Trabajo Fin de Grado

Reconocimiento facial mediante Machine Learning para solución de fichajes.

Autor

Guillermo Bambó Pueyo

Directores

Carmelo Marín Gavín (Director)

Jesús Gallardo Casero (Ponente)

Escuela Universitaria Politécnica de Teruel (Universidad Zaragoza)

Ingeniería Informática

2022

Reconocimiento facial mediante Machine Learning para solución de fichajes.

Resumen

Los sistemas de control de asistencia han sufrido cambios a lo largo de los años. Con la llegada del teletrabajo se innovó la forma de fichar en el trabajo, pero fue con la crisis sanitaria de 2020 que se puso de manifiesto la falta de seguridad de estos sistemas. Para solventar estos problemas de seguridad, se hace uso de tecnologías de *Machine Learning*, más concretamente el reconocimiento facial, el cual proporciona una solución para prevenir amenazas de seguridad, como pueden ser: la suplantación de identidad y el robo de credenciales.

Por tanto, este Trabajo Final de Grado (TFG) consiste en el desarrollo de una aplicación móvil que permitirá a los empleados de Movicoders S.L poder realizar el acto de fichar con un simple reconocimiento facial. Esta aplicación está enfocada a reducir el tiempo que necesitan los empleados para fichar y facilitar a la empresa el control de asistencia de una manera innovadora y sencilla.

Así pues, con este proyecto, aparte de intentar facilitar el fichaje a los empleados y a las empresas, se pretende aplicar los conocimientos adquiridos durante toda la carrera, así como los conseguidos durante el proceso de investigación de este trabajo.

Abstract

Attendance control systems had been changing through the last years. With the advent of the telework the way of signing in at work was innovated, but it was the health crisis of 2020 which highlighted the lack of security in these systems. To resolve this security problems, Machine Learning technologies are used, specifically facial recognition which provides a solution to prevent security threats, such as: identity fraud and credentials theft.

Therefore, this Final Degree Project (TFG) consist in the development of a mobile application (App) which will allow employees of Movicoders S.L to perform the act of sign in with a simple facial recognition. This application is focused on decreasing the time employees need to sign in and making it easier for the company to control attendance in a simple and innovative way.

Thus, this project, apart from trying to facilitate the registration of employees and companies is intended to apply the knowledge acquired during the entire degree, as well as those obtained during the research process of this project.

Índice

Tabla de ilustraciones

Siglas

AFIS: Automated Fingerprint Identification System, sistema automatizado de identificación por huella dactilar.

API: Application Programming Interface, conjunto de reglas y especificaciones para comunicar dos aplicaciones

APP: Application, se refiere normalmente a aplicaciones móvil.

iOS: Sistema operativo utilizado por los iPhone de Apple.

JSON: JavaScript Object Notation, formato ligero de intercambio de datos.

MVC: Modelo Vista Controlador

NPM: Node Package Manager, controlador e instalador de paquetes.

QR: Quick Response, código de barras

REST: Representational State Transfer, estilo de arquitectura.

SQL: Structured Query Language, lenguaje informático destinado a operar bases de datos.

TFG: Trabajo de Fin de Grado

UML: Unified Modelling Language

USB: Universal Serial Bus, tecnología usada para conectar ordenadores con dispositivos periféricos.

YARN: Yet Another Resource Negotiator, controlador e instalador de paquetes.

Capítulo 1: Introducción

1.1-Introducción

La forma de fichar en el trabajo suele estar basada y orientada a la utilización de un dispositivo físico, ya sea una tarjeta, una pulsera u otro objeto que verifique si el empleado a asistido al lugar de trabajo. En algunas empresas los ordenadores y equipos disponen de un lector de estos dispositivos integrado, por lo que, si no se dispone de ese dispositivo, no se puede acceder a fichar. La forma convencional consiste en apuntar a mano las horas en un papel o documento Word / Excel y aunque esta forma es propensa a errores es la más utilizada ya que muchos negocios no disponen de una gran cantidad de empleados.

Puesto que estas soluciones tienen amenazas de seguridad y precisan de la protección de los dispositivos o se comete errores en las horas apuntadas que difieren con la realidad se va a desarrollar una nueva solución que permita a cualquier usuario fichar utilizando tecnología de reconocimiento facial desde un dispositivo móvil.

La aplicación se integrará sobre una solución de fichajes ya existente que utiliza el método mencionado (dispositivo físico) y utiliza también otros métodos bastante seguros como son los códigos QR y el reconocimiento de huella dactilar (AFIS).

1.2-Estructura de este documento

Este documento consta de 8 capítulos:

El capítulo 1 muestra la introducción y la organización de este documento.

El capítulo 2 se centra en los objetivos y la motivación por hacer este proyecto.

El capítulo 3 es el marco teórico y se proporciona una descripción general del sistema, con el fin de conocer las principales funciones que debe efectuar, los datos asociados y los factores, restricciones, supuestos y dependencias que afectan al desarrollo, sin entrar en excesivos detalles. Se definirán los conceptos de forma básica.

El capítulo 4 consiste en el análisis donde se define detalladamente las funciones del producto, los usuarios a los que está destinada esta App y los requisitos que debe tener nuestro sistema tanto funcionales como no funcionales.

En el capítulo 5 se habla del diseño del proyecto, la arquitectura con la que está diseñada y los diagramas UML de actividades, estados y casos de uso.

El capítulo 6 se va a detallar el proceso de desarrollo del proyecto. Consiste en detallar las herramientas utilizadas y el código implementado para conseguir los objetivos. Se hablará también de las pruebas de evaluación realizadas.

El capítulo 7 se centra en el marco de investigación. Este se refiere a las posibles mejoras de la aplicación y las mejoras en el rendimiento durante el mantenimiento. También, otras posibles soluciones utilizando otras tecnologías.

En el capítulo 8 y final se da una valoración personal al documento.

Capítulo 2: Motivación y objetivos

2.1-Motivación

El 12 de mayo de 2020 entro en vigor la ley de control horario que obliga a registrar la hora de entrada y salida de cada uno de los empleados de la empresa. El objetivo de ésta, según el Gobierno español, es combatir el abuso laboral y las horas extra no abonadas.

Los controles de asistencia han ido cambiando a lo largo de los años. Al principio se realizaban en papel o en ordenador en un documento Word o Excel. Esta medida entrañaba un peligro: la existencia de errores al apuntar los datos o una posterior modificación, con lo que podría haber un desajuste entre la realidad y lo registrado. Los controles evolucionaron a sistemas que permitían fichar de forma presencial utilizando dispositivos físicos como una tarjeta, una pulsera u otros objetos con el fin de que conste que el empleado ha entrado o salido de la empresa. Esta forma presencial es la que más se utiliza hoy en día en todos los empleos que requieren que el empleado acuda a la empresa. Con la llegada del teletrabajo, los sistemas de control de asistencia se modificaron para que el empleado pueda fichar fuera del trabajo. Lo más común es utilizar aplicaciones de escritorio donde se ingresa las credenciales, también se pueden utilizar aplicaciones web. Estas medidas tienen distintas vulnerabilidades, entre ellas existe una que es muy peligrosa: la suplantación de identidad.

Se puede asumir que el nuevo diseño va a innovar la forma de fichar en el trabajo debido a que se va a usar tecnologías de reconocimiento facial y de inteligencia artificial las cuales son poco utilizadas y, a su vez, seguras y fiables. Con el uso de estas tecnologías se hace que la suplantación de identidad sea mucho más complicada.

Además, se va a hacer sobre un dispositivo móvil lo cual es también innovador ya que lo normal es fichar con una aplicación de escritorio, o un dispositivo físico habilitado para ello.

Se han analizado las tecnologías y entornos de desarrollo que se utilizan en la empresa para que la integración de la nueva funcionalidad se realice de la forma más sencilla y estructurada posible.

2.2-Objetivos

Objetivo general

El principal objetivo es el de analizar, diseñar e implementar una aplicación móvil utilizando un framework de React Native y tecnología Tensorflow para facilitar el control de asistencia de los empleados, a través del reconocimiento facial.

Objetivos específicos

1.- Desarrollar una aplicación móvil para controlar la asistencia de los empleados de la empresa Movicoders.

2.- Aportar una forma innovadora de controlar la asistencia de los empleados para la empresa Movicoders basada en la utilización de tecnología de reconocimiento facial.

3.- Integrar tecnologías novedosas que facilitan el desarrollo del proyecto y con las que no se ha trabajado.

4.- Realizar un proyecto de software real donde poner en práctica los conocimientos adquiridos en la carrera y en el desarrollo de este proyecto.

Capítulo 3: Marco teórico

3.1-Metodologías seguidas (SCRUM y de prototipado)

Se puede entender como la guía que debo seguir a lo largo de todas las etapas de mi proyecto (análisis, diseño, desarrollo y prueba) con el objetivo de cumplir con los requisitos del cliente y entregarle un producto de calidad en el menor tiempo posible.

SCRUM es una metodología ágil de desarrollo incremental que permite al cliente integrarse en el desarrollo. Aunque es una metodología de uso principalmente grupal, voy a utilizar aspectos de esta en mi metodología de trabajo para que a su vez me sirva de aprendizaje con esta forma de trabajar. Esta metodología se ejecuta en bloques temporales y fijos, donde a cada bloque se le denomina "**Sprint**" o **iteración**. Cada uno de ellos debe dar un resultado completo para ser entregado al cliente. [1]

Estos sprint suelen durar varios días y consisten en alcanzar los objetivos marcados en la reunión final del anterior sprint. Esto fomenta el trabajo en equipo y se lleva un seguimiento del proyecto con más detalle, ya que está bajo análisis durante todo el tiempo de desarrollo con un encargado de proyecto que es el que reparte las diferentes actividades y problemas a realizar y poder llevar un control sobre todos los demás integrantes del proyecto. En mi caso, al hacerlo individualmente, mi encargado de proyecto era mi tutor de la empresa, quien me decía qué estaba bien o estaba mal y si había alcanzado los objetivos o no.

