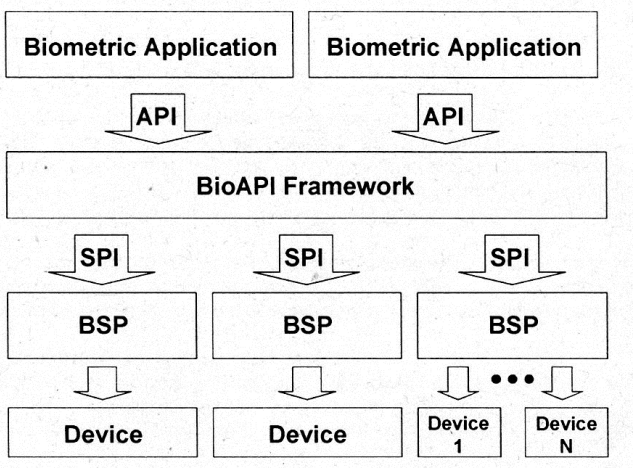


Figura 12. Arquitectura de BioAPI

Las aplicaciones, situadas en el nivel más alto de la arquitectura, están escritas de modo que invoquen las funciones a través del API, funciones a las que el Framework proporciona soporte. A continuación, el Framework invoca las funciones a través de la especificación SPI y el BSP proporciona soporte a las mismas.

BFPs y Unidades

Las Unidades de BioAPI son los bloques de construcción básicos que presentan los BSPs a las aplicaciones. Son, por decirlo de alguna manera, la idea abstracta de los dispositivos biométricos. Se encargan de encapsular y ocultar recursos de software y hardware, como sensores o archivos.

Cada Unidad de BioAPI modela o encapsula como máximo una pieza del hardware con su software correspondiente. Se clasifican en cuatro categorías, a saber:

- Unidad de sensor (Sensor Unit)
- Unidad de archivo (Archive Unit)
- Unidad de algoritmo de equiparación (Matching algorithm Unit)
- Unidad de algoritmo de procesamiento (Processing algorithm Unit)

Una Unidad de BioAPI puede ser gestionada directamente por un BSP o, de forma indirecta, a través de un BFP asociado a dicha Unidad. Cuando un BFP está creando la asociación temporal entre un conjunto de Unidades y un BSP (lo que se conoce como attach session) no puede asociar más de una Unidad de cada categoría. La clasificación de los BFP se realiza en función de las Unidades que pueden gestionar, siendo las categorías las siguientes:

- BFP de sensor (Sensor BFP)
- BFP de archivo (Archive BFP)
- BFP de algoritmo de equiparación (Matching algorithm BFP)
- BFP de algoritmo de procesamiento (Processing algorithm BFP)

La comunicación entre los BFPs y las Unidades durante la attach session se realiza mediante una interfaz adicional (que no se encuentra representada en la Figura 12) que se conoce como Interfaz de Provisión de Funciones o FPI (Function Provider Interface).

BIR

El Registro de Identificación Biométrica o BIR (Biometric Identification Record) es una estructura de datos que hace referencia a cualquier dato biométrico que es devuelto a la aplicación, ya sea "crudo" (en formato original), intermedio (pre-procesado) o procesado. Este elemento se crea con varias finalidades: facilitar el intercambio biométrico de datos entre diferentes componentes o sistemas, simplificar el software y el proceso de integración del mismo en el hardware, promover la interoperabilidad de los programas biométricos y garantizar la compatibilidad hacia delante de los sistemas

biométricos con los futuros adelantos tecnológicos, siempre que estos también continúen respetando el estándar.

El BIR es una instanciación de un objeto CBEFF (Common Biometric Exchange Formats Framework), cuya estructura (ver Figura 13) consta de una cabecera, la SBH (Standard Biometric Header), un bloque de datos o BDB (Biometric Data Block) y por último, de modo opcional, un bloque de seguridad o SB (Security Block). Cada una de estas partes contiene campos con información detallada sobre el archivo CBEFF, aunque solo algunos de ellos son obligatorios.

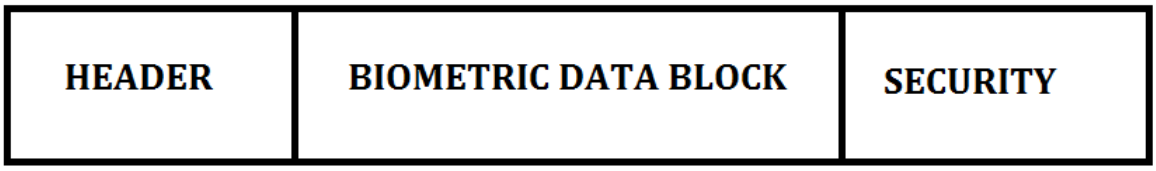


Figura 13. Estructura de un objeto CBEFF

El SBH contiene información que describe el contenido del BDB, y consta de tres campos obligatorios (SBH Security Options, BDB Format Owner y BDB Format Type) y diecisiete opcionales, entre los que se encuentran el Patron Header Version, el Biometric Type o el Biometric Data Quality, entre otros.

Por su parte, el BDB contiene las muestras biométricas e identifica detalladamente su formato según el campo Format ID del SBH y teniendo en cuenta los siguientes parámetros:

- Si es estándar o de propietario.
- Si es publicado o no publicado.
- Si es un dato "crudo" (directamente obtenido del sensor), intermedio (dato obtenido del procesamiento de un dato crudo, pero aún no dispuesto para la comparación) o procesado (dato ya completamente procesado, que puede compararse con la plantilla del usuario).
- Si es para reclutamiento, verificación o identificación.
- Si contiene una o más muestras.
- Si pertenece a un único tipo biométrico o a varios.
- Si es directo o encriptado, y si es con o sin signo.

Por último, el bloque SB contiene parámetros asociados con la firma y/o encriptación del BIR. Dentro de la arquitectura de BioAPI hay varias formas de garantizar la seguridad de los datos, siendo la original situarlos dentro de la propia estructura CBEFF, en el módulo de SB. Este campo contiene un algoritmo de identificación con todos los parámetros necesarios para crear un mecanismo de autenticación conocido como MAC (Message authentication code function). Este método solo asegura la integridad de los datos, pero el BDB puede ser encriptado para proporcionar además la

privacidad adecuada. En la versión 2.0 de BioAPI el Signature Block evoluciona a Security Block (manteniendo las siglas SB) y adquiere la capacidad de dirigir tanto la integridad como la privacidad de los datos.

El BIR hereda la estructura estándar del objeto CBEFF e inserta información dentro del SBH que hará posible que los dispositivos que implementan BioAPI sean capaces de interpretarlo. Su estructura se puede ver en la Figura 14[16]:

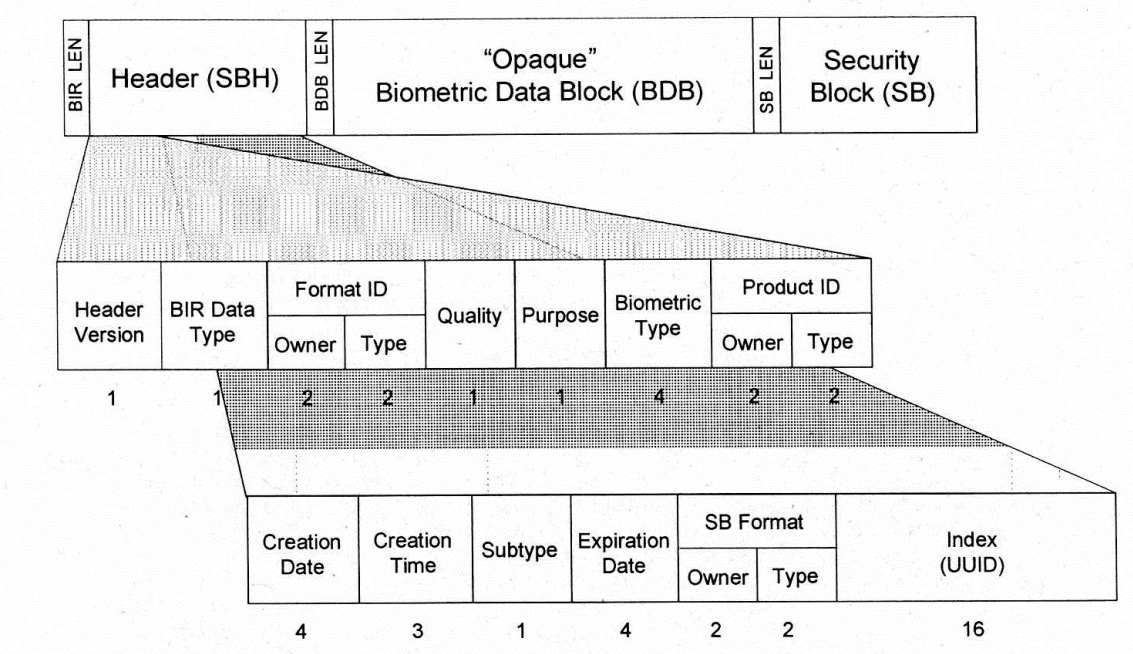


Figura 14. Estructura del BIR

Los campos "format owner" y "format type" contienen información única basada en una característica específica. El primero denota el proveedor, industria, grupo de trabajo o consorcio que define el formato del dato biométrico que contiene el BDB. Por otro lado, el valor del segundo campo indica el formato del BDB según lo especificado por el format owner. Mientras que el format owner debe respetar los estándares biométricos, el format type puede ser un formato de dato no publicado, propiedad de un proveedor determinado, u otro que sí siga los estándares de la industria.

Cuando un BSP crea un nuevo BIR, devuelve un parámetro que lo identifica, llamado controlador o "handle". La mayoría de las operaciones biométricas pueden realizarse sin extraer el BIR del BSP, lo cual favorece la rapidez ya que suele ser de gran tamaño. Pero si en algún caso es necesario gestionar el BIR independientemente (ya sea para almacenarlo en una base de datos, moverlo a otro BSP o enviarlo a un servidor para identificación o verificación) puede hacerse a través de dicho controlador.[17][18]

3.4.3. BioAPI Java

La arquitectura explicada en el apartado anterior es la aplicada en la única versión de BioAPI que se comercializa actualmente, la correspondiente al lenguaje de programación C. En este trabajo, el objetivo es crear una nueva versión de BioAPI para Android, y para ello se ha partido de una versión alternativa en Java y que, por tanto, ofrece más facilidades para el porte.

Sin embargo, la versión Java de este estándar biométrico no es una versión como tal. Es una primera aproximación, realizada por la Universidad de Purdue, que no está finalizada y cuya norma se encuentra actualmente en desarrollo (ISO/IEC 30106-2). A pesar de que su funcionamiento es correcto a un nivel básico, tiene una dificultad añadida, y es que no respeta la estructura por capas del BioAPI original.

El código del BioAPI C especifica, para cada función empleada y para cada porción del mismo, la capa a la que pertenece. El código del BioAPI Java no lo hace, elimina muchas funciones por ser innecesarias en programación orientada a objetos y combina otras. Debido a ello, y al no haber una normativa regulada donde se especifiquen las asignaciones, es muy complicado reconocer a primera vista las correspondencias entre funciones de C y Java, lo cual, como se verá más adelante, representa un importante inconveniente a la hora de implementar la arquitectura Framework Free.

El BioAPI Java está estructurado en cuatro grandes paquetes, a saber:

- Paquete org.bioapi: Contiene todas las funciones necesarias para fijar, gestionar y procesar los componentes y Unidades que interactúan con el Framework de BioAPI.
- Paquete org.bioapi.data: Contiene todas las estructuras de datos y modelos que describen interfaces para el paquete org.bioapi.
- Paquete org.bioapi.net: Contiene las interfaces que se usan en las operaciones de creación de redes.
- Paquete org.bioapi.template: Este paquete se crea para reducir la redundancia de código al escribir las aplicaciones. Ofrece a los desarrolladores un punto de inicio desde el que escribir su código, proporcionándoles directamente muchas de las interfaces que suelen ser implementadas de forma prácticamente idéntica en la mayoría de aplicaciones biométricas.[19]

3.5. BioAPI Framework Free

La arquitectura BioAPI Framework Free se crea a partir de la eliminación del Framework desde la arquitectura original de BioAPI. Este modelo permite la integración de BSPs que siguen la norma de BioAPI dentro de sistemas cuyo resto de componentes no están estandarizados, proporcionando la interfaz SPI (ya que, al no

haber dos niveles de comunicación, la API deja de ser necesaria). Los requisitos en este caso determinado se reducen a los mínimos imprescindibles para que el BSP tenga un funcionamiento fiable y el resto de componentes no necesitan cumplirlos. La arquitectura interna de los BSP que puede verse en la Figura 15[20] no cambia, aunque hay que tener en cuenta que la mayoría de BSPs destinados a un uso en sistemas con el modelo Framework Free son aplicaciones monolíticas de un solo proveedor.

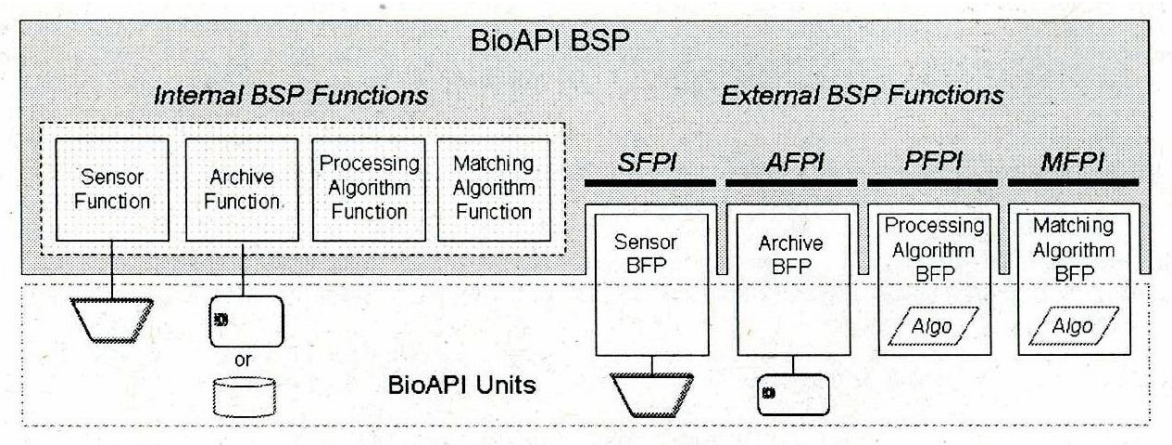


Figura 15. Arquitectura interna de un BSP

Para comprender bien las consecuencias de eliminar el Framework, hay que entender cuál es su función dentro de la arquitectura de BioAPI. En el apartado anterior se ha efectuado una aproximación, indicando que el Framework es la capa de BioAPI que permite la comunicación entre la aplicación biométrica y el BSP, mediante las interfaces API y SPI, respectivamente. Pero, yendo un poco más allá, se van a explicar a continuación algunas de las responsabilidades del Framework en un sistema biométrico: el registro de componentes y la carga de los BSP y asociación de los mismos con las Unidades de BioAPI.

El registro de componentes contiene información acerca de los BFPs y los BSPs instalados. En el modelo BioAPI hay un único Framework con un único registro de componentes asociado en cada sistema biométrico. En un sistema computacional real existirán múltiples registros de componentes apoyados en el mismo Framework. Para hacer esto posible, cada uno de ellos se modela como un sistema biométrico independiente dentro del sistema general.

Mediante el uso de unas determinadas funciones internas del Framework, una aplicación puede obtener información desde el registro de componentes acerca de los BSPs instalados, de los BFPs o del propio Framework. También puede preguntar a través de un SPI hacia un BSP concreto, en cuyo caso la información recibida es acerca de los BFPs instalados a los que proporciona soporte dicho BSP o de las Unidades de BioAPI que son accesibles desde el mismo.

La carga de los BSP y su posterior asociación con las Unidades de BioAPI es necesaria para que una aplicación pueda ejecutar operaciones biométricas y consta de una serie de pasos que están esquematizados en la Figura 16[21]. Para empezar, debe acceder al Framework y, a continuación, identificar y cargar uno o varios BSPs. Tras esto asocia los BSPs seleccionados con una o varias Unidades de BioAPI y establece una "attach session" para ese BSP teniendo en cuenta que puede haber como máximo una Unidad de BioAPI de cada categoría. Tras la asociación el resto de funciones pueden llamarse mediante una simple referencia al BSP usado, ya que las Unidades de BioAPI se encargarán de realizar la conexión necesaria para su funcionamiento.

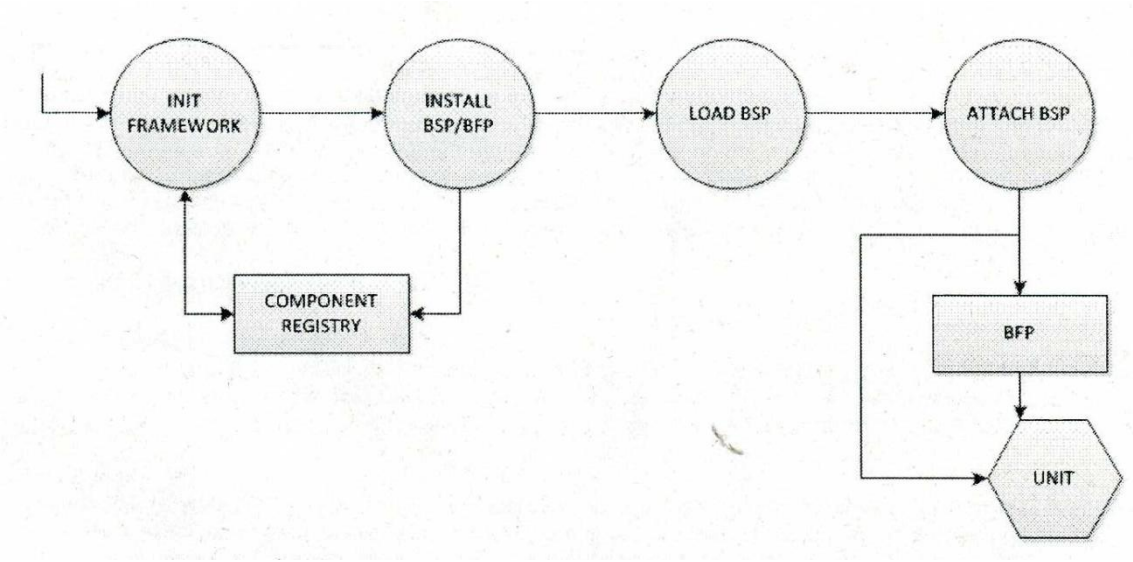


Figura 16. Pasos para la instalación de una aplicación biométrica

Estas dos son solo un ejemplo de las funcionalidades que pierde BioAPI eliminando su Framework; hay más, como por ejemplo la instalación y desinstalación de BFPs o BSPs. Dado que la arquitectura Framework Free está pensada para los sistemas biométricos más sencillos, muchas de estas funcionalidades pueden ser eliminadas directamente, ya que su ausencia no representará un problema al ejecutar la aplicación. Sin embargo, es posible que haya otras que deban ser implementadas en otras partes del código para evitar fallos.

