Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Vuzix FR: reconocimiento facial para residencias de ancianos

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Mariyan Georgiev Petkov

Tutor: David De Andrés Martínez

[2018/2019]

Índice de contenidos

1. Introducción....11
 1.1. Motivación....11
 1.2. Objetivos....12
  1.2.1. Desarrollo de aplicaciones para Vuzix Blade....12
  1.2.2. Computación en la nube e inteligencia artificial....15
 1.3. Estructura de la memoria....17
2. Estado del arte....18
 2.1. Gafas de realidad aumentada....18
  2.1.1. Vuzix Blade....18
  2.1.2. Microsoft Hololens 2....19
  2.1.3. Google Glass Enterprise Edition 2....19
  2.1.4. Comparativa entre Blade y otros modelos de gafas de realidad aumentada....20
 2.2. Aplicaciones de reconocimiento facial....22
3. Tecnologías y herramientas utilizadas....23
 3.1. Android Studio....23
 3.2. Microsoft Visual Studio 2017....23
 3.3. Microsoft Azure....24
 3.4. SQLite....24
 3.5. Postman....24
 3.6. GitLab....25
 3.7. Bibliotecas....25
  3.7.1. Bibliotecas en el proyecto Android....25
  3.7.2. Bibliotecas en el proyecto .Net....26
4. Especificación....27
 4.1. Requisitos funcionales....27
 4.2. Requisitos no funcionales....27
 4.3. Casos de uso....28
 4.4. Mock-ups....29
 4.5. Modelo de datos....31
5. Diseño....32
 5.1. Esquema de componentes....32
 5.2. Transferencia de información entre componentes....33
  5.2.1. Detección....33
  5.2.2. Identificación....34
  5.2.3. Recuperación de información....35
 5.3. Arquitectura....36
  5.3.1. Arquitectura de la aplicación Android....36
  5.3.2. Arquitectura de la aplicación .Net Core....39
 5.4. Estructura del modelo de aprendizaje entrenado....41
6. Implementación....42
 6.1. Implementación del proyecto Android....42
  6.1.1. Vista del proyecto Android....42
  6.1.2. Modelo del proyecto Android....44

6.1.3. Presentador del proyecto Android....48
6.2. Implementación del proyecto .Net Core....52
6.2.1. Puntos de acceso de la API web....52
6.2.2. Lógica del microservicio VZX....53
6.2.3. Acceso a recursos en el microservicio VZX....54
6.3. Entrenamiento del modelo de aprendizaje para Face API....55
7. Pruebas....58
7.1. Pruebas unitarias en el proyecto .Net Core....58
7.2. Pruebas de usuario....59
8. Conclusión....61
8.1. Relación del trabajo desarrollado con los estudios cursados....61
8.2. Trabajo futuro....62
9. Bibliografía....63

Índice de figuras

Figura 1. Diseño de navegación en anchura....13
Figura 2. Diseño de navegación en profundidad....14
Figura 3. Computación en la nube. 15
Figura 4. Características de Vuzix Blade....18
Figura 5. Gafas Hololens 2....19
Figura 6. Gafas Google Glass Enterprise Edition 2....19
Figura 7. Face2Gene. 22
Figura 8. iFalcon....22
Figura 9. Android Studio IDE....23
Figura 10. Visual Studio IDE....23
Figura 11. Azure....24
Figura 12. SQLite....24
Figura 13. Postman....24
Figura 14. GitLab....25
Figura 15. Bibliotecas del proyecto Android....25
Figura 16. .Net Core....26
Figura 17. EntityFrameworkCore....26
Figura 18. EntityFrameworkCore.Sqlite....26
Figura 19. Newtonsoft.Json....26
Figura 20. AspNetCore.Http....26
Figura 21. Diagrama de casos de uso....28
Figura 22. Pantalla principal....29
Figura 23. Pantalla de tomar captura....30
Figura 24. Pantalla de aceptar captura....30
Figura 25. Pantalla de selección de grupo....30
Figura 26. Pantalla de muestra de resultado....31
Figura 27. Tabla "Person" de la base de datos VZX....31
Figura 28. Esquema de componentes....32
Figura 29. Petición REST Detección....33
Figura 30. Resouesta REST Detección....33
Figura 31. Petición REST Identificación....34
Figura 32. Respuesta REST Identificación....34
Figura 33. Petición VZX web API....35
Figura 34. Respuesta REST VZX web API....35
Figura 35. Arquitectura MVP....36
Figura 36. Estructura del proyecto Android....36
Figura 37. Vista lógica del proyecto Android (MVP)....37
Figura 38. Vista XML del proyecto Android (MVP)....37
Figura 39. Modelo del proyecto Android (MVP)....37
Figura 40. Presentador del proyecto Android (MVP)....38
Figura 41. Arquitectura de microservicios....39
Figura 42. Estructura del proyecto .Net Core....39
Figura 43. Estructura de la implementación del microservicio VZX....40
Figura 44. Estructura de la configuración del microservicio VZX....41
Figura 45. Estructura de modelo de aprendizaje de la Face API....41

