# FINGERPAY

Pablo Fernández Grado
Francisco Javier Sánchez Platero
Robert Slavi Marinov

**GRADO EN INGENIERÍA INFORMÁTICA**
**FACULTAD DE INFORMÁTICA**
**Dpto. Arquitectura de Computadores y Automática**

UNIVERSIDAD COMPLUTENSE DE MADRID


TRABAJO FIN DE GRADO EN TECNOLOGÍA DE LA INFORMACIÓN

Junio 2016

**Director:**
José Luis Vázquez Poletti

**Codirectores:**
Eva Ullán Hernández
José Manuel Velasco Cabo

# Autorización de difusión y utilización

**AUTORIZACIÓN PARA LA DIFUSIÓN DEL TRABAJO FIN DE GRADO Y SU DEPÓSITO EN EL REPOSITORIO INSTITUCIONAL E-PRINTS COMPLUTENSE**

Los abajo firmantes, alumno/s y tutor/es del Trabajo Fin de Grado (TFG) en el Grado en ……de la Facultad de ……, autorizan a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el Trabajo Fin de Grado (TF) cuyos datos se detallan a continuación. Así mismo autorizan a la Universidad Complutense de Madrid a que sea depositado en acceso abierto en el repositorio institucional con el objeto de incrementar la difusión, uso e impacto del TFG en Internet y garantizar su preservación y acceso a largo plazo.

Periodo de embargo (opcional):

- ☐ 6 meses
- ☐ 12 meses

TÍTULO del TFG: .....
……
Curso académico: 20…... / 20……

Nombre del Alumno/s:
……
……
……

Tutor/es del TFG y departamento al que pertenece:
……
……
……

Firma del alumno/s Firma del tutor/es

# Dedicatoria

Quiero dedicar este trabajo toda mi familia, en especial a mi abuelo, al que he perdido durante la realización del mismo. A mis padres, gracias a ellos he conseguido llegar hasta donde he llegado, tanto académica como personalmente. A mi hermana, a mi cuñado y a la pequeña de la familia, que siempre tiene una 'risita' preparada.

Agradezco a mi amigo Carlos Quevedo por el préstamo del dispositivo lector de huellas, objeto principal para la realización de nuestro proyecto.

Y por último gracias a Silvia por sus ánimos, su apoyo y su ayuda durante toda la carrera. Ella siempre ha sido la mano que me ayudaba a levantarme cada vez que tropezaba.

Pablo

Quiero dar las gracias a todas aquellas personas que me han acompañado durante todos estos años (que ya son unos cuantos). Amigos, abuelos y en especial a mis padres y hermana que día tras día me han ayudado y apoyado en todo.

Javi

Por último queremos agradecer a nuestros tutores Eva, José Manuel y, especialmente, José Luis por su ayuda, dedicación y consejos en la realización del trabajo.

# Prólogo

En un mundo digital como el nuestro, los avances tecnológicos intentan abarcar todos los aspectos del día a día del ser humano. La tecnología se ha vuelto conveniente en procesos tan habituales como el pago de productos y servicios. Pero, ¿podemos también afirmar que sea segura?

Lo último en pago electrónico ahora mismo bascula entre las tarjetas *contactless* (aquellas en la que basta acercarlas al lector) y el teléfono móvil. La seguridad de ambas soluciones depende de dos aspectos principales: la propia seguridad del elemento tecnológico (robo, *hackeo*) y la unicidad del elemento identificativo (clonado).

FingerPay, el Trabajo Fin de Grado de Robert, Pablo y Francisco Javier, propone un giro de 180 grados en cuanto a pago seguro. La propia naturaleza nos ha dotado de un elemento identificativo único e irrepetible: las huellas dactilares. Incluso compartiendo ADN (gemelos, mellizos), es muy raro que dos individuos compartan huellas. Esto se debe a que cada hermano ocupa una posición distinta en el útero durante la gestación, recibiendo por tanto las fuerzas intrauterinas de forma diferente. La concentración de algunas hormonas y los factores de crecimiento inducen también a variaciones en el dibujo.

Pero el trabajo de estos tres "ingenieros de facto" no se ha limitado a implementar un sistema de pago usando la huella dactilar, cuya simplicidad en la gestión lo vuelve conveniente. Su propuesta abarca además el almacenamiento seguro de las credenciales a través de un ingenioso sistema de nubes de almacenamiento federadas, permitiendo así la división y deslocalización de los datos.

Sin miedo a equivocarme, puedo afirmar que el futuro de los pagos electrónicos está aquí y se llama FingerPay.

J.L. Vázquez Poletti

# Índice

Prólogo ..... VII
Índice de figuras ..... X
Índice de abreviaturas ..... XI
Resumen ..... XIII
Abstract ..... XIV
Capítulo 1. Introducción ..... 1
1.1 Antecedentes ..... 1
1.2 Objetivos ..... 3
1.3 Plan de trabajo ..... 3
Chapter 1. Introduction ..... 4
1.1 Background ..... 4
1.2 Objectives ..... 6
1.3 Work plan ..... 6
Capítulo 2. Materiales y métodos ..... 7
2.1 Hardware ..... 7
2.2 Lenguajes de programación ..... 7
2.3 Servidores ..... 7
2.4 Programas para el desarrollo ..... 7
2.5 Bases de datos ..... 8
2.6 Modelo de trabajo ..... 8
Capítulo 3. Resultados ..... 9
3.1 Registro ..... 9
3.2 Terminal de pago ..... 11
3.3 Tratamiento de huellas ..... 13
3.4 Base de datos de usuarios ..... 27
Capítulo 4. Trabajo realizado por cada componente del grupo ..... 33
Capítulo 5. Tecnologías descartadas. ..... 40
Capítulo 6. Discusión y conclusiones ..... 41
6.1 Discusión ..... 41
6.2 Conclusiones ..... 43
6.2 Conclusions ..... 43
Apéndice ..... 45
Apéndice 1. Diagramas de flujo ..... 45
Apéndice 2.Diagrama general de la aplicación ..... 52
Bibliografía ..... 53
Anexos ..... 57
Anexo 1: Métodos de pago con huellas dactilares. ..... 57
Anexo 2: Ley Orgánica de Protección de Datos ..... 59
Anexo 3: Estadísticas de uso de tarjeta ..... 60

# Índice de figuras

Imagen 1 - Tipos de huellas ..... 1
Imagen 2 - Características de la huella ..... 2
Imagen 3 - Fingerprint types ..... 4
Imagen 4 - Fingerprint characteristics ..... 5
Imagen 5 - Lector de huellas ..... 7
Imagen 6 - Diagrama general de la arquitectura de la aplicación ..... 9
Imagen 7 - Registro (I) ..... 9
Imagen 8 - Registro (II) ..... 10
Imagen 9 - Terminal (I) ..... 11
Imagen 10 - Terminal (II) ..... 11
Imagen 11 - Terminal (III) ..... 11
Imagen 12 - Terminal (IV) ..... 12
Imagen 13 - Terminal (V) ..... 12
Imagen 14 - Terminal (VI) ..... 12
Imagen 15 - Árbol de directorios ..... 14
Imagen 16 - Árbol de directorios del usuario ec2-user ..... 16
Imagen 17 - Estructura del directorio ..... 18
Imagen 18 - Cifrado de huella ..... 24
Imagen 19 - Particionado de huella ..... 25
Imagen 20 - Distribución de archivos ..... 25
Imagen 21 - OpenShift® ..... 28
Imagen 22 - Diagrama Entidad-Relación ..... 29
Imagen 23 - Tablas de la base de datos ..... 29
Imagen 24 - Directorio de la aplicación de conexión a base de datos ..... 31
Imagen 25 - Ejecución de dos procesos simultáneos en Amazon ..... 32
Imagen 26 - Primer software para recolecta de huellas ..... 33
Imagen 27 - Modificación del software para recolecta de huellas ..... 34
Imagen 28 - Boceto del registro ..... 34
Imagen 29 - Boceto del terminal ..... 35
Imagen 30 - Direcciones locales generadas para conectarse a OpenShift® ..... 36
Imagen 31 - Prototipo TPV ..... 41
Imagen 32 - Diagrama de flujo de tratamiento de imágenes de *watchfile* ..... 45
Imagen 33 - Diagrama de flujo del proceso de tratamiento de texto plano de watchfile ..... 46
Imagen 34 - Diagrama de flujo del proceso de tratamiento de ficheros de *watchfile*_forward 47
Imagen 35 - Diagrama de flujo del proceso de tratamiento de ficheros de watchfile_db ..... 48
Imagen 36 - Diagrama de flujo de un registro ..... 49
Imagen 37 - Diagrama de operación de pago ..... 50
Imagen 38 - Diagrama de flujo de operaciones en base de datos ..... 51
Imagen 39 - Diagrama general de la arquitectura de la aplicación ..... 52
Imagen 40 - Gráfica de extracciones/operaciones * ..... 60
Imagen 41 - Gráfica cajeros, TPVs y tarjetas * ..... 60

# Índice de abreviaturas

| | |
|---|---|
| Dpto. | Departamento |
| SSH | Secure SHell |
| NFC | Near Field Communication |
| ISO | International Organization of Standarization |
| ANSI | American National Standard Institute |
| INCITS | InterNational Committee for Information Technologies Standards |
| WSQ | Wavelet Scalar Quantization |
| NIST | National Institute of Standards and Technologies |
| MINEX | Minutiae Interoperability Exchange |
| NFIQ | NIST Fingerprint Image Quality |
| SDK | Software Development Kit |
| JDK | Java Development Kit |
| JDBC | Java Database Connectivity |
| SQL | Structured Query Language |
| EC2 | Elastic Cloud Computing |
| MVC | Model View Controller |
| IaaS | Infrastructure as a Service |
| PaaS | Platform as a Service |
| SaaS | Software as a Service |
| DNI | Documento Nacional de Identidad |
| GNU | GNU Not Unix |
| API | Application Programming Interface |
| AES | Advanced Encryption Standard |
| SO | Sistema Operativo |
| DNS | Domain Name System |
| rhc | red hat client |
| CUDA | Compute Unified Device Architecture |
| TPV | Terminal de Punto de Venta |
| LOPD | Ley Orgánica de Protección de Datos |

# Resumen

Estamos acostumbrados a pagar en los supermercados y comercios con tarjeta de crédito. Para ello necesitamos una tarjeta bancaria y un documento identificativo.

Hemos unido estos dos requisitos en uno de cara a hacer el pago más sencillo, rápido y cómodo. La solución que proponemos es registrar nuestra huella dactilar y asociarla a una cuenta bancaria, de forma que nuestro dedo sea suficiente para pagar.

Para ello, hemos desarrollado varios programas, para implementar los distintos puntos del proceso:

- Registro
- Terminal de punto de venta
- Verificación y tratamiento de huellas dactilares
- Consultas en la base de datos.

También hemos utilizado una base de datos en Dropbox™ para el almacenamiento de las imágenes de las huellas, todas ellas encriptadas, y otra relacional para los datos de los clientes. Para la intercomunicación dentro de la aplicación se han utilizado canales seguros SSH.

Con este nuevo método de pago bastará poner tu dedo en el lector de huellas para que el pago se realice automáticamente.

**Palabras clave:** Amazon Web Services, biometría, Dropbox™, FingerPay, huella dactilar, nube, OpenShift®, pago, sistema distribuido, SSH.

# Abstract

We are used to pay in a supermarket and other shops with a credit card, for that we need a credit card and an identification card. We joined these requirements to do the payment easier, faster and more comfortable. Our solution was to register the fingerprint linked to a bank account, so we only need our finger to pay.

We developed some software to implement all steps in the process:

- Register.
- Payment terminal.
- Fingerprint verification and treatment.
- Database queries.

We also used a database in Dropbox™ to save fingerprints encrypted images, and a relational database to save client data. We used SSH connection to the intercommunication within the app.

In this way, you only need to put your finger on the fingerprint reader and the payment will be made automatically.

**Keywords:** Amazon Web Services, biometric, cloud, distributed system, Dropbox™, FingerPay, fingerprint, OpenShift®, payment, SSH

# Capítulo 1. Introducción

## 1.1 Antecedentes

Hasta ahora los métodos de pago empleados en los comercios han sido con banda magnética, con chip, sin contacto y con *wallets* para dispositivos móviles, utilizando los dos últimos tecnología NFC. Para todos ellos se necesita además una identificación que lo acompañe.

Estos métodos tienen un gran problema, si alguien roba una tarjeta bancaria, puede hacer compras en nombre de otra persona sin permiso; evitarlo es uno de nuestros objetivos.

En el mundo de la identidad, decimos que ésta es robusta cuando se mezclan al menos dos de los siguientes términos. Algo que se conoce, algo que se tiene, algo que se es. Nosotros hemos elegido el pin como algo que se conoce y la huella como algo que se es.

### Tipos de dibujos en las huellas

Las huellas tienen 3 sistemas de crestas.

- Marginal, en la parte superior.
- Basilar, en la parte inferior.
- Nuclear en la parte central. Las huellas adeltas no tendrían esta cresta.

Cuando existen los tres sistemas de crestas en una huella podemos encontrar también un delta.

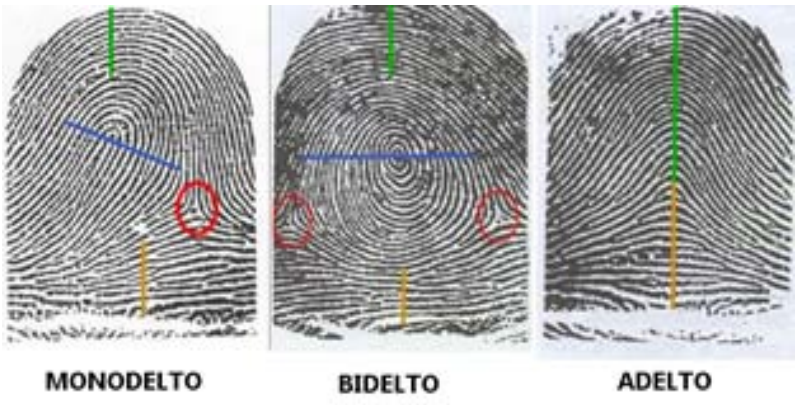


**Imagen 1 - Tipos de huellas**

Las huellas monodelta tienen un solo delta. Puede ser dextrodelto si está a la derecha o sinistrodelto si está a la izquierda. Como curiosidad, se puede decir que las monodeltas son las más habituales y que en la mano derecha solemos tener sinistrodelto y en la izquierda dextrodelto.

En las bidelta podemos encontrar dos deltas, uno a cada lado. La espiral central puede ser única como en la imagen o de doble espiral

Las últimas son las adelta en las que no aparece ni el núcleo ni el delta. En definitiva, según la forma de las crestas podemos encontrar tres tipos de huellas y cada tipo tiene dos subtipos.

- Monodelto
    - Bucle izquierdo (sinistrodelto)
    - Bucle derecho (dextrodelto)

- Bidelto
    - Espiral
    - Doble espiral
- Adelto
    - Arco
    - Arco entoldado

A simple vista, dos huellas del mismo tipo pueden parecer idénticas, pero realmente no es así. Para poder distinguirlas, necesitamos fijarnos en los puntos característicos o minucias, los cuales son los siguientes:

- **Empalme**: unión de dos crestas que van paralelas y las une un fragmento en forma de ángulo.
- **Convergencia**: dos crestas que van en sentido horario y se unen formando una sola.
- **Ojal**: es la unión de una convergencia y una bifurcación la cual forma un círculo en medio de las crestas.
- **Desviación**: dos crestas que van en sentido contrario y en el momento de cruzarse se desvían sin llegar a tocarse.
- **Transversal**: una cresta que pasa por medio del hueco que deja otra con la que se cruza.
- **Fragmento**: pequeña sucesión de puntos que forman una línea, la cual debería tener más de largo que de ancho sin exceder 7 veces de largo lo que tiene de ancho.
- **Abrupta**: una cresta cuyo recorrido termina y no vuelve a aparecer.
- **Bifurcación**: una única cresta en sentido horario y que se separa en dos.
- **Interrupción**: discontinuidad de una cresta dactilar la cual está formada por puntos y fragmentos con la condición de que continúe su recorrido.
- **Punto**: forma parte de una interrupción.
- **Vuelta**: fragmento de cresta que gira interrumpiendo su recorrido.
- **Cuña**: para que se configure este punto característico deberá estar formado por la vuelta de dos crestas paralelas, consistente en que un pedazo de cresta ve interrumpido su camino y evita su recorrido o continuidad.
- **Rama**: unión o función de varias convergencias o bifurcaciones, la cual se encuentra con mayor frecuencia en la parte bacilar del dactilograma.


**Imagen 2 - Características de la huella**

## 1.2 Objetivos

Nuestros objetivos principales son desarrollar un método seguro de pago, que en caso de ser interceptado, sea imposible de interpretar; con el que nadie te pueda suplantar; fácil de utilizar por cualquier persona sin tener en cuenta sus conocimientos tecnológicos; más cómodo que la tarjeta bancaria y que sea rápido en efectuar el pago.

## 1.3 Plan de trabajo

Fases del trabajo:

Se comenzó con una fase de investigación, repartiendo el trabajo de manera que cada uno pudiera buscar información acerca de su parte principal en el proyecto. Aunque se repartiera el trabajo, todos los componentes del grupo debían estar bien informados de todo el proyecto. Se encontraron algunas noticias y desarrollos relacionados con el pago mediante huellas dactilares, sobre los que se habla en los anexos.

Después se procedió a planificar el tiempo disponible para realizar el proyecto. Se tuvieron reuniones para detallar cómo se quería hacer el trabajo, qué partes iba a tener y cómo desarrollarlo.

El siguiente paso consistió en el desarrollo de las diferentes partes del trabajo por separado, para una vez acabadas, poder conectarlas montando los servidores.

Una vez finalizado el desarrollo de la aplicación se procedió a la creación de los servidores y volcado del código en ellos.

Para terminar se realizó la conexión de todas las partes del trabajo y las pruebas reales sobre las máquinas.

Métodos de comunicación a distancia utilizados:

- Skype™
- Whatsapp
- Google Drive
- Dropbox™

Reuniones

Se han realizado reuniones con el director y los codirectores del Trabajo de Fin de Grado los lunes a las 15:30h. En estas reuniones se explicaban los progresos sobre el trabajo realizado y se hablaba sobre posibles mejoras, alternativas, soluciones a problemas encontrados, otros caminos a tomar sobre ciertas decisiones, etc.

Los alumnos por nuestra parte, tuvimos más reuniones donde cada uno compartía sus avances con el resto y se decidían los siguientes pasos a tomar.

# Chapter 1. Introduction

## 1.1 Background.

Nowadays, payment methods used in shops use magnetic stripes, chips, mobile devices… For all of them an extra identification method is required. These methods represent a big problem.