A su vez, también he añadido aspectos de la metodología de **prototipado** para la creación de la aplicación. El prototipado es un modelo de desarrollo iterativo en el cual se elabora un diseño preliminar del prototipo, el mismo que es modificado junto al cliente. Además, permite detallar los requerimientos que se obtuvieron del cliente para luego ser presentados al usuario y buscar su aceptación. Es una estrategia que puede aplicarse en casi todas las actividades del proceso de software porque permite una constante modificación del modelo que se estará desarrollando, con el objetivo de que cada prueba que se realice se verifique el cumplimiento de los requisitos de usuario hasta alcanzar el producto final. [2]

La metodología del prototipado se divide en seis fases, de acuerdo con Pressman (2010):

Comunicación: en esta etapa se reúnen los participantes para definir los objetivos generales del software. En este proyecto se enumeraron los principales inconvenientes que genera el no poseer un sistema de control de asistencia que se detallará en posteriores apartados.

Plan rápido: con una entrevista se analizará los requerimientos principales para poder así diseñar un plan del prototipo inicial.

Modelado, diseño rápido: esta etapa se centra en la representación de aquellos aspectos del software que serán visibles para los usuarios finales (por ejemplo, disposición de la interfaz humana o formatos de la pantalla de salida).

Construcción del prototipo: el diseño rápido lleva a la construcción de un prototipo.

Desarrollo, entrega y retroalimentación: en este punto el modelo se entrega y es evaluado por los participantes, que dan retroalimentación para mejorar los requerimientos. La iteración ocurre a medida que el prototipo es afinado para satisfacer las necesidades de distintos participantes.

Entrega del desarrollo final: se realiza la entrega del producto final al cliente con todas las modificaciones solicitadas en la etapa de retroalimentación.

3.2-Definición del proyecto

Gracias a la implementación por parte de Facebook, el reconocimiento facial se popularizo para buscar nuevas funcionalidades, y fue con el modelo de teléfono IPhone X de Apple que se empezó a utilizar en el día a día por un gran número de personas como una medida de seguridad fiable. Esta medida de seguridad fue muy bien recibida por los consumidores ya que durante el 3er trimestre de su lanzamiento la empresa Apple vendió 41.3 millones de móviles y hoy en día sigue siendo utilizada en los modelos más recientes.

Los usuarios cada vez más apuestan por sistemas de seguridad para los controles de acceso y, en específico, el reconocimiento facial cada vez está siendo más utilizado. La confiabilidad que proporciona este sistema de acceso permite poder llegar a automatizar la mayor cantidad de procesos, reduciendo la necesidad de contratación de personal de vigilancia.

Con la llegada del Covid19, el teletrabajo estuvo a la orden del día y, con esto, llegaron los problemas de control de asistencia. Para los usuarios que no estuvieran relacionados con el teletrabajo, implementar una solución de fichajes fue tarea complicada y, en muchos casos, se optó por la compra de aplicaciones específicas con esta función lo que se traduce en un gasto adicional. Algunos de estos usuarios necesitaban más seguridad y es por esto que utilizaban el reconocimiento facial para evitar casi por completo los problemas de suplantación de identidad muy típicos en el teletrabajo y otros problemas de ciberseguridad.

Lo más común es hacer una aplicación de escritorio que permita al trabajador fichar en el trabajo, que normalmente es utilizando un usuario y contraseña para identificarse. En mi propuesta, voy a innovar el control de acceso, que será a través de una aplicación móvil, la cual facilita y agiliza el control de asistencia de la jornada de trabajo de cada trabajador.

Es por estas razones que el proyecto se centra en la integración de una función de reconocimiento facial en una solución de fichajes, detectando el rostro, obteniendo información detallada acerca de él y comparándolo con una lista de rostros ya verificados previamente almacenados en la base de datos y, por último, otorgando al usuario el permiso de "*login*" a la aplicación.

3.3-Fundamentación teórica

Como he comentado varias veces, la aplicación va a utilizar reconocimiento facial como proceso de identificación del empleado, pero antes de definirlo y explicar cómo se utiliza es mejor explicar primero lo que es un sistema biométrico y definir conceptos relacionados con este tipo de sistemas.

Sistema biométrico

Estrictamente hablando, el termino biometría se refiere a una ciencia que se ocupa del análisis estadístico de las características biológicas. En este sentido, la biometría es utilizada en un contexto de análisis de las características humanas con propósitos de seguridad.

En el caso biométrico, se refiere a aquellas características físicas y conductuales únicas que nos diferencian, características que son utilizadas para proporcionar un nivel más alto cuando hablamos de seguridad al unirse con la biometría.

La identificación o autentificación biométrica explota el hecho de que ciertas características son singulares e inalterables y, además, imposible de perder, transferir u olvidar, por lo que las convierte en confiables, amigables y seguras si las comparamos con las contraseñas (password).

Un sistema biométrico necesita de un sistema de captura mediante el cual se obtiene la imagen o muestra de la característica biométrica en cuestión. Posteriormente la información obtenida, debe ser tratada para que el ordenador pueda extraer de ésta los datos relevantes y necesarios para el buen funcionamiento del sistema, tras lo que, mediante algoritmos necesarios se obtiene la planilla con la cual se podrá hacer la identificación. [3]

Patrón facial

Es un rasgo biométrico que se puede diferenciar de cada persona, son una serie de puntos que tenemos en la cara que gracias a unas uniones y reglas podemos analizarlos y compararlos para determinar si es una misma persona o no.

Estos patrones se consiguen utilizando un conjunto de referencias o puntos concretos (suelen ser aproximadamente 68) cuya distancia entre ellos y configuración son distintos para cada

persona, confiriendo patrones únicos como una huella digital. También existen otros métodos más modernos como el análisis de textura superficial que mapea y cataloga la textura de la piel, como si cartografiase cada poro y cada arruga de la cara.

Hoy en día, aunque se ha investigado mucho y existen alternativas, solo se puede analizar una cara de frente. Pero esto se puede entrenar, ya que estamos utilizando un programa que, si se toman archivos históricos de esa misma persona de perfil o en escorzo, escalando y rotando la cara, el programa puede aprender a ajustar imágenes faciales orientadas de una manera diferente.

Antiguamente los algoritmos se escribían manualmente, ahora esta labor ha sido sustituida por el aprendizaje automático (Machine Learning), aunque los humanos aún están involucrados en su revisión y supervisión. [4]

Aprendizaje automático

Es un tipo de inteligencia artificial (AI) que proporciona a las computadoras la capacidad de aprender, sin ser programadas explícitamente. El aprendizaje automático se centra en el desarrollo de programas informáticos que pueden cambiar cuando se exponen a nuevos datos.

El proceso de aprendizaje automático es similar al de la minería de datos. Ambos sistemas buscan entre los datos para encontrar patrones. Sin embargo, en lugar de extraer los datos para la comprensión humana (como es el caso de las aplicaciones de minería de datos) el aprendizaje automático utiliza esos datos para detectar patrones en estos y ajustar las acciones del programa en consecuencia.

Los algoritmos del aprendizaje automático se clasifican a menudo como supervisados o no supervisados. Los algoritmos supervisados pueden aplicar lo que se ha aprendido en el pasado a nuevos datos. Los algoritmos no supervisados pueden extraer inferencias de conjuntos de datos. [5]

Reconocimiento facial

Una vez explicados los términos anteriores ya podemos entender el reconocimiento facial.

Es una tecnología capaz de identificar o verificar a un sujeto a través de una imagen, video o cualquier elemento audiovisual de su rostro. Reconociendo patrones de las imágenes o videos a partir de rasgos faciales que lo diferencian y junto a un algoritmo de aprendizaje automático que estará entrenado.

Existen dos modos de hacer un reconocimiento facial, en 2D y en 3D. En los últimos años, la velocidad y precisión de estos algoritmos ha experimentado nuevos avances gracias a la

investigación especialmente de las técnicas 3D. En la actualidad conviven diferentes tecnologías, pero todas siguen las mismas fases a la hora de reconocer una cara.

Las fases son las siguientes:

1. Capturan una imagen bidimensional o tridimensional del rostro a partir de una fotografía o vídeo, ya sea de una persona sola o entre la multitud.

2. Los algoritmos analizan las características faciales para obtener la información biométrica. En el reconocimiento 2D se utilizan los puntos de referencia como la nariz, la boca y los ojos, midiendo el ancho y la distancia entre ellos. En el reconocimiento 3D se utiliza además la forma, la textura, la profundidad, los contornos y la curva de la cara. Por eso son más precisos y suplen algunas deficiencias de iluminación, orientación o expresiones faciales de los sistemas 2D.

3. Antes de compararlas con otros rostros hay que normalizar los datos para que no haya errores no deseados y fácilmente evitables. Esta transformación, dependiendo el método de reconocimiento empleado, modifica unos parámetros u otros.

4. Después de hacer este análisis del rostro, el código registrado se coteja con las informaciones existentes en la base de datos que almacena otras *faceprints* (huella facial) y fotografías identificadas. En esta fase se hace evidente la necesidad de acumular grandes cantidades de información para obtener resultados.

Como resultado, el sistema devuelve un porcentaje de similitud, que dependerá del grado de coincidencia y de la información guardada en su base de datos. También puede mostrar otros detalles registrados, como el nombre, la dirección, edad, etc. [6]

Aplicación móvil

Consiste en una aplicación software destinada para dispositivos móviles y tabletas. El termino App es una abreviatura que suele utilizarse para referirse a una aplicación informática para dispositivos móviles y tabletas. Para el desarrollo de aplicaciones móviles tenemos diferentes sistemas operativos, algunos de ellos son: Android, iOS y Windows Phone entre otros. [7]

3.4-Perspectiva del proyecto

El sistema de información a implementarse es un software dependiente ya que se integrará en un software ya existente y hará uso de la base de datos.

- **Interfaz del sistema:**

Se utilizarán las imágenes de fotografías de los empleados que se albergan en la base de datos ya existente.

Se utilizará la librería React Native Camera para poder hacer uso de la cámara, la cual nos servirá para hacer una foto y aplicar sobre esta foto los distintos modelos de inteligencia artificial que servirán para detectar rostros, los puntos del rostro importantes y comparar el rostro obtenido de la foto con una lista de rostros almacenados en la base de datos que ya están verificados.

- **Interfaz con el usuario:**

Una primera propuesta de interfaz es la siguiente:

Con una dimensión de pantalla completa (teléfono móvil), nos mostrara la imagen previa de la cámara y pulsando el botón, podremos hacer una fotografía que guardaremos en el sistema del teléfono. Una vez guardada se aplicará la transformación a una imagen tensor para poder aplicar los modelos de Tensorflow para todo el reconocimiento facial. El sistema redirecciona a la vista de inicio donde verá un calendario y podrá fichar para entrar o para salir.

- **Interfaz con el hardware:**

Es necesario que los usuarios accedan mediante un dispositivo móvil ya sea Android o iOS.