En la Figura 17 se puede ver una idealización de la estructura que se intenta conseguir, organizada por capas. Aunque realmente el código no sigue esa estructura, es la forma más efectiva de comprender el significado de Framework Free.

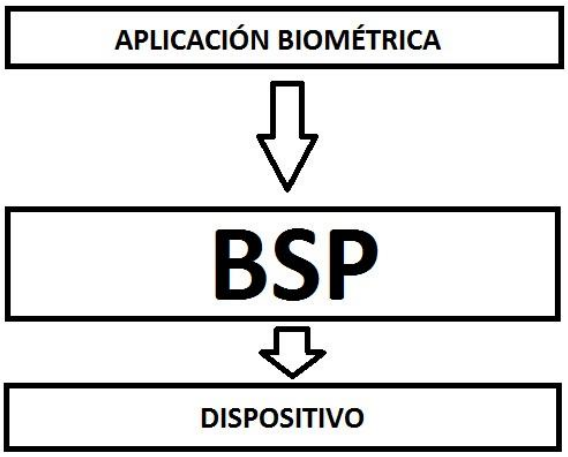


Figura 17. Estructura simplificada de BioAPI Framework Free

Como se ve, se ha eliminada la capa del Framework y por lo tanto una de las dos interfaces originales de BioAPI deja de ser necesaria, concretamente la API, realizándose todas las comunicaciones entre la aplicación y el BSP a través de la interfaz SPI. A niveles inferiores la arquitectura permanece igual que en la versión completa, y las comunicaciones entre BSP y dispositivo no se ven afectadas. A diferencia de la arquitectura general (ver Figura 12) sólo se ha representado un BSP y un dispositivo. Esto es debido a que BioAPI Framework Free está especialmente diseñado para sistemas sencillos, de un único componente, que necesitan un programa ligero para funcionar correctamente.[17]

3.5.1. Framework Free en BioAPI Java

Al no respetar la estructura por capas y no disponer de una normativa regulada que especifique qué funciones de BioAPI Java se corresponden con las de BioAPI C, es muy difícil saber qué partes del código conforman el Framework. Mirando en la normativa al respecto, aparece que el paquete org.bioapi contiene todas las funciones necesarias para fijar, gestionar y procesar los componentes y Unidades que interactúan con el Framework de BioAPI, mientras que el paquete org.bioapi.data contiene todas las estructuras de datos y modelos que describen interfaces para el paquete org.bioapi. En principio se podría pensar que hay que eliminar esos dos paquetes completos, pero es una simplificación de la realidad, ya que muchas de esas funciones no forman parte del Framework.

Localizar dichas funciones y eliminarlas, probando en cada momento las consecuencias que su desaparición provocaba en el funcionamiento del programa, ha sido el mayor inconveniente a la hora de realizar el presente trabajo, y se le dedicará una atención especial en los apartados posteriores.

4. DISEÑO DE LA SOLUCIÓN

En este apartado se va a analizar detalladamente el problema planteado, considerando las diversas soluciones que existen al mismo. Se indicará la solución escogida, haciendo un razonamiento de sus ventajas e inconvenientes frente a otras opciones de resolución. Finalmente, se describirá el diseño de dicha solución, tanto el código, mediante diagramas de flujo, como la interfaz gráfica.

4.1. Planteamiento del problema

La finalidad de este trabajo es adaptar el estándar BioAPI a plataformas Android, para poder diseñar sistemas biométricos en dispositivos móviles.

El hecho de proporcionar una plataforma móvil para sistemas biométricos podría suponer una mejora en muchas situaciones, sobre todo en lo referente a seguridad e identificación de personas por medio de organismos gubernamentales o privados.

Sin embargo habría que estudiar a fondo el marco legal existente, y en caso de que fuera necesario, añadirle las mejoras pertinentes, para evitar abusos legales cometidos mediante el uso de este sistema. En la Ley Orgánica 1/1992, de Protección Ciudadana, *"se regulan las condiciones en que los agentes de las Fuerzas y Cuerpos de Seguridad, siempre que ello fuese necesario para el ejercicio de las funciones de protección de la seguridad que les corresponden"*. El artículo 20 dice:

> *1. Los agentes de las Fuerzas y Cuerpos de Seguridad podrán requerir, en el ejercicio de sus funciones de indagación o prevención, la identificación de las personas y realizar las comprobaciones pertinentes en la vía pública o en el lugar donde se hubiere hecho el requerimiento, siempre que el conocimiento de la identidad de las personas requeridas fuere necesario para el ejercicio de las funciones de protección de la seguridad que a los agentes encomiendan la presente Ley y la Ley Orgánica de Fuerzas y Cuerpos de Seguridad.*
>
> *2. De no lograrse la identificación por cualquier medio, y cuando resulte necesario a los mismos fines del apartado anterior, los agentes, para impedir la comisión de un delito o falta, o al objeto de sancionar una infracción, podrán requerir a quienes no pudieran ser identificados a que les acompañen a dependencias próximas y que cuenten con medios adecuados para realizar las diligencias de identificación, a estos solos efectos y por el tiempo imprescindible.*
>
> *(...)*

> *4. En los casos de resistencia o negativa infundada a identificarse o a realizar voluntariamente las comprobaciones o prácticas de identificación, se estará a lo dispuesto en el Código Penal y en la Ley de Enjuiciamiento Criminal.*[22].

Actualmente el Documento Nacional de Identidad (DNI) se considera identificación suficiente para una persona. La identificación biométrica sería un método mucho más preciso y seguro, pero al tener unas connotaciones mucho más agresivas que la entrega del DNI, su uso debería quedar restringido para la aplicación en los casos más extremos, y no todos aquellos a los que hace referencia el apartado primero del artículo 20. Por otro lado, hay que tener en cuenta que sólo los agentes de los Cuerpos y Fuerzas de Seguridad del Estado tienen el derecho a exigir a una persona que se identifique, según la Ley Orgánica 15/1999, de 13 de diciembre, de Protección de Datos de Carácter Personal:

> *Artículo 6. Consentimiento del afectado.*
>
> *1. El tratamiento de los datos de carácter personal requerirá el consentimiento inequívoco del afectado, salvo que la ley disponga otra cosa.*
>
> *Artículo 44. Tipos de infracciones.*
>
> *Tratar datos de carácter personal sin recabar el consentimiento de las personas afectadas, cuando el mismo sea necesario conforme a lo dispuesto en esta Ley y sus disposiciones de desarrollo. (Infracción grave)*
>
> *Artículo 45. Tipo de sanciones.*
>
> *(...)*
>
> *2. Las infracciones graves serán sancionadas con multa de 40.001 a 300.000 euros*[22].

Por lo que el uso de métodos biométricos para la identificación por parte de una empresa privada sin consentimiento previo del usuario pasaría a ser un delito grave.

4.2. Planteamiento de la solución

Para llevar a cabo la adaptación de BioAPI en plataforma Android, lo primero que hay que hacer es redactar una nueva versión del código en ese sistema operativo, lo que se conoce en términos informáticos como “portar el código”.

A continuación hay dos caminos. Se puede eliminar el Framework del código para finalmente crear una aplicación que haga uso del mismo, lo que sería el método más tradicional, o se puede hacer a la inversa: crear la aplicación tras realizar la adaptación del código, y una vez comprobado que dicha aplicación funciona, eliminar el

Framework y se observa si la aplicación continúa funcionando. La principal ventaja de esta segunda estrategia es que permite la localización de errores a medida que se eliminan las funciones del Framework, que es la parte del trabajo más innovadora y complicada de implementar. De seguir el método tradicional, hasta no eliminar completamente la capa Framework no se comprobaría el funcionamiento, y en caso de que éste no fuera correcto habría que localizar los errores entre todo el código nuevo.

Por lo tanto, se debe crear una aplicación que haga uso de algunas de las funcionalidades del código portado, y tras ello, aplicarle los requisitos de la versión de Framework Free. Como dichos requisitos solo están indicados en la versión programada en lenguaje C, hay que encontrar la equivalencia de los mismos en Java. Simplificándolo hasta la mínima expresión, hay que quitarle a BioAPI la capa Framework, haciendo que la aplicación interactúe directamente con el BSP.

4.3. Diseño de la solución

Antes de comenzar con el desarrollo, es necesario diseñar la solución del problema. Mediante un análisis del diseño de la solución se busca llegar a aquella que es la óptima entre todas las posibles, buscando la que más eficientemente cumpla todos los requisitos e intentando que los alcance de una forma sencilla. Para ello, hay que valorar las ventajas y desventajas de cada una de las soluciones que se podrían llevar a cabo, escogiendo después la que mejor se adapte a los objetivos que se buscan.

Como se ha dicho en el apartado 4.2, el trabajo se puede diferenciar en tres grandes bloques: la adaptación del código de BioAPI de Java hacia Android, la creación de una aplicación en Android que haga uso del código adaptado y la eliminación de la capa del Framework de BioAPI. Para facilitar la lectura, se ha dividido el diseño de la solución en tres sub-apartados, siguiendo esa estructura.

4.3.1. Adaptación del código

No existe un método determinado para adaptar un código de un lenguaje de programación a otro. Al igual que ocurre con la traducción de textos, un porte depende del lenguaje inicial y del final, del tipo de programación que implementan (no es lo mismo hacerlo entre dos lenguajes de programación funcional, que entre un lenguaje de programación funcional y otro de programación lógica) y del tipo de código del que se quiere realizar la adaptación.

Cuando se quiere portar un código entre dos tipos diferentes de lenguaje de programación (por ejemplo al realizar la versión en Java de BioAPI desde el original en C, convirtiendo un programa escrito según el paradigma de la programación imperativa a la programación orientada a objetos) lo más habitual suele ser reescribir el código completamente. Se analizan sus funciones, sus métodos y sus capacidades y se estudia

el método más eficiente de imitar su comportamiento en la plataforma de destino. Se puede ver que es un método que deja muchos aspectos a la interpretación del programador que realiza el porte, y resulta en muchos casos largo y complicado, pero eficaz.

Algunas plataformas proporcionan otro método para realizar este tipo de conversiones, que es la posibilidad de implementar código nativo, escrito en el lenguaje de partida, directamente en la plataforma de destino. Por ejemplo, Android tiene el Kit de Desarrollo Nativo o NDK (Native Development Kit) que permite desarrollar aplicaciones en C o C++. Aunque el código usado sea diferente, las aplicaciones se compilan y ejecutan del mismo modo que al usar el SDK, por lo que el rendimiento no se ve afectado. Sin embargo, para la mayoría de aplicaciones el uso de esta herramienta no representa una ventaja, y desde la propia guía de desarrolladores de Android se recomienda evitar su empleo salvo que sea necesario utilizar alguna herramienta no incluida en el framework. Por lo tanto, este segundo método solo se usa para casos muy específicos, y la inmensa mayoría de desarrolladores optan por portar manualmente el código, ya que los beneficios que proporciona en materia de seguridad y robustez superan las desventajas de su complejidad[23].

Cuando lo que se quiere es portar un código entre lenguajes de programación que siguen el mismo paradigma, se usan métodos similares, pero mucho más simples. La reescritura del código se puede realizar manteniendo la misma estructura, sencillamente solucionando los errores de sintaxis que puedan aparecer y cambiando los apartados del código inicial que empleen herramientas inexistentes en la plataforma final. En ocasiones, cuando los lenguajes de programación inicial y final tienen la misma base, pueden no existir errores de sintaxis, con lo que el código funcionaría sin modificaciones. En cuanto a la conversión automática, existen muchos programas que convierten código de un lenguaje a otro, pero la dificultad de localizar los posibles errores usando este método, sumado a que entre lenguajes que comparten paradigma la conversión manual no es excesivamente complicada, hacen que, de nuevo, el primer método sea el más usado, dejando el segundo para casos con códigos simples y breves en los que la rapidez de la adaptación es un factor crítico.

En este trabajo se quiere portar un extenso código desde un lenguaje de programación orientada a objetos, Java, al SO Android, que está basado en Java. Esto hace que los errores de sintaxis que puedan surgir sean mínimos, ya que ambos lenguajes la comparten, aunque el hecho de que el código no es ni breve ni simple ya sería suficiente para desaconsejar el uso de cualquier programa automático.

Por tanto, la solución óptima es realizar una conversión manual, copiando el código sin modificaciones desde el original en Java y solucionando los errores producidos por el uso de herramientas Java inexistentes en Android.

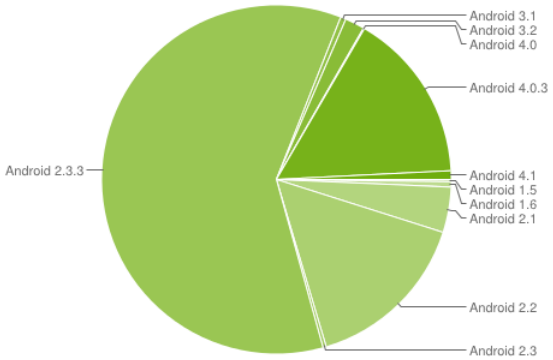
4.3.2. Aplicación móvil

El primer paso para la creación de una aplicación Android, antes incluso de comenzar a pensar en cómo programarla, es definir el mercado objetivo. Hay múltiples versiones de Android y a la hora de diseñar una aplicación es básico saber en cuál de ellas quiere implementarse, ya que a medida que van surgiendo actualizaciones, se crean nuevas funcionalidades que las anteriores no soportan. A continuación se listan las versiones existentes y algunas de sus características[24]:

- Versión 1.5, Cupcake: Lanzada en abril de 2009, incluye entre sus funciones un teclado QWERTY virtual, widgets, captura de vídeo o copiar y pegar.
- Versión 1.6, Donut: Lanzada en septiembre de 2009. Incluye navegación en Google Maps y búsquedas por voz.
- Versión 2.0, Eclair: Lanzada en diciembre de 2009, incluye zoom digital en la cámara de vídeo y fotos, un nuevo navegador de internet con soporte de vídeo y salvapantallas animados.
- Versión 2.2, Froyo: Lanzada en mayo de 2010. Mejora la gestión de la memoria con respecto a las versiones anteriores, siendo más veloz, y soporta Flash.
- Versión 2.3, Gingerbread: Lanzada en diciembre de 2010. Incluye mejoras de la gestión de la energía, del teclado virtual y soporte para pagos mediante NFC (Near Field Communication).
- Versión 3.0-3.4, Honey Comb: Lanzada en mayo de 2011, es una versión de Android especialmente diseñada para tablets.
- Versión 4.0, Ice Cream Sandwich: Lanzada a finales de 2011, para todo tipo de dispositivos móviles; incluye pantalla principal con imágenes 3D, barras de estado y la posibilidad de redimensionar los widgets. También proporciona soporte USB para teclados y controles para consolas.
- Versión 4.1, Jelly Bean: Lanzada en julio de 2012. Introduce una barra de notificaciones y mejora el rendimiento de los menús.

La versión Framework Free de BioAPI está pensada para operar en dispositivos de baja capacidad de memoria, es decir, aquellos con las versiones más antiguas de Android. El estudio de la cuota de mercado de las diferentes versiones de Android, con datos de agosto de 2012, que puede verse en la Figura 18[25], nos indica que Cupcake y Donut, las dos primeras versiones, apenas tienen peso, ya que sumando ambas no llegan a tener una presencia del 1%. Eclair, la versión 2.1, es la primera que aún mantiene cierta relevancia, ya que está presente en el 4.2% de los dispositivos móviles Android. Con el fin de llegar a la máxima cantidad de usuarios posible, se ha elegido esta última versión para realizar la aplicación, aprovechando la altísima compatibilidad hacia adelante que tiene Android; muchas de las aplicaciones diseñadas para versiones antiguas funcionan sin problemas en las nuevas, con lo que al elegir Eclair como versión principal garantizamos la compatibilidad con un 99.3% de los dispositivos Android. En cuanto al

idioma, se ha realizado en inglés, ya que es el más empleado por los desarrolladores Android y aquel en el que se encuentran la mayoría de tutoriales. De este modo se aseguraba la máxima difusión posible.



Version	Codename	API Level	Distribution
1.5	Cupcake	3	0.2%
1.6	Donut	4	0.5%
2.1	Eclair	7	4.2%
2.2	Froyo	8	15.5%
2.3 - 2.3.2	Gingerbread	9	0.3%
2.3.3 - 2.3.7		10	60.3%
3.1	Honeycomb	12	0.5%
3.2		13	1.8%
4.0 - 4.0.2	Ice Cream Sandwich	14	0.1%
4.0.3 - 4.0.4		15	15.8%
4.1	Jelly Bean	16	0.8%

Figura 18. Cuota de mercado de las distintas versiones Android (Agosto de 2012)

Tras elegir el mercado objetivo, el siguiente paso es definir la estructura que va a tener la aplicación. El objetivo de ésta es comprobar que el código portado funciona, por lo que debe hacer uso de BioAPI. Podría usarse una aplicación de comparación de huellas digitales o de iris, pero la instalación de dispositivos no forma parte de los objetivos del trabajo; lo más apropiado es buscar un método de comprobación lo más sencillo posible, como puede ser una captura de fotos, para poder implementarlo directamente en un smartphone, sin necesidad de instalarle dispositivos biométricos de captura de muestras adicionales.

La aplicación de captura de fotos tendría que acceder a la cámara del smartphone, hacer una captura de la cara del usuario para el reclutamiento y guardarla en la memoria. Tras esto, cuando un usuario quisiera acceder a la aplicación para verificar su identidad, ésta le haría una foto y la compararía con las muestras guardadas para así comprobar si se trata de una persona existente en el sistema. Según el resultado de la comparación, se mostraría un mensaje de aceptación o denegación y se regresaría a la actividad principal para permitir una nueva captura. El diagrama de flujo de esta opción se muestra en la Figura 19. Sin embargo, la creación de algoritmos de identificación, verificación o comparación tampoco forma parte del presente trabajo, y sin un algoritmo de comparación de rostro sería imposible realizar la implementación de esta aplicación.

Una posible solución sería no realizar el algoritmo de comparación, es decir: la aplicación toma la foto de muestra y la guarda como plantilla, y cuando el usuario quiera identificarse se le toma una foto pero la comparación con la base de datos se programa para que se permita el acceso siempre, como queda detallado en el diagrama

de flujo de la Figura 20. Aunque esto serviría para el propósito de comprobar el funcionamiento del código, ya que la foto pasaría por la aplicación en formato BIR, el resultado sería muy poco vistoso, y los fallos más difíciles de detectar.