If someone steals a bank card, shopping can be made in the name of the legitimate owner without permission. Solving this problem is one of our goals.

Identity is stronger while mixing some of the following: something you know, something you have and something you are. We chose a PIN as the first and the fingerprint as the third.

### Fingerprints draw types

Fingerprint has 3 systems of ridges:

- Marginal, on top
- Basilar, at the bottom
- Nuclear in the middle. The adeltas fingerprints haven't got these ridges.

When there are three systems of ridges in a fingerprint we can also find a delta.

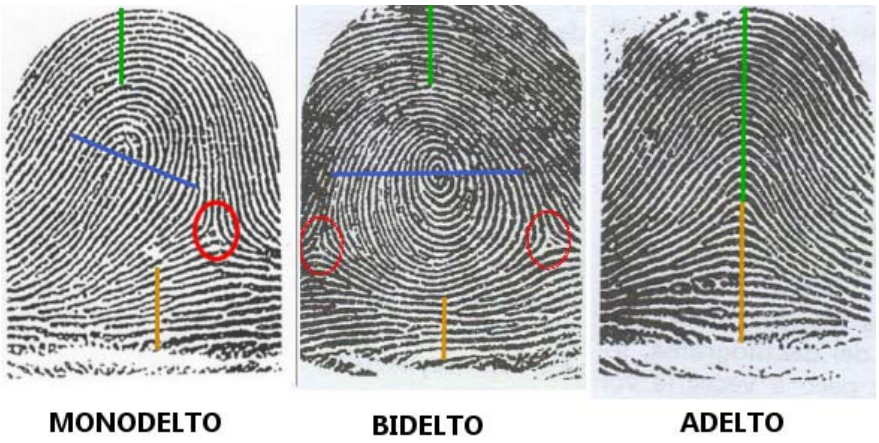


Imagen 3 - Fingerprint types

The monodelta fingerprints have a single delta. It may be dextrodelto if it's at the right or sinistrodelto if it's at the left. As a curiosity, we can say that monodeltas are the most common and in the right hand usually have sinistrodelto and in the left hand we usually have dextrodelto.

In bidelta fingerprints we can find two deltas, one on each side. The central spiral may be unique as in the image or double spiral are the latest. The latter are the adelta that appears neither the core nor the delta.

Definitely, according to the shape of the fingerprints we can find 3 types of fingerprints and each type has two subtypes.

- Monodelto
    - left loop (sinistrodelto)
    - right loop (dextrodelto)

- Bidelto
    - Spiral
    - Double Spiral
- Adelto
    - Arc
    - Tented arch

With a naked eye, two fingerprints of the same type may look identical, but this is not so. To distinguish need to look at the minutia or minutiae. These are:

- Splice: union of two ridges that run parallel and join a fragment in the form of angle.
- Convergence: two ridges that are clockwise and join to form one.
- Buttonhole: It is the union of a convergence and a bifurcation which forms a circle in the middle of the ridges.
- Deviation: two ridges that go in the opposite direction and at the time of crossing deviate without touching
- Cross: a ridge passing through gap left by another which intersects
- Fragment: Small sequence of points which form a line, that line should be longer than wide.
- Abrupt: a ridge whose tour ends and does not reappear.
- Fork: a single ridge in clockwise and it separates into two.
- Interruption: It's the discontinuity of a fingerprint ridge which is formed by dots and fragments provided continue its journey.
- Point: a point that is part of an interruption
- Return: Fragment of ridge that rotating interrupting its route
- Cradle: It consists of two parallel round ridges. It is that a piece of crest interrupted his way and prevents its route or continuity.
- Branch: Union or function of several convergences or bifurcations that it is most often in the basilar part of dactylogram.


**Imagen 4 - Fingerprint characteristics**

## 1.2 Objectives

Our principal objectives are to develop a payment secure method, that in case to intercept, it will be impossible to interpret; With nobody can replace you; Make it easy to use by anyone; That will be more comfortable that the credit card and will be quick in the payment.

## 1.3 Work plan

It started with a research phase, dividing the work so that everyone could find information about the main part in the project. Although the work would be shared, all members of the group should be fully informed of the whole project. Some news and developments related to payment fingerprint on which it is spoken in the chapter were found.

### Remote communication methods used:

- Skype™
- Whatsapp
- Google Drive
- Dropbox™

### Meetings

It has been realized meetings with the director and co-directors of the Final Grade on Mondays at 15:30h. At these meetings the progress on the work were explained and talked about possible improvements, alternatives, solutions to problems encountered, other paths to take over certain decisions, etc.

Students meanwhile, had more meetings where everyone shared their progress with the rest and the next steps were decided.

# Capítulo 2. Materiales y métodos

## 2.1 Hardware

Hemos empleado un lector de huellas de la marca DigitalPersona®, concretamente el modelo U.are.U 4500 el cual cumple con certificaciones: ISO 19794-4:2005, ANSI/INCITS 381-2004, ISO/IEC 19794-2:2005, ANSI INCITS 378-2004, WSQ, MINEX, NFIQ. El dispositivo ofrece un enrolamiento y una verificación interna.

Nosotros hemos decidido utilizar nuestro método propio haciéndolo así más seguro ya que el terminal lo único que hace es capturar una imagen del dedo sin necesidad de tratarla.

**Imagen 5 - Lector de huellas**

## 2.2 Lenguajes de programación

Python 2.7.11, Java 7 y Java 8, SQL

Python ha sido utilizado para el tratamiento y comparación de imágenes en el servidor.

Java lo hemos empleado para la realización del software con interfaz gráfica, el simulador del terminal de punto de venta y el programa para registrarse, y para el pequeño software que hace consultas en la base de datos. Java nos ofrecía compatibilidad para desarrollar con el lector de huellas, que Python por ejemplo no.

La creación de base de datos, y sus posteriores consultas, se ha desarrollado con SQL.

Para la conexión desde Java a la base de datos se ha utilizado la librería *mysql-connector-java-5.1.39*.

## 2.3 Servidores

Amazon Web Services con Elastic Cloud Computing (EC2)

Amazon Web Services nos ofrecía 750 horas al mes durante 12 meses de forma gratuita de instancias EC2 en Linux por cada cuenta. Ofrece muchos más elementos que no eran necesarios en nuestro caso.

## 2.4 Programas para el desarrollo

Eclipse, PyCharm

Estos dos programas han sido los utilizados para desarrollar el software necesario para nuestra aplicación de pago. Ambos de libre distribución.

## 2.5 Bases de datos

OpenShift®, Dropbox™

Tenemos una base de datos relacional en OpenShift®, en la que almacenamos los datos de los clientes y sus cuentas.

Para el almacenamiento de las huellas, la tecnología utilizada ha sido la de Dropbox™. Guardamos imágenes partidas en trozos y cifradas para que, si se ve comprometida, no se pueda obtener ningún tipo de información válida para el atacante.

## 2.6 Modelo de trabajo

Para el desarrollo de las dos aplicaciones con interfaz gráfica, el modelo utilizado ha sido el Modelo-Vista-Controlador (MVC), siendo así más sencilla la adaptación en caso de un cambio de interfaz gráfica, por ejemplo.

El *cloud computing* tiene tres modelos, infraestructura como servicio (IaaS), plataforma como servicio (PaaS) y software como servicio (SaaS). Hemos conseguido complementar los tres modelos:

IaaS → Amazon Web Services

PaaS → OpenShift®

SaaS → Dropbox™

La conexión entre todas las partes de la arquitectura se realiza mediante canales SSH para mantener en todo momento la seguridad y el compromiso de los datos, de tal forma que nadie ni nada sin permisos, pueda leer los datos que viajan de un lado a otro.

# Capítulo 3. Resultados

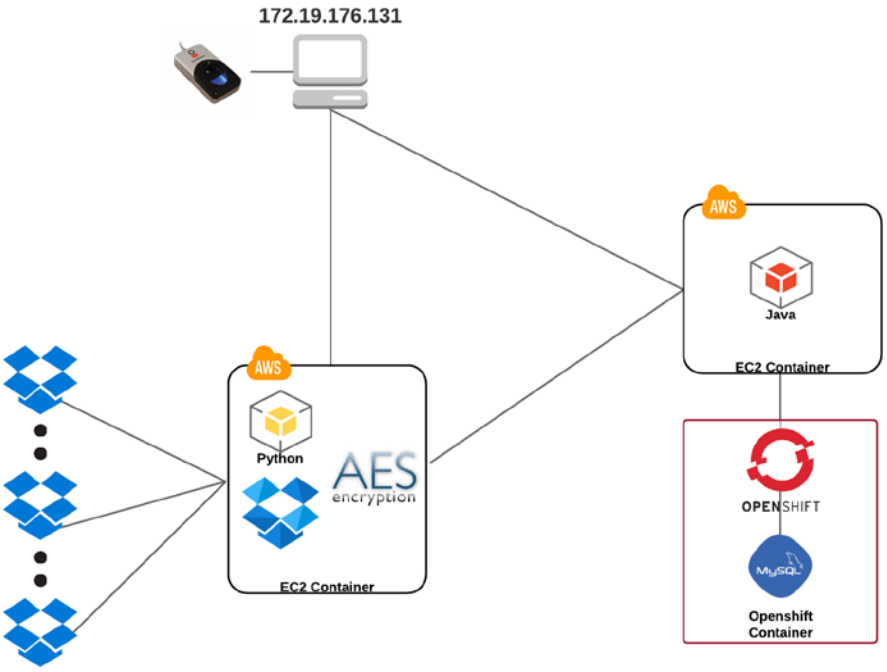


**Imagen 6 - Diagrama general de la arquitectura de la aplicación.**

## 3.1 Registro

Para realizar el registro bastará ejecutar el archivo 'FingerPayRegistro.jar'.

El registro tiene una primera ventana en la que introducir algunos de los datos del usuario que se muestran en la imagen, de los que el teléfono es opcional.

El botón con la imagen del cepillo permite borrar el contenido de todos los campos.

El botón con la imagen de la flecha de color nos permite continuar al siguiente paso.

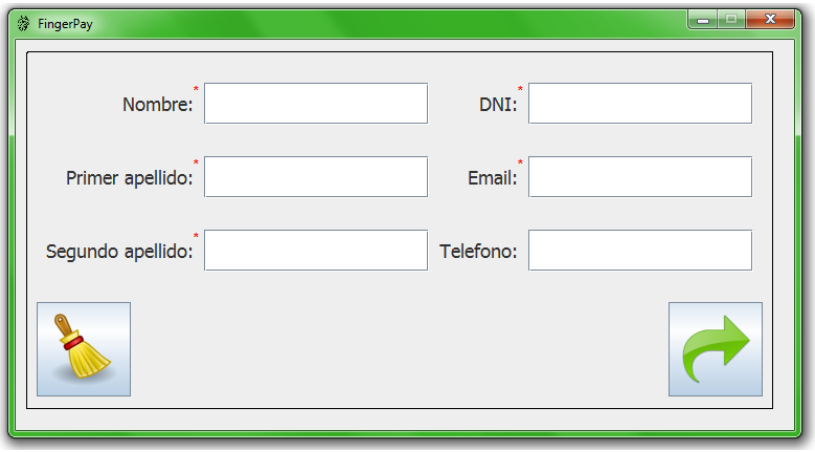


**Imagen 7 - Registro (I)**

Al pulsar en la flecha verde se hace automáticamente una consulta a la base de datos para saber si este usuario está ya registrado o no, debido a que actualmente solo se permite un registro por persona.

- Si está registrado, aparece un mensaje informando de que no se puede continuar con el proceso de registro.
- Si no está registrado, sale una ventana nueva donde podremos continuar con el registro.

En la siguiente ventana (Imagen 5) se puede ver un recuadro blanco donde, una vez haya sido leída la huella, aparecerá impresa. A su derecha, el botón naranja sirve para eliminar la huella y poder registrarla de nuevo en el caso de que el intento no haya salido como se deseaba.

En este paso también es necesario añadir un número pin para las futuras operaciones. Si queremos borrar el pin introducido para cambiarlo, se puede hacer con el botón 'borrar'.

Al igual que en la ventana anterior, el botón con la imagen del cepillo lo borra todo.

Existen dos botones con flecha de color verde. La de la izquierda lleva a la ventana anterior respetando los datos previamente introducidos, y la de la derecha permite continuar con el siguiente paso.


**Imagen 8 - Registro (II)**

Al pulsar sobre la flecha verde se envían los datos de la primera ventana y los de la segunda para guardarlos en las distintas bases de datos.

La conexión con ambas máquinas alojadas en Amazon, se realiza mediante SSH con el comando 'pscp'.

## 3.2 Terminal de pago

Para ejecutar el terminal de pago bastará ejecutar el archivo 'FingerPayTerminal.jar'.

La interfaz del proceso de pago (Imagen 6) se ha desarrollado de una forma muy minimalista para que sea fácil de utilizar por cualquier tipo de usuario, sean cuales sean sus conocimientos tecnológicos. Cabe destacar que no es necesario el uso de ratón, ya que es una reproducción de un terminal de punto de venta. Representa un terminal de pago como los actuales de tarjetas bancarias.

En la parte superior se introducirá, mediante un teclado numérico, el precio de la transacción.


**Imagen 9 - Terminal (I)**

A continuación, se pulsa 'intro' (lo que podría corresponder al botón verde del terminal) y la imagen de la huella que está de color gris pasará a verse de color naranja (Imagen 7), lo que significa que está a la espera de leer la huella del comprador.


**Imagen 10 - Terminal (II)**

Es el momento de que el comprador sitúe el dedo que previamente ha registrado, como se indica en la sección anterior, en el lector de huellas.

Si se lee correctamente, aparecerá de color verde y si no, de color rojo (Imagen 8). Esto no quiere decir que la transacción haya sido correcta o no, sino que la lectura de huella ha ido bien o no.


**Imagen 11 - Terminal (III)**

Posteriormente, solo cuando sea verde, se activa la introducción del pin. El usuario deberá teclear su código numérico, al igual que se hace actualmente con el pin de la tarjeta bancaria.

El pin va codificado con puntos para evitar que otras personas puedan visualizarlo, al igual que ocurre con los actuales (Imagen 9).

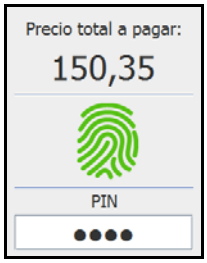


**Imagen 12 - Terminal (IV)**

Una vez hechos estos tres sencillos pasos, se envían los datos al servidor para ser verificados. Mientras tanto, se muestra una imagen que nos indica que la operación se está procesando (Imagen 10). Es una imagen en movimiento para indicar que la aplicación está trabajando.


**Imagen 13 - Terminal (V)**

Una vez se obtenga la respuesta, se muestra el resultado (Imagen 11), que puede ser:

- Correcto (tic verde): la huella y el pin han sido verificados correctamente y el pago se ha realizado.
- Erróneo (aspa roja): la huella y/o el pin no han podido ser verificados y por tanto el pago no se ha realizado.
- Tiempo excedido de espera (esfera azul con aspa roja) que puede tener lugar en dos situaciones:
    - La respuesta tarda demasiado en llegar al terminal.
    - El servidor detecta que la consulta a la base de datos de huellas está siendo demasiado larga y avisa al terminal.

En cualquier caso, la respuesta obtenida se muestra durante 5 segundos. Después desaparecerá para volver a visualizar la ventana inicial.


**Imagen 14 - Terminal (VI)**

La conexión a la máquina alojada en Amazon, se realiza mediante SSH con el comando 'pscp'.

## 3.3 Tratamiento de huellas

Características y estructura del sistema:

El sistema de tratamiento de huellas, se encarga de extraer, distribuir y manejar los datos de las huellas que le son enviados.

El sistema está diseñado para ser desplegado haciendo un número de modificaciones mínimas en el *host*. Está implementado para ser ejecutado en sistemas operativos basados en Linux, independientemente de su arquitectura (32 o 64 bits).

Los requisitos mínimos del *host* son:

- Kernel Linux > 3.18
- Herramientas GNU: cat, split, mv
- Openssl
- Python 2.7 (> 2.7.5)

El sistema ha sido probado en:

- Manjaro(arch) 64 bits
- Ubuntu 64 y 32 bits
- Debian 64 bits
- CentOS 64 bits

Estructura de código del sistema de tratamiento de huellas:

La mayor parte del funcionamiento del sistema se basa en operaciones de entrada/salida (lectura/escritura sobre fichero) y de tratamiento de ficheros. En este caso, el tiempo de los programadores y la claridad del código, prima frente al rendimiento del lenguaje.

La sintaxis de Python, la herencia de C y la gran cantidad de paquetes que ofrece, proporciona soluciones eficientes y simples a nuestras necesidades.

Utilizamos dos APIs Python de terceros para implementar el sistema de tratamiento de huellas:

- La *API* de Dropbox™ para comunicar y sincronizar los datos en la nube.
- El paquete distribuido por Google, *watchdog*, para monitorizar directorios y realizar acciones ante determinados cambios en los mismos.

Hemos aprovechado la modularidad de Python para estructurar el código, haciéndolo más fácil de mantener y adaptar:

- Mantenibilidad: Se mantienen separados los módulos que implementan la funcionalidad completa del sistema de los módulos que implementan una funcionalidad parcial.
- Adaptación: Flexibilidad ante posibles cambios que puedan surgir en las *APIs* externas, o incluso cambiarlas por otras en caso de optar por otros distribuidores, además de poder cambiar las nuestras.

## Estructura y permisos de directorios del sistema:

Para conseguir aislamiento y seguridad de datos, el sistema hace uso de los permisos de usuario que ofrece Linux.

El sistema tiene tres usuarios:

- **ec2-user**: usuario con privilegios, encargado de manejar el sistema. En su directorio raíz se encuentra la lógica del sistema, encargándose de organizar al resto de usuarios.
- **db_user**: usuario utilizado por el sistema de bases de datos para transmitir los resultados a nuestro sistema.
- **guest**: usuario utilizado por los terminales y puntos de acceso para transmitir y recoger información de nuestro sistema.

El sistema de base de datos (db_user) y el de tratamiento de huella (ec2-user) tienen la misma jerarquía en el sistema global: la comunicación entre ellos es entre iguales, aunque el usuario db_user tiene acceso limitado y supervisado al sistema de tratamiento de huellas.

Los terminales no forman parte de la misma jerarquía que los sistemas de arriba, por tanto su acceso al sistema de tratamiento de huellas es más restringido.

## Directorios:

A nivel de */home* se encuentran los directorios que son utilizados por el sistema para llevar a cabo sus tareas (Imagen 12).

Los directorios se pueden clasificar en dos grupos:

- **watchdog**: son escuchados y ante cambios registrados, se lanzan operaciones
- **carpetas de los usuarios**: contienen los ficheros necesarios para que cada usuario lleve a cabo su labor.