- **Interfaz con el software:**

La aplicación necesita una interfaz para poder visualizar la base de datos, además de poder ver la hora de entrada y salida de los empleados.

- **Interfaz de comunicación:**

La comunicación entre dispositivos será mediante una API sobre Swagger, el teléfono manda la información en forma de cadena JSON y la API devuelve la información de las credenciales del usuario relacionado con esa cadena.

La comunicación con la base de datos será mediante consultas SQL, cuyo lenguaje de programación será PostgreSQL. Sobre una base de datos de teléfono móvil llamada Firebase.

- **Restricciones de memoria:**

En principio no hay ninguna restricción.

3.5-Alcance del producto

La propuesta del proyecto tiene como finalidad el desarrollo de una aplicación móvil el control de acceso de los empleados por medio de la identificación con reconocimiento facial usando el framework React Native y usando modelos de inteligencia artificial Tensorflow.

Este prototipo esta desarrollado con el framework de aplicaciones móviles de código abierto, creado por la red social Facebook, React Native, mediante el entorno de desarrollo Visual Studio Code o IntelliJ, utilizando el lenguaje de programación TypeScript que es una versión de JavaScript más rigurosa. Adicionalmente, se utilizará la librería React Native Camera y la librería de Tensorflow, la cual posee los modelos de inteligencia artificial necesarios para detectar caras, los puntos importantes de la cara detectada y la comparación de caras.

El proyecto también constara de una base de datos llamada Firebase ya implementada por la empresa Movicoders donde se guardan la información de todos los usuarios de la aplicación y es de donde puedo extraer un listado con las imágenes de las caras de los empleados, las cuales usaré para la comparación**.**

La aplicación tendrá una interfaz interactiva e intuitiva desarrollada en Android, lo cual facilitará su manejo y permitirá el acceso a los empleados de Movicoders, mediante reconocimiento facial, de los cuales serán almacenados los datos de sus rostros previamente (imagen de sus rostros).

Esta propuesta está destinada únicamente a la empresa Movicoders, sin embargo, podría extenderse a más empresas o instituciones que necesiten el manejo de control de acceso y asistencia.

Capítulo 4: Análisis del proyecto

4.1-Funciones del producto

Las principales funciones que debe realizar el producto son las siguientes:

1.- Reconocimiento facial usando una cámara. Esta función llevara a cabo la tarea de identificar y cotejar en la base de datos el rostro del empleado que está intentando fichar.

2.- Aplicar la transformación a imagen compatible. La cámara guarda imágenes en un formato no normalizado, esta función permite transformar una imagen para poder aplicar sobre ella modelos de inteligencia artificial Tensorflow.

3.- Aplicar los modelos de inteligencia artificial. Con los mencionados modelos, debemos poder reconocer un rostro, detectar sus puntos importantes y poder compararla con la lista de rostros ya verificados.

4.- Redireccionar a la vista apropiada. La aplicación debe direccionarnos a una vista según el resultado del reconocimiento facial, si es negativo volverá a la cámara y aparecerá una ventana emergente de error y si es positivo redirigirá a la vista inicial.

5.- Guardar hora de entrada. Esta función llevará a cabo la tarea de guardar la hora de entrada de un empleado cuando ha fichado en el inicio de su jornada.

6.- Guardar hora de salida. Esta función llevará a cabo la tarea de guardar la hora de salida de un empleado cuando ha fichado en el fin de su jornada.

7.- Guardar hora de salida por incidencia. Esta función llevará a cabo la tarea de guardar la hora de salida por una incidencia, además del motivo de la incidencia. Las incidencias podrán ser de varias causas.

4.2-Usuarios de la aplicación

Esta aplicación está destinada a todos los empleados de la empresa que quieren fichar para que conste su hora de entrada y de salida del trabajo utilizando una aplicación móvil por lo que, quien disponga de este dispositivo, podrá utilizar la aplicación. Existirá un usuario administrador que gestionará todos los registros de los usuarios.

4.3-Características de los usuarios

Nuevamente siguiendo estándares de accesibilidad la aplicación tiene que estar destinada para todo tipo de público que pueda trabajar en la oficina. Por lo que debe ser fácil de aprender, de usar y de recordar.

El nivel medio para poder navegar por la aplicación es un nivel bajo ya que la aplicación consta de tres elementos que están bien definidos para que no haya ninguna pérdida.

Si el usuario no tiene experiencia y no es capaz de navegar por la aplicación de forma intuitiva, puede hacer uso del botón de ayuda para aprender lo básico y necesario.

4.4-Aplicaciones existentes

Una lista con algunas aplicaciones que utilizan reconocimiento facial:

Veriff, proporciona a las organizaciones procesos inteligentes, precisos y automatizados de verificación de identidad en línea, previniendo del fraude de identidad y los malos actores. Mediante la base de datos más extensa del mercado, el motor de decisión inteligente de Veriff analiza miles de variables tecnológicas y conductuales en solo unos segundos, y compara personas con más de 9800 documentos de identidad oficiales sobre 190 países. Se utiliza en ámbitos muy variados como en educación, cripto, metaverso, *gaming*, cuidado de la salud y finanzas. [8]

Luxand, es una compañía privada de alta tecnología formada en 2005. Se centra en investigar las tecnologías de inteligencia artificial e identificación biométrica que permite a la empresa desarrollar un conjunto completo de herramientas y librerías para realizar reconocimiento facial automático. Los productos y tecnologías de Luxand se utilizan en compañías de identificación biométrica y de seguridad, bancos, industria de entretenimiento, medicina e industria cosmética sobre portales de entretenimiento en línea, chats y páginas web de todo el mundo. [9]

Face phi, empresa fundada en 2012 con sede en España, se expandió con tres sedes más localizadas en: Seúl, Corea del Sur (2019), Montevideo, Uruguay (2022) y Reino Unido (2022). Es una empresa experta en verificación de identidad digital de usuarios, especializada en *onboarding* digital y soluciones biométricas de autenticación. Proporciona procesos digitales más seguros, accesibles y libres de fraude apostando por la innovación con inteligencia artificial y *Machine Learning*, aplicando tecnología *blockchain* e introduciendo la identidad digital descentralizada. La empresa cuenta ya con 300 millones de usuarios en todo el mundo y más de 120 clientes, con una tasa de retención superior al 95%. [10]

4.5-Requerimientos del sistema

En esta sección se van a detallar los requisitos funcionales y no funcionales de la aplicación a desarrollar. Todas las funcionalidades desde las más simples hasta las más complejas. Los requerimientos que se van a mencionar están relacionados con el módulo de reconocimiento facial, obviando los otros módulos.

REQUERIMIENTOS FUNCIONALES

RF.1 La aplicación muestra una pantalla de carga al ingresar en la aplicación donde aparecerá el logotipo de la empresa.

RF.2 La aplicación mostrará una pantalla donde el usuario elige la forma de autentificarse para acceder al fichaje entre las opciones disponibles: *login* (usuario y contraseña), reconocimiento facial, códigos QR o reconocimiento de huella dactilar.

RF.3 La aplicación permitirá a los usuarios cerrar sesión en cualquier momento.

RF.4 La pantalla de reconocimiento facial dispondrá de un botón para realizar una fotografía.

RF.5 La aplicación transforma la imagen fotografiada a un formato Tensor.

RF.6 La aplicación aplicará modelos informáticos de TensorFlow para reconocer patrones faciales de la imagen fotografiada por la cámara.

RF.7 La aplicación recogerá el listado de rostros verificados de los empleados de la base de datos.

RF.8 La aplicación comparará el rostro fotografiado y transformado con el listado de rostros verificados.

RF.9 La aplicación mostrará una ventana emergente en la pantalla donde se mostrará el resultado del reconocimiento facial.

RF.10 La aplicación crea una cadena JSON para comunicar con la API alojada en Swagger para obtener las credenciales del empleado reconocido.

RF.11 La aplicación navegará a la pantalla correspondiente dependiendo del resultado del reconocimiento facial.

REQUERIMIENTOS NO FUNCIONALES

RNF.1 La aplicación deberá funcionar sobre dispositivos Android.

RNF.2 El listado de los rostros verificados se almacenarán en un servidor alojado en la nube.

RNF.3 La aplicación necesitará de conexión a la red para comunicarse con el servidor.

4.6-Análisis de requerimientos

Voy a analizar los requerimientos que posee este proyecto. En primer lugar, los **funcionales**:

RF.1: las pantallas de carga sirven para saber qué está haciendo el sistema, en este caso cuando se inicia.

Importancia: Es relativamente importante controlar el estado de la aplicación.

RF.2: gracias a la navegación podemos acceder a cada funcionalidad de la aplicación pulsando el botón correspondiente.

Importancia: Es muy importante poder navegar por la aplicación de forma fluida.

RF.3: la aplicación se puede cerrar en cualquier momento.

Importancia: el usuario tiene la capacidad de cerrar la aplicación en cualquier momento que quiera.

RF.4: existe un botón para poder hacer la fotografía.

Importancia: no es muy importante pero necesario.

RF.5: la imagen que se obtiene es en formato base64 y se tiene que transformar a un formato Tensor, que es compatible con los modelos que se aplicarán.

Importancia: es una parte esencial del proceso de reconocimiento facial utilizando modelos de TensorFlow.

RF.6: se aplican los modelos elegidos de TensorFlow para analizar las imágenes en busca de patrones faciales, estos modelos son: *face-detection* y *face-landmarks-detection*.

Importancia: parte esencial del proceso de reconocimiento facial.

RF.7: el sistema tiene que disponer del listado de rostros ya verificados que esta almacenado en la base de datos.

Importancia: es otra parte esencial del proceso de reconocimiento facial.

RF.8: se realiza una comparación de los resultados de la aplicación de los modelos para poder reconocer a un empleado.

Importancia: es la parte final del reconocimiento facial, que es muy importante ya que el resultado determina si el empleado es reconocido o no.

RF.9: cuando acabe el reconocimiento facial se mostrará en pantalla un mensaje con el resultado del proceso.

Importancia: es importante que el usuario conozca el resultado del reconocimiento facial, también se consideran buenas prácticas de diseño mantener informado al usuario de la aplicación.

RF.10: se utiliza una API de Swagger para realizar la autenticación del usuario empleado en el sistema, obteniendo las credenciales que permiten el acceso.

Importancia: es una parte fundamental para acceder a la aplicación.

RF.11: la navegación a la pantalla correspondiente se realiza dependiendo del resultado obtenido en el reconocimiento.