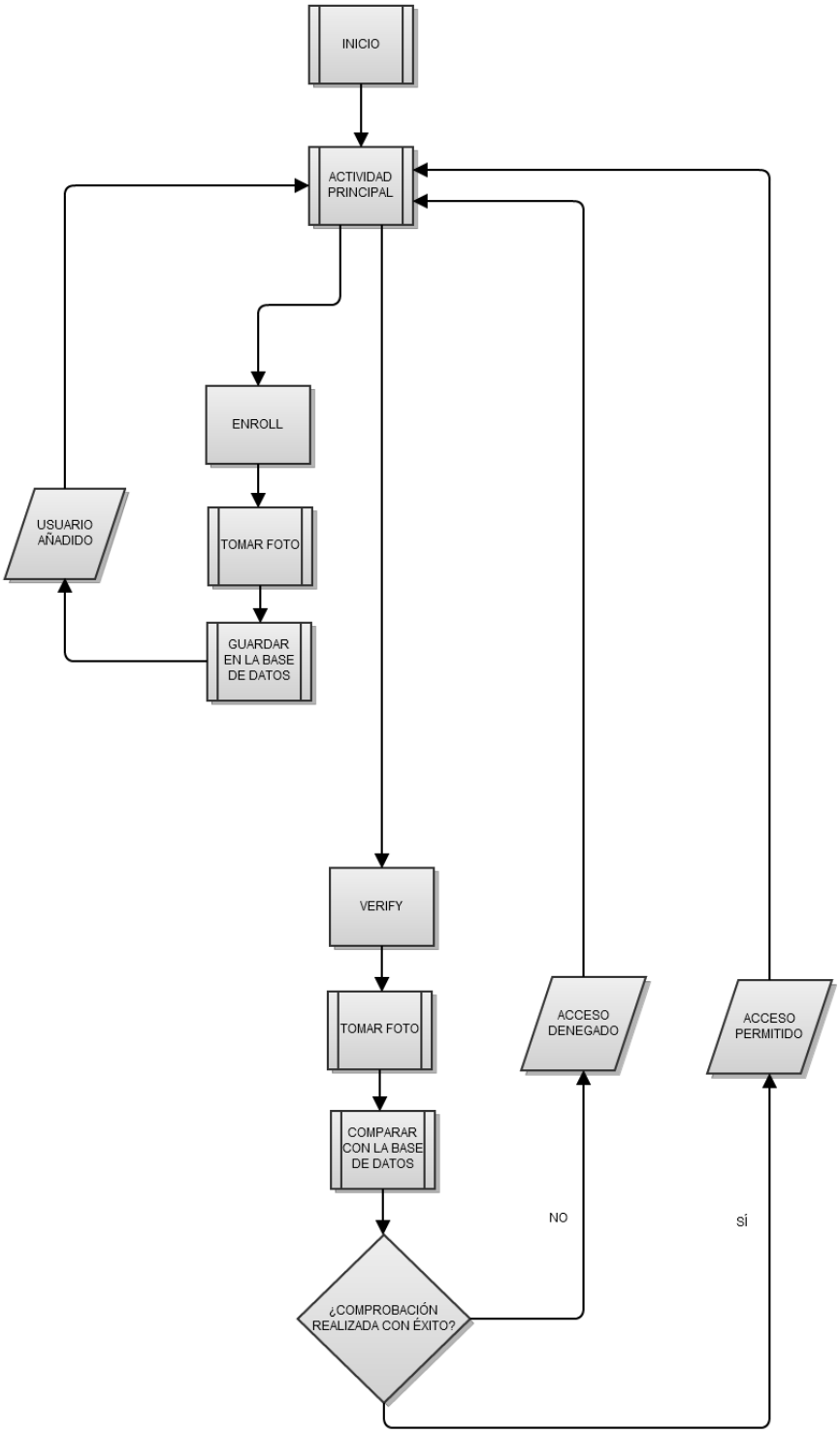


Figura 19. Diagrama de flujo de la aplicación de captura de fotos

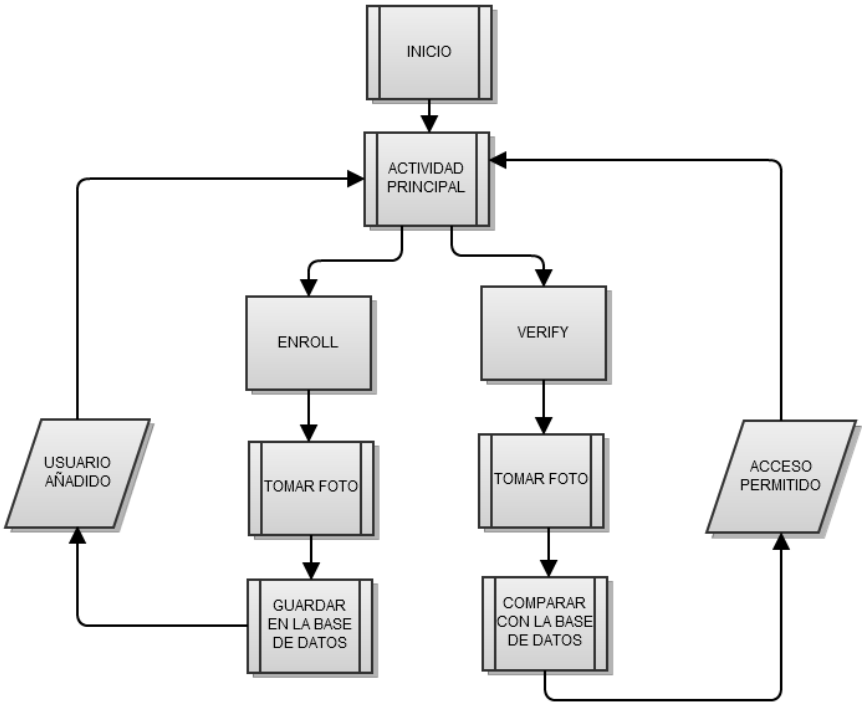


Figura 20. Diagrama de flujo de la aplicación de captura de fotos sin comparación

Una vez descartadas las dos opciones anteriores, se ha optado por una solución diferente: una aplicación de usuario y contraseña. Cuando el usuario quiera entrar en el sistema por primera vez, se le pedirá que introduzca su nombre de usuario y contraseña para el reclutamiento o "enroll". Aunque no sean biométricos propiamente dichos, la aplicación guardará estos datos en formato BIR, en una base de datos. Tras esto, el usuario podrá identificarse en cualquier momento reintroduciendo su usuario y contraseña para realizar la verificación o "verify". La aplicación comprobará si los datos introducidos existen en su base de datos, y en caso afirmativo mostrará un mensaje de acceso permitido. Si la contraseña no coincide, el mensaje mostrado será de error al intentar el acceso, y si el usuario no coincide con ninguno de la base de datos el mensaje indicará que dicho usuario no existe en el sistema. El diagrama de flujo de la aplicación se puede ver en la Figura 21 y como puede observarse es prácticamente idéntico al diagrama de la idea original de captura de fotos, sustituyendo la comparación de rostro por la de usuario y contraseña.

Como añadido, indicar que al principio se pensó en habilitar una funcionalidad extra para la aplicación que permitiría modificar la base de datos manualmente. Finalmente se desechó esta idea debido a la peligrosidad que suponía, ya que un usuario recién registrado podría acceder a la base de datos común y borrar todas las plantillas guardadas. Aunque no se ha implementado en la solución final, en el Apartado 8: Pruebas sí se ha utilizado esta funcionalidad para comprobar la puesta en marcha de la base de datos.

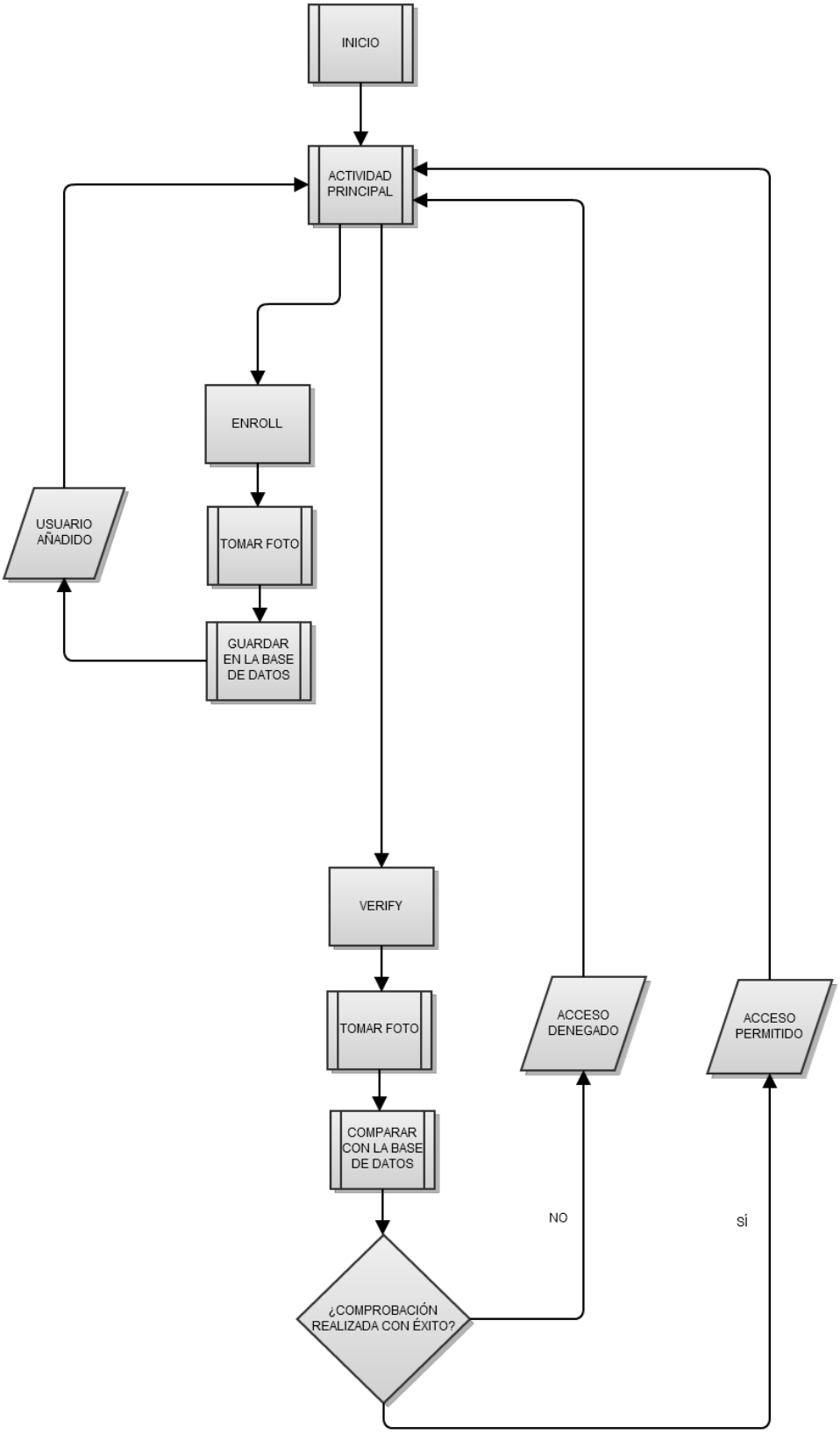


Figura 21. Diagrama de flujo de la aplicación de usuario y contraseña

Una vez definidas las funcionalidades de la aplicación y la estructura que ésta va a seguir, el último paso del diseño es la interfaz gráfica. La primera interfaz ideada constaba de cinco pantallas, como puede verse en la Figura 22 y en la Figura 23. Al iniciar la aplicación, se mostraba la pantalla de bienvenida, que solo tenía un campo de texto fijo con la leyenda "BioAPI Password Application", común a todas las demás pantallas, y un botón de inicio, el start. Al pulsarlo se pasaba a la pantalla de selección de opciones, que constaba de dos botones: uno, el botón enroll, para el reclutamiento, y el segundo, el botón verify, para la verificación. Ambos botones llevaban a la tercera pantalla, la de introducción de datos, que incluía dos campos de texto editable o edit texts, donde el usuario podía escribir su nombre y su contraseña. En esta misma pantalla había un solo botón, el de ejecutar, llamado simplemente Go!, que daba a la aplicación la orden de procesar los datos. Si la opción escogida había sido enroll y el usuario no existía previamente, se mostraba una pantalla final con el texto "Usuario introducido correctamente". Si, por el contrario, el usuario ya existía en el sistema, se mostraba otra con el texto "El usuario introducido ya existe". Por otro lado, si la opción escogida había sido verify y el usuario había metido sus datos correctamente, aparecería la pantalla final de éxito, con el texto "Usuario correcto. Acceso permitido". En el caso de que el usuario hubiera metido mal sus datos o simplemente no estuviera en el sistema, la pantalla final tendría el texto "Usuario incorrecto. Acceso denegado".

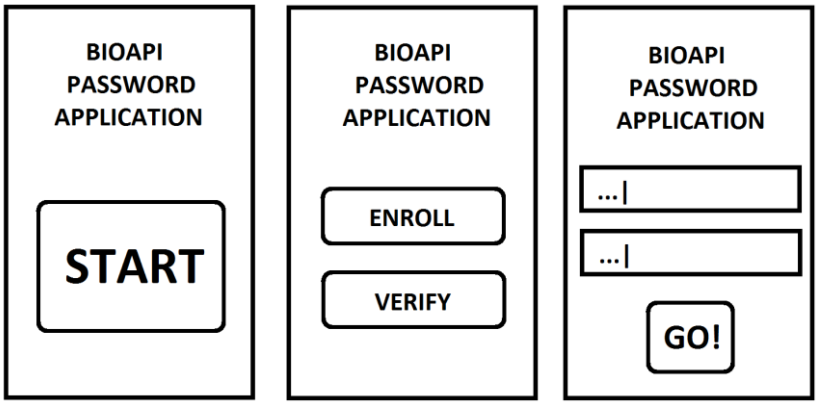


Figura 22. Pantallas de la aplicación de usuario y contraseña

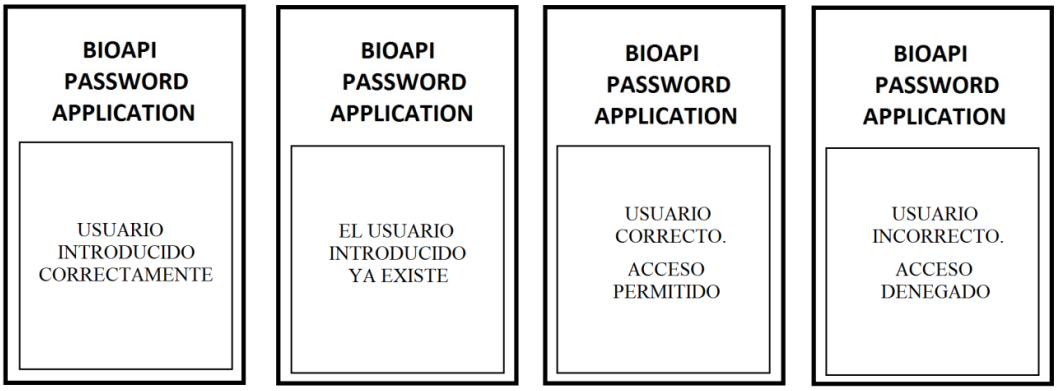


Figura 23. Pantallas finales de la aplicación de usuario y contraseña

Esta opción se descartó por la incomodidad que suponía tener que regresar desde la pantalla final, fuese cual fuese, hasta la inicial. Se pensó en añadir un botón de retorno, pero incluso de este modo era un programa pesado de usar. Por ello, se decidió sustituir las pantallas finales por notificaciones en mensajes sobreescritos, conocidas como toasts. Un toast es un mensaje que se muestra en un rectángulo encima de la pantalla durante unos segundos y con el que el usuario no puede interaccionar (ver Figura 24[26]). Son útiles para el caso que nos ocupa ya que no producen ningún cambio en la actividad que se está realizando, con lo que sería posible reclutar varios usuarios seguidos sin necesidad de estar volviendo a la pantalla principal tras cada introducción.

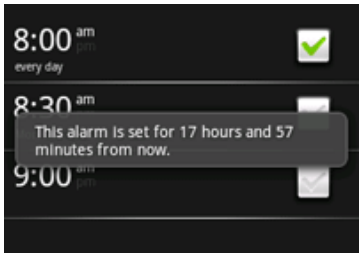


Figura 24. Toast

Además se decidió eliminar el mensaje de bienvenida ya que no aportaba nada al programa y fusionar en una sola la pantalla de selección de opciones y la de recogida de datos. El programa se condensaba así en una sola pantalla que aglomeraba todos los elementos necesarios: los campos de texto editables, indicando cual era para usuario y cuál para contraseña; los botones de selección de opción, que incluían en sus funciones la ejecución de la opción escogida (se eliminaba de este modo la necesidad del botón Go!) y, por último, los toasts, que aparecían tras el reclutamiento o verificación del usuario, ya fuera correcta o incorrecta. En la Figura 25 puede verse la esquematización de la interfaz gráfica de la aplicación que se hizo de forma previa y su equivalente final en el layout gráfico de Eclipse.

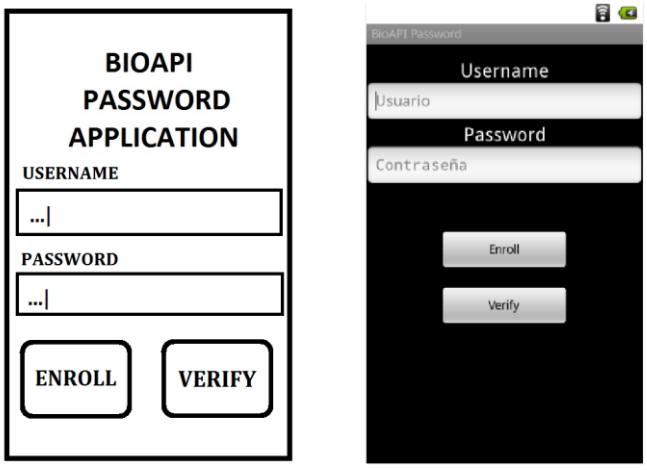


Figura 25. Interfaz gráfica final. Esquematización y simulación en Eclipse

4.3.1. Eliminación del Framework

La eliminación de la capa Framework ha sido el mayor obstáculo a la hora de diseñar la solución, como ya se ha mencionado en el apartado 3.5.1. Hay que tener en cuenta que este trabajo es la primera adaptación que se crea del estándar BioAPI en Framework Free no ya en Android, sino también en programación orientada a objetos, luego la dificultad aumenta al no disponer de ninguna base de la que partir. Incluso, de la versión en C, lo único que existe es la documentación del estándar, pero no existe actualmente ninguna implementación de esta versión de BioAPI.