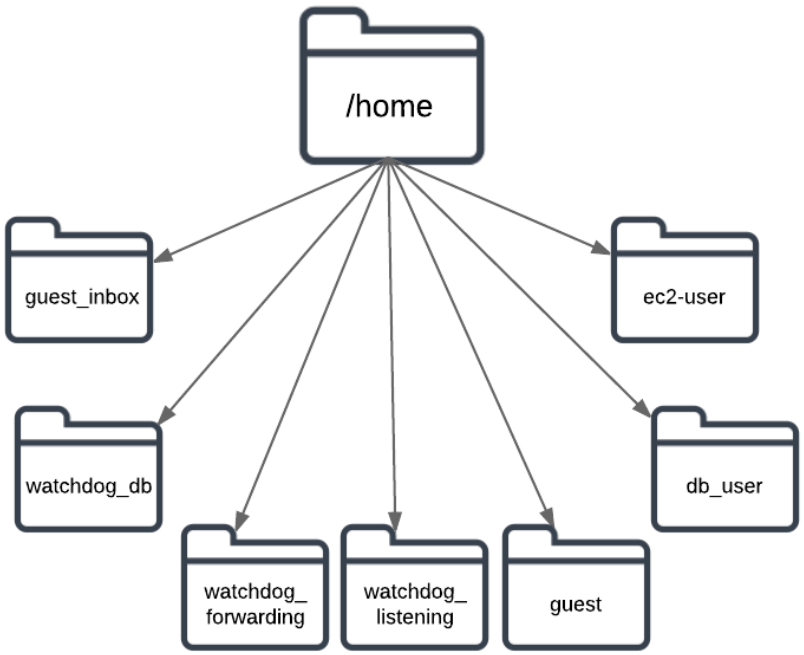


**Imagen 15 - Árbol de directorios**

Watchdogs:

**'Watchdog listening':**

Directorio monitorizado a la espera de ficheros entrantes procedentes de los terminales. Dependiendo del tipo de fichero y extensión, se llevan a cabo distintos tipos de operación (registro/carga). En el caso de no coincidir con el tipo de fichero admitido, no se lanzan operaciones y los ficheros son borrados.

| permisos | propietario | grupo |
|---|---|---|
| **drwx-wx---** | **ec2-user** | **guest** |

El propietario del directorio es el usuario del sistema, quien es el encargado de monitorizarlo y tratar los ficheros entrantes.

El grupo con permisos es *guest* que tiene como usuario único el usuario *guest*, el cual tiene permisos de ejecución y escritura, para poder entrar en el fichero y escribir en él (crear ficheros).

**'Watchdog forwarding':**

Espera los resultados de las operaciones que fueron lanzadas en *watchdog listening* y los reenvía a la base de datos.

| permisos | propietario | grupo |
|---|---|---|
| **drwxrwx---** | **ec2-user** | **ec2-user** |

Este directorio monitoriza ficheros entrantes que son creados dentro del mismo del sistema. Por esto, tanto el propietario como el grupo son el mismo y tienen todos los permisos.

**'Watchdog_db':**

Directorio monitorizado a la espera de conexiones procedentes de la base de datos o del propio sistema de huellas con los resultados de las operaciones. Una vez recibidas, son tratadas por el sistema. Dependiendo del tipo de respuesta puede llevar a cabo alguna función, pero siempre prepara las respuestas para ser enviadas al directorio de recogida 'buzón' del terminal.

| permisos | propietario | grupo |
|---|---|---|
| **drwx-wx---** | **ec2-user** | **db_use**r |

El propietario que trata los ficheros es de nuevo el usuario del sistema con todos los permisos sobre el directorio. El grupo asociado tiene un único usuario con el mismo nombre. Este usuario tiene permisos de ejecución para poder entrar en el directorio y de escritura para poder crear los ficheros que envía del sistema de base de datos remota.

**'guest_inbox':**

Este directorio hace la función de buzón de recogida de mensajes de los terminales. Cada terminal tiene un directorio propio (lleva por nombre el ID del terminal) dentro de 'guest_inbox'. A él son enviados los mensajes procesados de las respuestas de la base de datos a 'watchdog_db'. No es un directorio monitorizado por el sistema, sino que cada terminal está a la espera de respuesta y revisa su buzón durante un periodo de tiempo.

| permisos | propietario | grupo |
|---|---|---|
| **drwx--x---** | **ec2-user** | **guest** |

El usuario del sistema es el encargado de hacer la escritura en el buzón correspondiente y, en caso de no existir, crearlo. El usuario de grupo es el terminal que se conecta a nuestro buzón, teniendo solo permiso de lectura. Además el acceso está limitado a su propio buzón, no permitiéndosele listar el directorio, para evitar que vea las IDs de otros terminales. Un rasgo común de todos los directorios anteriores es que el usuario con privilegios del sistema es el propietario de todos ellos, al ser el encargado de tratar todos los ficheros entrantes en los *watchdogs*, o de crearlos en el caso del directorio 'buzón'. En el caso de los *watchdogs*, aunque no tenga permisos sobre los ficheros que han sido generados por los otros usuarios, puede leerlos y borrarlos al estar éstos en su directorio. En el caso del 'buzón', necesita estos permisos para poder crear directorios y ficheros que serán leídos.

Aunque no haga uso de todos los permisos de los que dispone en cada uno de los directorios, siguiendo la política de restringir al máximo los privilegios de los usuarios de fuera, hace falta uno que pueda tratar el directorio a nivel administrativo en caso de ser necesario.

## Directorios de usuarios:

**a) ec2-user** (Imagen 13)

Es el directorio del usuario con privilegios. Dentro del directorio 'work_unit' reside toda la lógica del sistema, código, claves, archivos de configuración, binarios y librerías auxiliares. En él se crean los directorios (volátiles) para las operaciones de tratamiento de huellas. El acceso es restringido solo a este usuario.

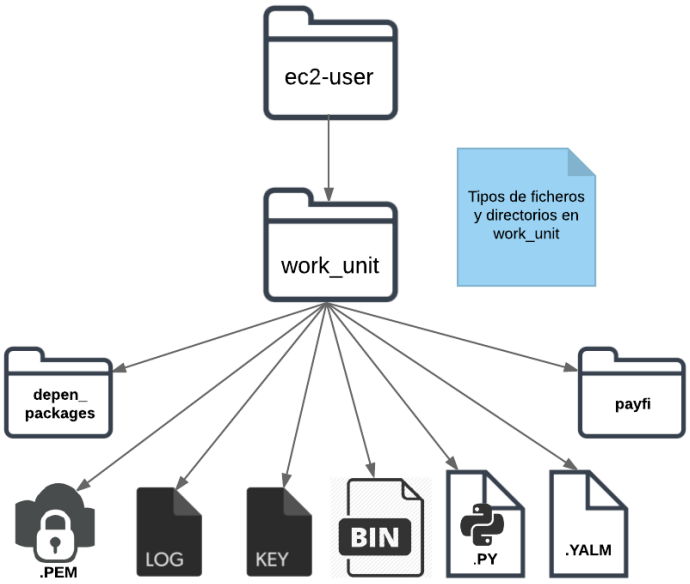


**Imagen 16 - Árbol de directorios del usuario ec2-user**

**Payfi:** Este directorio es el paquete que se ha implementado para poder desarrollar la aplicación. Contiene tanto módulos implementados por nosotros como paquetes de terceros.

**depen_packages:** Tiene los paquetes de dependencias de los paquetes de terceros que no son distribuidos por Cpython.

**LOG:** Los ficheros con extensión *.log* son utilizados para registrar incidencias en diferentes etapas del sistema de huellas:

1. 'my_log.log': registra las incidencias que puedan ocurrir a la hora de tratar las conexiones entrantes en 'watchdog_listening' y el posterior tratamiento de datos.
2. 'db_log.log': registra las incidencias que pueden surgir en las conexiones a 'watchdog_db' y el posterior tratamiento de datos.

**KEY:** Los ficheros con extensión *.key* son claves simétricas de AES con longitud de 64 caracteres. Hay dos ficheros de este tipo: uno es utilizado para cifrar los ficheros de las huellas y el otro los *.tar.gz* que llevan los trozos de los ficheros de huella.

**YALM:** Los ficheros con extensión *.yalm* son utilizados para cargar los parámetros a los *watchdogs*. De esta manera se evita tener hardcodeadas en el sistema variables del sistema, haciendo mucho más fácil el mantenimiento de código y la configuración y adaptación del sistema:

1. 'app_config.yalm': Contiene datos y variables que son cargados en 'watchfile.py'. Entre ellos destacan las claves de autentificación (OAuth) de Dropbox™, la ventana de espera de una operación y el valor mínimo para que el resultado de una comparación sea válido. El resto de variables son *paths* y extensiones aceptadas.
2. 'forwarding_listener_config.yalm': Contiene datos y variables que carga 'watchfile_forward'. Entre ellos destacan los datos de la conexión al sistema de bases de datos para poder enviar los resultados. Aparte, contiene *paths* y extensiones de fichero válidos.
3. 'db_listener_config.yalm': Contiene datos y variables que son cargados por 'watchfile_db.py'. Entre ellos destaca la variable que se utiliza para formar el *path* del buzón correspondiente al terminal.

**BIN:** Se hace uso de tres binarios para llevar a cabo el tratamiento de huellas:

1. mindtct_(32/64)_V2: Es el algoritmo que extrae las minucias de las imágenes que llegan de los terminales. Acaba en V2 porque es la versión que ha sido adaptada a nuestro proyecto generando solo los ficheros necesarios que se utilizarán para comparar huellas.
2. bozorth3_(32/64): Es el algoritmo encargado de comparar las huellas.
3. aescrypt_(32/64): Se encarga de cifrar/descifrar ficheros con clave simétrica (AES) dada. El número total de binarios es de 3 o 6 dependiendo del tipo de arquitectura (32 o 64 bits), diferenciándose por los sufijos '_32' y '_64'.

**py:** Son los archivos python de los *watchdogs* que son lanzados para que el sistema funcione. Se componen por 'watchfile.py', 'watchfile_forward.py' y 'watchfile_db.py'.

**PEM:** Los ficheros con extensión *.pem* contienen las claves privadas. Sirven para establecer conexiones a sistemas remotos. En el caso del sistema de huellas se conecta al sistema de base de datos.

La estructura de los directorios para hacer tareas requeridas es la siguiente:

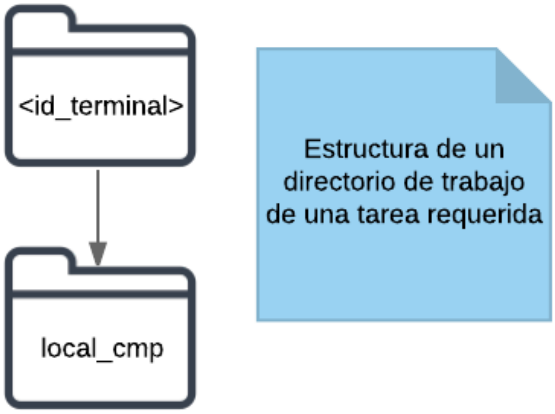


**Imagen 17 - Estructura del directorio**

**<id_terminal>:** Lleva por nombre el del terminal que requiere la operación, para así poder enviar los datos entrantes que faltan para completar la operación. Además, de esta forma no hay colisión de nombres de tareas simultáneas procedentes de otros terminales.

A este directorio se redirige la salida del algoritmo de extracción de minucias de la imagen.

**local_cmp:** Es el directorio donde se descargan/suben los trozos de huella de la nube y en el que son tratados, tanto para operaciones de cobro como de registro.

**b) db_user y guest:**

Los directorios de estos usuarios contienen una carpeta no visible *.ssh* que contiene el archivo 'authorized_keys', en el que son almacenadas las claves públicas.

Todas las carpetas pertenecientes a los usuarios tienen los mismos privilegios. Esto se debe a las estrictas reglas que impone SSH para poder establecer conexión remota al sistema.

| permisos | propietario | grupo |
|---|---|---|
| **drwx------** | **<user>** | **<user>** |

## Inicio del sistema de tratamiento de huellas:

El inicio conlleva unos pasos para que se configure y se establezcan los parámetros del mismo. Hace falta lanzar tres *scripts* en Python para que el sistema funcione.

**Lanzar 'watchfile.py':**

1. Averigua la arquitectura (i386, x86_64) del SO sobre la que va a funcionar el sistema. De esta manera se eligen los binarios a utilizar para arquitectura de 32 bits o de 64 bits.
2. Acto seguido, se define el formato que va a seguir el *logger* de incidencias para registrarlas en el fichero.
3. Después se cargan del archivo de configuración '.yalm' las variables del sistema y datos relevantes para el funcionamiento del mismo.
4. Ahora el sistema está listo para registrar la ruta de escucha de conexiones entrantes 'watchdog_listening'. Este paso se averigua por las variables del paso anterior.

5. Por último, se inicia el *handler* que actuará ante los cambios en el directorio monitorizado, se registra un observador con el *handler* y el *path* definido, y se inicializa.

**Lanzar 'watchdog_forward.py':**

Salvo el primer paso, este programasigue los mismos pasos y en el mismo orden que 'watchfile.py',. El directorio de escucha es 'watchdog_forwarding'.

**Lanzar 'watchdog_db.py':**

Sigue los mismos pasos y orden que 'watchdog_forward.py'. El directorio de escucha es 'watchdog_db'.

## 'Watchfile.py':

Una vez inicializado y monitorizando, el sistema está listo para recibir conexiones. Dependiendo de los datos entrantes, el sistema responderá de diferentes formas.

Cuando llega un fichero de una conexión (el cual es creado en el directorio), se registra por el *observer* y éste ejecuta la función sobrescrita del *handler* para objetos creados en el directorio.

La función sigue una serie de pasos para tratar los diferentes tipos de fichero que son creados. Averigua la extensión del fichero y, en el caso de no disponer de ella o de no estar entre las definidas para ser tratadas, lo elimina de manera inmediata. Si se trata de un fichero plano con extensión aceptada, se llevan a cabo una serie de pasos comunes, independientemente del tipo de operación requerida:

1. Abrir y leer los datos del fichero. Si están bien estructurados, se obtienen los datos para el tipo de operación y se devuelve un diccionario con los datos que serán utilizados en las fases siguientes. En el caso contrario, si los datos no siguen un formato correcto, la operación no sigue adelante y el archivo es borrado.
2. Acto seguido, se registra una cola de dos espacios que servirá para comunicar los procesos (procesos Unix, no flujos) que se lanzarán.
3. Se crean dos directorios jerarquizados (uno dentro del otro), los cuales servirán de *workspace* para la operación requerida. El padre llevará por nombre la ID del terminal que requiere la operación. Éste es uno de los datos obligatorios en el fichero *.txt*.
4. Guardamos la cola registrada anteriormente como valor en un *hash* global (diccionario) cuya *key* será el ID del terminal.
5. Se forman los *paths* de los cuales se obtendrán las huellas del *cloud* utilizando el pin cargado del fichero.
6. Tanto para la operación de pago, como la del registro, se lanzan dos procesos independientes simultáneamente:
    a. Uno consigue las huellas de la nube y las prepara para ser tratadas.
    b. El otro proceso es el que se encarga de la comparación/registro. Éste espera a ser avisado por dos procesos (el primero que lanzamos y el que trata la imagen de la huella) para poder llevar a cabo la operación.

Estos procesos son los encargados de registrar un *timer* de operación. El *timer* es la ventana de tiempo máximo que una operación puede estar a la espera de un dato(en este caso la imagen o la obtención de las huellas de la nube). Pasado el tiempo máximo sin haber recibido respuesta, el directorio de trabajo de la operación se destruye y se registra una incidencia de *timeout* en el fichero log del *watchfile*. El valor del *timer* es cargado del fichero de configuración.

7. Por último, una vez finalizados los procesos mencionados arriba, independientemente del resultado de los mismos, los datos de la huella llegados a 'watchfile' son borrados.

En el caso de tratarse de una imagen con una extensión aceptada, los pasos a seguir son:

1. Obtener el ID del terminal del que proviene la imagen (el nombre de la imagen es el ID del terminal)
2. Comprobar que existe la carpeta con el mismo nombre que el ID del terminal. En el caso de no existir, la imagen es borrada; en caso contrario, se sigue adelante con la operación.
3. Obtener del *hash* global (cuyo *key* es el ID del terminal) el objeto cola que se comunicará con el proceso a la espera (comparar/registrar).
4. Lanzar un proceso independiente para obtener las minucias de la imagen; éste, una vez acabado, avisará depositando un mensaje en la cola al proceso que espera.
5. Al final la imagen es borrada.

### 'Watchfile_forward.py':

Una vez lanzado con éxito el *script,* monitoriza el directorio a la espera de ficheros entrantes y los reenvía a la base de datos.

Al monitorizar un archivo que recibe datos generados y transmitidos dentro del mismo sistema el único *parser* que se le aplica es de extensión de fichero. Si el fichero tiene una extensión aceptada es reenviado directamente.

El fichero siempre es borrado del directorio, tanto si es válido y reenviado, como si no.

Los datos de la conexión así como los ficheros aceptados son cargados por el fichero de configuración del script.

### 'Watchfile_db.py':

Lanzado el *script,* monitoriza el directorio a la espera de ficheros entrantes de la base de datos con respuestas de operaciones o del propio sistema.

Al recibir los datos de un sistema de confianza y de la misma jerarquía (sistema de base de datos o reenvío dentro del sistema de huellas), solo se remite a leer los datos en el orden esperado:

1. De la primera línea del fichero lee el tipo de operación que se llevó a cabo (registro o pago).
2. De la segunda línea lee el resultado de la operación (éxito o fallo).
3. De la tercera y última línea obtiene el identificador del terminal que encargó la operación.

Estos datos son suficientes para generar un archivo para responder al terminal en su buzón. Para ello el *script*:

1. Genera la ruta del buzón utilizando una variable cargada por el fichero de configuración que contiene la ruta de los buzones y añade a ella el nombre del ID del terminal.
2. Comprueba si existe el directorio en la ruta de buzones, o en caso de no existir (primera operación de ese terminal) lo crea.
3. Acto seguido, crea un fichero en la ruta del buzón del terminal y escribe el mensaje para el terminal. El mensaje se compone de:
    a. El *timestamp*, que obtiene en el momento en el que se crea el fichero. De esta forma el terminal distingue cuándo está leyendo datos actualizados y cuándo no.
    b. El resultado de la operación. El terminal sabe qué operación hace, solo necesita el resultado de la misma.

Se transmite la menor información posible al terminal por razones de seguridad. De esta forma, si el medio por el que se transmite la información o incluso el terminal quedan comprometidos, no expone la manera en que son tratados los datos por el sistema y tampoco viajan datos del usuario por la red en vano.