Importancia: es importante que la aplicación no se quede atascada y que el usuario no sepa lo que tiene que hacer.

Analizaré ahora los **no funcionales**:

RNF.1: la aplicación desarrollada está destinada principalmente a dispositivos Android, pero se desarrolla sobre un framework multiplataforma que permite, realizando pequeños cambios en el código, crear la aplicación para otro sistema operativo. React Native también ejecuta iOS.

Importancia: es importante que la aplicación se desarrolle y funcione sobre dispositivos Android como prioridad.

RNF.2: el listado de rostros de los empleados que ya estén verificados los almacenará el administrador en una base de datos Firebase que esta alojada en la nube.

Importancia: los rostros ya verificados forman parte del final del proceso de reconocimiento facial por lo que es muy importante.

RNF.3: se precisa de conexión a la red de internet para comunicar la base de datos con la aplicación móvil desarrollada.

Importancia: es muy importante que el dispositivo tenga acceso a internet.

Capítulo 5: Diseño del proyecto

5.1-Arquitectura del sistema

Se centra en una aplicación móvil diseñada con el framework React Native, que permite utilizar vistas (views) las cuales son un componente fundamental en React Native. Gracias a la navegación por estas vistas, el usuario puede elegir la forma de hacer el fichaje. Si el empleado elige la opción de reconocimiento facial, deberá centrarse en la cámara de su dispositivo móvil utilizando la cámara frontal.

La aplicación, con la ayuda del framework React Native Camera, captura la imagen de la cámara y se procede a transformar el resultado a un formato de Tensor compatible con los modelos de TensorFlow.

Estos modelos se aplican sobre la imagen y predicen la detección de rostros y los patrones faciales encontrados. Los modelos de TensorFlow están pre-entrenados con muchísimas imágenes por lo que los hace muy fiables y la confianza tiene valores muy altos. Utilizaremos el modelo *face-detection* para detectar el rostro del empleado y el *face-landmarks-detection* para detectar los patrones de la cara detectada previamente.

El sistema, al mismo tiempo, obtiene el listado de rostros ya verificados de la base de datos que utilizará para compararla a la imagen capturada por la cámara.

Si la comparación tiene un rango de confianza de más del 90%, se puede asegurar que es un empleado de la empresa, y al ser tan seguro se genera una cadena JSON que será enviada a la API de Swagger. La API nos devolverá las credenciales del empleado relacionado con ese JSON y ya podrá acceder a la vista donde podrá realizar el fichaje.

La vista de fichaje consiste en un calendario donde seleccionas el día (por defecto se selecciona el día actual) y dos botones: uno para las entradas y otro para las salidas.

Si el reconocimiento facial no ha superado el mínimo de confianza, se pedirá al usuario volver a repetir el proceso.

5.2-Diagramas UML (Unified Modelling Language)

Son gráficos que sirven para representar el comportamiento de los diferentes procesos que integran un sistema. Estos gráficos están escritos con lenguaje unificado de modelación (UML). Existen una diversidad de esquemas UML, sin embargo, para el desarrollo de este proyecto se utilizaron los siguientes: diagramas de casos de uso, diagramas de estado y diagramas de actividades.

Para el proyecto se logró establecer dos roles para la utilización del sistema. Estos son el usuario administrador y el usuario empleado. Estos dos serán los roles que puede adoptar un usuario cuando está interaccionando con el sistema.

Diagramas de casos de uso

En estos diagramas se detallan las actividades que tienen que realizar tanto el actor como el sistema.

El primer diagrama que voy a detallar es el de administrador:

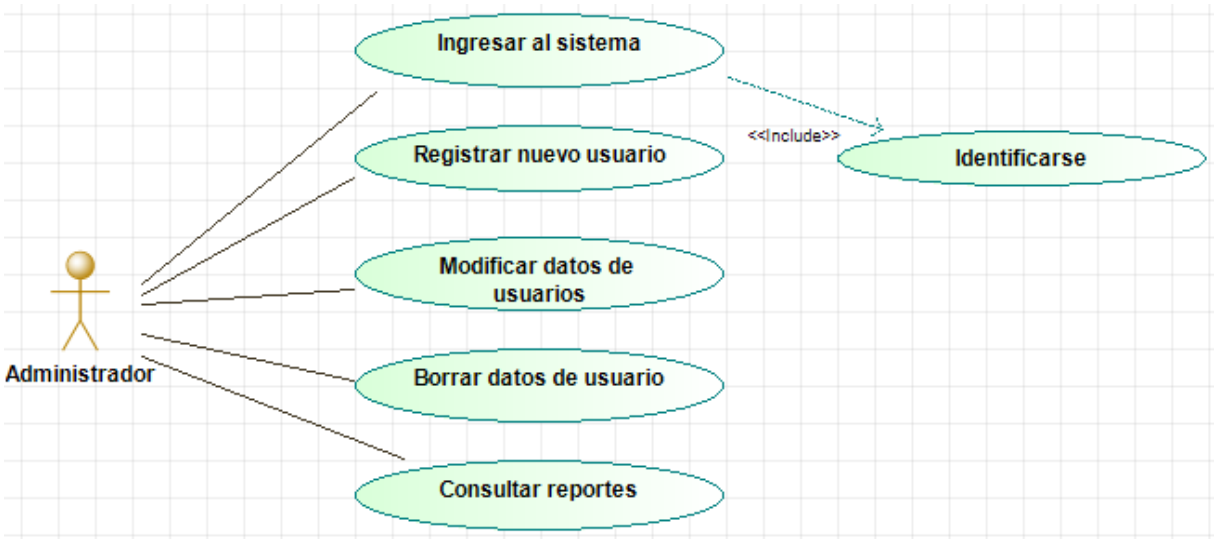

Ilustración 1. Diagrama de casos de uso del administrador

El siguiente diagrama es el del empleado:

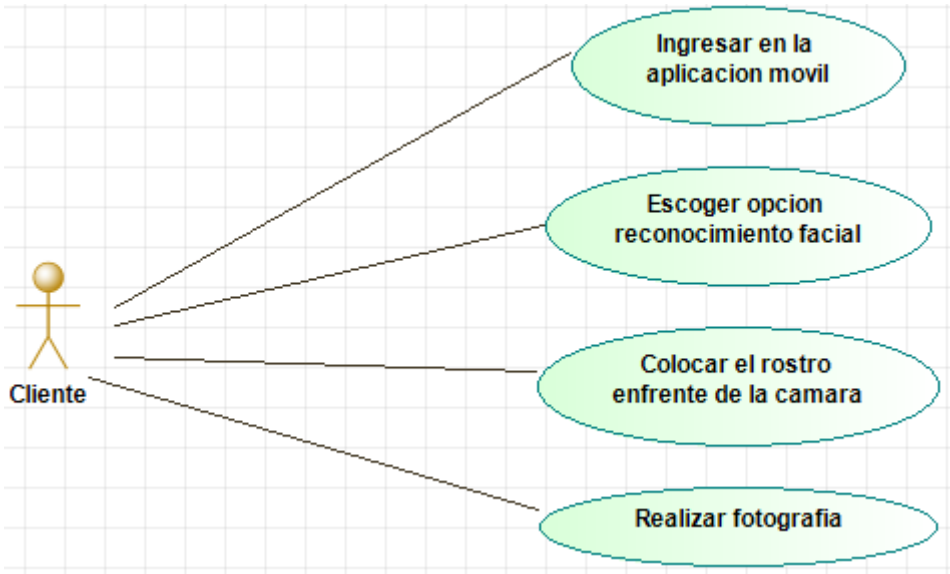

Ilustración 2. Diagrama de casos de uso del cliente

Diagramas de estados

Estos muestran todos los posibles estados por los que va a pasar un actor.

Ilustración 3. Diagrama de estados del administrador

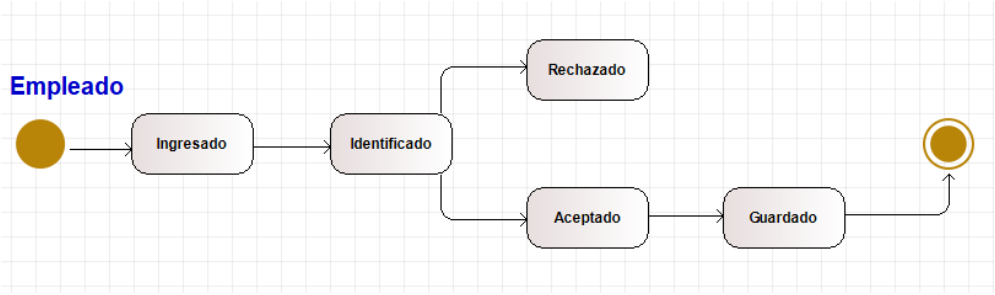

Ilustración 4. Diagrama de estados del empleado

Diagramas de actividades

Las actividades que registra un sistema tienen que ver con lo que realizan los usuarios que lo conforman.

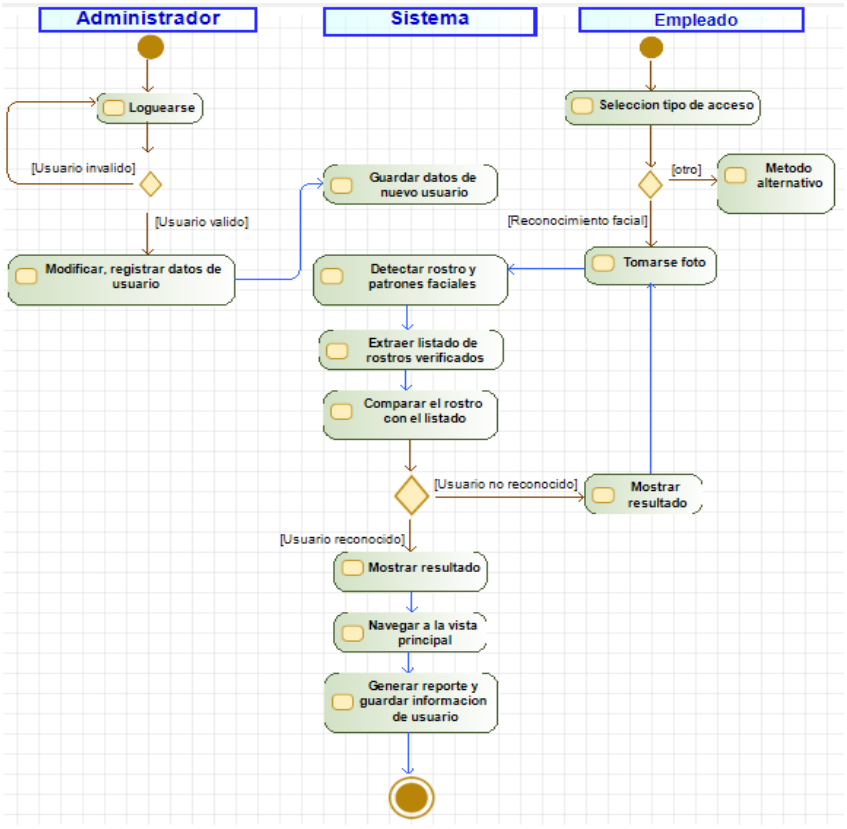

Ilustración 5. Diagrama de actividades

Capítulo 6: Desarrollo del proyecto

6.1-Herramientas utilizadas

React Native:

Es un framework de JavaScript usado para el desarrollo de aplicaciones móviles nativas para iOS y Android. Se basa en React, la librería de JavaScript elaborada por Facebook para crear interfaces de usuario, pero en lugar de dirigirse al navegador, se dirige a plataformas móviles.