Si bien en la norma ISO/IEC 19784-1 del BioAPI original de C se especifica claramente que funciones no son necesarias en la versión Framework Free y pueden eliminarse y cuales son absolutamente imprescindibles, en la norma (actualmente en desarrollo) ISO/IEC 30106-2 de BioAPI Java solo se realiza una breve descripción general de los paquetes, y ni se menciona la existencia de una versión Framework Free. Además, en BioAPI Java no se respeta la estructura por capas, lo que dificulta aún más la localización no solo de las funciones que no son necesarias en la versión Framework Free, sino de todas aquellas que forman parte del Framework. Debido a esta falta de recursos, la solución más lógica, que habría sido aplicar los requisitos de Framework Free según vienen en la norma al código adaptado, es completamente imposible de implementar.

Una vez descartada esta primera solución, hay dos caminos posibles. El primero es establecer una equivalencia entre el BioAPI C y el de Java, asignando a cada función en Java su equivalente en C. Para ello, habría que mirar en la norma de ambas versiones función a función, comparando parámetros de entrada y salida y las habilidades que proporcionan. Una vez obtenida la equivalencia en C de cada función existente en el BioAPI Java, se utilizaría la norma de C, que sí recoge los requisitos de la versión Framework Free, para saber si dichas funciones son prescindibles o necesarias. Toda esta información se recogería en una tabla como la de la Figura 26 y tras ello se procedería a la eliminación de las funciones superfluas.

Función en BioAPI C	Función en BioAPI Java	Prescindible en F. Free
Función_A	*Función_H*	Sí
Función_B	*Función_B*	No
Función_C	*Inexistente*	-
Función_D	*Función_E*	Sí
Función_E	*Función_E*	Sí
Función_F	*Función_R*	No

Figura 26. Ejemplo de equivalencias entre funciones de BioAPI C y BioAPI Java

Sin embargo, muchas de las funciones existentes en el BioAPI C se eliminaron en la versión de Java, otras se fusionaron, y la inmensa mayoría cambiaron los nombres de sus parámetros. Por ello, su identificación, que ya de por sí es difícil, se convierte en un trabajo prácticamente imposible de llevar a cabo. Además, la extensión del código convierte este trabajo en algo inabarcable teniendo en cuenta el tiempo del que se dispone.

El segundo camino es mucho menos metódico y más atrevido. Consiste en analizar las funciones del BioAPI Java, guiándose por las breves descripciones que proporciona su normativa, y, sabiendo la utilidad del Framework, localizar una por una aquellas que forman parte del mismo para, a continuación, eliminarlas. Como se menciona en el apartado 3.5.1, dentro del BioAPI Java, el paquete org.bioapi contiene todas las funciones necesarias para fijar, gestionar y procesar los componentes y Unidades que interactúan con el Framework de BioAPI, mientras que el paquete org.bioapi.data contiene todas las estructuras de datos y modelos que describen interfaces para el paquete org.bioapi. Intuitivamente puede suponerse que el equivalente a la capa Framework en el BioAPI C, y por tanto, las funciones a eliminar, se encontrarán en esos dos paquetes. A pesar de que conceptualmente esta opción es más difícil que la otra, la imposibilidad de llevar a cabo el primer camino ha hecho necesario elegirla.

5. DESARROLLO

En este apartado se van a explicar los detalles de la realización del trabajo. Una vez fijados los objetivos y el diseño final del mismo, ya puede llevarse a cabo su creación. A lo largo de las próximas páginas se resumirá el proceso de desarrollo, describiendo los métodos empleados, los problemas encontrados y las soluciones aplicadas. Al igual que en anteriores apartados el texto se ha dividido en cuatro módulos para facilitar la lectura: una introducción, donde se habla de los problemas que hicieron aparición antes incluso de comenzar el desarrollo y se describe la instalación de los programas necesarios para la realización del trabajo, y los respectivos desarrollos de la adaptación del código, de la aplicación móvil y de la eliminación del Framework.

5.1. Introducción

Antes de comenzar el desarrollo del trabajo, se decidió realizar una primera toma de contacto con el BioAPI Java. Para ello, tras descargarlo, se instaló el IDE NetBeans y MySQL Server, un gestor de bases de datos. Aunque podría haberse instalado directamente en Eclipse, se optó por respetar las recomendaciones sobre Entornos de Desarrollo de la Universidad de Purdue, desarrolladora de la versión Java de BioAPI[27]. A continuación se configuró la conexión de BioAPI con la base de datos siguiendo al pie de la letra las instrucciones proporcionadas por el desarrollador. Al realizar este paso, se observó un detalle que, si bien en el momento no pasaba de ser una anécdota, se reveló como un importante problema al crear la aplicación móvil: era necesario crear manualmente la base de datos, como se ve en la Figura 27, especificando las columnas, su tamaño y el tipo de datos a almacenar; la aplicación de prueba proporcionada por el desarrollador no implementaba la funcionalidad de crearla automáticamente al iniciarse.

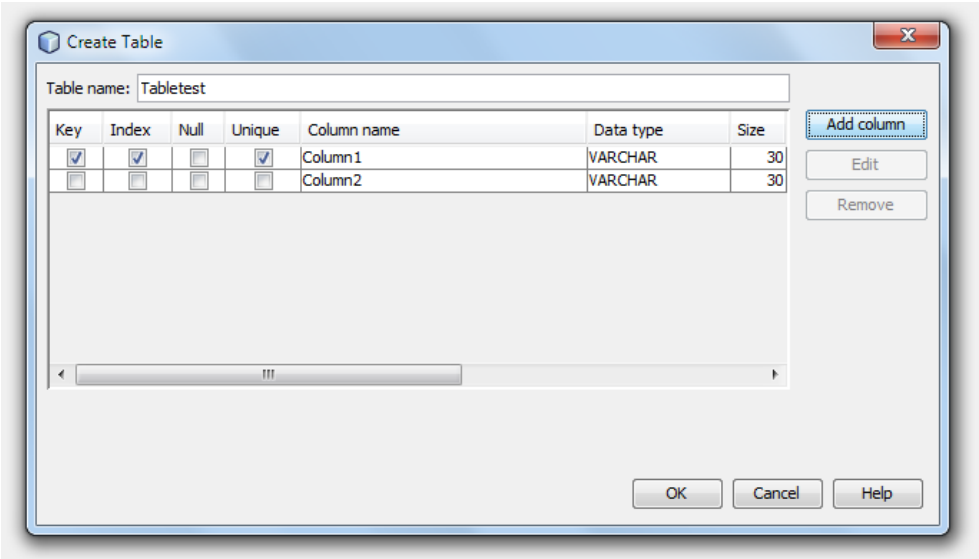


Figura 27. Creación de una base de datos en NetBeans

Al finalizar las instrucciones del fabricante, el código de BioAPI presentaba el aspecto de la Figura 28, organizado en los cuatro paquetes indicados en apartados anteriores: bioapi, data, net y template.

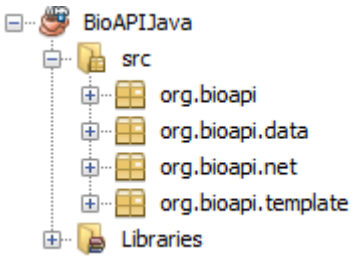


Figura 28. Estructura del BioAPI Java en NetBeans

Una vez instalado el BioAPI Java, se creó la base necesaria para el desarrollo del trabajo. Esto incluía la instalación del IDE Eclipse y la creación de un nuevo proyecto Android, en el que se llevaría a cabo todo el desarrollo posterior.

La instalación de Eclipse no representó ningún problema, al ser un IDE muy intuitivo y existir múltiples tutoriales al respecto. Al crear el nuevo proyecto hubo que escoger el API Android necesario, es decir, la versión objetivo. Siguiendo el razonamiento explicado en el apartado de diseño, se seleccionó la versión 2.1 Eclair, correspondiente con el API 7, como se puede ver en la Figura 29.

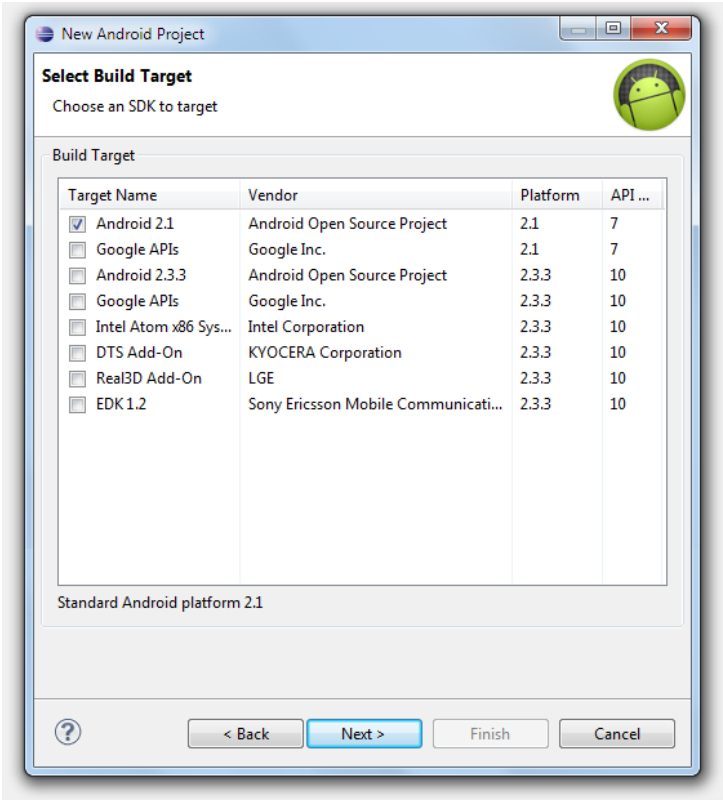


Figura 29. Selección de la versión Android

Además del API, había que elegir un nombre para el proyecto: BioAPI Android. También había que especificar el espacio de nombres (del inglés namespace), que sirve para identificar cada bloque; en el caso del BioAPI Java es "org.bioapi", y mediante añadidos (data, net o template) se identifican todos los paquetes. Se optó por imitar ese apartado y usar el mismo espacio de nombres. El resultado inicial, nada más crear el proyecto, es el de la Figura 30.

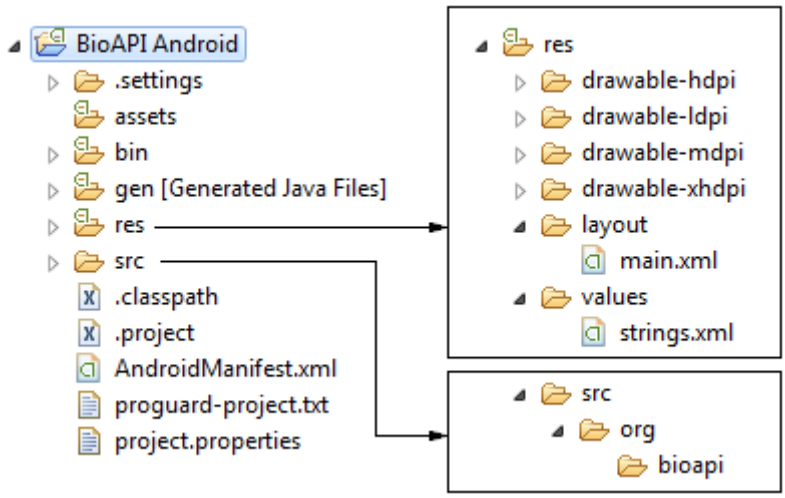


Figura 30. Estructura inicial del BioAPI Android en Eclipse

El último paso era la creación de un emulador. Aunque Eclipse permite realizar la depuración de las aplicaciones en un dispositivo externo, por comodidad se decidió usar un emulador en el propio ordenador para hacer las comprobaciones necesarias a medida que se programaba, y dejar la prueba en un móvil real para cuando se hubiese concluido el trabajo entero. Desde el AVD (Android Virtual Device Manager) se creó un nuevo emulador que operaba para la versión 2.1 de Android. Existe la opción de añadirle funcionalidades al mismo: cámara de fotos, acelerómetro, GPS… pero al no ser necesarias para la aplicación diseñada no fueron usadas. .

Una vez instalados todos los programas necesarios para el desarrollo del trabajo y creada la base del proyecto Android, se inició el proceso de creación.

5.2. Desarrollo del portado del código

Para el portado del código se había decidido realizar una copia literal del mismo y partir de ella para solucionar los problemas de sintaxis que pudieran aparecer.

Se creó el mismo número de clases que existían en el BioAPI Java, respetando los nombres usados en el mismo. A continuación, el código se transcribió literalmente, sin aplicarle ningún cambio. La consola de errores se muestra en la Figura 31, donde se puede ver como gracias a la base común de Java y Android, no existe ninguno a reparar,

tan sólo varias advertencias (en inglés warnings) que indican código que podría llegar a suponer un error, pero que con la implementación existente no representan un problema.

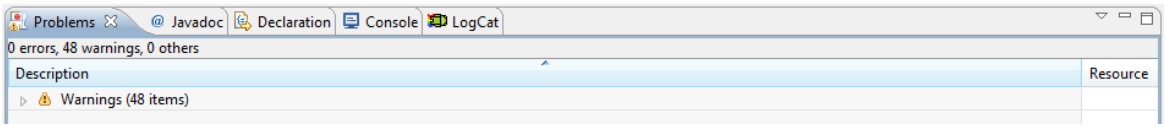


Figura 31. Errores encontrados al portar el código

Al no existir errores, teóricamente ya se podía iniciar la creación de la aplicación móvil, pero es conveniente comprobar primero las advertencias que muestra el programa, para evitar que posteriormente se conviertan en un problema. Al expandir la pestaña de advertencias, se observó que todas ellas eran del tipo "arreglo rápido" o *warning with quick fix*, lo que quiere decir que son comunes y el propio IDE puede solucionarlas automáticamente, sin necesidad de la intervención del desarrollador. Por otro lado, de las 48 advertencias existentes, 40 formaban parte de uno de los dos siguientes tipos y cuatro más dependían de ellos:

- *The import [*] is never used.*
- *[*] is a raw type. References to generic type [*] should be parameterized.*

Donde el símbolo [*] representa un parámetro determinado.

La primera de las dos advertencias indica que en una determinada clase se importa algo (un método, acceso a las funciones de otra clase, una herramienta, etc.) que luego no se emplea en ninguna parte del código de esa clase. Esto no representa ningún problema, ya que lo que significa es que la llamada no tiene ningún efecto y por lo tanto no repercute en el programa. Es un aviso al desarrollador para que elimine la importación inútil, con vistas a una mejor comprensión por parte de otro desarrollador diferente o por la suya propia.

En cuanto a la segunda advertencia, dice literalmente: *[*] es un tipo crudo. Las referencias al tipo genérico [*] deben hacerse mediante parámetros.* Es decir, que es necesario indicar que tipo de objetos van a almacenarse dentro de [*]. Por mostrarlo con un ejemplo concreto, la declaración:

```
static Hashtable formats=null;
```

Provoca la advertencia:

```
Hashtable is a raw type. References to generic type Hashtable<K,V>
should be parameterized.
```

Para solucionar este caso, es necesario indicar al programa qué tipo de datos van a almacenarse dentro de Hashtable. El tipo hashtable genérico está formado por dos parámetros: Key y Value. Habría que mirar en el programa qué tipo de dato son ambos

y luego inicializarlos, es decir, si Key es un entero y Value una cadena, habría que escribir algo como:

```
static HashTable<Integer, String> formats=null;
```

Sin embargo, como la advertencia es de arreglo rápido, es más sencillo situarse encima y marcar la opción que recomienda el IDE, en este caso *"Add type arguments to Hashtable"*. Al hacer click, directamente cambia la declaración de Hashtable por:

```
static Hashtable<UUID, OwnerTypePair> formats=null;
```

Es decir, encuentra automáticamente el tipo de dato al que pertenecen Key y Value, en este caso los datos personalizados Universally Unique Identifier (UUID) y OwnerTypePair respectivamente, e inicializa Hashtable según esa información.

Tras comprobar que ninguna de las advertencias señaladas representa un problema grave para la futura aplicación, ya que todas ellas tienen una solución sencilla, se dio por finalizado el portado del código de BioAPI de Java hacia Android, resultando el conjunto de la Figura 32.

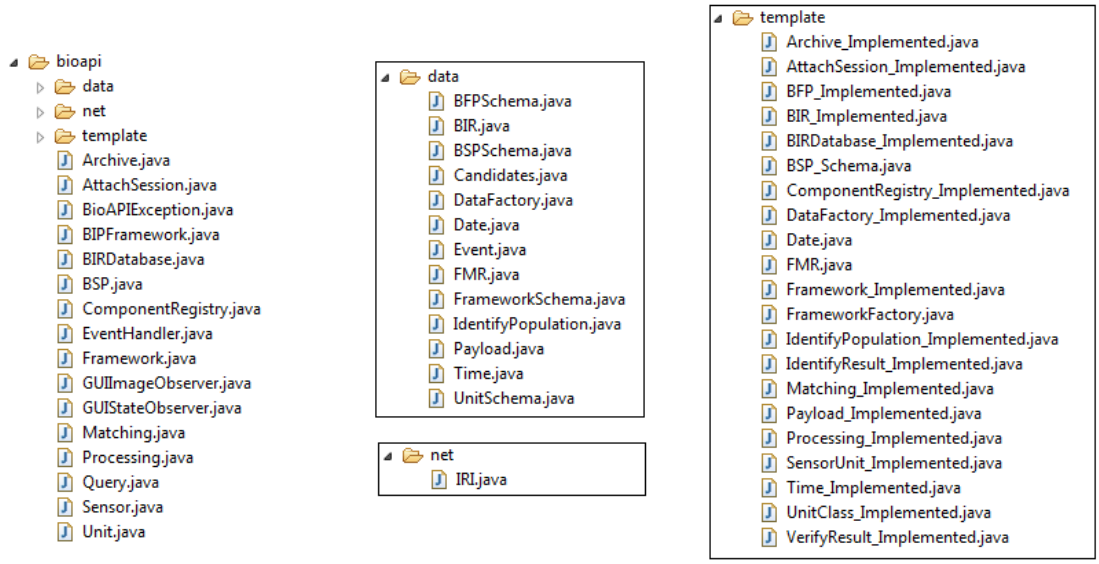


Figura 32. BioAPI en Android

5.3. Desarrollo de la aplicación móvil

El desarrollo de la aplicación consta de dos grandes bloques conceptuales: la creación de una aplicación de usuario y contraseña que gestione una base de datos, por un lado, y conseguir que esté estructurada siguiendo el estándar BioAPI, por otro. Aunque en la práctica ambos bloques se desarrollan simultáneamente, es conveniente recordar su

existencia por separado ya que su unión representa el objetivo final de este trabajo: la implementación de BioAPI en Android. Para la resolución de las dudas surgidas se han empleado diversos tutoriales hallados en internet[28].