El tipo de operación, aunque no es utilizado para generar la respuesta al terminal, sí es utilizado por el sistema. En el caso de tratarse de una operación fallida de registro de usuario, nuestro sistema debe borrar los datos de la huella que han sido guardados en la nube.

Al final de la operación los ficheros procedentes del sistema de base de datos o dentro del propio sistema son borrados.

### Operaciones de tratamiento de huella:

Por ahora la aplicación FingerPay soporta dos tipos de operaciones: registro de usuario y pago.

Cabe destacar que para llevar a cabo cualquier operación, los ficheros han de tener extensión y formato válido, además de llegar en el orden establecido.

Primero ha de llegar a 'watchdog_listening' el fichero con extensión *.txt* y la estructura válida dependiendo del tipo de operación que se requiera.

A continuación, debe llegar la imagen que lleve por nombre el ID del terminal del que proviene.

En caso de llegar primero la imagen o llegar después de expirar el tiempo de espera, ésta es borrada por el *watchdog*.

Las funciones principales que intervienen en las operaciones de registro y pago se encuentran en *'*download_chunks.py'*.*

### Pago:

Una vez recibido y validado el fichero con los datos del pago por el *watchfile*, éste lanza dos procesos simultáneos. Como se explica en el apartado de 'tratamiento de los ficheros de huella'

El primer método lanzado como proceso es '**download_tar'***:* Este método se encarga de obtener las huellas de las distintas cuentas de Dropbox™, descifrarlas y unirlas:

1. Descarga los *.tar.gz.aes* cifrados de las cuentas Dropbox™. Para ello se lanzan 'n' procesos paralelos para obtener de forma simultánea los ficheros, siendo 'n' el número de cuentas de Dropbox™ que tienen ficheros con huellas.
2. Acto seguido, se descifran los ficheros *.tar.gz.aes* en paralelo, lanzando un proceso para cada fichero *.tar.gz*.
3. A continuación, se extraen los trozos de las huellas de los ficheros *.tar.gz*.
4. Una vez extraídos todos los trozos de las huellas en el mismo directorio, se procede a su unión y descifrado. Para unirlos se utiliza la herramienta 'cat', razón por la que es importante que los ficheros de las huellas tengan un formato bien definido. Una vez unidas las huellas, se procede a su descifrado con la otra clave AES. Tanto la unión como el descifrado de los ficheros de huella se hace en orden y en paralelo, para lo cual se lanza un *pool* de procesos que une y descifra varios ficheros a la vez.
5. Una vez obtenidos todos los ficheros con los datos de las huellas unidos y descifrados, se procede a su listado. Se genera un fichero con el *path* en el que se encuentran los ficheros y los nombres de cada fichero de huella. Este fichero es utilizado por el algoritmo de comparación bozorth3 para localizar la galería de huellas contra las que tiene que comparar la huella entrante. El fichero tiene un formato *.lis*.
6. En el caso de que todo el proceso haya tenido lugar sin problemas, se avisa por la cola con un mensaje 'done' al proceso que esté esperando (en este caso el de comparación). En caso contrario se registra en el fichero *log* el error y se envía al proceso en espera un mensaje con *flag* '1'.

El segundo proceso en lanzarse es '**compare'**: Este método está a la espera de señales para poder hacer la comparación de huellas y encontrar el resultado. Espera el aviso de 'download_tar' que fue lanzado de forma simultánea y del proceso que extrae las minucias de la huella 'extract_mindtct'.

1. Al iniciarse este proceso, lo primero que hace es esperar durante un tiempo determinado (se puede ajustar en la variable de archivo de configuración) a recibir las señales correspondientes para continuar.
2. En caso de recibir ambas señales positivas a tiempo, el proceso ejecuta el binario bozorth3 y obtiene su salida. En caso contrario registra un error en el fichero *log* y borra el directorio de trabajo para este pago.
3. La salida de bozorth3 es un fichero con un número por fila que representa la puntuación de similitud entre la huella a comparar y la huella en esa fila del fichero lista *.lis* contra la que fue comparada.
4. En el fichero de salida se busca el mayor número, y la posición que éste ocupa dentro del fichero. De esta forma se obtiene la puntuación máxima, que representa la mayor similitud entre huellas y a partir de su posición se obtiene de la lista *.lis* el nombre de la huella.

5. Obtenida la puntuación de similitud y el nombre de huella, se comprueba que la puntuación supera la barrera mínima para que se pueda considerar un *match* seguro. Este valor se define en el fichero de configuración y su valor se establece a 30 en base a las diferentes pruebas que hemos realizado para evitar falsos positivos.
6. En caso de superar la barrera mínima, el nombre de la huella es añadida al final del fichero al el resto de datos enviados por el terminal y es enviado a 'watchdog_forwarding'. En caso de no superar la barrera mínima, el máximo obtenido se genera y envía un fichero con el error a 'watchdog_bg'.

Al final de la operación, independientemente del resultado de la misma, el directorio de trabajo para la tarea es borrado.

En el caso de recibir una imagen, lo primero que hace *watchfile,* es obtener su nombre (ID del terminal) y lo busca en el *hash* global (diccionario). En caso de existir, éste devuelve una cola de comunicación. Eso significa que la imagen ha llegado dentro de la ventana de tiempo para ser tratada. En caso contrario, devuelve *false* y la imagen de la huella es borrada.

A continuación es lanzado un proceso '**extract_mindtct'.** Este método se encarga de obtener las minucias de las huellas. El resultado es un fichero con extensión *.xyt* que es guardado en el directorio de trabajo de la tarea de pago y será utilizado por bozorth3 para compararlo con el resto de huellas obtenidas de la nube.

Una vez extraídas las minucias, se avisa al proceso en espera (en este caso a '*compare'* por la cola con un 'done' en caso de éxito y el *flag* '1' en caso de error, registrando la incidencia en el fichero de *log*).

Al final de la operación, independientemente del resultado, la imagen de la huella es borrada de 'watchdog_listening'.

<u>Registro:</u>

Recibido y validado el fichero con los datos de registro del usuario por *watchfile*, éste lanza dos procesos simultáneos.

El primero es '**download_tar_up':** Este método, al igual que 'download_tar' en el pago, se encarga de obtener las huellas de Dropbox™. Pero en lugar de hacer todos los pasos y dejarlas unidas y descifradas, implementa los tres primeros pasos, ya que para el registro necesitamos dejar las partes cifradas de la nueva huella en los directorios con el resto de partes que son comprimidas y cifradas.

En caso de éxito, avisa al proceso en espera 'upload_tar'; en caso contrario, registra el error en el fichero *log* y envía el *flag* '1'.

El otro proceso lanzado simultáneamente es '**upload_tar'**. Al igual que '*compare'*, este método espera recibir señal de otros dos procesos para poder continuar con su tarea.

Si recibe las señales a tiempo:

1. Cifra el fichero *.xyt* extraído de la imagen y lo renombra de forma que su nombre sea único, y luego lo parte en trozos.

2. Después mueve cada uno de los trozos a las carpetas correspondientes que serán comprimidas.
3. Una vez las piezas han sido movidas, las carpetas son comprimidas en formato *.tar.gz* y renombradas según el convenio establecido en el fichero de configuración, en este caso *<pin>%<parte>.tar.gz*.
4. Generados los archivos comprimidos, éstos son cifrados con la clave AES que les corresponde.
5. Una vez cifrados los archivos, éstos son subidos a las distintas cuentas de Dropbox™, con la nueva huella.
6. Por último, la función genera el archivo con los datos del usuario que fueron enviados por el terminal junto con el nombre de la huella generado y los reenvía a 'watchdog forwarding'.

Si no recibe alguna de las señales de los procesos, registra el error en el fichero *log* y borra el directorio de la tarea de registro.

Al final de la operación, el directorio de trabajo de la tarea es borrado.

El otro proceso del que 'upload_tar' espera recibir señal es '**extract_mindtct'**, explicado en el apartado de pago.

## Tratamiento de los ficheros de huella:

Los ficheros que contienen la información de la huella de los usuarios son tratados y almacenados de diferente forma al resto de los datos de usuario.

Los pasos a seguir para preparar una huella para la nube son los siguientes:

1. El fichero de la huella es cifrado por AES con una clave de 64 caracteres.

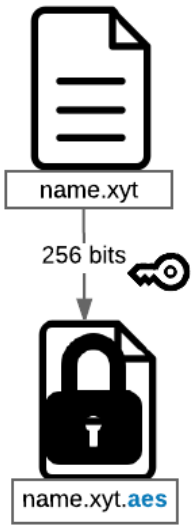


**Imagen 18 - Cifrado de huella**

2. Una vez cifrado, el fichero se parte en tantas piezas como servicios de almacenamiento en la nube se utilicen para almacenarlos. Al partir las huellas éstas son renombradas:

- Su nombre raíz se mantiene.
- Las extensiones .xyt y .aes son borradas.
- Se añaden sufijos ordenados distintos a cada trozo. Éstos son separados del nombre por un carácter singular (el orden de los sufijos viene dado en el archivo de configuración).

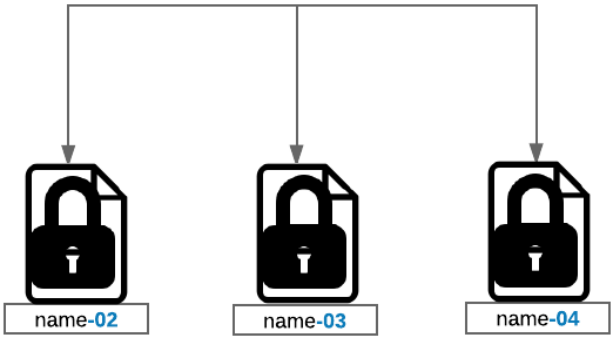


**Imagen 19 - Particionado de huella**

3. Obtenidos los trozos renombrados, éstos son distribuidos en las carpetas junto al resto de trozos con el mismo sufijo y pin.

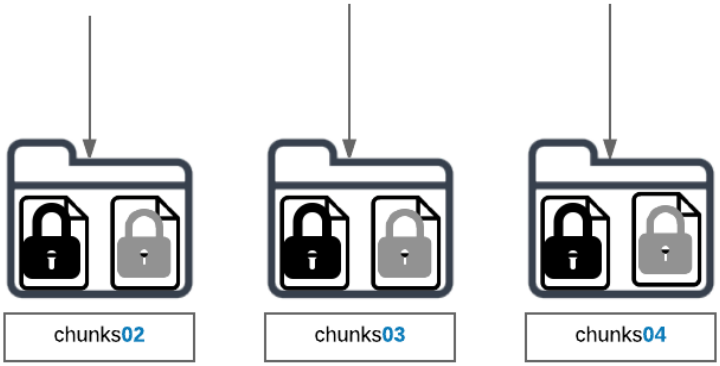


**Imagen 20 - Distribución de archivos**

Los directorios son comprimidos en formato *.tar.gz*. El nombre del archivo resultante tiene la siguiente forma *<pin>%<número>.tar.gz:*

- Pin: Para no depender de la estructura en la nube para mantener el orden.
- Carácter singular '%': Para separar el pin del número. Esto hace posible que se puedan parsear y automatizar tareas para tratar los archivos.
- Número: Sirve para enumerar y diferenciar los distintos archivos generados, no teniendo que ver con los sufijos de los ficheros. Se empieza a numerar por 1 hasta 'n', siendo éste el número de ficheros resultantes (3 en nuestro ejemplo).

Los ficheros resultantes de la compresión son cifrados por AES con una clave de 64 caracteres (clave distinta a la que se ha utilizado para cifrar las huellas individuales). El resultado de la compresión son archivos con la extensión de cifrado AES. Estos archivos finales son los que serán subidos a la nube para ser almacenados.

El proceso inverso es el que se lleva a cabo para obtener las huellas de las nubes y tratarlas en el sistema.

Las razones por las que se ha elegido este tipo de estructura para organizar los ficheros de las huellas son por rapidez, seguridad y eficiencia en cuanto a recursos.

- **Tiempo:**
    - La API del servicio en la nube que utilizamos no ofrece la posibilidad de descargar un directorio remoto entero, permitiendo obtener solo un archivo por conexión, lo cual implica una conexión por descarga.

      Ahora el sistema tarda considerablemente menos tiempo en recuperar menos de una decena de archivos grandes, que en recuperar como mínimo cientos de archivos más pequeños.
    - Permite poder descifrar de una sola vez cientos de archivos, lo que supone un ahorro de tiempo importante.
- **Seguridad:** Frente a la implementación anterior, la seguridad de los ficheros no ha variado. Sigue utilizando dos claves simétricas de 64 caracteres para cifrar los datos, distinguiéndose simplemente en que ahora los tiene agrupados. La forma en que se almacenan en la nube no ha variado; allí los directorios siguen teniendo la misma estructura.
- **Eficiencia:** Aquí se observa un ahorro de recursos importante por varias razones:
    - hacer pocas conexiones de red frente a muchas.
    - recorrer cientos/miles de archivos para cifrarlos dos veces.

## Algoritmos de tratamiento de huellas:

**MINDTCT – Detección de minucias (*Minutiae detection*)**:

Para llevar a cabo la comparación entre dos huellas se usa la técnica de detección de minucias. Las minucias que se tienen en cuenta son las terminaciones de cresta y las bifurcaciones de cresta.

Suele haber alrededor de 100 minucias en una huella. Para poder identificarlas se tienen en cuenta las coordenadas y la orientación de la cresta a la que pertenecen.

El algoritmo MINDTCT se encarga de detectar y localizar todas las minucias en una imagen dada, asignando a cada minucia un punto de localización, orientación, tipo y calidad de ésta.

MINDTCT sigue una estructura totalmente modular, ejecutándose cada módulo en una subrutina siguiendo una secuencia.

Este diseño del algoritmo permite que los pasos se puedan alterar o incluso ignorar haciendo más fácil ajustar el algoritmo a las necesidades, pruebas o exigencias de rendimiento.

- Entrada**:**

  El algoritmo acepta por entrada ficheros en formato ANSI/NIST, WSQ, JPEGB, JPEGL, IHEAD y png.

- Salida**:**

  El código fuente de MINTCT fue adaptado para que la salida conste solo del fichero que necesitamos.

  El fichero con extensión .xyt contiene en cada fila las características de una minucia de la huella.

  En la primera y segunda columna se encuentran respectivamente las coordenadas 'x' e 'y' de la minucia; en la tercera columna, el ángulo de la minucia; y en la cuarta, la calidad de la minucia (calidad de la imagen en ese punto).

**BOZORTH3– Comparación de huellas (Fingerprint matcher)**:

Bozorth3 es un *matcher* que utiliza la localización y la orientación de las minucias para hacer comparaciones entre huellas. El algoritmo es invariante en rotación y translación.

Compara creando una tabla para cada huella. En las tablas están definidas distancias y orientación de cada minucia de la huella.

Cuando dos tablas son comparadas se crea una tercera en la que se guarda información sobre la compatibilidad entre las dos huellas. Los datos de la tercera tabla son utilizados para crear una puntuación de similitud entre huellas.

- Entrada**:**

  Ficheros con extensión *.xyt*, resultado de la salida del binario que implementa el algoritmo de extracción de minucias MINDTCT.

  Ficheros con extensión *.lis* que contienen *paths* de las huellas contra las que se va a comparar una huella.

  - Dos ficheros con extensión *.xyt*
  - Fichero con extensión *.xyt* y fichero con extensión *.lis*

- Salida**:**

Un *stream* por consola con un número entero por fila. El número representa la similitud que hay entre dos huellas, basándose en el número de puntos comunes que formen el camino más largo al ser unidas.

El número de filas mostradas depende de las huellas listadas en el fichero con extensión *.lis*. Muestra los resultados de la comparación en el orden del fichero de la lista. En caso de comparar dos huellas, la salida es solo un número entero.

## 3.4 Base de datos de usuarios

Para alojar toda la información de cuentas y usuarios en la base de datos, vamos a abordar cuatro puntos:

- Cómo se ha alojado la base de datos en OpenShift®.
- Cómo es el diseño de esa Base de Datos.
- Cómo es el programa que la gestiona y cómo funciona.

- Dónde se aloja ese programa (Amazon Web Services).


**Imagen 21 - OpenShift®**

Para alojar la base de datos en OpenShift® se ha creado una nueva aplicación con las *Cartridges* siguientes:

- MySQL 5.5
- phpMyAdmin 4.0

El diseño de la base de datos es sencillo ya que se compone únicamente de dos tablas:

- Usuario: Almacena los datos de una persona, la cual puede ser propietaria de una o varias cuentas. Se compone de:
    - DNI: Clave primaria de la tabla. Identifica al usuario.
    - Nombre: El nombre del usuario.
    - Apellido 1: El primer apellido del cliente.
    - Apellido 2: El segundo apellido del cliente.
    - Email: un correo electrónico de contacto del cliente.
    - Teléfono: Columna opcional.
- Cuenta: Almacena los datos de una cuenta de un usuario. Esta tabla está relacionada con la de usuario mediante la columna DNI. Se compone de:
    - DNI: Identifica al usuario de la cuenta. Es la clave foránea.
    - NombreCuenta: Clave primaria de la tabla. Se trata del identificador principal del usuario, obtenido a través de su pin y su huella.
    - Pin: Cuatro números que proporcionan una seguridad extra a la cuenta.
    - Cantidad: Dinero del que dispone la cuenta.