Este framework usa una técnica que realiza llamadas asincrónicas al sistema operativo donde se ejecuta la aplicación, que llama a las API de widget nativas. Esto permite construir aplicaciones móviles que se ven como aplicaciones verdaderamente nativas usando una biblioteca de JavaScript y la API de React para diseño de aplicaciones web.

He elegido este framework ya que permite de manera sencilla la creación de aplicaciones móviles. La mayoría del código que se escribe se puede compartir entre plataformas por lo que React Native facilita el desarrollo simultaneo para Android y iOS.

React Native permite la utilización de TypeScript en lugar de JavaScript, que es una versión de JavaScript mucho más rigurosa con la escritura de código. [11]

Bibliotecas y componentes de React Native:

React Native tiene una gran comunidad detrás de él, así que hay un montón de librerías y componentes que puede utilizar. Sin embargo, se van a detallar sus áreas principales: navegación, administración del estado, animaciones, bibliotecas y componentes comúnmente usados.

Librerías importantes utilizadas en el proyecto

React Navigation

Es una biblioteca robusta que le permite implementar la navegación en las aplicaciones nativas mediante sus navegadores, como la pila (stack) de navegación, que es la que usaré en la aplicación. Esta navegación utiliza las rutas definidas en la aplicación que indican las pantallas almacenadas en la pila. [12]

Mobx

Es una herramienta para actualizar y administrar el estado de la aplicación. Tiene una corta curva de aprendizaje y es simple. Es el que se ha seleccionado por la empresa en lugar de Redux, que es igual de recomendable. [13]

React Native Camera:

Es una librería creada por la comunidad de React Native en 2015, con 281 colaboradores y 29 liberaciones, la cual, como dice el propio nombre, sirve para el uso de la cámara de un dispositivo móvil permitiendo realizar varias acciones con esta como tomar fotografías, grabar videos, reconocimiento facial, escaneo de códigos de barras (1D y 2D) y reconocimiento de texto.

Aunque posee la función de reconocer los rostros y los puntos importantes de cada rostro, voy a utilizar Tensorflow para realizar estos análisis de imagen, determinando con mayor precisión los nombrados puntos importantes faciales para crear patrones de rostros. [14]

TensorFlow

Creado por Google Brain, es una biblioteca de código abierto para la computación numérica y *Machine Learning* a gran escala. TensorFlow reúne una serie de modelos y algoritmos de *Machine Learning* y *Deep Learning* y los hace útiles mediante una metáfora común. TensorFlow permite a los desarrolladores crear gráficos de flujo de datos, estructuras que describen cómo los datos se mueven a través de un gráfico, o una serie de nodos de procesamiento. Cada nodo del gráfico representa una operación matemática, y cada conexión o arista entre nodos es una matriz de datos multidimensional, o tensor.

TensorFlow proporciona unos modelos ya creados y entrenados con una gran cantidad de datos. En específico, utilizaremos los siguientes modelos: face-detection y face-landmarks-detection para reconocer el rostro y los patrones faciales. También permite la creación de modelos usando el lenguaje de programación Python, de esta forma los modelos se ajustan a nuestras necesidades, pero hay que realizar un proceso de transformación para poder utilizarlo con React Native. [15]

Android

Es el sistema operativo más utilizado que, a día de hoy, cuenta con un 83% de todos los teléfonos inteligentes vendidos de España, siendo precedido de iOS con el 17% restante. Es por esto que se ha elegido realizar la aplicación utilizando este sistema operativo, aunque trabajando en nativo se puede utilizar el sistema operativo iOS haciendo alguna pequeña modificación ya que React Native permite la multiplataforma.

Creado por Google para dispositivos móviles con kernel de Linux, su idioma nativo es Java.

Como el proyecto se centra en implementar una aplicación nativa, estas se encuentran desarrolladas en los lenguajes de programación nativo que ofrece cada sistema operativo, con

las mejores prestaciones y rendimiento de potencial del hardware. Las aplicaciones nativas se diseñan y programan dependiendo de la plataforma que utilicemos, es por esto que la empresa había elegido como plataforma React Native que trabaja con los lenguajes de programación JavaScript y TypeScript siendo el ultimo el lenguaje que utilizare. [16]

Android Studio

La herramienta que permite vincular la aplicación que estamos diseñando a un emulador virtual o a un teléfono móvil que tengamos, lo cual facilita el proceso de mejora ya que puedes ver el progreso de tu aplicación.

La forma recomendada es utilizar un dispositivo móvil conectado por USB al ordenador ya que el emulador precisa de mucha memoria RAM, además de tener una buena cantidad de espacio de almacenamiento ya que existen distintas versiones de Android.

En la aplicación que voy a diseñar se ha utilizado una versión de Android 10 ya que es la más estable y recomendada por Android para el desarrollo de Apps. [17]

Firebase

Una de las herramientas más destacadas y esenciales de Firebase son las bases de datos en tiempo real. Estas se alojan en la nube, son No SQL y almacenan los datos como JSON. Permiten alojar y disponer de los datos e información de la aplicación en tiempo real, manteniéndose actualizados, aunque el usuario no realice ninguna acción.

Firebase envía automáticamente eventos a las aplicaciones cuando los datos cambian, almacenando los datos nuevos en el disco. [18]

NodeJS

Node.js es una librería y entorno de ejecución para JavaScript de código abierto, asíncrono y dirigido por eventos. Como código, es empleado con el objetivo de desarrollar aplicaciones web lo más optimizadas posible, rápidas y que soportan peticiones simultáneas de un gran número de usuarios**.**

Aunque además de JavaScript, Node.js también puede ejecutar programas codificados en otros lenguajes de programación que puedan trasladarse a JavaScript, tales como TypeScript o CoffeeScript.

Las tecnologías que se emplean para ejecutar las acciones en el servidor funcionan a través de peticiones aisladas**.** De esta forma, Node.js logra atender a un gran volumen de peticiones simultáneas**.** Es una solución tecnológica que ofrece buenos resultados para gestionar un gran

número de conexiones que coinciden en el tiempo en nuestro servidor. Esto es ideal para aplicaciones móviles.

Además, Node.js utiliza un I/O (entrada/salida) de tipo asincrónico a través de su Bucle de Eventos (Event Loop). Las tareas se ejecutan de forma simultánea, a diferencia de lo que ocurre en otros lenguajes de programación tradicionales cuya forma es lineal. La simultaneidad hace que los procesos de trabajo se aligeren y tengan tiempos de ejecución más cortos, se consiguen mayores velocidades de procesamiento y, por tanto, se obtienen aplicaciones de mayor rendimiento. [19]

NPM

Node Package Manager (controlador de paquetes de nodo) es un gestor de paquetes para JavaScript, es decir, sirve para instalar y gestionar versiones de paquetes y librerías js.

Esta herramienta funciona de dos formas: como un repositorio utilizado para la publicación de proyectos Node.js de código abierto y como herramienta de línea de comandos que ayuda a interactuar con plataformas en línea como navegadores y servidores, que ayuda a instalar y desinstalar paquetes, gestión de versiones y gestión de dependencias necesarias para ejecutar un proyecto. [20]

Visual Studio Code

Se va a utilizar como entorno de trabajo, podemos utilizar cualquier lenguaje de programación y utilizar React Native. Posee infinidad de extensiones que crean los propios usuarios y ayuda al desarrollo de proyectos.

Con el controlador de paquetes NPM, instalaremos las librerías y los módulos necesarios para escribir todo el código, el controlador instalará las dependencias necesarias. [21]

6.2-Propuesta de código

Como he comentado, la aplicación va a ser integrada en una aplicación ya existente. Es por esto que voy a detallar aquí la parte de código que he implementado y utilizado para realizar la función de reconocimiento facial y como lo he integrado en la aplicación ya existente facilitada por la empresa.

La funcionalidad a integrar se centra en una solución de *frontend* por lo que se hace mucho más sencilla la integración.

El reconocimiento facial se divide en cuatro fases que utilizaré para desglosar el código y se comprenda mejor la explicación del proceso utilizado. Estas fases son: adquisición de la

imagen, transformación de la imagen, detección de rostro, detección de patrones faciales y reconocimiento facial mediante comparación con la lista de rostros verificados.

Adquisición de la imagen

Para capturar la imagen es necesario utilizar el componente de React Native Camera dentro del método de renderización de la vista. También es estrictamente necesario asignar la referencia del componente en una variable global o un atributo de la clase de la vista que será renderizada donde se esté implementando. He creado un componente que será la cámara que importare en la vista donde la tengo que utilizar. [22]

```
render() {
  return (
    <>
      <RNCamera
        ref={ref => {
          this.camera = ref;
        }}
        captureAudio={false}
        style={FRecStyles.preview}
        type={RNCamera.Constants.Type.front}
        androidCameraPermissionOptions={{
          title: 'Permission to use camera',
          message: 'We need your permission to use your camera',
          buttonPositive: 'Ok',
          buttonNegative: 'Cancel',
        }}>
        {' '}
        <TouchableOpacity
          activeOpacity={0.5}
          style={FRecStyles.btnAlignment}
          onPress={this.takePicture}>
          <Icon name="camera" />
        </TouchableOpacity>
      </RNCamera>
    </>
  );
}
```

Ilustración 6. Código para utilizar la librería React Native Camera

Como podemos observar en la imagen se crea la referencia de la cámara con ref, se piden los permisos de utilización de la cámara y se crea un botón para realizar la fotografía con el evento que se genera al presionar el botón (onPress) que ejecuta la función takePicture (tomarFotografia).