Como ya se ha mencionado anteriormente, la estructura de BioAPI en Java no está definida en ninguna normativa oficial, y difiere notablemente de la del BioAPI C. Por ello, no se puede establecer una equivalencia exacta entre un BSP en C y un BSP en Java: en el primer caso el BSP es un concepto en sí mismo y tiene su propio código independiente; en el segundo, no existe la idea de un BSP sin una aplicación asociada. En el BioAPI Java se proporcionan a los desarrolladores una serie de plantillas, almacenadas en el paquete template, con el fin de minimizar las redundancias de código. Este método tiene la ventaja de que estandariza el código de las aplicaciones en un grado mucho mayor, y el inconveniente de que elimina la frontera entre BSP y aplicación, convirtiéndolos en dos entes inseparables. A la hora de escribir código esta unión obligada no resulta un problema, pero sí lo es cuando se intenta concebir la arquitectura del sistema, que se convierte en algo mucho más abstracto. Por lo tanto, aunque a lo largo del presente documento por simplicidad se emplea en todo momento el término "aplicación", sería más correcto hablar de "aplicación con BSP asociado".

La aplicación está basada en una única activity. Este término, propio de Android, denomina a las clases que contienen las interfaces de las aplicaciones; en este caso, solo existe una pantalla, por lo que el uso de más activities es innecesario[29]. Tras crearla, bajo el nombre PasswordEntry, Android obliga a realizar su registro en el Android Manifest ya que, de lo contrario, el programa no funcionaría. Automáticamente se genera el siguiente código:

```
public class PasswordEntry extends Activity {

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        }
    }
```

Al iniciarse, la aplicación ejecutará la activity y mostrará la interfaz main. El siguiente paso es, por tanto, adaptar la interfaz al diseño elegido, que puede verse en el apartado 4.3.2, para luego programar las funciones de cada componente del mismo. Las interfaces gráficas pueden programarse en lenguaje XML o bien haciendo uso del interfaz gráfico de Eclipse. En este caso se han empleado ambas herramientas; la base ha sido el editor gráfico pero algunos detalles se han pulido mediante código directo, por ejemplo la programación de los campos de texto editables para que muestren un texto determinado antes de empezar a escribir en ellos mediante la utilidad AndroidHint:

```
android:hint="@string/HintUsername"
```

```
<string name="HintUsername">Username</string>
```

El resultado de ese código en el emulador es el de la Figura 33; el texto de la cadena HintUsername, que es *"Username"*, aparece en el campo de texto editable mientras el usuario no introduce nada, y al comenzar a escribir, desaparece y es sustituido por los datos que entran por teclado.

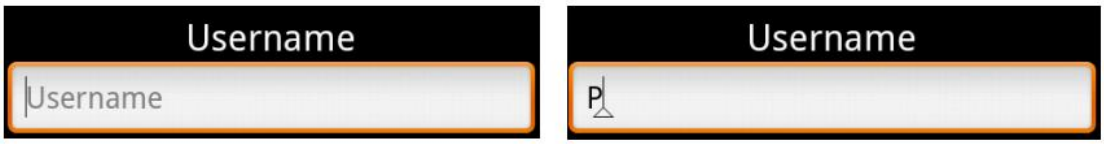


Figura 33. Aplicación de AndroidHint

La interfaz gráfica final de la aplicación vista en el emulador es la de la Figura 34.

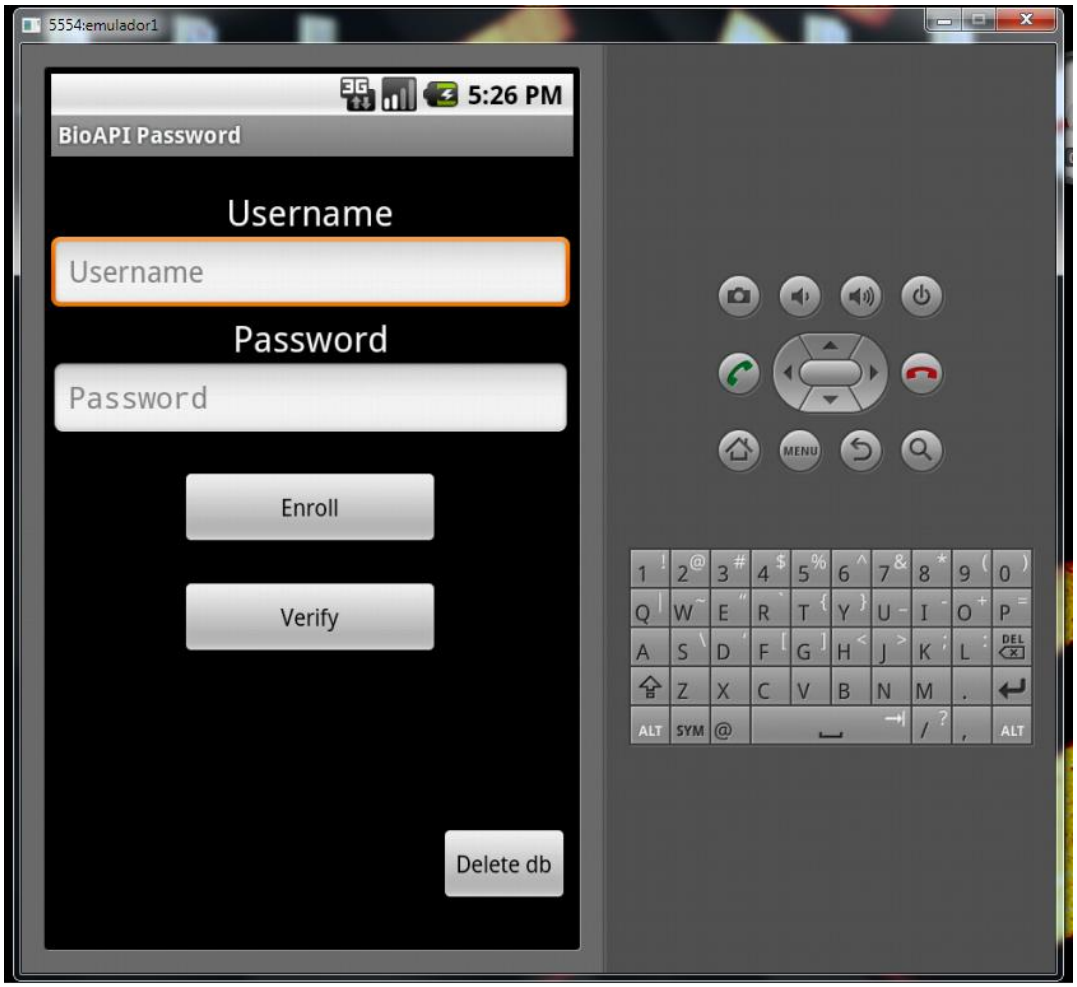


Figura 34. Interfaz final de la aplicación Android

Obviamente, la interfaz sufrió cambios durante el proceso de creación del código; por ejemplo, la ya mencionada desaparición de la funcionalidad de gestionar la base de

datos, que originalmente estaba implementada mediante el botón “Delete db”. La razón por la que se optó por la estrategia de diseñar primero la interfaz es que de este modo es más fácil ver la aplicación en conjunto, de forma global; si se van añadiendo los componentes por separado, programando todas las funciones de cada uno de ellos antes de situar el siguiente es más sencillo cometer errores al realizar las conexiones internas y suele ser necesario rehacer partes del código para adaptarlas al conjunto.

Una vez creada la interfaz gráfica se procedió a la escritura del código de la aplicación. En el presente documento se ha realizado una división en los siguientes apartados para simplificar su lectura: clases heredadas de Java, creación de la base de datos y por último, gestión de los datos introducidos. Sin embargo, esta estructuración obedece únicamente a motivos estéticos, ya que el código se escribió de forma global avanzando en los tres apartados de forma simultánea.

5.3.1. Clases heredadas de Java

Junto con el código de BioAPI Java se distribuía una aplicación de ejemplo que hacía uso del mismo. A pesar de que estaba incompleta –la gestión de la base de datos se realizaba externamente– tres clases de su código se reciclaron para su adaptación al presente trabajo, ya que al usar las plantillas dadas estaban completamente estandarizadas y servían para cualquier propósito biométrico.

La primera de ellas era la clase PasswordBSP, en la que se definían los parámetros necesarios para la creación de un BSP, mediante la importación de una de las plantillas para código de BioAPI: la BSP_Schema.

```
import org.bioapi.template.BSP_Schema;
```

El código simplemente creaba una carcasa vacía, que representaba la estructura del BSP, y que debía rellenarse con los parámetros deseados:

```
public class PasswordBSP extends BSP_Schema
{

      public PasswordBSP(String _BSPDescription, Format[]
_BSPSupportedFormats, UUID _BSPUuid, int _DefaultCalibrateTimeout, int
_DefaultCaptureTimeout, int _DefaultEnrollTimeout, int
_DefaultIdentifyTimeout, int _DefaultVerifyTimeout, Type[]
_FactorsMask, long _MaxBSPDbSize, long _MaxIdentify, long
_MaxPayloadSize, Operation[] _Operations, Option[] _Options, String
_Path, FMR _PayloadPolicy, String _ProductVersion, String
_SpecVersion, String _Vendor)

            {
            super(_BSPDescription, _BSPSupportedFormats, _BSPUuid,
            _DefaultCalibrateTimeout, _DefaultCaptureTimeout,
            _DefaultEnrollTimeout, _DefaultIdentifyTimeout,
            _DefaultVerifyTimeout, _FactorsMask, _MaxBSPDbSize,
```

```
_MaxIdentify, _MaxPayloadSize, _Operations, _Options,
_Path, _PayloadPolicy, _ProductVersion, _SpecVersion,
_Vendor);
}
```

La segunda era la clase Main, que establecía los parámetros necesarios para conformar el BSP mediante llamadas al BioAPI, completando así la clase anterior. Las funciones que proporcionaban los parámetros necesarios, devolviendo después el BSP creado, tenía el código reproducido a continuación:

```
public void createBSP()
	{
		bsp = new PasswordBSP("Sample Password BSP",
				bspSupFormat,
				bspUUID,
				0x7FFFFFFF,
				0x7FFFFFFF,
				0x7FFFFFFF,
				0x7FFFFFFF,
				0x7FFFFFFF,
				FactorsMask,
				0,
				0,
				0,
				operations,
				options,
				null,
				fmr,
				"2.0.1",
				"2.0",
				"BioAPI 2.0 Reference Implementation");
	}

public PasswordBSP getBSP()
{
    return bsp;
}
```

Mediante estas dos clases, Main y PasswordBSP, se creaba un BSP que podía usarse para una amplia variedad de aplicaciones biométricas. Se decidió añadirlas al código porque facilitaban la comprobación del funcionamiento de la implementación en Android de BioAPI; si el código funcionaba en Java usando un BSP genérico debía funcionar también en Android en las mismas condiciones.

La tercera clase de Java empleada en Android recibía el nombre de PasswordBIR, e importaba la plantilla BIR_Implemented:

```
import org.bioapi.template.BIR_Implemented;

public class PasswordBIR extends BIR_Implemented
{
	String username = null;
	String password = null;
```

```
        UUID id;

        public PasswordBIR(String username, String password, UUID id)
        {
                super(null, null, false, null, null, null, null, null);

                this.username = username;
                this.password = password;
                this.id = id;
        }

}
```

De las tres era posiblemente la más importante. Definía la estructura del BIR en el que se basaría la aplicación, que en este caso debía estar formado por dos cadenas de texto, para los campos de usuario y contraseña, y un UUID para identificar sin género de dudas cada uno de los usuarios introducidos. La importancia extrema de esta clase radica en que la comparación de la identidad de cada usuario se realizaría en formato BIR, por lo que este debía estar perfectamente definido para poder usar el mismo tipo de parámetros en todo momento.

5.3.2. Creación de la base de datos

Cuando se empezó a diseñar la base de datos apareció el problema más grave de todos los surgidos durante la creación de la aplicación. Como ya se ha comentado en anteriores apartados, ni BioAPI C ni la versión de Java implementan una base de datos propia, sino que gestionan una creada externamente, conectando con ella. Esto es un método fiable y seguro para un dispositivo fijo, pero roza lo absurdo en un dispositivo móvil. ¿Para qué realizar una implementación móvil si va a ser necesario conectarse a un dispositivo fijo para usarla? Por supuesto, podría crearse una base de datos propia externamente, introducirla en el dispositivo móvil, y realizar la conexión con ella una vez dentro, haciendo innecesaria la conexión fija posterior. Sin embargo, esta solución también presenta un problema: la conexión fija sigue siendo necesaria, aunque solo sea para introducir inicialmente la base de datos en el dispositivo móvil. Por otro lado, la arquitectura Framework Free está pensada para dispositivos de poca memoria, y las bases de datos genéricas podrían resultar demasiado pesadas para los mismos. Además; ya que se realiza una implementación completamente nueva, ¿por qué no mejorar las anteriores?

Con esa filosofía presente, se llegó a una conclusión final: la solución óptima era que la aplicación creara su propia base de datos al iniciarse, haciendo innecesaria la creación externa. Esto solucionaba los problemas de la conexión fija, pero aún quedaba el del tamaño. Por suerte, Android proporciona una funcionalidad interna, llamada SQLite, que permite la gestión de pequeñas bases de datos optimizadas para este SO[30].

Una vez solucionados los problemas logísticos se procedió a la programación de la base de datos, siguiendo uno de los tutoriales hallados al respecto[31]. En una clase nueva, llamada SQLiteAdapter, se escribieron las funciones necesarias para la creación y gestión de una base de datos con dos columnas de campos de tipo cadena, uno para el usuario y otro para la contraseña. La gestión incluía diversas funcionalidades, como la apertura de la base de datos para lectura:

```
public SQLiteAdapter openToRead() throws android.database.SQLException
{
	sqLiteHelper = new SQLiteHelper(context, MYDATABASE_NAME, null,
	MYDATABASE_VERSION);
	sqLiteDatabase = sqLiteHelper.getReadableDatabase();
	return this;
}
```

La apertura para escritura:

```
public SQLiteAdapter openToWrite()throws android.database.SQLException
{
	sqLiteHelper = new SQLiteHelper(context, MYDATABASE_NAME, null,
	MYDATABASE_VERSION);
	sqLiteDatabase = sqLiteHelper.getWritableDatabase();
	return this;
}
```

El cierre de la base de datos, altamente recomendable para evitar errores tras abrirla, ya sea para escritura o para lectura:

```
public void close()
{
	sqLiteHelper.close();
}
```

La inserción de datos:

```
public long insert(String content, String content2)
{
	ContentValues contentValues = new ContentValues();
	contentValues.put(KEY_NAME, content);
	contentValues.put(KEY_PASSWORD, content2);
	return sqLiteDatabase.insert(MYDATABASE_TABLE, null,
	contentValues);
}
```

La indexación, imprescindible a la hora de comparar:

```
public Cursor queueAll()
{
	String[] columns = new String[]{KEY_ID, KEY_NAME, KEY_PASSWORD};
	Cursor cursor = sqLiteDatabase.query(MYDATABASE_TABLE, columns,
	null, null, null, null, null);
	return cursor;
}
```

Y el borrado total:

```
public int deleteAll()
{
	return sqLiteDatabase.delete(MYDATABASE_TABLE, null, null);
}
```

Para que la base de datos se creara bastaba con crear un objeto de la clase SQLiteAdapter, de este modo:

```
private SQLiteAdapter mySQLiteAdapter;
```

Tras esto, se le podía aplicar cualquiera de las funciones anteriores, sin necesidad de preocuparse por las conexiones de la aplicación con la base de datos, ya que era interna, ni por la dependencia con un dispositivo fijo encargado de la gestión, ya que era móvil.

5.3.3. Gestión de los datos introducidos

La gestión de los datos del usuario incluye su captura desde el campo de texto editable, la comparación de los mismos con los almacenados en la base de datos y su introducción en la misma. Esto se realiza mediante el uso de una serie de funciones, escritas en diferentes clases, que interactúan entre sí. A continuación se describirán las más importantes: las funciones de captura getsUsername y getsPassword, la de introducción de datos InsertRecord, la de reclutamiento CompareWithDatabase, la de verificación VerifyDatabase, las de ejecución de opción Enroll y Verify y por último las funciones que son activadas directamente mediante la interacción del usuario con la interfaz gráfica.

Funciones de captura

Las funciones de captura de datos están implementadas en la clase Activity de la aplicación y simplemente recogen los datos que el usuario introduce en los campos de texto editables. Son dos, getsUsername, que recoge la cadena que el usuario introduce en el campo de texto de nombre, y getsPassword, que recoge la cadena que el usuario introduce en el campo de texto de contraseña. Ambas siguen la misma estructura: primero se definen cuatro parámetros, dos cadenas de texto y dos campos de texto editables:

```
static String sUserName, sPassword;
static EditText Username, Password;
```

Luego se inicializan en la función onCreate de la clase PasswordEntry los campos de texto editables, dándole a cada uno un nombre:

```
Username = (EditText)findViewById(R.id.etext1);
Password = (EditText)findViewById(R.id.etext2);
```

Y por último, es copiado en las cadenas el texto que el usuario introduce, mediante las funciones de captura:

```
public static String getsUserName()
{
	sUserName = Username.getText().toString();
	return sUserName;
}

public static String getsPassword()
{
	sPassword = Password.getText().toString();
	return sPassword;
}
```

Al finalizar estos pasos, los datos estarán en almacenados en la memoria interna, en forma de cadenas de texto. En un sistema biométrico real, sería necesario su procesamiento en función del tipo de datos que fuesen, ya que los datos crudos no son válidos para la comparación. Sin embargo, esta aplicación no busca la creación de un sistema biométrico completo, únicamente la comprobación de que el BioAPI ha sido correctamente portado, por lo que en este caso se ha optado por el tipo de dato más simple posible, y no es necesario aplicarle ningún cambio antes de su comparación.