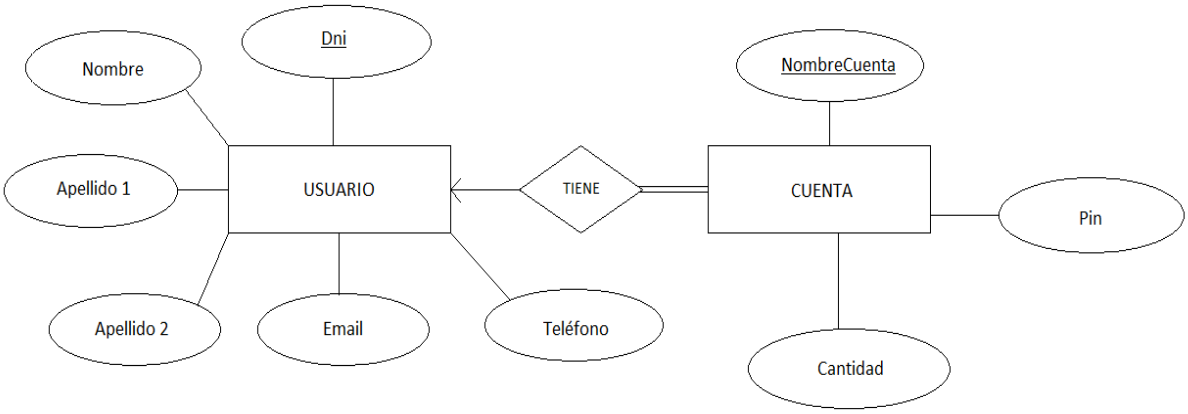


**Imagen 22 - Diagrama Entidad-Relación**

**cuenta**

| Columna | Tipo | Nulo | Predeterminado | Comentarios |
|---|---|---|---|---|
| DNI | varchar(9) | No | | |
| NombreCuenta | varchar(30) | No | | |
| Pin | int(4) | No | | |
| Cantidad | int(9) | No | | |

**Índices**

| Nombre de la clave | Tipo | Único | Empaquetado | Columna | Cardinalidad | Cotejamiento | Nulo | Comentario |
|---|---|---|---|---|---|---|---|---|
| PRIMARY | BTREE | Sí | No | DNI | 10 | A | No | |
| | | | | NombreCuenta | 10 | A | No | |

**usuario**

| Columna | Tipo | Nulo | Predeterminado | Comentarios |
|---|---|---|---|---|
| DNI | varchar(9) | No | | |
| Nombre | varchar(15) | No | | |
| Apellidos | varchar(30) | No | | |
| Email | varchar(30) | No | | |
| Telefono | int(9) | No | | |

**Índices**

| Nombre de la clave | Tipo | Único | Empaquetado | Columna | Cardinalidad | Cotejamiento | Nulo | Comentario |
|---|---|---|---|---|---|---|---|---|
| PRIMARY | BTREE | Sí | No | DNI | 11 | A | No | |

**Imagen 23 - Tablas de la base de datos**

Para la gestión de la base de datos, se ha creado un programa en Java que se encarga de comprobar, insertar, actualizar y/o eliminar los datos que le lleguen.

Estos datos son enviados desde otras partes de la aplicación a una carpeta llamada ‘Servidor’, sobre la cual la aplicación está escuchando de forma continua, esperando que lleguen nuevos archivos sobre los que actuar.

Cuando llega un archivo a la carpeta mencionada, el programa lo coge y, según el tipo, realiza unas acciones u otras.

Tipos de archivos que se manejan:

- Consulta (*SELECT*):
    - ID de la máquina que generó el archivo.
    - Tipo de operación (en este caso, *SELECT*).
    - DNI del nuevo usuario.
- Registro (*REGISTER*):
    - ID de la máquina que generó el archivo.
    - Tipo de operación (en este caso, *REGISTER*).
    - Pin de la cuenta (compuesto por cuatro números).
    - Nombre del nuevo usuario a registrar.
    - Apellido 1 del nuevo usuario.
    - Apellido 2 del nuevo usuario.
    - DNI del nuevo usuario.
    - Email del nuevo usuario.
    - Teléfono (opcional, en caso de que no se indique se recibirá un cero).
- Pago (*CHARGE*):
    - ID de la máquina que generó el archivo.
    - Tipo de operación (en este caso, *CHARGE*).
    - Pin de la cuenta (compuesto por cuatro números)
    - Cantidad a pagar.
    - Nombre de la cuenta.

El programa los recoge y comprueba que la cuenta tenga la cantidad necesaria de dinero para poder ejecutar ese pago. En caso de que se pueda pasar a la operación de pago, la aplicación hace la actualización necesaria en la base de datos.

Tanto en el caso de que la operación se realice correctamente como en el caso contrario, se genera un nuevo archivo (con el identificador del terminal que creó el inicial) para notificar el resultado.

El formato de los archivos es:

- Tipo de operación que se ha realizado (*REGISTER, CHARGE* o *SELECT*)
- Si se ha completado con éxito o se ha producido algún error (*SUCCESS* o *FAILURE*)
- Número de la máquina que envió el primer archivo.

La gestión de la conexión a la base de datos se lleva a cabo mediante JDBC (com.mysql.jdbc.Driver). A continuación, para establecer la conexión se hace uso de la clase DriverManager.

Al iniciarse el programa, éste se queda escuchando continuamente en la carpeta 'Servidor'. Cuando detecta un nuevo archivo inicia la ejecución de su clase principal (Principal.class) que hace las siguientes operaciones:

- Tratamiento del archivo (*OperacionesFichero.class*): Abre el archivo recibido y lee lo que hay dentro, guardando sus datos temporalmente. Si la línea correspondiente al tipo de la operación es *REGISTER, CHARGE* o *SELECT* se continúa al siguiente apartado y se borra el archivo. Si no, aquí acaba el proceso y se borra el archivo.
- Ejecución de la operación solicitada (*OperacionesBD.class*): Con los datos recogidos se irá a la parte del programa que se encarga de la gestión de la base de datos. Así, si recibe un *SELECT*, buscará con una consulta *select* en la base de datos. Si recibe *REGISTER*, hará dos *insert*, uno para el usuario y otro para la cuenta asociada a él. Si hace un *CHARGE*, hará un *select* para comprobar que la cantidad a pagar no sea mayor que la que dispone el cliente y se procederá al pago con un *update*.
- Comunicar los resultados (*OperacionesFichero.class*): Se crea un archivo de salida informando de si han salido bien o no las operaciones requeridas. Para enviar este archivo a la máquina destino, usamos el comando scp de la siguiente manera:

*scp -i db_user.pem Respuesta/<nombre_Máquina*>.txt*
*db_user@<dir_VMAmazon>/home/watchdog_db*

*Máquina física que ha recogido la información inicial.

## Elementos de la máquina:

En esta máquina disponemos de los siguientes elementos (en /home/ubuntu/App):

- 'app.jar': Aplicación que se ejecuta continuamente en la máquina.
- Carpeta 'Servidor': Carpeta donde el programa 'app.jar' está escuchando continuamente esperando nuevos archivos. Cuando llega uno nuevo, comprueba su tipo y realiza la actividad necesaria para modificar o no la base de datos alojada en OpenShift®.
- Carpeta 'Respuesta': Una vez se han realizado todas las operaciones con un archivo de la carpeta 'Servidor', la aplicación crea un archivo de salida en la carpeta 'Respuesta' informando de lo ocurrido y la hora del sistema en la que ha finalizado.
- 'db_user.pem': Archivo .pem que permite a 'app.jar' conectarse a la otra máquina de Amazon para enviar los archivos que necesita.
- El archivo de propiedades para 'app.jar'. En él se guarda la información de:
    - La DNS de la otra máquina de Amazon que se encarga de la gestión de las huellas.
    - El usuario con el que se conectará la aplicación a dicha máquina.
    - La ruta donde la aplicación enviará los archivos de respuesta.
    - La dirección de OpenShift®.


**Imagen 24 - Directorio de la aplicación de conexión a base de datos**

- Para conectar la aplicación a la base de datos alojada en OpenShift® hacemos uso de rhc de Client tools. Gracias a esta herramienta podemos conectarnos a nuestra cuenta de OpenShift® y a nuestras aplicaciones alojadas en la nube a través de port-forward:

  *rhc port-forward accessbd*

- Las Client Tools son unas herramientas para la conectividad. Para su uso necesitamos tener Ruby 1.8.7 instalado en el sistema donde vaya a ejecutarse.

Por lo tanto, la máquina alojada en Amazon está ejecutando dos procesos simultáneos:

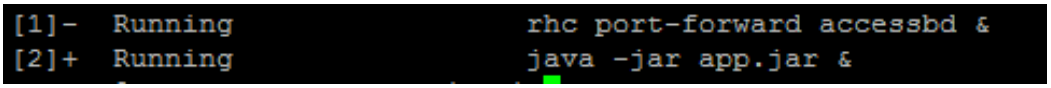


**Imagen 25 - Ejecución de dos procesos simultáneos en Amazon**

# Capítulo 4. Trabajo realizado por cada componente del grupo

## Participación de Pablo

### Fase 1. Aprendizaje

Desde un principio mi cometido fue el lector de huellas. No teníamos más materiales que el lector (nos lo habían prestado), y para conseguir que funcionara había que buscar los *drivers* y el SDK para poder programar sobre él.

Una vez instalado todo, era momento de entender cómo funcionaba. Por suerte el SDK traía consigo unos ejecutables a modo de ejemplo con los que se podían enrolar dedos y verificarlos de diferentes maneras y con diferentes interfaces gráficas. También incluía sus fuentes para poder ver el código y aprender mejor el funcionamiento.

### Fase 2. Primeras pruebas

Las primeras pruebas fueron dirigidas a desarrollar mis propios programas con los que funcionara el lector de huellas, tanto en enrolamiento como en verificación. Sin embargo, finalmente el enrolamiento lo hemos programado de la forma explicada más adelante y para la verificación hemos utilizado otro método separado, alojado en Amazon de forma que los terminales no tuvieran que procesar nada y fueran más sencillos.

### Fase 3. Recolección de huellas

Desarrollé un programa que recolectaba huellas, de modo que con la ayuda de amigos y familiares que colaboraron registrando las suyas, pude comenzar a hacer pruebas.

En un principio, para la recolección el software necesitaba capturar 4 veces la huella, pero era un error de entendimiento del programa. Seguí aprendiendo el funcionamiento y vi que en lugar de hacer una mezcla de todas para conseguir una imagen de mejor precisión y calidad, que era lo que yo pensaba, solo estaba almacenando la última.

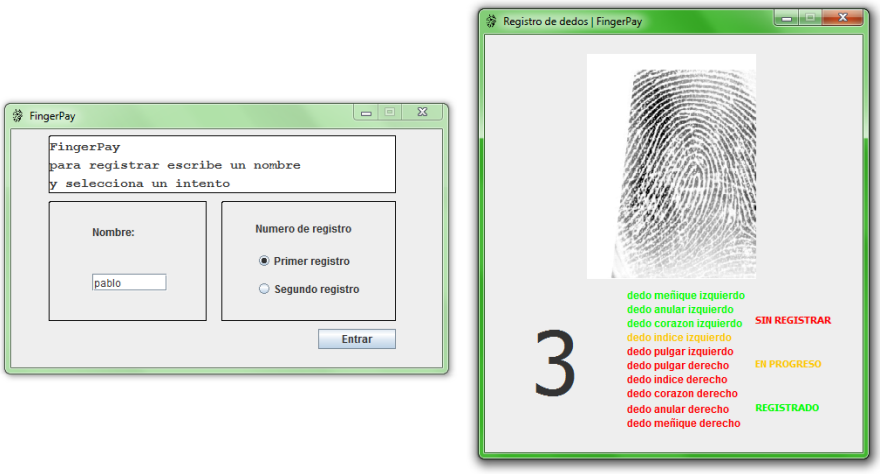


**Imagen 26 - Primer software para recolecta de huellas**

El error venía producido porque el método que utiliza el dispositivo para el enrolamiento hace 4 capturas de huella para obtener minucias, en la verificación obtiene minucias de una huella y las compara con las que se registraron en el enrolamiento.

Pero no era exactamente eso lo que queríamos, ya que lo único que necesitábamos era una imagen de la huella para poder almacenarla en la parte del registro y otra durante el pago para compararla. Por ello modifiqué el software para que solo tomara una huella por dedo. El proceso lo repetíamos dos veces para poder comparar una toma con otra en nuestras pruebas.

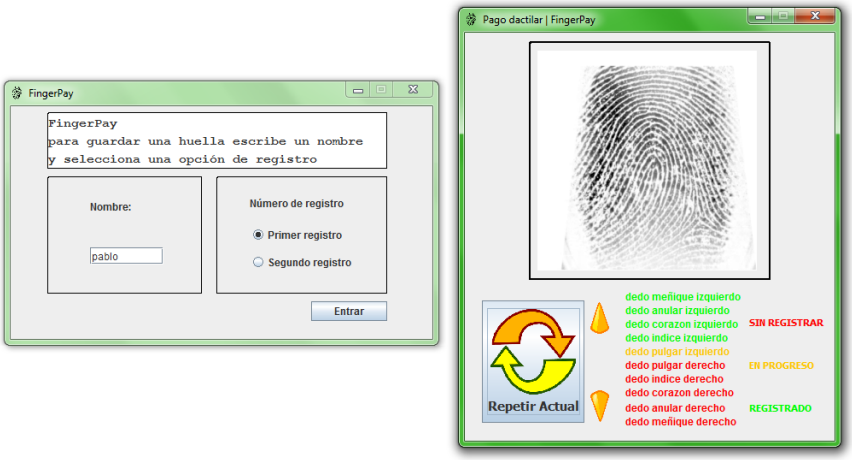


**Imagen 27 - Modificación del software para recolecta de huellas**

## Fase 4. El software final

Por último, desarrollé tanto el software de registro como el del terminal de pago(apartados 3.1 y 3.2).

Lo primero fue hacer unos bocetos de cómo quería que fuera la interfaz gráfica. La interfaz definitiva no es exacta a los bocetos.

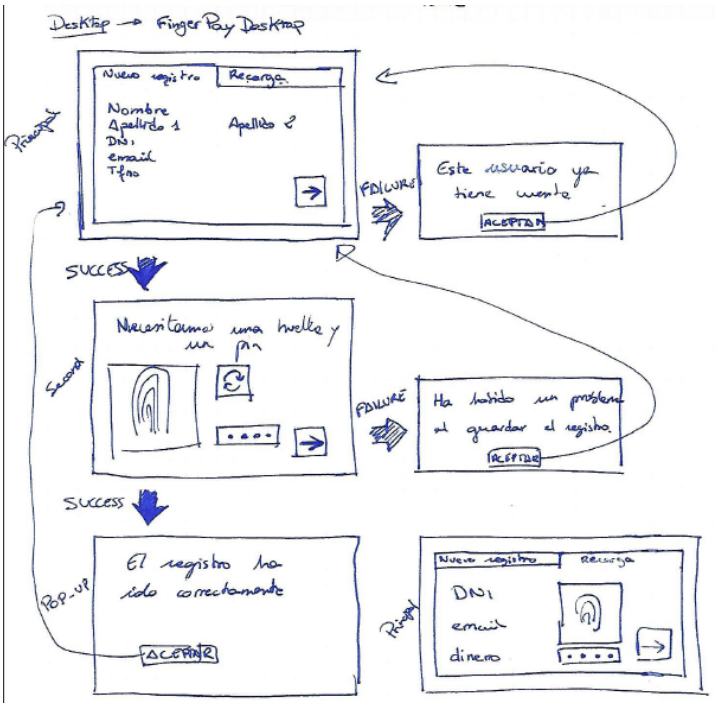


**Imagen 28 - Boceto del registro**

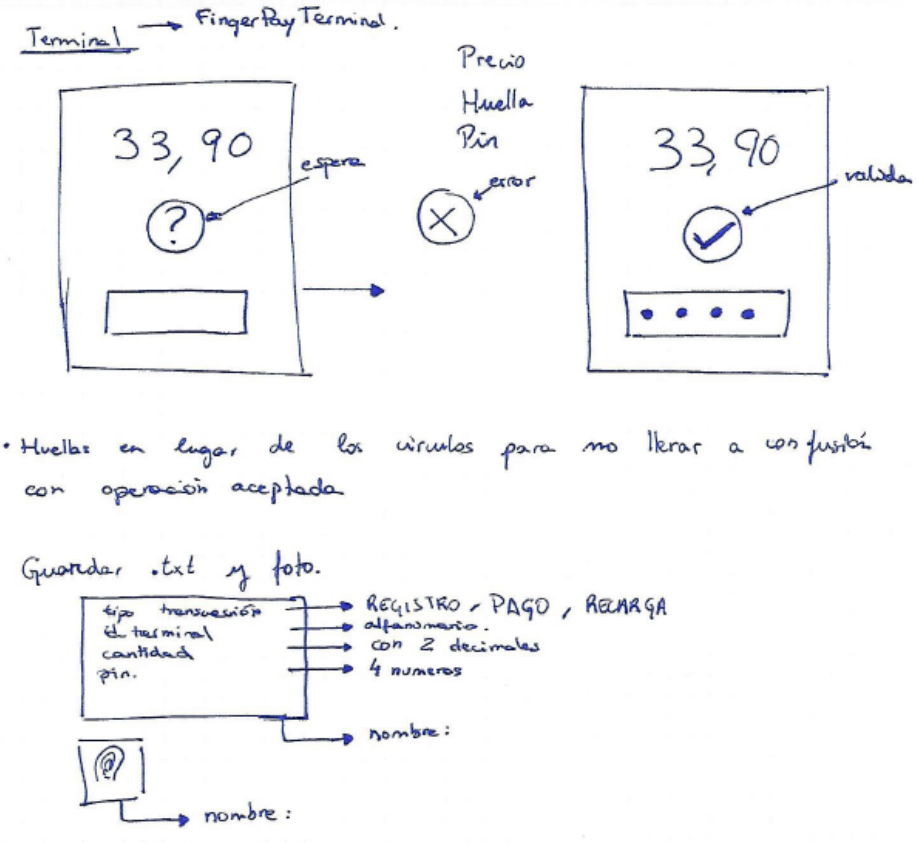


**Imagen 29 - Boceto del terminal**

En segundo lugar, volqué todas las ideas en la programación en Java. Hubo algún problema, como por ejemplo al intentar que toda la aplicación trabajara en una sola hebra de ejecución, ya que de hacerlo así, mientras el programa espera a recibir una respuesta, podía parecer que se había colgado. Para que fuera más satisfactorio para el usuario se separó en dos hebras. Gracias a esta solución podemos ver la imagen animada mientras esperamos a que se realice el pago.

El último paso fue unir mi parte con la de mis compañeros realizando la conexión por canal seguro a las máquinas de Amazon. Para ello me ayudé de las herramientas PUTTY ya que desde Windows no hay disponible ningún comando nativo con el que conectarse por SSH. El programa *pscp.exe* hace las veces del comando *scp* que hay en Unix, por lo que puedo ejecutar el comando desde la aplicación añadiéndole una clave privada *.ppk* a modo de identificación, el archivo que quiero enviar y por último la dirección donde se quiere mandar.

He hecho los diagramas que a mi parte corresponden, redactado mi parte de la memoria, unido todas las partes de mis compañeros y dado formato a toda ella.

## Participación de Francisco Javier:

### Investigación

Uno de los puntos que debíamos tratar era el almacenamiento de los datos de un usuario y sus cuentas en una base de datos. Se buscaron varios servicios donde se podría almacenar, como iCloud o Azure. Ambos descartados, después de hacer pruebas, por limitaciones a causa de las cuentas de pago, o porque sus servicios se quedaban cortos para lo que buscábamos.

Finalmente la elegida fue OpenShift®. Tenía lo que buscábamos al permitir tantas opciones con sus *Cartridges* y sus pocas limitaciones con cuentas gratuitas.

## OpenShift®

Una vez que elegimos OpenShift®, comencé a mirar cómo era su funcionamiento. En un principio, el objetivo era el de almacenar en OpenShift® la base de datos y el programa que la gestionase, por lo que empecé a documentarme por Internet acerca de los distintos tipos de *Cartridges* de los que disponía este servicio.