Es necesario que tenga unas opciones configuradas para que las imágenes sean lo más similares posible a las almacenadas en la base de datos. Además, cuando se realiza una fotografía con la cámara hay que asegurarse de que la orientación es la correcta.

La iluminación es importante, ya que el reconocimiento facial es muy sensible a los cambios de iluminación y puede afectar a la hora de detectar los puntos de los patrones faciales.

Aunque puede funcionar con fotos de rostros perfilados, lo recomendable es que el usuario este mirando al frente y lo más centrado posible.

```
const takePicture = async () => {
  if (this.camera && !this.state.takingPic) {
    let options = {
      quality: 0.85,
      fixOrientation: true,
      forceUpOrientation: true,
      base64: true,
    };
    this.setState({takingPic: true});
    try {
      const data = await this.camera.takePictureAsync(options);
      Alert.alert('Success', JSON.stringify(data));
    } catch (err) {
      Alert.alert('Error', 'Failed to take picture: ' + (err.message || err));
      return;
    } finally {
      this.setState({takingPic: false});
    }
  }
};
```

Ilustración 7. Código para realizar una fotografía

La imagen se guarda con formato base 64 en un *hook* llamado *image* que se declara al principio de la clase y se rellena de información utilizando el siguiente comando.

```
setImage([data]);
```

Ilustración 8. Hook para guardar la imagen

El setImage debe ir dentro del método de tomar la fotografía, después de la primera alerta.

Transformación de la imagen

La imagen hay que transformarla a un formato compatible con los modelos de TensorFlow, que tienen que estar normalizados ya que están pre entrenados con imágenes Tensor. Para esto he creado una función en la vista donde tengo que utilizarlos. [23]

```
const transformImageToTensor = async data => {
  //.ts: const transformImageToTensor = async (uri:string):Promise<tf.Tensor>=>{
  //read the image as base64
  const img64 = await FileSystem.readAsStringAsync(data, {
    encoding: FileSystem.EncodingType.Base64,
  });
  const imgBuffer = tf.util.encodeString(img64, 'base64').buffer;
  const raw = new Uint8Array(imgBuffer);
  let imgTensor = decodeJpeg(raw);
  const scalar = tf.scalar(255);
  //resize the image
  imgTensor = tf.image.resizeNearestNeighbor(imgTensor, [300, 300]);
  //normalize; if a normalization layer is in the model, this step can be skipped
  const tensorScaled = imgTensor.div(scalar);
  //final shape of the tensor
  const img = tf.reshape(tensorScaled, [1, 300, 300, 3]);
  return img;
};
```

Ilustración 9. Función para transformar la imagen a formato Tensor

He utilizado un sistema de ficheros para poder acceder con facilidad a la imagen y métodos de la librería de TensorFlow que harán la transformación.

Antes he tenido que importar las librerías relacionadas con TensorFlow:

```
import * as tf from '@tensorflow/tfjs';
import {bundleResourceIO, decodeJpeg} from '@tensorflow/tfjs-react-native';
```

Ilustración 10. Importaciones necesarias para TensorFlow

Detección del rostro y patrones faciales

Necesito ahora utilizar el modelo de TensorFlow para detectar el rostro de la imagen que ha capturado la cámara. TensorFlow tiene un proceso de utilización de los modelos que difiere con otras tecnologías. Necesitamos crear un detector con una configuración específica para utilizar el modelo.

Para descargar los modelos simplemente hay que utilizar el controlador de paquetes NPM con el comando:

npm i @tensorflow-models/face-detection y ***npm i @tensorflow-models/face-landmarks-detection***

Es necesario importarlos donde se van a utilizar:

```
import * as faceDetection from '@tensorflow-models/face-detection';
import * as faceDetectionLandmarks from '@tensorflow-models/face-landmarks-detection';
```

Ilustración 11. Importaciones de los modelos a utilizar

El primer modelo detectará la cara y el segundo obtendrá todos los puntos importantes del rostro. Se devolverá una predicción que contiene un cuadro para delimitar el rostro y los puntos importantes. En la siguiente imagen se muestra la forma de esta predicción. [24]

```
[
  {
    box: {
      xMin: 304.6476503248806,
      xMax: 502.5079975897382,
      yMin: 102.16298762367356,
      yMax: 349.035215984403,
      width: 197.86034726485758,
      height: 246.87222836072945
    },
    keypoints: [
      {x: 446.544237446397, y: 256.8054528661723, name: "rightEye"},
      {x: 406.53152857172876, y: 255.8, "leftEye },
      ...
    ],
  }
]
```

Ilustración 12. Ejemplo del resultado de la predicción

El cuadro es representado por el *box* y tiene los 4 puntos necesarios para hacer un cuadrado alrededor de la cara.

Los *keypoints* son los puntos importantes y están representados por x e y que son las coordenadas actuales del punto en los pixeles de la imagen. También, alguno de los puntos tiene un nombre especifico, algunos de estos son: ojoDerecho, ojoIzquierdo, Nariz, centroBoca, orejaIzquierda, etc.

Estos puntos se obtienen gracias a una plantilla que se utiliza, y que sirve para todos los rostros. Cuando se unen los puntos que hay en la plantilla se forma una malla que en cada rostro forma un patrón diferente.

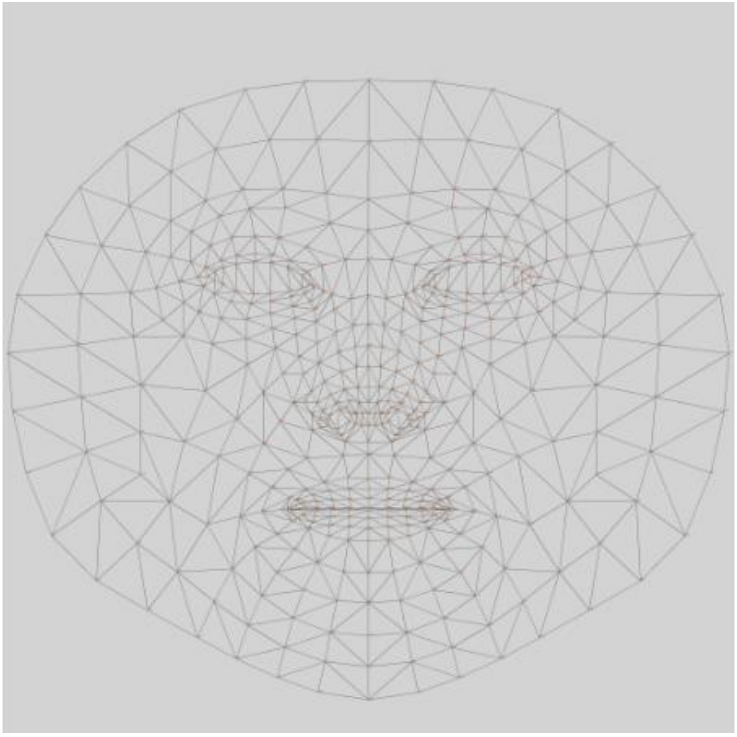

Ilustración 13. Plantilla utilizada por los modelos para detectar patrones faciales

Con estos puntos podemos definir los patrones faciales, pero es más adecuado obtener los puntos utilizando el otro modelo ya que tiene en cuenta más puntos. El modelo face-detection solo toma en cuenta 68 puntos mientras que el face-landmarks-detection toma en cuenta más de 130. [25]

Para poder aplicar los modelos primero tenemos que cargarlo de la siguiente forma:

```
const loadModel = async () => {
  //.ts: const loadModel = async ():Promise<void|tf.LayersModel>=>{
  const model = await tf
    .loadLayersModel(bundleResourceIO(modelJSON, modelWeights))
    .catch(e => {
      console.log('[LOADING ERROR] info:', e);
    });
  return model;
};
```

Ilustración 14. Cargar el modelo elegido.

Como se observa, vamos a utilizar el modelo almacenado en JSON y sus pesos (weights) que vienen a ser las opciones de configuración de nuestro modelo. Haría falta dos cargas, una para cada modelo ya que tienen diferente JSON y pesos.

Cuando ya tenemos cargado el modelo vamos a utilizarlo para hacer predicciones de la siguiente manera:

```
const makePredictions = async (batch, model, imagesTensor) => {
  const predictionsdata = model.predict(imagesTensor);
  //.ts: const predictionsdata:tf.Tensor = model.predict(imagesTensor) as tf.Tensor
  let pred = predictionsdata.split(batch); //split by batch size
  //return predictions
  return pred;
};
```

Ilustración 15. Método para utilizar el modelo informático

Como solo tenemos que analizar una cara, el *batch* siempre será 1 por lo que las predicciones serán todas de la misma imagen. Si hubiera más de una cara, el modelo devolvería todas las predicciones realizadas.

El siguiente método combina todo lo mencionado anteriormente, si se implementa en una clase auxiliar lo podríamos exportar gracias a la palabra reservada *"export"* y poder utilizarlo donde fuera necesario aplicar el modelo.

```
const getPredictions = async image => {
  await tf.ready();
  const model = await loadModel();
  const tensor_image = await transformImageToTensor(image);
  const predictions = await makePredictions(1, model, tensor_image);
  return predictions;
};
```

Ilustración 16. Método que se puede importar para utilizar el modelo informático

Sin la necesidad de importar los modelos de nuevo, simplemente importaríamos la función.

Reconocimiento facial

Se debe hacer una comparación de la imagen analizada y la lista de rostros ya verificados que obtendremos de la base de datos.

El listado contiene las imágenes de los rostros que tenemos que comparar, ya están transformadas al formato Tensor y se obtienen siguiendo un determinado proceso. Este proceso consiste en conectar con la base de datos y obtener el listado para que la aplicación lo pueda utilizar.