Figura 46. Vista lógica del proyecto Android (MVP)....42
Figura 47. Clase MainView.java....42
Figura 48. Clase IdentificationView.java....43
Figura 49. Modelo del proyecto Android (MVP)....44
Figura 50. Detección en la clase AzureCognitiveServices.java....44
Figura 51. Detección en la clase AzureCognitiveServices.java....45
Figura 52. Tarea de detección en la clase DetectionTask.java....46
Figura 53. Tarea de identificación en la clase IdentificationTask.java....47
Figura 54. Clase VzxMicroservice.java....48
Figura 55. Presentador del proyecto Android (MVP)....48
Figura 56. Construcción de la vista en MainActivity.java....49
Figura 57. Métodos de tomar captura en MainActivity.java....49
Figura 58. Proceso post-captura....50
Figura 59. Inicio de instancias de vista y servicios de IdentificationActivity.java....50
Figura 60. Opciones del menú de seleccionar grupo de búsqueda....51
Figura 61. Proceso post identificación....51
Figura 62. Estructura de la implementación del microservicio VZX....52
Figura 63. Clase VzxController.cs....53
Figura 64. Clase VzxService.cs....53
Figura 65. Clase PersonRepository.cs....54
Figura 66. Operación tipo CRUD en la calse Repository.cs....54
Figura 67.Flujo completo de identificación....55
Figura 68. Colecci'on de llamadas REST de Postman....56
Figura 69. Llamada REST para crear grupo de personas....57
Figura 70. Llamada REST para crear una persona....57
Figura 71. Llamada REST para crear una cara....57
Figura 72. Llamada REST para entrenar un grupo....57
Figura 73. Pruebas unitarias del microservicio....58
Figura 74. Prueba unitaria de objeto nulo....58
Figura 75. Prueba unitaria de funcionamiento base....59
Figura 76. Objeto esperado del servicio....59

Índice de tablas

Tabla 1. Comparativa técnica entre gafas de realidad aumentada....20
Tabla 2. Caso de uso Identificar....28
Tabla 3. Caso de uso Mostrar información....29
Tabla 4. Tiempos de uso por primera vez....60
Tabla 5. Tiempos de uso por usuarios conocedores del dispositivo y la aplicación....60

Resumen

El presente trabajo fin de grado consiste en el desarrollo de una aplicación que tiene como objetivo reconocer el rostro de una persona y sacar información relevante a ella. El usuario final es el personal de asistencia sanitaria en residencias geriátricas.

La aplicación es desarrollada para ser utilizada con las gafas de realidad aumentada Vuzix Blade, fabricadas por el desarrollador líder de realidad inteligente y aumentada, Vuzix.

Desarrollar aplicaciones para las gafas Vuzix Blade requiere utilizar tecnologías Android, en parte, con bibliotecas propias de Vuzix que facilitan el desarrollo en este tipo de dispositivo. Por otro lado, se utilizan los servicios de computación en la nube de Azure para reconocimiento facial.

El funcionamiento principal, sin entrar en detalles, se basa en tomar una captura del rostro de una persona con la cámara del dispositivo, utilizar el servicio de reconocimiento facial de Azure para identificarla y obtener un identificador que será utilizado como criterio de búsqueda en una consulta a la base de datos de la residencia de ancianos para recuperar y visualizar la información relevante sobre dicha persona identificada. Para acceder a la base de datos y recuperar la información se llamará a un servicio desplegado en Azure.

Un ejemplo de caso de uso sería cuando el asistente sanitario quiere consultar información sobre la habitación, la cama o dependencia de la persona residente.