Finalmente se llegó a la conclusión de que el almacenamiento de la aplicación sería más adecuado que fuese en la nube de Amazon Web Services y que ésta se conectase a la base de datos en OpenShift®, ya que tendríamos un desacoplamiento mayor y por tanto mejor.

A OpenShift® le añadí las *Cartridges*:

- MySQL 5.5
- phpMyAdmin 4.0: Añadido para así poder ver y desarrollar de una manera más intuitiva la base de datos.

## Conexión a OpenShift

Una vez almacenada la base de datos en OpenShift®, llegó el momento de ver cómo conectarse a ésta.

En un principio, las primeras pruebas fueron llevadas a cabo con la dirección general proporcionada por OpenShift®, pero no funcionó.

Finalmente, después de ir viendo distintos métodos que no llegaron a funcionar bien, encontré un tutorial de *rhc* (Usando Ruby y Git) que sirvió para realizar una conexión a la base de datos a través de una dirección local ligada a la del servidor.

Ahora ya, simplemente usando en segundo plano el comando *rhc port-forward <nombre_de_la_base de datos>,* éste me proporcionaría una dirección y un puerto con los que acceder desde la máquina que lo ejecutase.

```
To connect to a service running on OpenShift, use the Local address

Service Local               OpenShift
------- ------------- ---- ------------------
httpd   127.0.0.1:8080  =>  127.5.145.131:8080
java    127.0.0.1:3528  =>  127.5.145.129:3528
java    127.0.0.1:4447  =>  127.5.145.129:4447
java    127.0.0.1:5445  =>  127.5.145.129:5445
java    127.0.0.1:5455  =>  127.5.145.129:5455
java    127.0.0.1:8081  =>  127.5.145.129:8080
java    127.0.0.1:9990  =>  127.5.145.129:9990
java    127.0.0.1:9999  =>  127.5.145.129:9999
mysql   127.0.0.1:3306  =>  127.5.145.130:3306

Press CTRL-C to terminate port forwarding
```

**Imagen 30 - Direcciones locales generadas para conectarse a OpenShift®**

## Desarrollo del Programa

Teniendo la base de datos creada y sabiendo cómo conectarme a ella, llegó el turno del desarrollo de una aplicación que gestionase las distintas consultas que se iban a realizar. La aplicación la hice en Java, ya que es el lenguaje con el que más documentación conseguí encontrar en lo referente a OpenShift®. Además hice uso de un *plugin* proporcionado por OpenShift® para Eclipse que facilitaba la interacción entre el entorno de desarrollo y el servidor. Esto facilitó mucho el proceso de volcado del código en OpenShift® antes de que se decidiese que la aplicación se alojaría en Amazon Web Services.

El programa se dividió en tres partes:

- El watchdog que monitorizaba una carpeta llamada 'Servidor'. Cuando llegaban varias peticiones en un corto periodo de tiempo, la aplicación daba un error al intentar acceder a un archivo que ya estaba en uso, por lo que se hizo un *Thread.sleep* haciendo que un proceso esperase a que el otro liberase el fichero.
- La lectura del fichero que llegase a dicha carpeta.
- Las operaciones que se realizan sobre la base de datos. Esas operaciones las hice en lenguaje SQL.

Para esa conexión a Openshift®, utilicé la librería necesaria para usar *JDBC*, acompañada del comando mencionado en el punto anterior (*rhc*), para así poder usar la dirección generada.

Además, una vez terminada, se añadió un archivo de propiedades que almacenase direcciones, puertos, localizaciones de carpetas, usuarios y contraseñas de las que hiciese uso el programa, teniendo así un fichero al que acceder para modificar fácilmente cualquiera de estos parámetros, evitando tener que hacer cambios en la aplicación de manera interna.

Una vez acabado, subí a la máquina alojada en Amazon un ejecutable generado *.jar* (junto con sus carpetas necesarias para su funcionamiento) donde se ejecutaría junto al comando *rhc*. Además tuve que instalar los elementos necesarios para que todo ello funcionase sin problemas: Ruby, Git, JDK y *rhc*.

## Conexión a la otra máquina de Amazon

Una de las partes de esta aplicación consistía en responder a las peticiones, generando un fichero de respuesta indicando si había salido bien la operación o no. Había dos formas de respuesta dependiendo de quién hiciese dicha petición:

- Al terminal local que hacía una petición para ver si existía un dato. En este caso, este terminal era el que venía a recoger el fichero generado a la carpeta 'Respuestas', ya que mi máquina no tenía por qué conocer su dirección.
- A la otra máquina de Amazon. En este caso sí tenía que enviar el archivo a una carpeta en la que estuviese leyendo la otra máquina. Después de varias pruebas, conseguí realizar una conexión mediante *scp* y usando un archivo *.pem* que me permitía acceder con una *private key* y usuario personalizado.

## Corrección de errores

Una vez acabado todo, se unió junto al resto de partes del trabajo e hice los cambios necesarios en la aplicación y en la base de datos para que todo quedase coherente con el resto de elementos del sistema, además de resolver errores que fueron surgiendo de conexión o en el formato de los datos almacenados en OpenShift®.

## Participación de Robert:

He participado en el diseño e implementación de la arquitectura del sistema (compuesto por un terminal con lector de huella, el sistema de tratamiento de huellas y el sistema de base de datos).

He diseñado, implementado y documentado el sistema de tratamiento de huellas, lo cual ha implicado:

- Elegir (por consenso) el servicio de almacenamiento en la nube (Dropbox™) que se adapte a las necesidades del sistema, familiarizarse con la API que proporciona y programar el código.
- Buscar y elegir el software biométrico utilizado para tratar las huellas.
    - Familiarizarse con la documentación y la API de Python, de Opencv y realizar pruebas sobre imágenes.
    - Familiarizarse con la documentación del software proporcionado por el NIST, adaptar parte del código fuente a nuestras necesidades, compilar y probar.
- Estructurar los directorios y permisos del sistema tratamiento de huellas:
    - Establecer los directorios necesarios con los permisos adecuados, buscando la mayor encapsulación y seguridad posible.
    - Definir los usuarios necesarios con los permisos adecuados.
- Establecer el protocolo de distribución, descarga y formato de los ficheros con la información de huella, teniendo en cuenta:
    - El rendimiento, buscando que el sistema consuma lo menos posible para obtener y tratar las huellas.
    - El tiempo, tratando de que las operaciones ejecutadas por el sistema tarden el menor tiempo posible.
    - La seguridad, priorizando el tratamiento y distribución de las huellas de la forma más segura.
- Establecer el protocolo de administración de las operaciones requeridas por los terminales, implicando esto:
    - La distribución y estructura de los directorios de cada operación requerida.
    - La comunicación del sistema de tratamiento de huellas con los terminales y el sistema de base de datos para llevar a cabo las tareas.
- Programar toda la lógica de las operaciones llevadas por el sistema de tratamiento de huellas, la cual conlleva los dos puntos anteriores.

- Elegir un servicio en la nube (Amazon Web Services) donde hospedar el sistema de tratamiento de huellas, lo que supone:
    - Configurar los host, máquina Ubuntu x86_y x64 y máquina CentOS x86_y x64.
    - Crear *snapshots* del sistema de tratamiento de huellas para poder ser distribuido más fácilmente y tener *backup* del sistema.

He redactado mi parte correspondiente de la memoria, implementando los diagramas de flujo complementarios.

# Capítulo 5. Tecnologías descartadas.

La primera biblioteca que encontramos para tratar las huellas fue OpenCV (Open Source Computer Vision). Esta biblioteca implementada en C++ proporciona APIs con *bindings* en varios lenguajes, entre ellos Python, para tratar imágenes.

Para comparar huellas hicimos uso de varios algoritmos de detección de bordes. Pero los resultados eran demasiado inestables como para que éstos fueran utilizados para comparar huellas.

Todos las implementaciones exitosas que encontramos que usaban OpenCV para comparar huellas aplicaban *machine learning*. Lo que suponía tener entre 8-10 muestras de una misma huella para poder ser comparada con éxito. Por esta razón tuvimos que buscar otra opción para la comparación de huellas.

En cuanto al lector de huellas en un principio quisimos utilizar Python, pero no fue posible ya que el lector no dispone de un SDK para ello.

# Capítulo 6. Discusión y conclusiones

## 6.1 Discusión

### Comparación con lo existente

Este método de pago es más cómodo para el usuario ya que si olvida la tarjeta en casa o no la lleva porque está haciendo deporte o cualquier otro motivo, puede seguir pagando en caso de necesidad.

Con respecto a los modelos que ya existen que utilizan huellas dactilares, nuestro modelo tiene la gran ventaja de que las huellas no se almacenan ni se verifican en el terminal, de tal modo que si alguien roba un terminal no podrá obtener datos de ningún usuario, y menos suplantarle. Otra gran ventaja que podemos encontrar en nuestro método es que el usuario no tiene que aportar ningún dispositivo específico como puede ser con 'Apple Pay' o 'Samsung Pay'. El sistema más parecido al nuestro creemos que es PayTouch, inventado en España, con el que se puede pagar utilizando dos huellas en lugar de una huella y el pin. Nos diferenciamos de este método en el almacenamiento y verificación.

No hemos encontrado ningún modelo que trate como nosotros las huellas, ya que nosotros las encriptamos, particionamos y volvemos a encriptar para una gran seguridad e integridad de los datos de nuestros clientes.

### Líneas futuras

**Terminal de pago**

Para el funcionamiento de nuestro software en los comercios se requiere de un terminal de punto de venta. Nosotros hacemos una propuesta innovadora y práctica de terminal con un teclado a la izquierda, un lector de huellas a la derecha y la pantalla en el centro.


**Imagen 31 - Prototipo TPV**

**Lectura de huellas vivas**

Un buen adelanto en este método de pago sería la lectura de huellas vivas, el cual solamente lee la huella si nota que por dentro de ese dedo circula la sangre, ya que el lector actual podría leer de forma correcta una huella falsa que no sea un dedo real, es decir, una falsificación de dedo.

***Matching***

Mejorar el *matching* de huellas ya que cabe la remota posibilidad de que el mayor resultado de la comparación sea de otra huella y no la del usuario.

Solución: Terminales con espacio para escanear dos huellas (dedo índice y corazón). Todos los *matchs* de los dedos índices que superan la barrera son comparados con sus respectivos en la segunda huella del usuario, y el mayor *match* en conjunto es considerado el correcto. Esta forma de comparar ahorra recurso y tiempo, y reduce de forma considerable los falsos positivos.

**Borrado simultáneo**

Este prototipo de sistema impide registrar las huellas de más de un usuario de forma simultánea. Cuando la huella del primero es subida a la nube, los archivos comprimidos en los que va la del segundo usuario los sobrescribirá.

Solución: Utilizar una base de datos Redis en la nube. De esta forma al registrar usuarios éstos son guardados en Redis y todos los datos son sincronizados a la vez. Es decir, cada cierto tiempo se obtienen los datos de Dropbox™ y son añadidas todas las huellas nuevas.

**Capa de abstracción**

Hacer que los terminales no se comuniquen directamente con los sistemas encargados de realizar cálculos y consultas.

Solución: Añadir una capa más que se ocupe de comunicar con los terminales. En ella estarían los *watchdog* encargados de esa tarea, esta capa servirá de intermediario entre el *backend* (sistema de tratamiento de huellas y sistema de base de datos) y los terminales. Esto añadiría mayor seguridad a nuestra aplicación además de hacerla más estructurada.

***Hash* de huella**

Encontrar un algoritmo hash que clasifique los ficheros con los datos de la huella en la nube. Éste sustituirá al pin, que por el momento es el más eficiente que hemos encontrado.

Solución: A día de hoy no disponemos de una solución viable.

**Paralelización de datos**

El algoritmo de comparación bozorth3, a pesar de ser muy rápido, es un proceso que ejecuta comparaciones en serie.

Solución: La forma que implementa las comparaciones permite que éste se paralelice lanzando varios procesos o, lo más óptimo, que se implemente en CUDA. De esta forma podrá hacer una gran cantidad de operaciones en un tiempo razonable, incluso se podría sustituir el pin como *hash* por otro más simple que no implique la intervención del usuario.

## 6.2 Conclusiones

- Hemos conseguido un método altamente seguro de pago.
- El método actual es difícil de suplantar. En cualquier caso, las pocas probabilidades que existen de suplantación se eliminarían con el lector de huellas vivas.
- Esta solución de pago es bastante minimalista, lo cual la hace bastante fácil de utilizar por cualquier usuario.
- Nuestro método de pago libera del uso de la tarjeta. Para pagar no haría falta llevar nada, lo cual lo hace muy cómodo.
- El método implementado tarda una media de entre 15 y 20 segundos para la realización de cada pago

## 6.2 Conclusions

- We achieved a highly secure method of payment.
- Using our method, nobody could replace you. Any way, we can overcome the few possibilities to this happens with live fingerprint reader.
- This payment solution is very minimalist and it’s easy to use by anybody.
- Our pay method releases the use of cards: you wouldn’t need anything to pay and this is very comfortable.
- The implemented method takes between 15 and 20 seconds for each payment.