Para la comparación se pueden seguir dos métodos: se puede analizar las dos imágenes utilizando como métrica de similitud la distancia euclídea que devolverá la diferencia entre las dos y la confianza de que sean el mismo rostro o simplemente comparando la cadena JSON generada por el rostro fotografiado y todos los rostros ya verificados. [26]

El primer método es mucho más fiable y es el que vamos a utilizar, pero es más difícil de implementar. Para medir la distancia euclídea vamos a crear una clase nueva llamada FaceRecognizer.ts donde implementaré un método que utilizare para comparar los rostros. La distancia euclídea se mide a partir de descriptores de rostro que se obtienen al analizar los patrones faciales. Estos descriptores contienen un label (campo) que indica el nombre del dueño del rostro y una distancia de *threshold* (límite). Cuando el *threshold* entre dos imágenes es muy pequeño, podemos decir que los rostros analizados son muy similares, ya que el valor medio para una imagen de 150x150 es de 0.6 y por debajo de ese valor es bastante seguro que los rostros son similares.

```
private _labeledDescriptors: LabeledFaceDescriptors[]
private _distanceThreshold: number

constructor(
  inputs: LabeledFaceDescriptors | WithFaceDescriptor<any> | Float32Array | Array<LabeledFaceDescriptors | Wit
  distanceThreshold: number = 0.6
) {
```

Ilustración 17. Constructor de la clase FaceRecognizer

Creamos dos variables y el constructor de FaceRecognizer.ts, que servirá para instanciarlo fuera de la clase.

```
public get labeledDescriptors(): LabeledFaceDescriptors[] { return this._labeledDescriptors }
public get distanceThreshold(): number { return this._distanceThreshold }
```

Ilustración 18. Asignación de variables con las propiedades del constructor

Asignamos las variables creadas con las propiedades del constructor y empezamos a calcular la distancia. Primero debemos hacer el método que calculara la media de la distancia euclídea.

```
public computeMeanDistance(queryDescriptor: Float32Array, descriptors: Float32Array[]): number {
  return descriptors
    .map(d => euclideanDistance(d, queryDescriptor))
    .reduce((d1, d2) => d1 + d2, 0)
    / (descriptors.length || 1)
}
```

Ilustración 19. Calculo de la distancia euclídea media

Ahora utilizamos la función para juntar los descriptores de la siguiente forma:

```
public matchDescriptor(queryDescriptor: Float32Array): FaceMatch {
  return this.labeledDescriptors
    .map(({ descriptors, label }) => new FaceMatch(
      label,
      this.computeMeanDistance(queryDescriptor, descriptors)
    ))
    .reduce((best, curr) => best.distance < curr.distance ? best : curr)
}
```

Ilustración 20. Junta los descriptores de rostro

Por último, para calcular el mejor resultado (el valor de threshold más bajo que 0.6):

```
public findBestMatch(queryDescriptor: Float32Array): FaceMatch {
  const bestMatch = this.matchDescriptor(queryDescriptor)
  return bestMatch.distance < this.distanceThreshold
    ? bestMatch
    : new FaceMatch('unknown', bestMatch.distance)
}
```

Ilustración 21. Función para buscar el mejor valor

El resultado se puede transformar a JSON de la siguiente manera:

```
public toJSON(): any {
  return {
    distanceThreshold: this.distanceThreshold,
    labeledDescriptors: this.labeledDescriptors.map((ld) => ld.toJSON())
  };
}
```

Ilustración 22. Función para transformar a JSON

Una vez tenemos este JSON, se lo tenemos que enviar mediante un *get* a la REST API de Swagger para que nos devuelva las credenciales del usuario y seguir navegando por la vista correspondiente; en caso de ser reconocido, navega hasta la vista de fichar y, si no es reconocido, vuelve a la vista de la cámara para realizar una foto de nuevo, ya que puede fallar por iluminación.

La integración en la aplicación ya existente es bastante sencilla ya que solo tengo que añadir el componente y las vistas creadas. He creado dos clases FRecView y FRecViewModel las cuales son la vista y su modelo. Los estilos también los he incluido dentro de otra clase llamada FRecStyles ya que se considera una buena práctica de programación. El modelo se va a usar para comunicar con la API, que nos devolverá las credenciales relacionadas con el rostro reconocido. La vista utilizará el componente cámara creado en una clase auxiliar (Camara.ts).

Es necesario incluirlas en la navegación utilizando la pila, para ello primero tengo que crear una pantalla (*screen*) y una ruta en la enumeración *ROUTES* (rutas) que se encuentra en el fichero Constants.ts.

```
const FRecScreen = () => <FRecView vm={new FRecViewModel()} />;
```

Ilustración 23. Creación de la pantalla FRecScreen

```
export enum ROUTES {
  LOGIN = 'Login',
  HOME = 'Home',
  NFC_SIGN = 'NFCSign',
  QR_SIGN = 'QRSign',
  FREC_SIGN = 'FRecSign',
  HISTORY = 'History',
  SETTINGS = 'Settings',
  CALENDAR = 'Calendar',
  SKIP = 'Skip',
}
```

Ilustración 24. Lista de rutas dentro de la aplicación

Por tanto, la pila se utiliza de la siguiente forma:

```
<Stack.Screen
  name={ROUTES.FREC_SIGN}
  component={FRecScreen}
  options={{headerShown: false}}
/>
```

Ilustración 25. Uso de pila de navegación

Es necesario que la navegación se realice en todas las vistas que sea necesario, por ejemplo, en la vista login (loginView) el usuario puede navegar hasta la función de reconocimiento facial por lo que habrá que incluir la navegación. Se hace así:

```
const toFRec = () => {
  navigate(ROUTES.FREC_SIGN, null);
};
```

Ilustración 26. Función para navegar a la pantalla de reconocimiento facial

Es una función que utiliza el navegador para poder acceder al a ruta especificada.

```
<TouchableOpacity style={{marginBottom: 80}}>
  <MaterialCommunityIcons
    name="face-recognition"
    size={100}
    color={COLORS.text}
    style={{alignSelf: 'center'}}
    onPress={toFRec}
  />
</TouchableOpacity>
```

Ilustración 27. Botón para navegar a la pantalla de reconocimiento facial

6.3-Problemas encontrados

El principal problema con el que me he encontrado es con las versiones de NPM, dependiendo de la librería que queramos utilizar la versión de NPM tiene que cambiar. Esto ocurre cuando las librerías están desactualizadas o ya no tienen revisión. Este problema ocurre con React Native Camera, que con la última versión de Node y NPM da error de descarga.

Para solventar estos problemas he utilizado la herramienta NMV, que permite organizar todos los paquetes que tenemos instalados de Node y podemos utilizarlo para descargar nuevos y seleccionar el adecuado para su empleo. Cuando se descarga una versión diferente de Node también se descarga el controlador NPM asociado a esa versión.

Otro problema que me apareció con continuidad está relacionado con la versión de Android, algunas librerías no aceptan versiones nuevas de Android ya que se crearon con una versión anterior y no han sido actualizadas.

La solución a este problema es simplemente buscar la versión de Android más antigua que sea compatible con todas las librerías del proyecto.

6.4-Pruebas

Para la evaluación del sistema se ha realizado una serie de pruebas con las cuales se busca encontrar los posibles errores que tenga el prototipo, para poder modificarlo conforme al gusto del cliente interesado (en este caso la empresa Movicoders).

También se busca la correcta cumplimentación de los requerimientos funcionales propuestos en esta aplicación móvil. El objetivo principal de las pruebas es evidenciar qué está mal y

proponer un cambio para que la entrega final se haga correctamente y habiendo cumplido los requerimientos.

Se van a realizar tres pruebas: **prueba de interfaz de usuario, prueba de integración y prueba de modelos**. Las pruebas tienen un objetivo, una técnica, el procedimiento y los criterios de éxito.

La primera es la prueba de interfaz de usuario, cuyo objetivo es verificar que la interfaz ofrece al usuario la navegación y el acceso adecuado a través de cada vista de la aplicación, en específico, lo relacionado con el reconocimiento. Se crearán pruebas para cada caso: uno con un reconocimiento facial positivo y otro con un reconocimiento facial negativo. Se debe confirmar que la navegación redirige correctamente a la pantalla correspondiente en cada caso y que informa al usuario mediante una alerta.

En la prueba de integración se validará el correcto funcionamiento del módulo de reconocimiento facial añadido a la solución de fichajes ya existente, comprobando que se incluyen todas las dependencias y que se garantiza su correcta operatividad. Para realizar esta prueba se ha instalado la aplicación en un emulador de Android con la versión mínima capaz de ejecutarla. Comprobando que la cámara no se desconfigura y que no reporta ningún error a la hora de comunicar con la base de datos para obtener el listado.

La última prueba a realizar está relacionada con los modelos que se aplican a las imágenes. Esta prueba consiste en entrenar los modelos con las imágenes de los empleados para luego almacenar los resultados obtenidos en forma de listado para el proceso de reconocimiento facial. Esta prueba busca el mejor rendimiento de los modelos aplicados; cuanto mayor sea la cantidad de imágenes con las que entrenar el modelo, mayor será el porcentaje de acierto a la hora de reconocer un rostro.

Capítulo 7: Investigación realizada

Para la realización de este documento se han investigado en profundidad las tecnologías que se han utilizado y las posibles alternativas que se han descartado por falta de conocimientos, por tener una curva de aprendizaje difícil y larga y porque se han elegido unas tecnologías con mejores rendimientos y valoraciones por parte de los usuarios. La investigación no solo se centra en la tecnología, si no que se ha investigado las alternativas disponibles a la tecnología finalmente utilizada.

Puesto que esta investigación se ha realizado a lo largo de todo el proceso de desarrollo de la aplicación, existen muchos puntos en los que centrarse para describir en profundidad las partes más importantes que precisaban de investigación previa. Entre otras, las partes que más necesitaban una investigación eran el *framework* React Native y el *Machine Learning* (aprendizaje automático). Estas dos partes son fundamentales para la creación de esta aplicación.

El *framework* React Native sirve para el desarrollo multiplaforma de aplicaciones. React Native utiliza módulos y librerías que se descargan utilizando uno de los controladores de paquetes más utilizados que son: NPM o YARN. Existen una infinidad de estos paquetes ya que son creados por los usuarios de React Native (aunque sirven para otros *frameworks* y entornos de trabajo) y se comparten en la propia plataforma de NPM. Estos paquetes son soluciones a problemas recurrentes de la informática y que sirven para que otras personas que sufren esos problemas puedan resolverlos. Con una simple importación se pueden utilizar las funciones y los métodos implementados dentro del paquete.

Junto con el *framework*, existe una estructura de diseño y de escritura para la realización del código. En las empresas se utiliza una estructura diferente adaptada a las necesidades de cada una. En Movicoders la estructura utilizada es la comentada en el apartado de análisis de este documento, basada en vistas y utilizando aspectos de la estructura MVC (Modelo Vista Controlador). La estructura se ha tenido que analizar para entender su funcionamiento y poder integrar de forma fácil y correcta la funcionalidad de reconocimiento facial.