Función de introducción de datos

Para introducir los datos en la base de datos se emplea una función llamada InsertRecord, que se encuentra en la clase PasswordArchiveUnit. Dicha función recibe por parámetros dos cadenas de texto: name y password; luego conecta con la base de datos existente (o la crea si no existe ninguna) e introduce las dos cadenas en su interior, para cerrarla inmediatamente después. Si la operación se lleva a cabo con éxito, devuelve el booleano true, y si hay algún fallo, muestra la excepción pertinente y devuelve false. Su código se reproduce a continuación:

```
public boolean insertRecord(String name, String password)
{
    try{
        mySQLiteAdapter = new SQLiteAdapter(PasswordEntry.c);
        mySQLiteAdapter.openToWrite();
        mySQLiteAdapter.insert(name, password);
        mySQLiteAdapter.close();

        return true;
    }
    catch(Exception e) {
        System.err.println("Exception: " + e.getMessage());
        return false;
    }

}
```

Función de reclutamiento

Para llevar a cabo la comparación de los datos introducidos con los existentes en la base de datos, es necesario realizar un listado de estos últimos, para lo que se hace uso de la función GetEntries. Primero conecta con la base de datos y la abre para lectura. A continuación crea un cursor que la recorrerá, mediante la función de indexación queueAll descrita en el apartado 5.3.2. El cursor se moverá fila por fila a lo largo de la base de datos, aumentado un contador con cada movimiento. Al finalizar, devolverá un vector de datos UUID del tamaño del contador, que contendrá el mismo número de elementos que existen en la base de datos. Si no existe la base de datos cuando se llama a esta función, se crea vacía, y se devuelve un vector nulo. Su código es el siguiente:

```
public UUID[] getEntries()
{
    mySQLiteAdapter = new SQLiteAdapter(PasswordEntry.c);
    mySQLiteAdapter.openToRead();
  cursor = mySQLiteAdapter.queueAll();
    int count=0;

    if(cursor.moveToFirst()){
         do
         {
               count++;
         }while(cursor.moveToNext());

         entries = new UUID[count];
         return entries;}

    entries = null;
    return entries;
}
```

La función anterior es imprescindible porque no pueden compararse los datos del interior de una base de datos si no se sabe cuántos hay. Una vez llevado a cabo este paso, se inicia la comparación a través de la función CompareWithDatabase, de la clase PasswordArchiveUnit. Esta función conecta con la base de datos, la abre para escritura y solicita el tamaño de la misma mediante una llamada a la función GetEntries. A continuación se crea un cursor (independiente del creado en GetEntries) que recorrerá la base de datos. Desde la primera a la última fila, los datos serán convertidos a formato BIR. En el apartado 5.3.1 ya se ha mencionado que la estructura del BIR de esta aplicación estará formada por dos campos de texto y un identificador:

```
public PasswordBIR(String username, String password, UUID id)
```

Los dos primeros campos serán extraídos de la base de datos mediante la función predefinida en Android getColumnIndex. El identificador será un UUID obtenido de la llamada anterior a la función GetEntries, relacionado con el número de elementos contenido en la base de datos:

```
_bir= new PasswordBIR
(cursor.getString(cursor.getColumnIndex(SQLiteAdapter.KEY_NAME)),
cursor.getString(cursor.getColumnIndex(SQLiteAdapter.KEY_PASSWORD)),
entries[count-1]);
```

Durante el recorrido por la base de datos, cada BIR obtenido de esta forma será comparado (únicamente en su campo de nombre) con los datos introducidos por el usuario. La comparación de contraseña es innecesaria para el reclutamiento, ya que lo que se persigue es que no existan dos usuarios con el mismo nombre; la contraseña puede coincidir con la de otra persona. Si en algún momento existe coincidencia, la comparación finaliza y se devuelve el booleano false, de lo contrario se devuelve true. El código completo de la función es el siguiente:

```
public boolean CompareWithDatabase()

{
    mySQLiteAdapter = new SQLiteAdapter(PasswordEntry.c);
    mySQLiteAdapter.openToRead();
    cursor = mySQLiteAdapter.queueAll();
    int count=0;
    entries = getEntries();
    cursor.moveToFirst();
    do{

       count++;
       _bir= new PasswordBIR (cursor.getString
       (cursor.getColumnIndex(SQLiteAdapter.KEY_NAME)),
       cursor.getString
       (cursor.getColumnIndex(SQLiteAdapter.KEY_PASSWORD)),
       entries[count-1]);

       if(_bir.getUsername().equals(PasswordEntry.getsUserName()))
       {return false;}

       }while(cursor.moveToNext());

return true;
}
```

Función de verificación

La función de verificación, llamada VerifyDatabase y situada en la clase PasswordArchiveUnit, sigue exactamente los mismos pasos que la de reclutamiento: primero conecta con la base de datos, la abre para escritura y solicita el tamaño de la misma mediante una llamada a la función GetEntries. A continuación crea un cursor para recorrerla, se convierten los datos a formato BIR y se comparan. La diferencia con el reclutamiento es que en este caso se compara tanto el usuario como la contraseña, ya que se quiere comprobar si el usuario ha introducido sus datos correctamente. Si la comparación encuentra alguna coincidencia, se devuelve el booleano true, y si tras comparar todos los datos no ha habido coincidencias, se devuelve false. El código de la función es el siguiente:

```
public boolean verifyDatabase()
{
   mySQLiteAdapter = new SQLiteAdapter(PasswordEntry.c);
   mySQLiteAdapter.openToRead();
   cursor = mySQLiteAdapter.queueAll();
   int count2 = 0;

   entries = getEntries();

   cursor.moveToFirst();
   do{
       count2++;
       _bir= new PasswordBIR (cursor.getString
       (cursor.getColumnIndex(SQLiteAdapter.KEY_NAME)),
       cursor.getString
       (cursor.getColumnIndex(SQLiteAdapter.KEY_PASSWORD)),
       entries[count2-1]);

       if((_bir.getUsername().equals(PasswordEntry.getsUserName())) &&
       (_bir.getPassword().equals(PasswordEntry.getsPassword())))
       {return true;}

   }while(cursor.moveToNext());

   return false;
   }
```

Funciones de ejecución

Las funciones de ejecución se encuentran en la clase PasswordBFP, y son activadas mediante una llamada desde la Activity cuando el usuario pulsa el botón de Enroll o Verify en la interfaz gráfica. Se ocupan de llamar a las funciones anteriores en el orden adecuado para llevar a cabo la opción escogida. Para su uso es necesario definir un objeto del tipo PasswordArchiveUnit, que permita el acceso a las funciones de esa clase, y tras ello activar la función GetEntries:

```
private PasswordArchiveUnit _pass = new PasswordArchiveUnit();

entries = _pass.getEntries();
```

Tras estos pasos previos, ya pueden ser implementadas. La función Enroll comprueba si GetEntries ha devuelto un vector nulo, y en caso afirmativo, indicativo de que la base de datos está vacía, inserta automáticamente los valores introducidos por el usuario, ya que al no existir usuarios en el sistema es imposible que se produzca una coincidencia de nombre. Si, por el contrario, GetEntries no ha devuelto un vector nulo, se compara el dato introducido con la base de datos mediante CompareWithDatabase. Si se recorren todas las filas de la base de datos sin encontrar coincidencias, los datos introducidos por el usuario se añaden a la misma mediante una llamada a la función InsertRecord:

```
public boolean Enroll()
{
int count =0;
```

```
if (entries == null)
{
       _pass.insertRecord
       (PasswordEntry.getsUserName(),PasswordEntry.getsPassword());
       return true;
}
while(count < entries.length)
{
       if (!_pass.CompareWithDatabase()){return false;}
       count ++;
}

_pass.insertRecord
(PasswordEntry.getsUserName(),PasswordEntry.getsPassword());
return true;
}
```

La función Verify realiza los mismos pasos, exceptuando la inserción de datos. Si la base de datos está vacía, o si no se encuentran coincidencias, devuelve un error en la verificación, y si los datos del usuario coinciden con alguna identidad existente en el sistema, devuelve el booleano true:

```
public boolean Verify()
{
int count =0;
if (entries == null){return false;}

while(count < entries.length)
{
      if (_pass.verifyDatabase())
      return true;
      count ++;
}
return false;
}
```

Funciones de interfaz gráfica

Las funciones de interfaz gráfica son activadas directamente por el usuario. Son dos, una para el botón Enroll y otra para el botón Verify. Ambas comparten el primer paso: mediante una llamada getBSP de la clase Main se crea un nuevo BSP. Por otro lado, se crea un objeto del tipo PasswordBFP, para tener acceso a las funciones de esa clase, de la siguiente forma:

```
Main m = new Main();
PasswordBFP bfp = new PasswordBFP
((PasswordAttachSession)m.getBSP().newAttachSession());
return bfp;
```

A continuación, se llama a la función Enroll o Verify, según lo que el usuario desee, y se muestran toasts según el resultado de las comparaciones. Dentro del reclutamiento hay dos opciones: que sea correcto, en cuyo caso se mostraría el texto *"User X*

successfully registered in database", o que se encuentre alguna coincidencia, tras lo que se mostraría el texto *"Username X already exists"*, mediante el siguiente código:

```
if (bfp.Enroll())
{
	toast = Toast.makeText(c, "User '" + sUserName + "' successfully
	registered in database", Toast.LENGTH_LONG);
	toast.setGravity(Gravity.CENTER, 0, 0);
	toast.show();
}
else
{
	toast = Toast.makeText(c, "Username '" + sUserName + "' already
	exists", Toast.LENGTH_LONG);
	toast.setGravity(Gravity.CENTER, 0, 0);
	toast.show();
}
```

Dentro de la verificación hay tres opciones: si el usuario la solicita cuando la base de datos está vacía, se mostrará el texto *"No users entered yet"*. Si hay usuarios y la verificación se lleva a cabo con éxito, se mostrará el texto *"Login for user X sucessful"* y si falla, el toast contendrá la leyenda *"Login for user X failed"*.

```
if (bfp.Verify())
{
	toast = Toast.makeText(c, "Login for user '" + sUserName + "'
	successful", Toast.LENGTH_LONG);
	toast.setGravity(Gravity.CENTER, 0, 0);
	toast.show();
}

else
{
	if (entries == null)
	{
		toast = Toast.makeText(PasswordEntry.c, "No users entered
		yet", Toast.LENGTH_SHORT);
		toast.setGravity(Gravity.CENTER, 0, 0);
		toast.show();
	}

	else
	{
		toast = Toast.makeText(c, "Login for user '" + sUserName
		+ "' failed", Toast.LENGTH_LONG);
		toast.setGravity(Gravity.CENTER, 0, 0);
		toast.show();
	}
}
```

Otra función gráfica era la DeleteDatabase, que permitía el borrado total de la base de datos. Aunque finalmente se desechó su uso, el código sigue en la aplicación, por lo que podría ser usado si otro desarrollador decide añadirle funcionalidades a este proyecto.

```
public void DeleteDatabase(View view)
{
	mySQLiteAdapter = new SQLiteAdapter(c);
	mySQLiteAdapter.openToWrite();
	mySQLiteAdapter.deleteAll();
	mySQLiteAdapter.close();

	toast = Toast.makeText(c, "Database deleted",
	Toast.LENGTH_LONG);
	toast.setGravity(Gravity.CENTER, 0, 0);
	toast.show();
}
```

5.4. Desarrollo de la eliminación del Framework

Para esta parte del proyecto, la más innovadora de todas, se había optado por ir función a función razonando cuales de ellas realizaban una función de intermediación entre el BSP y la aplicación, ya que serían parte del Framework y debían ser eliminadas. Como se ha dicho en el apartado de diseño, la descripción de la normativa no oficial del BioAPI Java reduce la búsqueda a dos únicos paquetes: bioapi y data. Dentro de ellos se encuentran las funciones necesarias para fijar, gestionar y procesar los componentes y Unidades que interactúan con el Framework de BioAPI, así como las estructuras de datos y modelos que describen interfaces para dichas funciones.

La eliminación del Framework se ha efectuado analizando una por una las clases de esos dos paquetes y razonando cuales de ellas formaban parte del Framework. A continuación se realizará una breve descripción de dichas clases y del razonamiento seguido, bien para su eliminación, bien para mantenerlas en el código.

En cuanto al borrado en sí, el código de BioAPI Java no contiene apenas funciones, siendo la mayoría de clases interfaces y enumeraciones. Por lo tanto, no ha habido ningún problema de interconexión al realizar el borrado de las clases seleccionadas, ni ha sido necesario sustituir su código.

5.4.1. Paquete bioapi

AttachSession

Contiene las funcionalidades necesarias para asociar temporalmente una determinada Unidad a un determinado BSP, lo que se conoce como "attach session". En función de la asociación formada y de la categoría de las Unidades empleadas, existen diferentes grupos de operaciones biométricas accesibles a través de Archive, Matching, Processing o Sensor. Esta clase también proporciona control sobre una Unidad genérica a través de la interfaz Unit.

Puesto que la asociación de BSPs con las Unidades de BioAPI es innecesaria en un sistema Framework Free, ya que está pensado para sistemas con BSP único, esta clase como es irrelevante y puede ser eliminada del código. Sin embargo, ese BSP podrá realizar sus funciones propias, por lo que las clases asociadas (Archive, Matching, Processing, Sensor y Unit) no deben borrarse.

Archive

Esta interfaz contiene las funcionalidades biométricas disponibles cuando una Unidad de archivo es adjuntada al BSP.

El BSP del sistema biométrico debe poder realizar funciones de almacenar información si es necesario, por lo que está clase no debe ser eliminada.

BioAPIException

Contiene un índice de algunos de los problemas que pueden surgir durante una operación de un sistema biométrico. Para cada uno de ellos incluye información adicional, como el código numérico que lo identifica o el ID del componente que lo genera.

Esta clase no pertenece al Framework, y proporciona una información que no solo no es irrelevante, sino que en muchos casos es necesaria para los desarrolladores. Por lo tanto, es indispensable que permanezca en el código.

BIPFramework

Define las extensiones BIP a BioAPI, para la posible implementación de un método de identificación de recursos basado en el International Resource Identifier (IRI).

El "Framework" de su nombre no se refiere a la capa Framework de BioAPI, sino al marco regulador de la clase IRI, situada en el paquete net. Proporciona una primera guía para un método no implementado en el código. Actualmente no aporta nada, pero puede ser útil si algún desarrollador decide actualizarlo en un futuro. Por lo tanto, se ha optado por mantenerla.

BIRDatabase

Contiene una base de datos gestionada internamente por el BSP que contiene, en formato BIR y con un UUID asociado, todos los datos biométricos almacenados por el sistema.

Esta clase no forma parte del Framework y además ha sido empleada en la aplicación del proyecto, por lo que debe permanecer en el código.

BSP

Contiene la interfaz necesaria para representar el BSP.

Esta clase no forma parte del Framework y además ha sido empleada en la aplicación del proyecto, por lo que debe permanecer en el código.

ComponentRegistry

Contiene la interfaz para el registro de componentes, herramienta que proporciona información a las aplicaciones acerca de los BFPs y los BSPs instalados, mediante una serie de llamadas al Framework.

El registro de componentes es una función del Framework que no puede ser implementada en un sistema Framework Free. Por lo tanto, debe ser eliminada.

EventHandler

La aplicación implementa la interfaz EventHandler para controlar los eventos que provoca el BSP. Se debe evitar cualquier llamada a BioAPI mientras que el controlador de eventos es ejecutado, o el sistema puede sufrir bloqueos.

El controlador es útil en un sistema con varios componentes, en el que el Framework debe evitar que se produzcan múltiples llamadas simultáneas a diferentes dispositivos o BSPs. En un sistema Framework Free, que consta de un solo BSP, un solo dispositivo biométrico y una única aplicación, un controlador carece de sentido. Por lo tanto, se ha optado por su eliminación.

Framework

La interfaz más básica del Framework. Controla los BSPs de los sistemas biométricos con varios componentes de diversas tecnologías, mediante el empleo de los BFPs, librerías que contienen las funciones necesarias para la gestión de los mismos.

Es la base de la capa que se quiere eliminar, por lo que no tiene cabida en el código.

GUIImageObserver

Contiene la interfaz que una aplicación biométrica debe usar para permitir a un BSP mostrar datos por pantalla, dirigiéndolos hacia ella en forma de mapa de bits a través del Framework.

Esta clase necesita de un Framework para su correcto funcionamiento, luego no es posible su implementación en este sistema.

GUIStateObserver

Contiene la interfaz que una aplicación biométrica debe usar para permitir a un BSP proporcionarle información sobre el estado de la interfaz gráfica de usuario o GUI (Graphical User Interface) a través del Framework.

Esta clase necesita de un Framework para su correcto funcionamiento, luego no es posible su implementación en este sistema.

Matching

Esta interfaz contiene las funcionalidades biométricas disponibles cuando una Unidad de algoritmo de equiparación es adjuntada al BSP.

El BSP del sistema biométrico debe poder realizar funciones de comparar información si es necesario, por lo que está clase no debe ser eliminada.

Processing

Esta interfaz contiene las funcionalidades biométricas disponibles cuando una Unidad de algoritmo de procesamiento es adjuntada al BSP.

El BSP del sistema biométrico debe poder realizar funciones de procesar información si es necesario, por lo que está clase no debe ser eliminada.

Query

Es una interfaz de consulta, que filtra las peticiones que devuelve el Framework.

Es otra de las clases que no tienen sentido sin el Framework, por lo que se ha optado por su eliminación.

Sensor

Esta interfaz contiene las funcionalidades biométricas disponibles cuando una Unidad de sensor es adjuntada al BSP.

El BSP del sistema biométrico debe poder realizar funciones de capturar información si es necesario, por lo que está clase no debe ser eliminada.

Unit

Esta interfaz contiene las funcionalidades biométricas genéricas disponibles cuando una Unidad, del tipo que sea, es adjuntada al BSP.

La Unidad del sistema biométrico debe poder realizar sus funciones propias cuando sea necesario, por lo que esta clase debe permanecer en el código.

5.4.2. Paquete data

BFPSchema

Define las propiedades y la arquitectura de los BFP instalados en el sistema.

No es una clase perteneciente al Framework, y además se ha usado en la aplicación del presente proyecto. Debe permanecer en el código.

BIR

Contiene el esquema del BIR. Para modificarlo, debe usarse la clase DataFactory para crear un BIR con un nuevo esquema. Este nuevo BIR podrá ser independiente o estar unido a un BSP asociado a una Unidad mediante una attach session. En ese último caso, el BIR se borrará cuando concluya la asociación.

No es una clase perteneciente al Framework, y además se ha usado en la aplicación del presente proyecto. Debe permanecer en el código.

BSPSchema

Define las propiedades y la arquitectura de los BSP instalados en el sistema.

No es una clase perteneciente al Framework, y además se ha usado en la aplicación del presente proyecto. Debe permanecer en el código.

Candidates

Contiene la colección de BIRs identificados como resultado de los procesos de comparación.

No forma parte del Framework, por lo que permanecerá en el código.

DataFactory

Proporciona a la aplicación una herramienta de instanciación de tipos de datos de BioAPI, como Date o BIR.

Puede ser útil en muchos casos, por ejemplo, si se quiere instanciar un BIR con un esquema diferente al proporcionado. No forma parte del Framework por lo que no debe eliminarse.

Date

Contiene la información necesaria para representar la fecha del calendario.

No forma parte del Framework, por lo que no se eliminará del código.

Event

Enumera los tipos de evento que se envían a la aplicación cuando cambia el estado de una Unidad.

Comunica la aplicación con el BSP, es una función de Framework. Por lo tanto, debe ser eliminada.

FMR

Contiene la tasa de falso negativo, un valor entero que indica la probabilidad de obtener un error en la verificación. A mayor valor, peor fiabilidad del sistema.

No es una función del Framework y es necesaria para indicar la fortaleza del sistema biométrico, por lo que no debe eliminarse.

FrameworkSchema

Contiene la parte del registro de componentes que define las propiedades del Framework instalado en el sistema.

Al ser una clase con funciones propias del registro de componentes, que se ha eliminado de la versión Framework Free, debe ser extraída del código.

IdentifyPopulation

Devuelve el número de BIRs con el que debe compararse cada nuevo dato introducido.