# Apéndice

## Apéndice 1. Diagramas de flujo

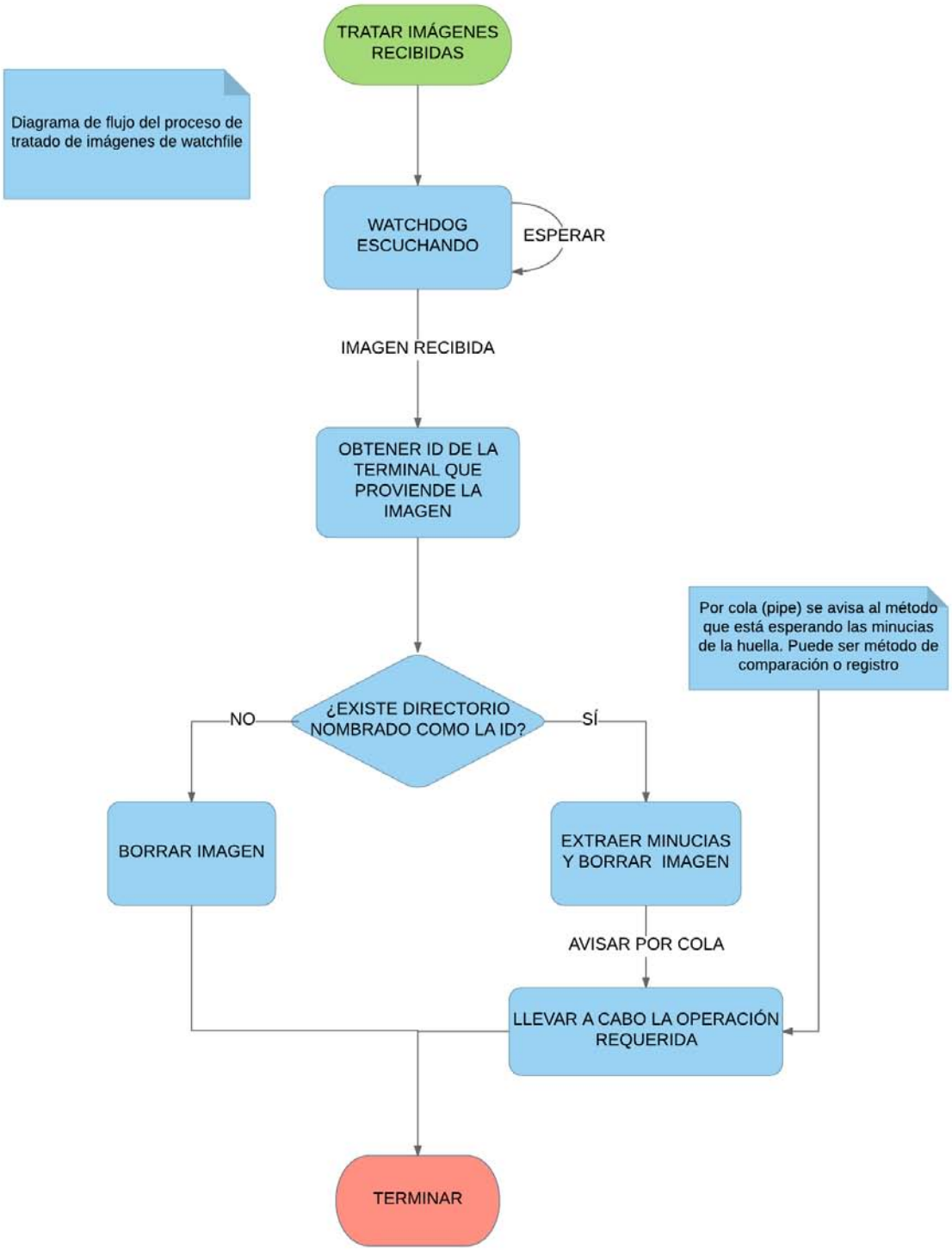


**Imagen 32 - Diagrama de flujo de tratamiento de imágenes de *watchfile***

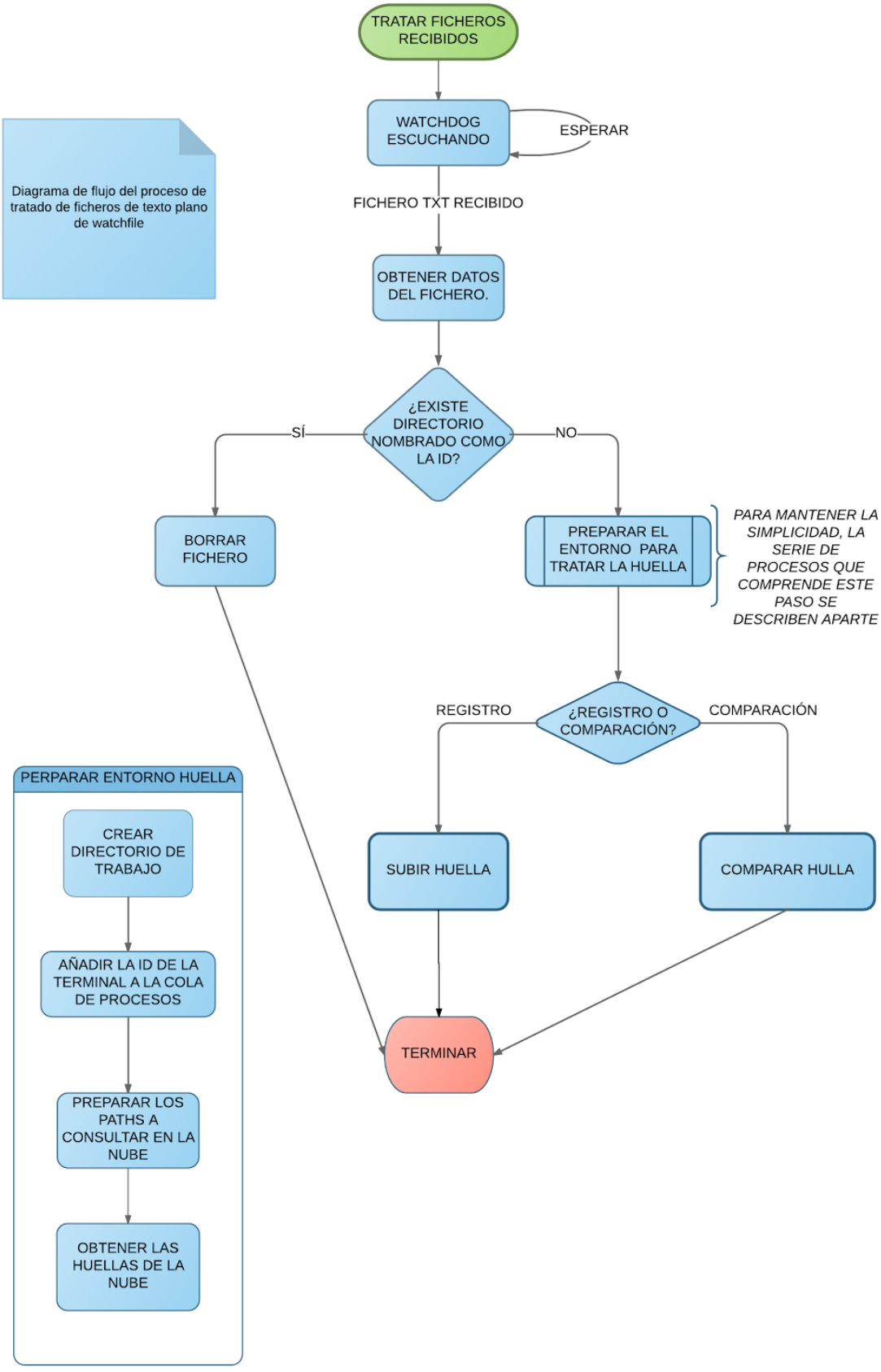


**Imagen 33 - Diagrama de flujo del proceso de tratamiento de texto plano de watchfile**

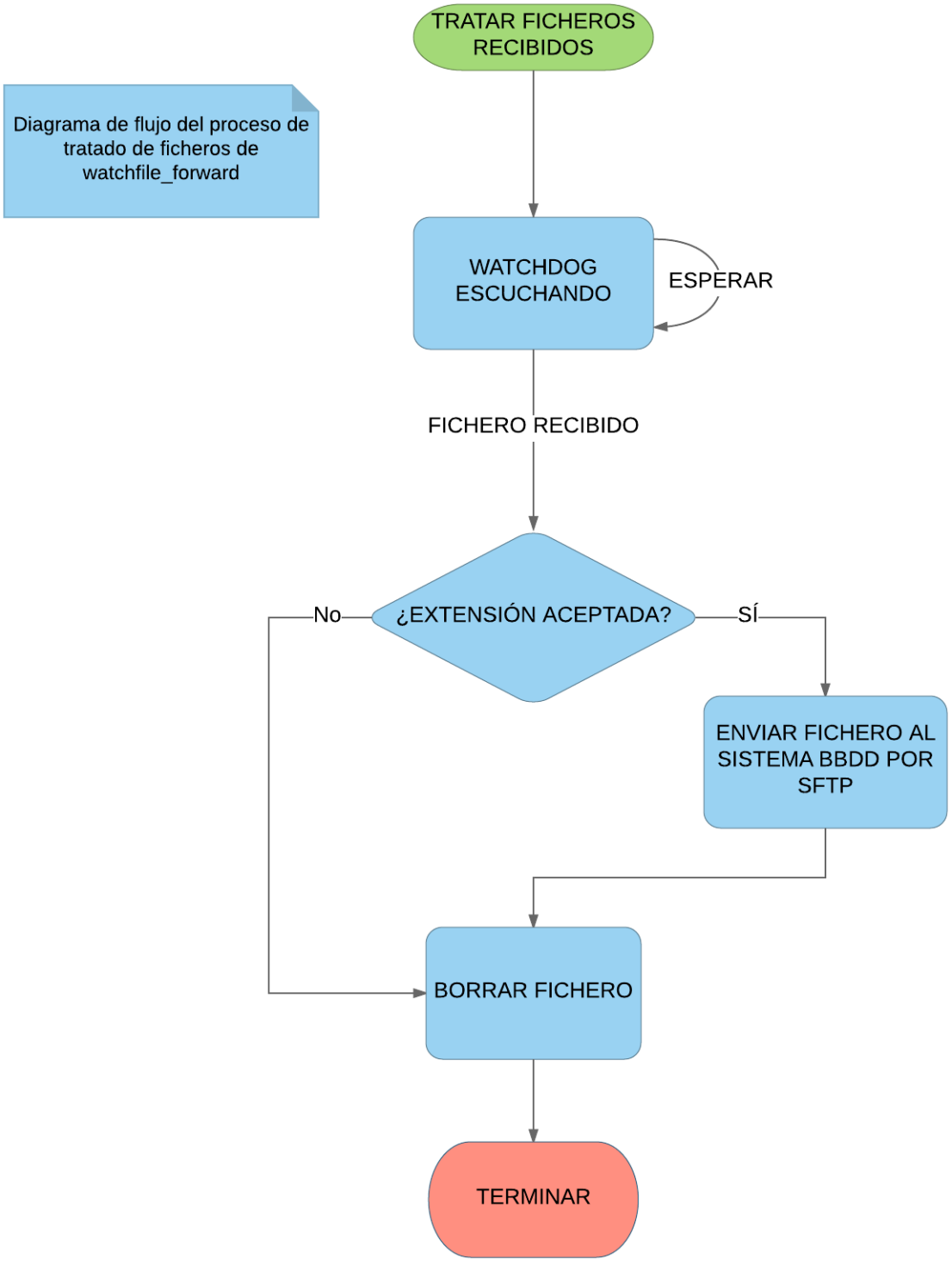


**Imagen 34 - Diagrama de flujo del proceso de tratamiento de ficheros de *watchfile*_forward**

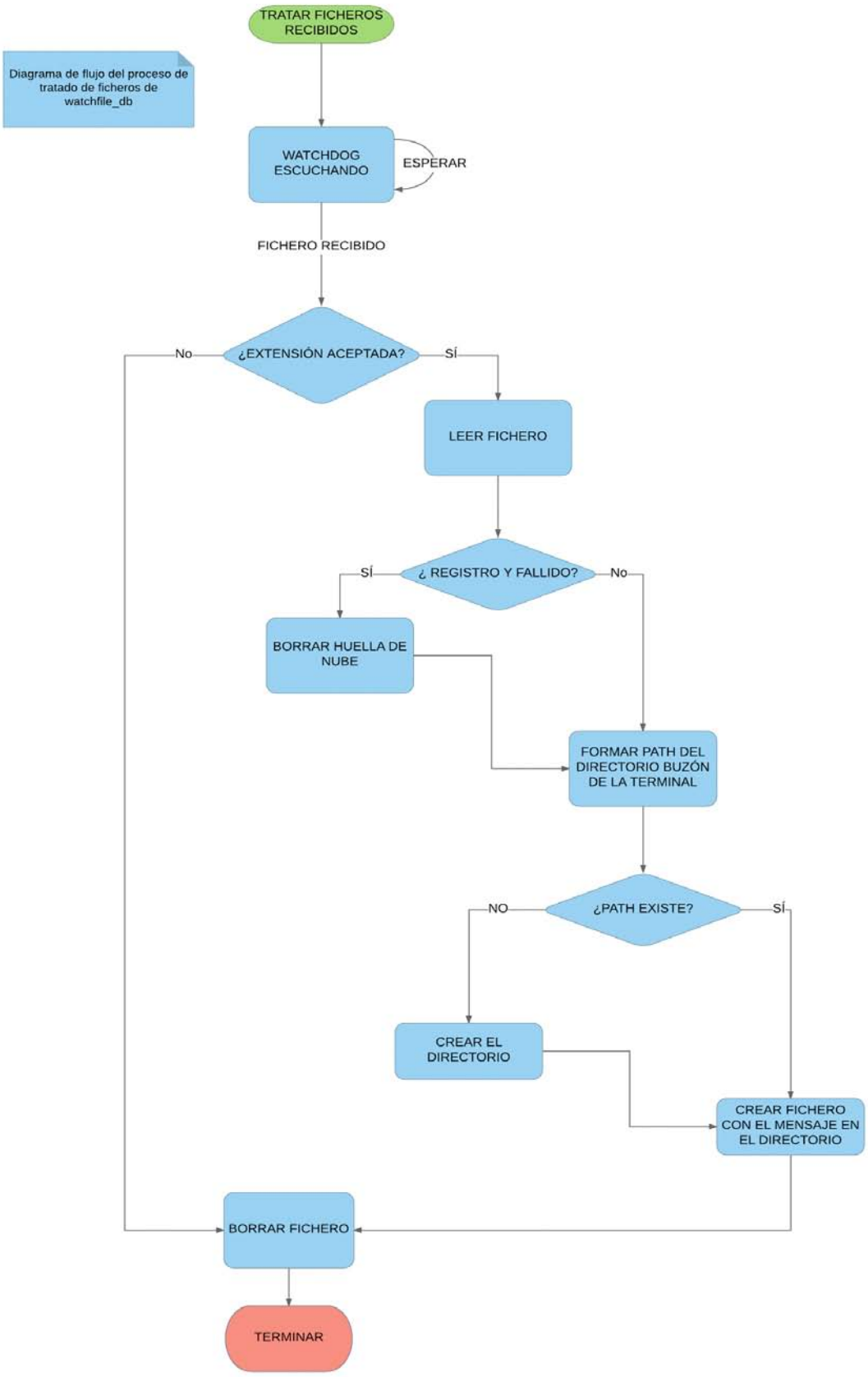


**Imagen 35 - Diagrama de flujo del proceso de tratamiento de ficheros de watchfile_db**

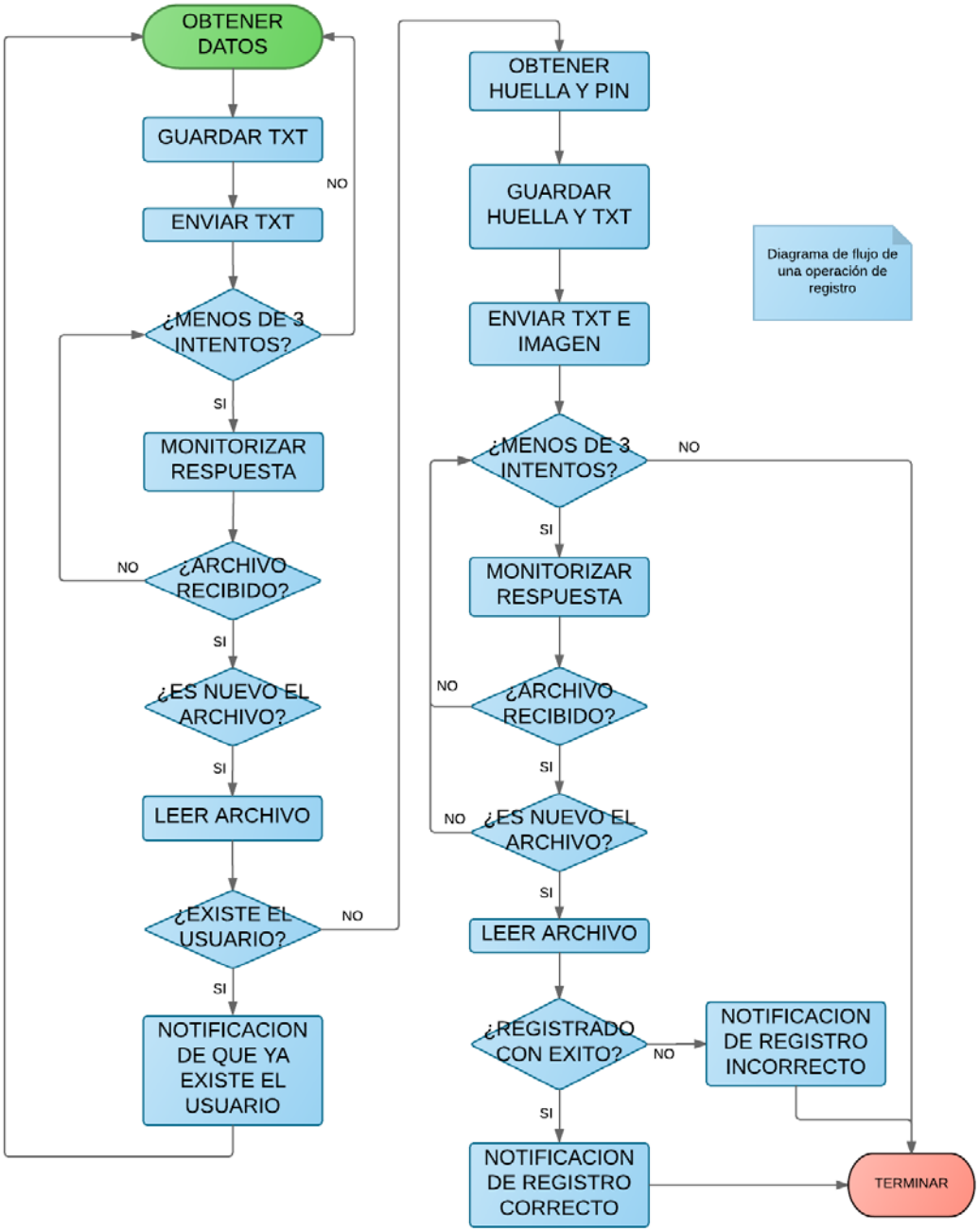


**Imagen 36 - Diagrama de flujo de un registro**

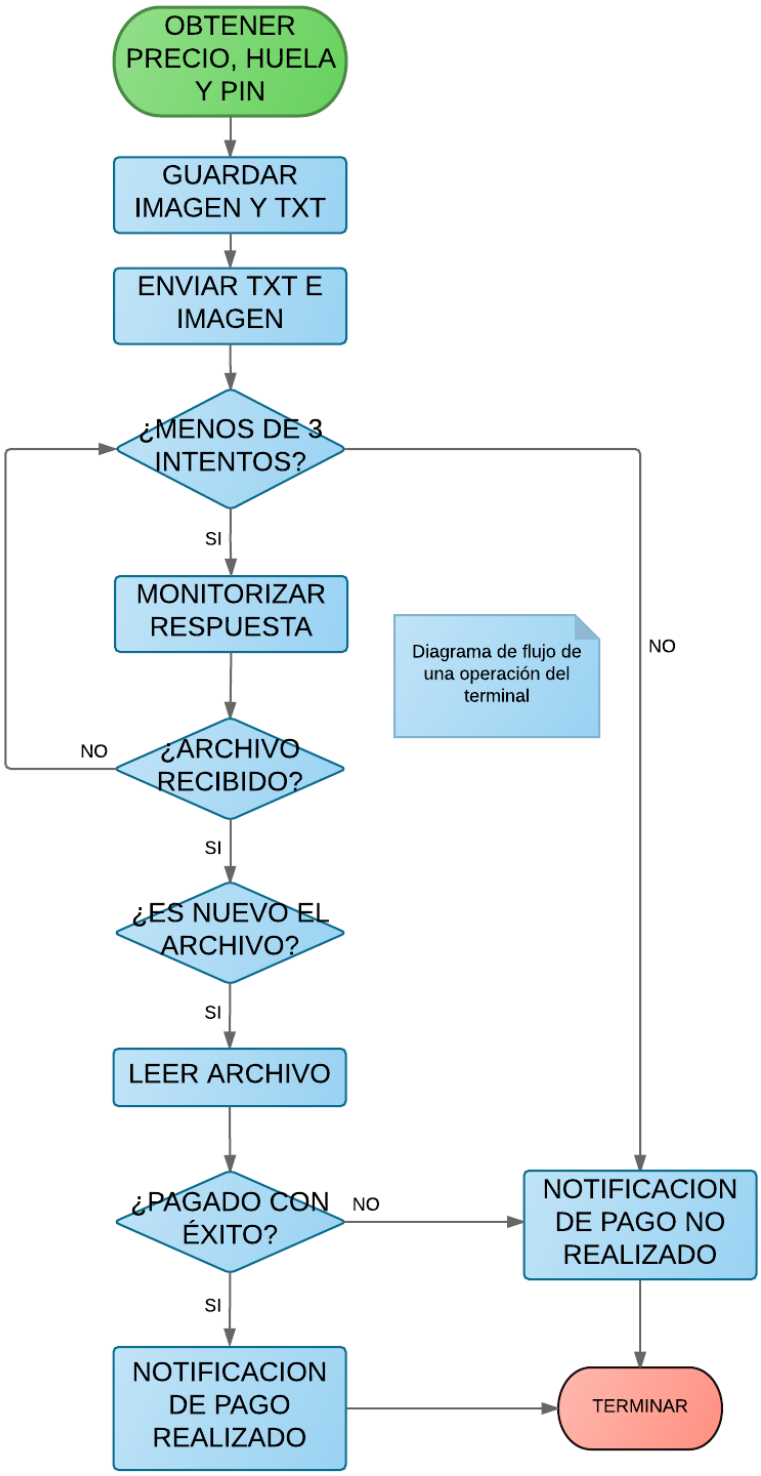


**Imagen 37 - Diagrama de operación de pago**

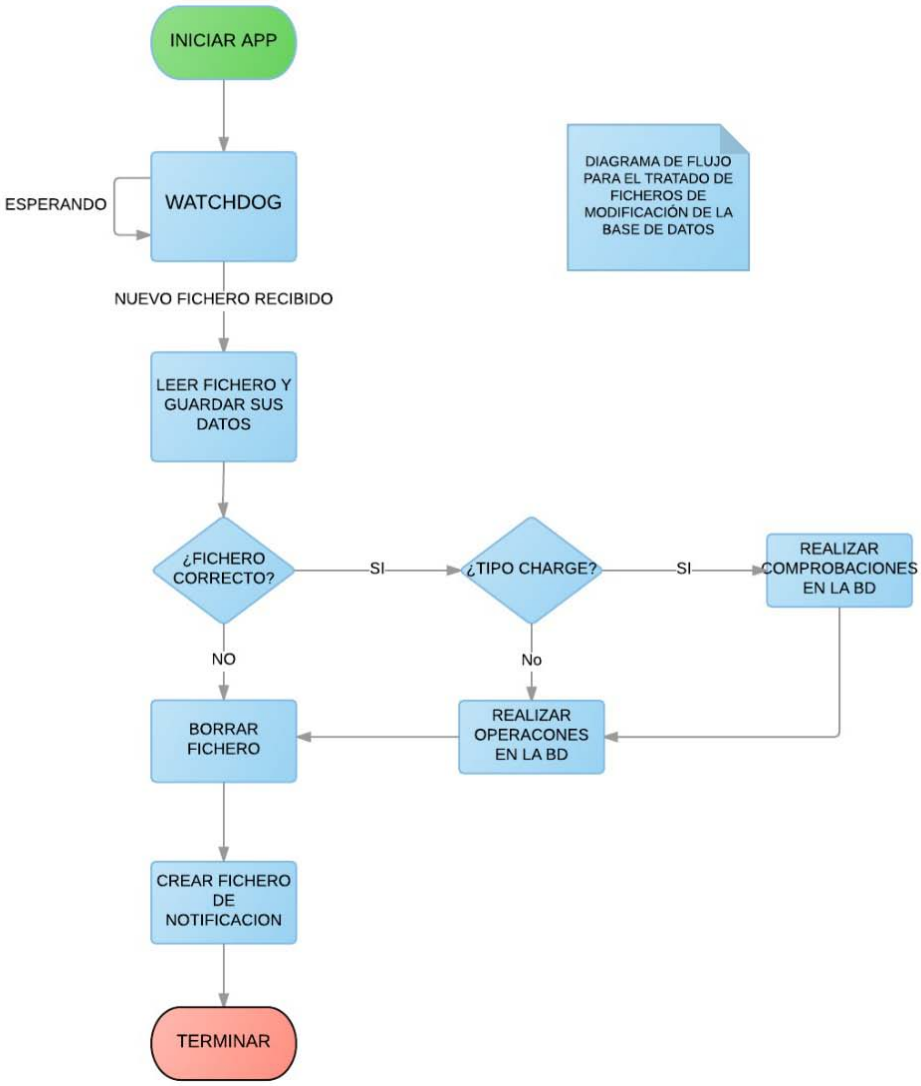


**Imagen 38 - Diagrama de flujo de operaciones en base de datos**

## Apéndice 2.Diagrama general de la aplicación

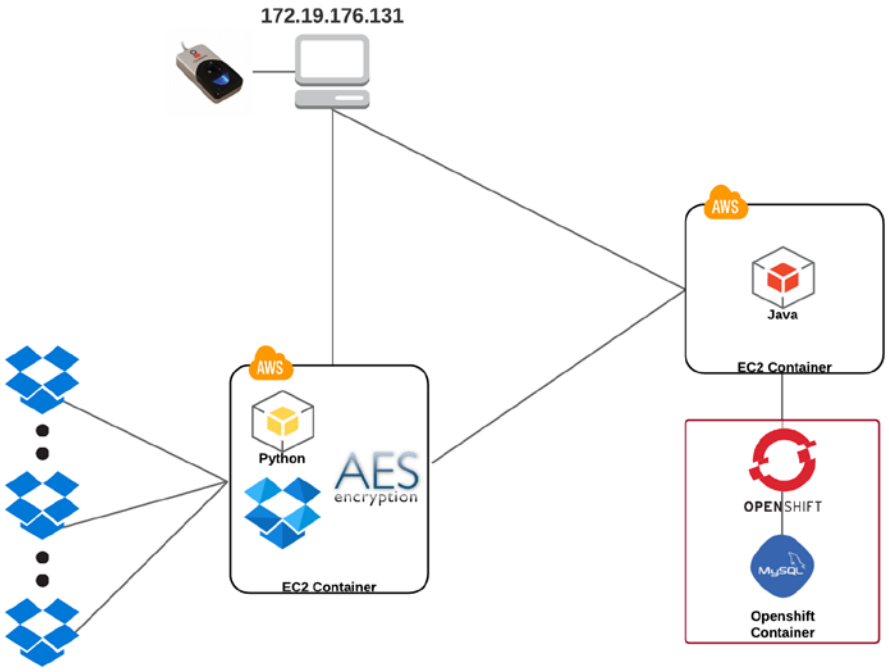


**Imagen 39 - Diagrama general de la arquitectura de la aplicación**

# Bibliografía


[1] Dropbox™ (9-12-2015), *Dropbox™ para desarrolladores* [en línea]. Available: https://www.dropbox.com/developers

[2] Drive (11-12-2015), *Drive para desarrolladores* [en línea]. Available: https://developers.google.com/drive/

[3] Azure (11-12-2015).*Azure* [en línea]. Available: https://azure.microsoft.com/es-es/

[4] iCloud (15/12/2015), *iCloud* [en línea]. Available: https://www.icloud.com/

[6] OpenShift® (23-3-2016), *OpenShift® para desarrolladores* [en línea]. Available: https://developers.OpenShift.com/managing-your-applications/client-tools.html

[7] Dzone (30-3-2016), Uso de OpenShift® en el entorno Eclipse [en línea]. Available: https://dzone.com/articles/OpenShift-how-create-web-0

[8] Amazon Web Services (19-5-2016), *Guia de usuario de acceso desde Linux a una VM* [en línea]. Available: http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AccessingInstancesLinux.html

[9] Linux Hispano (20/5/2016), *Copiar ficheros mediante scp usando un fichero pem para autenticarse* [en línea]. Available: http://www.linuxhispano.net/2013/03/31/copiar-ficheros-mediante-scp-usando-un-fichero-pem-para-autenticarse/

[10] Linux Academy (20-5-2016), C*onnect to Amazon ec2 using putty private key on windows* [en línea]. Available: https://linuxacademy.com/blog/linux/connect-to-amazon-ec2-using-putty-private-key-on-windows/

[11] Banco de España (25-05-2016), Encuesta uso tarjetas y cajeros [en línea]. Available http://www.bde.es/f/webbde/SPA/sispago/ficheros/es/estadisticas.pdf

[12] Garron (29-05-2016), *Copia de archivos* [en línea]. Available: - https://www.garron.me/es/articulos/scp.html#windows

[13] Mianamnesis (30-05-2016), Tipos de huellas [en línea]. Available: - http://www.mianamnesis.com/2011/12/dime-como-es-tu-huella-dactilar-y-te-dire-quien-eres/

[14] Huellas Forenses (30-05-2016), *Tipos de huellas* [en línea]. Available: - http://ellegadoenlascenadeldelito.blogspot.com.es/2014/05/puntos-caracteristicos-de-las-huellas.html

[15] BOE LOPD (31-05-2016), *BOE LOPD* [en línea]. Available https://www.boe.es/boe/dias/1999/12/14/pdfs/A43088-43099.pdf

[16] Signtechbiotronic (1-06-2016) Certificaciones lector huellas [en línea]. Available http://signtechbiometric.com/digitalpersona-crossmatch/

[17] Pysftp (25-05-2016). [en línea] Available: https://pypi.python.org/pypi/pysftp

[18] Pysftp (25-05-2016) Api's documentation. Available: http://pysftp.readthedocs.io/en/release_0.2.8/index.html

[19] Dropbox (10-10-2015) Python's Documentation. [en línea] Available: http://dropbox-sdk-python.readthedocs.io/en/master/index.html

[20] watchdog (3-11-2015) [en línea] Available: https://pypi.python.org/pypi/watchdog

[21] Watchdog (3-11-2015) Documentation. [en línea] Available: https://pythonhosted.org/watchdog/index.html

[22] Python 2.7 (1-10-2015) Documentation. [en línea] Available: https://docs.python.org/2/

[23] Amazon web services (15-05-2016). [en línea] Available: https://aws.amazon.com/es/

[24] NIST (20-09-2015) Image Group Fingerprint Overview. [en línea] Available: http://www.nist.gov/itl/iad/ig/fingerprint.cfm

[25] NIST (20-09-2015) Image Group Fingerprint Overview. [en línea] Available: http://www.nist.gov/itl/iad/ig/nigos.cfm

[26] NIST Release 5.0.0 - Test 5.0.0 - stable version - (03-04-2015) – http://biometrics.nist.gov/cs_links/nigos/docs/CHANGELOG.txt

[27] donde se ha descargado el código fuente [en línea] Available: nigos.nist.gov:8080/nist/nbis/nbis_v5_0_0.zip

[28] NIST (20-09-2015) Biometric Image Software. [en línea] Available: http://www.nist.gov/itl/iad/ig/nbis.cfm

[29] OpenCV (15-08-2015). [en línea] Available: http://opencv.org/

[30] OpenCV (15-08-2015) Python's Api documentation. [en línea] Available: https://opencv-python-tutroals.readthedocs.io/en/latest/index.html

[31] Eroski consumer (20-05-2016) estadísticas de pago dactilar por Eroski consumer [en línea] Available: http://www.consumer.es/web/es/economia_domestica/sociedad-y-consumo/2016/05/11/223730.php

[32] Digital Trends (13-05-2016) Pago en Japon [en línea] Avaliable: http://es.digitaltrends.com/seguridad/huella-dactilar-pagos-japon/

[33] ApplePay (25-05-2016) Apple muestra su método de pago con huellas dactilares [en línea] Available: http://www.apple.com/apple-pay/

[34] Samsung Pay (6-06-2016) Ya se puede pagar con Samsung Pay en España [en línea] Available: http://www.samsung.com/es/samsung-pay/

[35] Kimaldi (30-05-2016) productos de biometría [en línea] Available: http://www.kimaldi.com/productos/sistemas_biometricos/huella_vascular_y_facial/biometria_de_huella_dactilar

[36] Kimaldi (30-05-2016) pago por huella digital para restaurantes [en línea] Available: http://www.kimaldi.com/sectores/hoteles_y_restauracion/pago_por_huella_digital_para_restaurantes

[37] El periodico (15-12-2015) PayTouch para pagar con dos dedos [en línea] Available: http://www.elperiodico.com/es/noticias/tecnologia/sistema-pago-huella-dactilar-2512790

[38] Feed Back Today (15-123-2015) PayTouch para pagar con dos dedos [en línea] Available: http://www.feedbacktoday.net/entrevista/684/javier-peso-director-ejecutivo-de-paytouch

[39] Xataka (15-12-2015) [en línea] PayTouch para pagar con dos dedos Available: http://www.xataka.com/otros/paytouch-propone-pagar-usando-nuestra-huella-dactilar

# Anexos

## Anexo 1: Métodos de pago con huellas dactilares.

### Japón

Esta noticia la conocimos en el mes de abril de este año. Al parecer, en Japón se usa mucho este método. Según el artículo de 'digital trends', ya tiene más de 300 establecimientos (tiendas, hoteles, restaurantes…) que participan en las primeras pruebas para que poco a poco se estandarice de cara a los Juegos Olímpicos de 2020. En su caso necesitan una documentación añadida, como puede ser el pasaporte, al realizar un pago.