La curva de aprendizaje de React Native ha sido más larga y pronunciada que la curva de aprendizaje de la estructura empleada en la solución de fichajes. Esto se debe a que en la carrera los proyectos se estructuran de forma muy parecida y también se utiliza la estructura MVC en profundidad. Sin embargo, React Native fue creado para facilitar los procesos de desarrollo de aplicaciones móviles y se puede asegurar que el tiempo de aprendizaje fue menor del esperado.

Existe una dificultad añadida: el lenguaje de programación. React Native puede trabajar en multitud de lenguajes de programación e incluso en algunos casos combinar dos o más lenguajes en la misma aplicación. El problema se halla en que no se conocía el lenguaje

utilizado TypeScript, por lo que se ha tenido que invertir tiempo y esfuerzo en entender y conocer la sintaxis. TypeScript es un lenguaje muy riguroso fuertemente tipado que nace a partir de JavaScript.

Las alternativas de tecnología de aprendizaje automático son escasas, esta tecnología no está muy empleada ya que tienen una curva de aprendizaje larga y en algunas ocasiones costosa.

En la empresa se optó por una tecnología de código libre llamada TensorFlow, que es una muy buena opción gratuita, una de las mejores del mercado. La ventaja de TensorFlow es que dispone de modelos entrenados con una extensa cantidad de datos por lo que son muy eficaces cumpliendo la función con la que fueron creados. La alternativa principal era utilizar OpenCV que es un motor de Inteligencia Artificial como TensorFlow, pero dispone de menos modelos y no están entrenados con la misma cantidad de datos que TensorFlow.

Se ha estudiado la forma que tienen de trabajar los modelos de aprendizaje automático, que hacen uso de redes neuronales. Aunque las redes neuronales se estudian en la carrera, no se ha tenido que aprender a crearlas ya que se han usado los modelos ya creados y entrenados. Los modelos de TensorFlow se escriben en Python y luego deben ser traducidos y transformados para utilizarlos en React Native. Existe una librería llamada *react-native-tfjs* que posee un método que transforma el modelo de forma automática.

7.1-Optimización

Con los conocimientos adquiridos en la fase de investigación, se van a proponer optimizaciones para la aplicación que, por falta de tiempo, no se han podido llevar a cabo y que podrían suponer una mejora importante en la velocidad de respuesta de la aplicación.

El reconocimiento facial se realiza sobre una fotografía, la primera optimización sería poder hacerlo en un vídeo en directo. Esta forma es mucho más eficiente y rápida, pero más complicada de implementar. Investigando sobre este tema, se debería modificar la cámara a una cámara Tensor. Esta cámara transforma la imagen automáticamente al formato necesario para aplicar modelos sobre ella. No se implementó de esta forma desde el principio ya que las versiones de Android y NPM no eran compatibles y al detectar un rostro el sistema se cerraba sin reportar errores por lo que se tenía que invertir tiempo en buscar el error y solventarlo.

Otra optimización sería construir un modelo totalmente nuevo en el que crear una red neuronal para analizar y predecir patrones faciales en las imágenes. Esta optimización se basa en el uso del lenguaje de programación Python para crear el modelo. Python es un lenguaje muy utilizado cuya curva de aprendizaje es muy sencilla y corta. Los modelos de TensorFlow se escriben utilizando este lenguaje. La única pega de esta optimización es que hay que entrenar el modelo con un gran número de datos para asegurarnos que funciona correctamente. Estos datos se

pueden obtener de bancos de datos disponibles en la red si la empresa, como Movicoders, no posee muchos empleados.

Existe un problema de iluminación cuando se aplican los modelos sobre las imágenes. Si las imágenes son muy oscuras los patrones faciales son difíciles de predecir en una cara de una persona negra y si las imágenes están muy iluminadas la detección del rostro es difícil de predecir en una persona blanca. Se puede decir que el aprendizaje automático es racista, ya que las caras blancas son más fáciles de analizar. Este siempre ha sido un problema desde el inicio para esta tecnología, en parte porque se han entrenado a los modelos con más cantidad de caras blancas que de caras negras. Para solventar los problemas de iluminación algunos dispositivos incorporan un flash en la parte frontal del dispositivo. Para solventar el racismo del aprendizaje habría que entrenar los modelos con más cantidad de caras negras para que mejore y consiga detectar y reconocer patrones faciales con más facilidad.

Por último, una optimización que es recomendable para la aplicación es que, debido a que la librería de React Native Camera está sin supervisión desde hace un tiempo, se debería cambiar a una librería que sí tenga supervisión como React Native Vision o Expo Camera. React Native Vision es la continuación de React Native Camera pero, esta vez, supervisada y con parches que resuelven problemas encontrados en anteriores versiones. Expo Camera que viene de Expo es una librería que suele usarse junto a React Native; las aplicaciones de React Native que utilizan Expo son perfectas para la creación de aplicaciones móviles.

Capítulo 8: Conclusión

En este trabajo de fin de grado se han alcanzado los objetivos y requisitos iniciales planteados al comienzo de este proyecto. Se ha realizado una aplicación móvil que utiliza reconocimiento facial para controlar el acceso de los empleados integrándola en una solución de fichajes de la empresa Movicoders. Se ha intentado crear una aplicación ligera y compatible con la mayor cantidad de versiones Android debido a que es el *smartphone* más vendido en España. Para ello se ha utilizado el *framework* y las librerías de React Native.

Los modelos TensorFlow para el proceso de reconocimiento facial han sido elegidos por su rendimiento, por la cantidad de datos con los que se entrenan y por la probabilidad de acierto; son guardados como librerías y se utilizan para detectar rostros, detectar patrones faciales y comparar los rostros detectados con el listado de rostros ya verificados.

Gracias a la base de datos Firebase, los datos se mantienen actualizados y disponibles en todo momento. Esta base de datos se aloja en la nube y se guarda en ella el listado de rostros ya verificados de los empleados.

A nivel personal, el diseño y desarrollo de esta aplicación me ha supuesto un gran desafío. He tenido que poner en práctica los conocimientos adquiridos durante toda la carrera, pero lo más difícil fue la enorme labor de investigación y de autoaprendizaje que he realizado, ya que se han empleado tecnologías que eran desconocidas para mí y con las que, por supuesto, no había trabajado.

Todas las herramientas han sido empleadas por primera vez por lo que fue necesaria una inversión de mi tiempo en estudiar las nuevas herramientas y el lenguaje de programación TypeScript. Aunque TypeScript está basado en JavaScript, y este, a su vez, está basado en Java (lenguaje ampliamente estudiado en el grado), me costó adaptarme a la sintaxis nueva y las pautas de diseño que tenían en la empresa. El entorno de trabajo elegido React Native fue lo que más tiempo me ocupó cuando lo estudié; es un entorno multiplataforma muy popular para el desarrollo de aplicaciones móviles.

La labor autodidacta y de solución de errores la realicé haciendo uso de los manuales y guías de las propias herramientas y los foros de internet (como GitHub o StackOverflow, muy populares en el ámbito informático). [28] [29]

En el desarrollo de aplicaciones en un ámbito real, he aprendido a la fuerza que no siempre va a funcionar como se desea y que van a aparecer errores en cualquier momento menos esperado. Gracias a este proyecto he aprendido a lidiar con los problemas que me iban surgiendo y también a saber cómo buscar información sobre tecnologías nuevas y emergentes analizando sus ventajas y desventajas frente a otras y eligiendo la que más me convengan.

Las soluciones a problemas que me surgían normalmente las encontraba en los foros donde los usuarios comparten problemas que tienen y ellos mismos u otros usuarios comparten la solución que les funcionó.

Aunque la aplicación no tiene un gran tamaño, me he dado cuenta de que soy capaz de afrontar el diseño y desarrollo gracias a los conocimientos adquiridos durante la carrera y también que puedo aplicar esos conocimientos para crear una aplicación que sea útil para la vida cotidiana de las personas.

Referencias

[1] Metodología SCRUM: https://proyectosagiles.org/que-es-scrum/

[2] Metodología de prototipado: https://freed.tools/blogs/ux-cx/prototipo#:~:text=cumple%20o%20no.-,Metodolog%C3%ADa%20de%20prototipo%20o%20prototipado,medir%20y%20ajustar%20un%20plan.

[3] Sistema biométrico: https://www.incibe.es/protege-tu-empresa/guias/tecnologias-biometricas-aplicadas-ciberseguridad-guia-aproximacion-el

[4] Patrón facial: https://protecciondatos-lopd.com/empresas/reconocimiento-facial/

[5] Aprendizaje automático: https://www.netapp.com/es/artificial-intelligence/what-is-machine-learning/

[6] Reconocimiento facial: https://www.interpol.int/es/Como-trabajamos/Policia-cientifica/Reconocimiento-facial

[7] Aplicación móvil: https://anincubator.com/que-es-una-aplicacion-movil/

[8] Veriff: https://www.veriff.com/

[9] Luxand: https://www.luxand.com/about/

[10] Face Phi: https://facephi.com/

[11] React Native: https://reactnative.dev/

[12] React Navigation: https://reactnavigation.org/

[13] Mobx: https://mobx.js.org/README.html

[14] React Native Camera: https://react-native-camera.github.io/react-native-camera/docs/rncamera

[15] TensorFlow: https://www.tensorflow.org/

[16] Android: https://www.android.com/

[17] Android Studio: https://developer.android.com/studio

[18] Firebase: https://firebase.google.com/

[19] NodeJS: https://nodejs.org/en/

[20] NPM: https://www.npmjs.com/
[21] Visual Studio Code: https://code.visualstudio.com/
[22] Adquisición de la imagen con React Native Camera:
https://www.fullstacklabs.co/blog/react-native-camera
[23] Transformación de la imagen:
https://www.sitepoint.com/use-react-native-to-a-create-a-face-recognition-app/
[24] Modelo face-detection:
https://github.com/tensorflow/tfjs-models/tree/master/face-detection
[25] Modelo face-landmarks-detection:
https://github.com/tensorflow/tfjs-models/tree/master/face-landmarks-detection
[26] Librería para reconocimiento facial:
https://justadudewhohacks.github.io/face-api.js/docs/index.html
[27] Face recognition paper: https://arxiv.org/pdf/1907.06724.pdf
[28] Stack Overflow: https://es.stackoverflow.com/
[29] GitHub: https://github.com/