Es un método que necesita el Framework para pasar su valor a la aplicación, por lo que debe eliminarse. Si su uso es imprescindible, debe ser sustituido dentro del código de la propia aplicación. Por ejemplo, en la del presente proyecto se ha implementado mediante la función GetEntries.

Payload

Representa la capacidad de carga que la aplicación puede asociar a la plantilla biométrica obtenida.

No es un método del Framework, por lo que no debe eliminarse.

Time

Contiene la información necesaria para representar la hora del día.

No forma parte del Framework, por lo que no se eliminará del código.

UnitSchema

Define las propiedades y la arquitectura de las Unidades instalados en el sistema.

No es una clase perteneciente al Framework, y además se ha usado en la aplicación del presente proyecto. Debe permanecer en el código.

5.4.3. Resultado final

Tras eliminar el Framework y algunas de las plantillas del paquete template –ComponentRegistry_Implemented, AttachSession_Implemented, FrameworkFactory, Framework_Implemented e IdentifyPopulation_Implemented- la implementación presentaba el aspecto de la Figura 35.

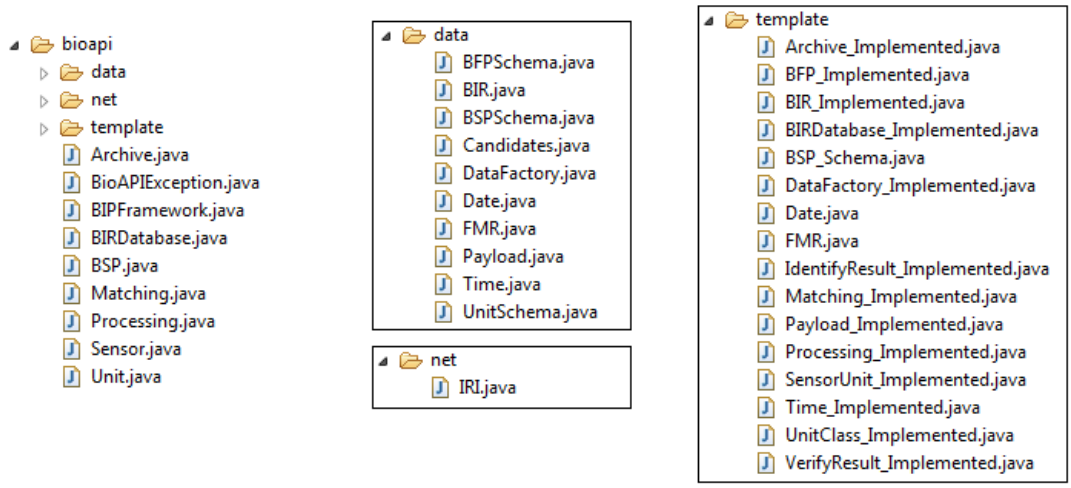


Figura 35. BioAPI Framework Free en Android

6. PRUEBAS

En el presente apartado se presentarán las pruebas realizadas para comprobar el funcionamiento de la aplicación creada, que a su vez sirve para comprobar el funcionamiento de la implementación de BioAPI en Android. Se van a establecer dos subconjuntos diferentes: el primero estará formado por las pruebas realizadas durante el proceso de creación, llevadas a cabo en el emulador de Android, y el segundo contendrá las pruebas finales, llevadas a cabo en un dispositivo móvil real una vez finalizado el proyecto.

6.1. Pruebas en emulador

Las pruebas durante el proceso se han llevado a cabo en su mayor parte por medio de toasts, ya que era el método más rápido de comprobar si una función o un método operaban correctamente. A continuación se mostrarán algunas de las pruebas empleadas para cada una de las funciones esenciales del programa, junto con una breve descripción de cada una y su resultado en el emulador.

6.1.1. Captura de datos

Para garantizar que los datos introducidos por el usuario eran capturados correctamente por el sistema, y comprobar que su movimiento entre clases no provocaba errores, se definió una función nueva en PasswordArchiveUnit llamada Prueba1. Su código simplemente mostraba en un toast el usuario y la contraseña elegidos por el usuario. A continuación, se adaptó el código del botón Enroll para que su única acción fuese llamar a Prueba1 desde la Activity.

```
public void Prueba1()
{
toast = Toast.makeText(PasswordEntry.c, "Usuario = '" +
PasswordEntry.getsUserName() + "' \nContraseña = '"
+ PasswordEntry.getsPassword() + "'", Toast.LENGTH_LONG);
toast.setGravity(Gravity.CENTER, 0, 0);
toast.show();
}
```

El resultado en emulador se puede ver en la Figura 36

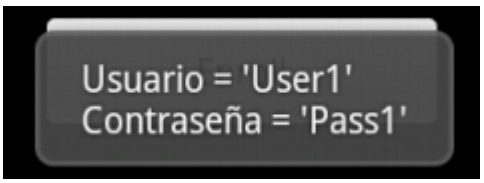


Figura 36. Prueba de captura de datos

6.1.2. Introducción en la base de datos

Para comprobar que la introducción en la base de datos no presentaba errores, se empleó un método parecido al de las funciones de reclutamiento y verificación directamente implementado en la interfaz gráfica, a través del botón Enroll. Al presionarlo, se realizaría una llamada a GetEntries para obtener el tamaño de la base de datos, que se mostraría por pantalla. A continuación, la aplicación recorrería la base de datos mediante un cursor, mostrando cada valor almacenado mediante dos toast, uno para el campo de usuario y otro para la contraseña:

```
entries=_pass.getEntries();
toast = Toast.makeText(c, "Number of entries = " + entries.length,
Toast.LENGTH_LONG);
toast.setGravity(Gravity.CENTER, 0, 0);
toast.show();

toast = Toast.makeText(c, "Name: " + cursor.getString
(cursor.getColumnIndex(SQLiteAdapter.KEY_NAME)), Toast.LENGTH_SHORT);
toast.setGravity(Gravity.CENTER, 0, 0);
toast.show();

toast = Toast.makeText(c, "Password: " + cursor.getString
(cursor.getColumnIndex(SQLiteAdapter.KEY_PASSWORD)),
Toast.LENGTH_SHORT);
toast.setGravity(Gravity.CENTER, 0, 0);
toast.show();
```

El resultado de esta prueba obtenido en el emulador, para la introducción de cuatro usuarios, se muestra en la Figura 37.

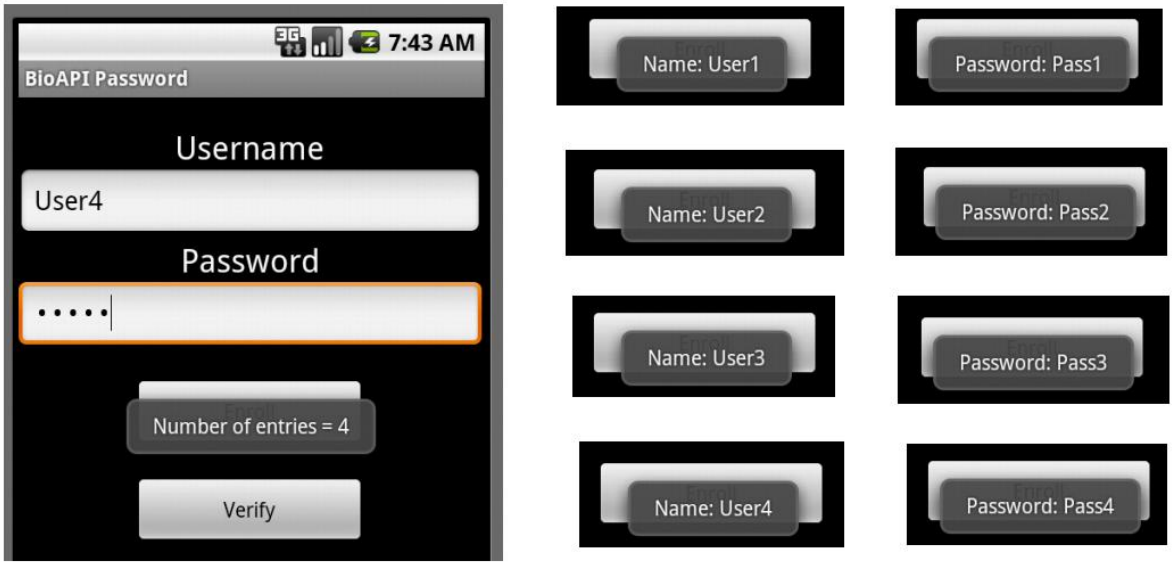


Figura 37. Prueba de introducción de datos en la base de datos

6.1.3. Pruebas de la aplicación final

Mediante la simulación de todos los casos posibles, se comprobó la respuesta del funcionamiento final del programa. Estas pruebas fueron realizadas dos veces, la primera al finalizar la creación de la aplicación y la segunda tras eliminar la capa Framework, obteniéndose idénticos resultados en ambas ocasiones. Se muestran aquí los resultados de la segunda tanda de pruebas, con el proyecto finalizado completamente en todos sus apartados.

Reclutamiento correcto

La introducción de un usuario nuevo hacía que el programa mostrara el toast de la Figura 38.

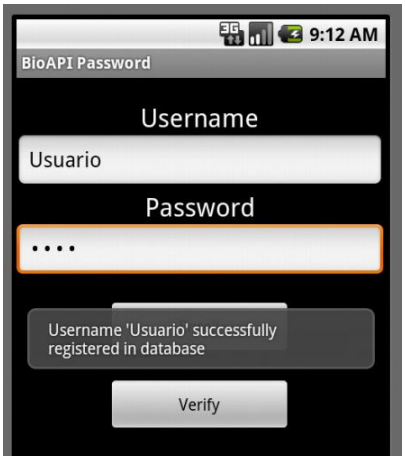


Figura 38. Reclutamiento correcto en emulador

Reclutamiento erróneo

El intento de reclutamiento con un nombre ya existente en la base de datos provocaba el resultado de la Figura 39.

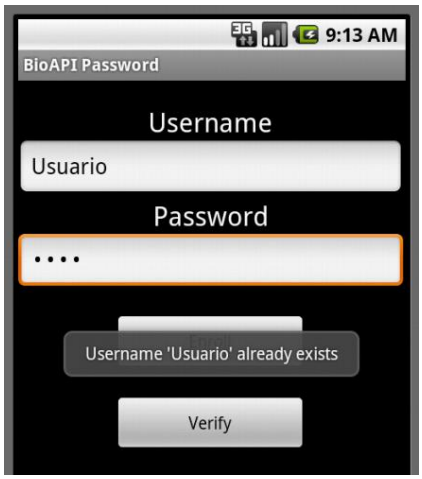


Figura 39. Reclutamiento erróneo en emulador

Verificación correcta

Si el usuario introducía sus datos correctamente, y había sido reclutado previamente para el sistema, el programa reconocía su identidad, como se ve en la Figura 40.

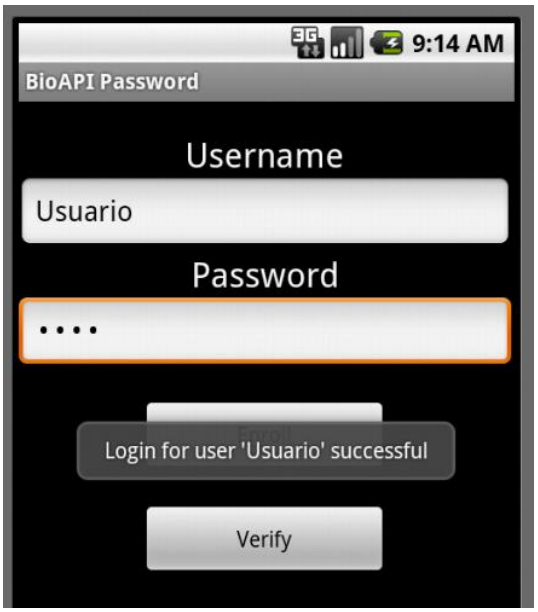


Figura 40. Verificación correcta en emulador

Verificación errónea

Si el usuario cometía un error al introducir sus datos tras haber sido previamente reclutado, o bien intentaba entrar al sistema con una identidad no registrada, el programa devolvía un fallo de registro como el de la Figura 41.

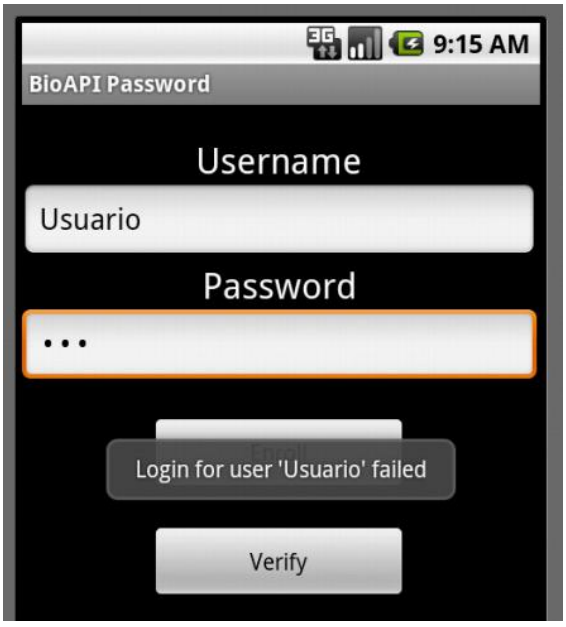


Figura 41. Verificación errónea en emulador

Solicitud de verificación antes de la introducción de usuarios

Si el usuario intentaba obtener su verificación cuando aún no existían registros en la base de datos, el sistema informaba de que no se habían producido reclutamientos aún, como se ve en la Figura 42.

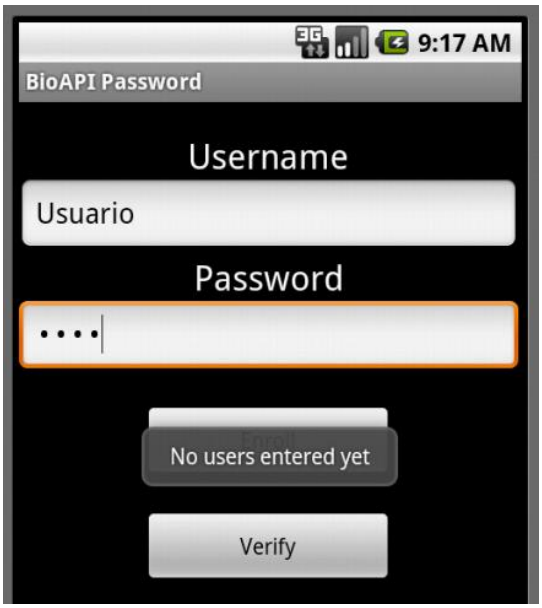


Figura 42. Intento de verificación previo a la introducción de usuarios en emulador

6.1.4. Borrado de la base de datos

Este método no fue implementado en la aplicación final, pero proporcionó una gran ayuda durante las pruebas, al evitar tener que borrar la base de datos manualmente cada vez que se quería probar un nuevo caso, como el de la solicitud de verificación previo a la introducción de usuarios. Para comprobar que el borrado se realizaba correctamente, tras borrar la base de datos aparecía en pantalla un toast con la leyenda *"Database deleted"*, como el de la Figura 43.

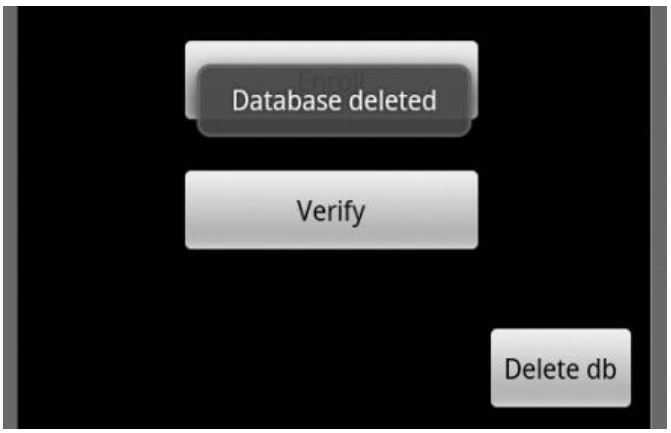


Figura 43. Borrado de la base de datos

6.2. Pruebas en dispositivo móvil

Las pruebas en un dispositivo móvil real han sido llevadas a cabo en un terminal Sony Ericsson Xperia X8 con sistema operativo actualizado a Android 2.1 Eclair. Para poder instalar aplicaciones propias ha sido necesario configurar el teléfono, marcando "ajustes" y accediendo desde ahí a "aplicaciones" y luego a "desarrollo" para activar la opción "Depuración USB". Una vez hecho esto ya se puede cargar la aplicación en el móvil y comprobar su respuesta en un entorno real.

El icono escogido para la aplicación ha sido el logo de BioAPI. En la Figura 44[32] puede verse el logo original y en la Figura 45 el resultado de su uso en la aplicación:

Figura 44. Logo de BioAPI

Figura 45. Icono de la aplicación usando el logo de BioAPI

Las pruebas realizadas han sido las mismas que las de aplicación final del emulador, obteniéndose los siguientes resultados:

Inicio de la aplicación

A la izquierda de la Figura 46, la aplicación nada más iniciarla, antes de escribir nada. A la derecha, una vez introducidos los datos, sin pulsar ningún botón.

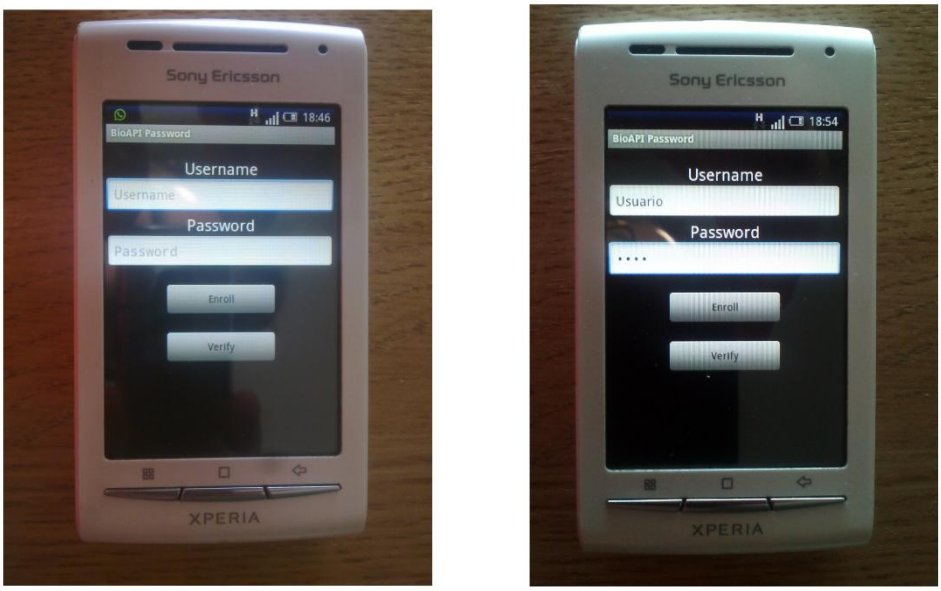


Figura 46. Inicio de la aplicación en un dispositivo real

Reclutamiento correcto

La introducción de un usuario nuevo hacía que el programa mostrara el toast de la Figura 47.