### Apple Pay

Una de las empresas más importantes, Apple, ya hace uso del pago por huella dactilar. Mediante su TouchID permite pagar de manera rápida y sencilla a través de los móviles de su marca. El número de tarjeta nunca se almacena en el dispositivo, por lo que el pago mediante huella permite perder el miedo a usar tus tarjetas o cuentas bancarias. Para poder utilizar Apple Pay necesitas el móvil, y además que sea de la compañía Apple.

En su página web se puede ver un video en el que demuestran su método.

### Samsun Pay

Este método es similar al de Apple, hay que ingresar las diferentes tarjetas y vincularlas con la huella dactilar. Al fin y al cabo no es un método de pago estrictamente hablando, al seleccionar que tarjeta se quiere utilizar en la pantalla del dispositivo, se activa con la huella dactilar y el pago se hace a través de NFC como con una tarjeta contactless.

### Lectores de huellas para todos

El mercado de las huellas dactilares es muy grande, y las capacidades para explotarlo muy extensas como hemos querido demostrar con este proyecto. En este enlace podemos ver la cantidad de diferentes tipos de tecnología que se está haciendo con el fin de avanzar en este campo. Diferentes dispositivos, cada día más, pensados para el uso doméstico.

Lectores de huella, controles de acceso, lectores en ratones…

### Ventajas del pago mediante huellas dactilares

Kimaldi nos explica cómo se implementa este método en un restaurante. Nuevamente se nos habla de lo rápido y sencillo que es de integrar en los distintos modelos de negocio y lo implementado que está en sitios como Corea y Japón.

Además nos indica cómo actúa cada tipo de usuario, desde el administrador, el encargado de gestionar la aplicación, hasta los clientes y camareros, que se encontrarán con un sistema cómodo y con muchas ventajas una vez se ha dado de alta el usuario interesado.

Una de las ventajas especificadas por la web es que estos sistemas combaten la suplantación de identidad después del robo de una tarjeta.

PayTouch

Es un método de pago nacido en Cataluña, en el año 2013 comenzó en el hotel Ushuaïa Ibiza Beach y Bricmania en búsqueda de financiación. Utiliza dos huellas para el pago, es bastante rápido, y guardan una función matemática en lugar de guardar las propias huellas para poder realizar la comprobación de qué cliente está pagando.

## Anexo 2: Ley Orgánica de Protección de Datos

La Agencia de Protección de Datos determina que los datos biométricos tienen condición de datos de carácter personal.

En el artículo 3a) de la LOPD expone que: "Datos de carácter personal: cualquier información concerniente a personas físicas identificadas o identificables."

Según lo que dice el artículo 3a) la huella digital identifica a un sujeto, su tratamiento no tiene mayor trascendencia que el de los datos relativos a un número de identificación personal.

En cuanto a que las huellas no van a ser tratadas sin consentimiento del interesado, el artículo 6.1 de la LOPD determina que "El tratamiento de los datos de carácter personal requerirá el consentimiento inequívoco del afectado, salvo que la ley disponga otra cosa." Por lo que previo al registro se tendría que firmar el documento donde se concretaría el uso que se le daría a la huella.

Además el artículo 5.1 de la misma nos recuerda que:

"Los interesados a los que se soliciten datos personales deberán ser previamente informados de modo expreso, preciso e inequívoco:

a) De la existencia de un fichero o tratamiento de datos de carácter personal, de la finalidad de la recogida de éstos y de los destinatarios de la información.
b) Del carácter obligatorio o facultativo de su respuesta a las preguntas que les sean planteadas.
c) De las consecuencias de la obtención de los datos o de la negativa a suministrarlos.
d) De la posibilidad de ejercitar los derechos de acceso, rectificación, cancelación y oposición.
e) De la identidad y dirección del responsable del tratamiento o, en su caso, de su representante. Cuando el responsable del tratamiento no esté establecido en el territorio de la Unión Europea y utilice en el tratamiento de datos medios situados en territorio español, deberá designar, salvo que tales medios se utilicen con fines de trámite, un representante en España, sin perjuicio de las acciones que pudieran emprenderse contra el propio responsable del tratamiento."

## Anexo 3: Estadísticas de uso de tarjeta

Encuesta del Banco de España

Con el tiempo, da la sensación de que ha habido un gran aumento del uso de tarjetas de crédito y debito, es decir, operaciones no pagadas con dinero. El Banco de España nos proporciona una estadística que ha realizado con la que esa sensación se traduce en realidad. Podemos observar que el número de transacciones realizadas a través de TPVs ha incrementado de una manera muy elevada, manteniéndose estable el número de retiradas de dinero en efectivo en cajeros automáticos. Sin embargo, el importe de los pagos por TPVs sigue siendo inferior a las extracciones en efectivo.

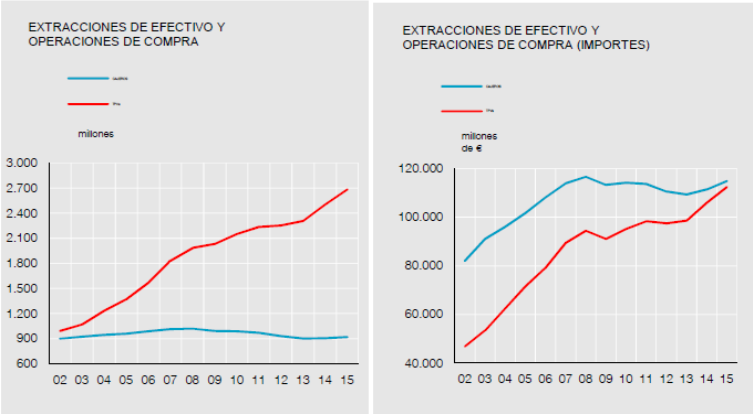


**Imagen 40 - Gráfica de extracciones/operaciones ***

Por otro lado el número de cajeros automáticos en los últimos años ha disminuido casi a la mitad y los TPVs, pese a decrecer en el año 2013, ahora están en el momento más álgido.

El número de tarjetas también ha aumentado desde el año 2000, aunque ha decrecido en los últimos 7 años.

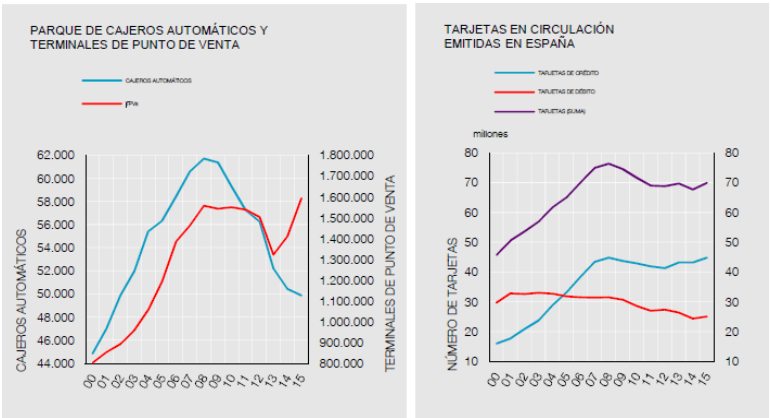


**Imagen 41 - Gráfica cajeros, TPVs y tarjetas ***

De todos estos datos, sacamos la conclusión de que los ciudadanos españoles preferimos cada año más el pago digital frente al metálico, lo que nos lleva a pensar que nuestra propuesta de pago mediante huellas dactilares tendría éxito en la sociedad.

* operaciones en miles e importes en millones de euros

## Estadística realizada por Eroski consumer

Eroski consumer ha publicado un artículo en el que se nos cuenta cómo la población española estaría dispuesta a hacer pagos mediante su huella dactilar. En concreto nos cuenta que un 45% lo ve con buenos ojos.

*"España, según Nicolas Huss, el director ejecutivo de Visa Europa, es un "terreno abonado para la tecnología de pago con huella digital, porque a los clientes españoles les gusta la innovación". Según la última encuesta de esta compañía, el 45% de la población estaría dispuesto a utilizar el móvil y su huella digital para hacer pagos, un porcentaje superior al de otros países.*

Además nos dice que es un método seguro y que es una tecnología ya integrada con el conocido servicio de PayPal en el móvil.

Podemos sacar en claro en ambos casos que la población española está dispuesta a dar este paso, que es un gran cambio. Por lo que pensamos que nuestro método de pago es un futuro tan cercano que es casi un presente.