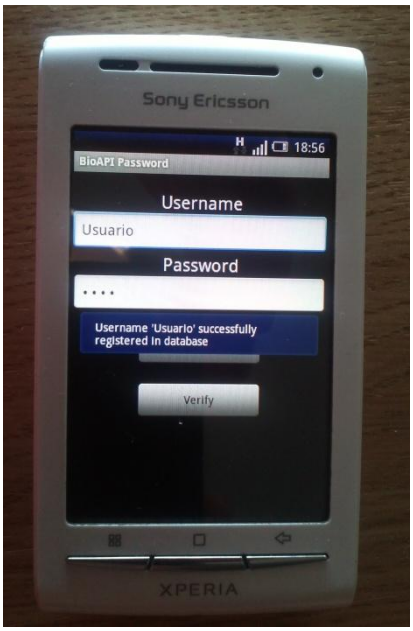


Figura 47. Reclutamiento correcto en dispositivo real

Reclutamiento erróneo

El intento de reclutamiento con un nombre ya existente en la base de datos provocaba el resultado de la Figura 48.

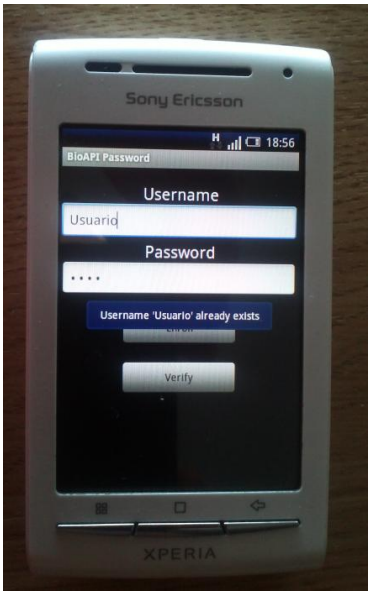


Figura 48. Reclutamiento erróneo en dispositivo real

Solicitud de verificación antes de la introducción de usuarios

Si el usuario intentaba obtener su verificación cuando aún no existían registros en la base de datos, el sistema informaba de que no se habían producido reclutamientos aún, como se ve en la Figura 49.

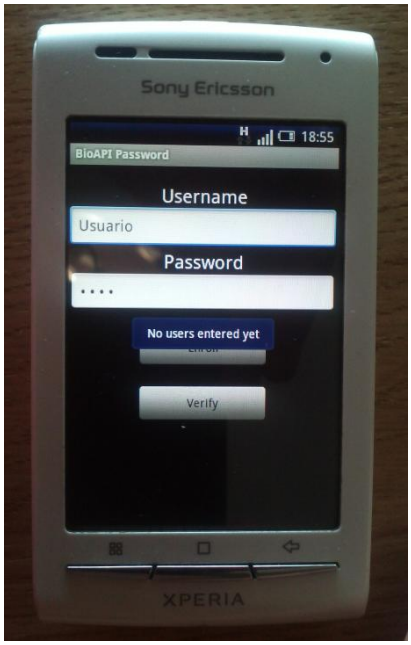


Figura 49. Solicitud de verificación sin usuarios en dispositivo real

Verificación correcta

Si el usuario introducía sus datos correctamente, y había sido reclutado previamente para el sistema, el programa reconocía su identidad, como se ve en la Figura 50.

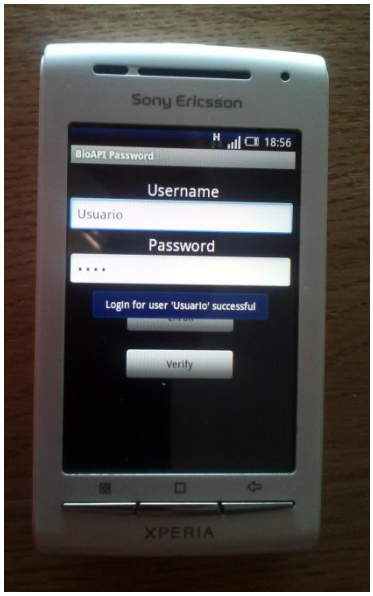


Figura 50. Verificación correcta en dispositivo real

Verificación errónea

Si el usuario cometía un error al introducir sus datos o intentaba entrar al sistema con una identidad no registrada, el programa devolvía un error como el de la Figura 51.

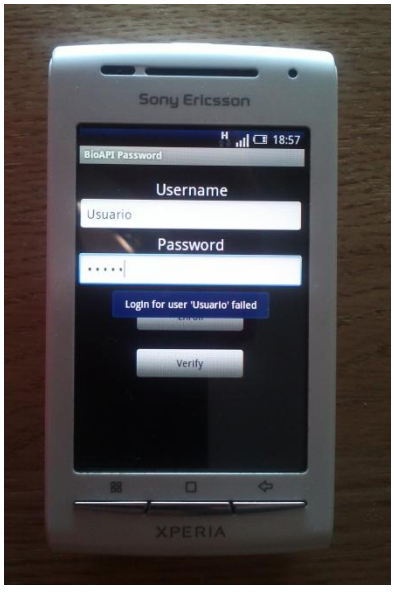


Figura 51. Verificación errónea en dispositivo real

Los resultados de las pruebas han sido los esperados, y demuestran que la implementación de BioAPI en Android se ha llevado a cabo satisfactoriamente.

7. CONCLUSIONES Y LÍNEAS FUTURAS DE INVESTIGACIÓN

7.1. Conclusiones

Al finalizar el trabajo se ha obtenido una implementación completamente funcional del estándar para aplicaciones biométricas BioAPI Framework Free en plataformas móviles bajo el sistema operativo Android. También se dispone de una aplicación de usuario y contraseña que permite el reclutamiento y posterior verificación de distintos usuarios en un único móvil mediante el uso de la implementación anterior.

Los principales problemas surgidos durante el proceso de creación han estado referidos a las partes del proyecto de las que no se tenía una base sólida en la que apoyarse, como la eliminación del Framework y, en menor medida, la creación de una base de datos que implementara BioAPI. Sin embargo, gracias a un elaborado proceso de diseño, dichos problemas fueron previstos antes de que aparecieran, lo que permitió la búsqueda de la solución más adecuada para cada uno.

El conjunto de la implementación de BioAPI Framework Free y su aplicación de prueba es el primer paso de lo que se espera que sea una nueva vía de desarrollo tecnológico, y por lo tanto no se planea su salida al mercado; se considera más bien como el apoyo necesario proporcionado a futuros desarrolladores que quieran hacer uso del enorme potencial que las aplicaciones biométricas móviles poseen. Además, este trabajo puede utilizarse como ilustración de las carencias que tiene actualmente el proyecto de norma de BioAPI Java, de forma que se pueda mejorar dicho proyecto y conseguir en un futuro próximo un estándar mucho más detallado y completo.

Por último, y como conclusión personal, el alumno partía desde un desconocimiento casi absoluto del tema a tratar, y concluye el trabajo con los conocimientos y habilidades suficientes para poder seguir investigando en el terreno de las aplicaciones biométricas, por lo que su consideración es que se han cumplido todos los objetivos, tanto técnico como personales, especificados al inicio del presente documento.

7.2. Líneas futuras de investigación

El presente proyecto permitirá a los desarrolladores con conocimientos Android diseñar sus propios sistemas biométricos respetando los estándares que exige la industria. Las posibilidades existentes en cuanto a aplicaciones son enormes. Por ejemplo:

- La policía científica podría comparar muestras biométricas descubiertas en la escena de un crimen in situ, sin tener que trasladarlas a un laboratorio, reduciendo de este modo las probabilidades de contaminarlas.

- Se podría comprobar la identidad de una persona en cualquier lugar, sin necesidad de trasladarla al sitio donde esté situado el sistema biométrico; esto aumentaría la comodidad de los trabajadores que tengan que emplear uno de estos métodos para acceder a su puesto de trabajo.

En materia de seguridad también se obtendrían ventajas, ya que podrían diseñarse cajas fuertes de tamaño reducido, funcionando en plataforma Android, que implementaran una prueba de acceso biométrico en sustitución de los métodos mecánicos actuales (o de forma añadida).

También podría ayudar a que las personas con alguna enfermedad mental degenerativa como el Alzheimer, que en ocasiones no recuerdan su nombre, sean identificadas rápidamente para poder avisar a sus familiares sin necesidad de trasladarlas a una comisaría, aumentando así su confusión.

Todos estos ejemplos son solo algunas de las opciones que deja abiertas este trabajo, que, como ya se ha dicho, es simplemente uno de los primeros pasos en esta dirección.

8. REFERENCIAS

[1] ERICSSON. *Ericsson predicts Mobile Data Traffic to grow 10-fold by 2016, Graph 15*. Ericsson Press Release, November 7, 2011. Disponible en http://www.ericsson.com/thecompany/press/releases/2011/11/1561267. (28 julio 2012).

[2] Handie Talkie H12-16. Recuperado de http://www.tuereselorigen.com/el-tatarabuelo-de-tu-smartphone/ (5 septiembre 2012)

[3] Motorola DynaTAC 8000X. Recuperado de http://www.taringa.net/posts/info/6855778/La-evolucion-de-los-celulares.html (5 septiembre 2012).

[4] MARTÍNEZ, Evelio. La historia de la telefonía móvil. *Revista RED*, Mayo 2001

[5] THE MOBILE WORLD, World Telephone Connections (Mb), 2011. Recuperado de http://www.totaltele.com/view.aspx?ID=464922 (5 septiembre 2012).

[6] Sistemas operativos móviles. (2009, 28 de abril [últ. mod. 2012, 10 de junio]). En *Wikipedia, la enciclopedia libre*. Disponible en: http://es.wikipedia.org/wiki/Sistema_operativo_m%C3%B3vil (29 julio 2012).

[7] ANDRADE, Jose. La gran comparación de los sistemas operativos móviles. *Engadget*, 19 de marzo de 2009. Disponible en http://es.engadget.com/2009/03/19/la-gran-comparacion-de-los-sistemas-operativos-moviles/. (29 de julio de 2012)

[8] Alphonse Bertillon. (2006, 20 de agosto [últ. mod. 2012, 12 de julio]). En *Wikipedia, la enciclopedia libre*. Disponible en: http://es.wikipedia.org/wiki/Alphonse_Bertillon (29 julio 2012).

[9] Historia de la biometría. En *Biometría*. Disponible en: http://www.biometria.gov.ar/acerca-de-la-biometria/historia-de-la-biometria.aspx (29 julio 2012).

[10] PEREZ, J.M. Biometrics error, 2005. Recuperado de http://commons.wikimedia.org/wiki/File:Biometrics_error.jpg?uselang=es (5 septiembre 2012).

[11] Biometría. (2005, 10 de octubre [últ. mod. 2012, 18 de julio]). En *Wikipedia, la enciclopedia libre*. Disponible en: http://es.wikipedia.org/wiki/Biometr%C3%ADa . (29 julio 2012).

[12] SÁNCHEZ REÍLLO, RAÚL. Identificación biométrica y su unión con las tarjetas inteligentes. Revista SIC. Vol 19. Agora. (Abril 2000)

[13] Arquitectura Android. Recuperado de http://droideka.blogspot.com.es/2012/01/entender-androidsus-inicios.html (5 septiembre 2012)

[14] Arquitectura de Android. (2011, 18 de octubre [últ. mod. 2012, 24 de julio]). En *Wikipedia, la enciclopedia libre*. Disponible en: http://es.wikipedia.org/wiki/Android#Arquitectura (30 julio 2012).

[15] BioAPI API/SPI Model. Del Text of 2nd Working Draft 19784-1, BioAPI - Part 1: Specification. American National Standards Institute. ISO/IEC WD 19784-1.2. 06 de septiembre, 2011.

[16] Biometric Information Record (serialized BIR for storage and transmission. Del Text of 2nd Working Draft 19784-1, BioAPI - Part 1: Specification. American National Standards Institute. ISO/IEC WD 19784-1.2. 06 de septiembre, 2011.

[17] AMERICAN NATIONAL STANDARDS INSTITUTE. *Text of 2nd Working Draft 19784-1, BioAPI - Part 1: Specification*. ISO/IEC WD 19784-1.2. 06 de septiembre, 2011.

[18] MATTHEW YOUNG. *History of the API and Relationship To Other Standards.* En *BioAPI Consortium Home Page*, junio 2005. Disponible en: http://www.bioapi.org/history.asp (6 agosto 2012)

[19] AMERICAN NATIONAL STANDARDS INSTITUTE. *Text of 3rd Working Draft 30106-2, Object Oriented BioAPI - Part 2: Java Implementtion*. ISO/IEC WD 30106-2. 08 de septiembre, 2011.

[20] Illustration of BSP architecture. Del Text of 2nd Working Draft 19784-1, BioAPI - Part 1: Specification. American National Standards Institute. ISO/IEC WD 19784-1.2. 06 de septiembre, 2011.

[21] BioAPI Installation Process. Del Text of 2nd Working Draft 19784-1, BioAPI - Part 1: Specification. American National Standards Institute. ISO/IEC WD 19784-1.2. 06 de septiembre, 2011.

[22] España. Ley orgánica 1/1992, de 21 de febrero, del Código penal. Boletín Oficial del Estado, 22 de febrero de 1992, núm. 46, p. 6209 – 6214.

[23] *Android NDK*. En *Android Developers*. Disponible en World Wide Web: http://developer.android.com/tools/sdk/ndk/index.html (6 agosto 2012)

[24] XPERTOS. *Versiones de Android - ¿Las conocías todas?*. En *Xpertos TV*, 2 de junio de 2012. Disponible en World Wide Web: http://www.xpertos.tv/tecnologia/Articulos/VistadeArt%C3%ADculos/tabid/221/Article/749/versiones-de-android-las-conocas-todas.aspx (7 agosto 2012)

[25] COSMOS. *Android Ice Cream Sandwich ya es la segunda versión más usada con el 15,9%, Jelly Bean logra el 0,8%*. En *Xataka*, 2 de agosto 2012. Disponible en: http://www.xatakandroid.com/mercado/android-ice-cream-sandwich-ya-es-la-segunda-version-mas-usada-con-el-15-9-jelly-bean-logra-el-0-8 (7 agosto 2012)

[26] Toast. Recuperado de http://developer.android.com/guide/topics/ui/notifiers/toasts.html (5 septiembre 2012)

[27] ASHWIN MOHAN, KEITH WATSON, SHIMON MODI. *BioAPI Java* [En línea]. (2 de junio, 2009 [últ. mod. 8 de diciembre 2009]). En *SourceForge - Download, Develop and Publish Free Open Source Software*. Disponible en http://sourceforge.net/projects/bioapijava/. (5 de marzo 2012)

[28] *Stack Overflow*. Disponible en http://stackoverflow.com/

[29] *Activity*. En *Android Developers*. Ref. de 9 agosto 2012. Disponible en http://developer.android.com/reference/android/app/Activity.html (5 septiembre 2012)

[30] *SQLite Database*. En *Android Developers*. Disponible en http://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html .(10 agosto 2012)

[31] ANDROIDER. *A simple example using Android's SQLite database, exposes data from Cursor to a ListView*. En *android-er*, 2 de junio 2011. Disponible en World Wide Web: http://android-er.blogspot.com.es/2011/06/simple-example-using-androids-sqlite_02.html (20 junio 2012)

[32] Logo BioAPI. Recuperado de http://www.secugen.com/company/index.htm (5 septiembre 2012)

ANEXO 1: PRESUPUESTO

En el presente anexo se calculará de forma precisa y detallando cada coste por separado el valor del proyecto presentado a lo largo del documento.

Costes de personal

Los costes de personal se han separado siguiendo dos parámetros diferentes: el primero, el tipo de personal, y el segundo, el apartado del proyecto. La primera división obedece a la diferencia de salarios existente entre el ingeniero y el director de proyecto, las dos personas que han intervenido en la creación. La segunda división es para detallar las horas empleadas de una forma más precisa, dada la amplitud del trabajo.

La división del proyecto en apartados se ha realizado siguiendo la siguiente estructura:

- Planteamiento: Incluye la familiarización con los entornos de desarrollo y los lenguajes de programación, la búsqueda de información sobre el proyecto y los tutoriales necesarios para llevarlo a cabo y el diseño de la solución.
- Desarrollo: Incluye los apartados de porte del código, desarrollo de la aplicación y eliminación del Framework.
- Pruebas: Incluye las pruebas en emulador y en dispositivo real.
- Redacción de la memoria: Incluye tanto la redacción como la corrección de la misma.

Tabla 1. Costes desglosados de personal

		Director de proyecto	Ingeniero
Planteamiento	Familiarización (h)	0	20
	Búsqueda información (h)	10	20
	Diseño (h)	0	45
Desarrollo	Porte de código (h)	0	30
	Desarrollo aplicación (h)	5	60
	Eliminación Framework (h)	5	60
Pruebas	Pruebas emulador (h)	0	30
	Pruebas dispositivo (h)	0	5
Escritura memoria	Redacción (h)	0	60
	Corrección (h)	15	10
Total	Total (h)	35	340
	Salario/hora	50	30
	Salario total (€)	1750	10200

Costes materiales

Los costes materiales incluyen los costes de hardware, los de software y los recursos de oficina o fungibles. Dentro del hardware se incluye un ordenador portátil de gama alta, necesario para ejecutar el entorno de desarrollo y para la búsqueda de información y un móvil Sony Ericsson Xperia X8, indispensable para las pruebas en dispositivo real. Ambos dispositivos tienen un plazo de amortización de tres años, y se han tenido en cuenta los dos meses (340 horas) de realización del proyecto. En cuanto a recursos fungibles, únicamente hay que tener en cuenta el acceso a internet. No se han contabilizado gastos de software ya que todo el que se ha empleado es libre y gratuito.

Tabla 2. Costes desglosados de material

			Coste proyecto
Hardware	Ordenador	900 (€ total)	50
	Móvil	115 (€ total)	6,4
Software	Software	0	0
Fungibles	Internet (€/mes)	30 (€/mes)	60
Total (€)			116,4

Total

Sumando los dos apartados anteriores, añadiéndole los costes indirectos, de un veinte por ciento, y el IVA, de un veintiuno por ciento, el precio total del proyecto asciende a *diecisiete mil quinientos veinte euros con cuarenta y un céntimos.*

Tabla 3. Costes desglosados totales

	Coste (€)	
Personal	Director: 1750	11950
	Ingeniero: 10200	
Materiales	116,4	
Costes indirectos (20%)	2413,28	
Subtotal	14479,68	
IVA (21%)	3040,73	
Total	17520,41	

Leganés, 5 de Septiembre de 2012

El ingeniero