Palabras clave: desarrollo gafas de realidad aumentada, Vuzix Blade, Azure, reconocimiento facial, Android, residencias geriátricas

Resum

El present treball final de grau consisteix en el desenvolupament d'una aplicació que té com a objectiu reconèixer el rostre d'una persona i treure informació rellevant a ella. L'usuari final és el personal d'assistència sanitària en residències geriàtriques.

L'aplicació és desenvolupada per a ser utilitzada amb les ulleres de realitat augmentada Vuzix Blade, fabricades pel desenvolupador líder de realitat intel·ligent i augmentada, Vuzix.

Desenvolupar aplicacions per a les ulleres Vuzix Blade requereix utilitzar tecnologies Android, en part, amb llibreries pròpies de Vuzix que faciliten el desenvolupament en aquest tipus de dispositiu. D'altra banda, s'utilitzen els serveis de computació en el núvol d'Azure per a reconeixement facial.

El funcionament principal, sense entrar en detalls, es basa en prendre una captura de la cara d'una persona amb la càmera del dispositiu, utilitzar el servei de reconeixement facial d'Azure per identificar-la i obtenir un identificador que serà utilitzat com a criteri de cerca en una consulta a la base de dades de la residència d'avis per recuperar i visualitzar la informació rellevant sobre aquesta persona identificada. Per accedir a la base de dades i recuperar la informació es dirà a un servei desplegat en Azure.

Un exemple de cas d'ús seria quan l'assistent sanitari vol consultar informació sobre l'habitació, el llit o dependència de la persona resident.

Paraules clau: desenvolupament ulleres de realitat augmentada, Vuzix Blade, Azure, reconeixement facial, Android, residències geriàtriques..

Abstract

This final degree project consists in the development of an application that aims to recognize the face of a person and draw relevant information. The end user is the health care staff in nursing homes.

The application is developed for use with Vuzix Blade augmented reality glasses, manufactured by the leading intelligent and augmented reality developer, Vuzix.

Developing applications for Vuzix Blade glasses requires using Android technologies, in part, with Vuzix's own libraries that facilitate the development of this type of device. On the other hand, Azure cloud computing services are used for facial recognition.

The main functionality, without going into details, is based on taking a screenshot of a person's face with the device's camera, using Azure's facial recognition service to identify it and obtaining an identifier that will be used as a search criteria in a query to the database of the nursing home to retrieve and view the relevant information about said identified person. To access the database and retrieve the information, a service deployed in Azure will be called.

An example of a use case would be when the health worker wants to consult information about the room, bed or unit of the resident.

Keywords: augmented reality glasses development, Vuzix Blade, Azure, facial recognition, Android, nursing homes

1. Introducción

En este primer apartado vamos a comentar cuál es el motivación para llevar a cabo este proyecto, qué objetivos tenemos para conseguirlo y la estructura del presente documento.

1.1. Motivación

La realidad aumentada surgió por primera vez en los años 70, como una tecnología orientada a las experiencias en mundos virtuales. El término fue acuñado por Tom Caudell en 1992, y a partir de ese momento se sucedieron diferentes aplicaciones y plataformas para desarrollar más tecnología y aplicaciones de realidad aumentada.

Ya en el siglo XXI la realidad aumentada ha entrado en un periodo de auge, que se ha dividido en las tres etapas siguientes [1]:

- Realidad aumentada en ordenadores personales: entre el año 2006 y 2008, gracias al mundo de los video juegos y a la mejora de las capacidades computacionales de ordenadores y tarjetas gráficas, aparecieron en el mercado las primeras herramientas de programación de realidad aumentada de alto nivel (D'Fusion de Total Immersion o Metaio SDK) y proliferaron las empresas especializadas en este campo.

- Realidad aumentada en smartphones: la revolución social y tecnológica provocada unos años más tarde por el visionario Steve Jobs con la invención de los smartphones dio pie a la aparición de las primeras Apps en el sector turístico que vinculaban información de internet a una capa superpuesta a la cámara del móvil, en función de la orientación y la localización de un usuario (Wikitude o Layar).

- Realidad aumentada en gafas y visores: en la actualidad estamos viviendo la siguiente revolución de la tecnología de realidad aumentada gracias al empujón mediático que Google propició para sus gafas de realidad aumentada. Aunque todavía los dispositivos que existen en el mercado son un tanto toscos y la experiencia visual es muy mejorable, ya se intuyen numerosas aplicaciones y negocios.

En esta última etapa es donde entra *ADD Informática*, empresa española destinada a satisfacer las necesidades de gestión administrativa de Residencias Geriátricas.

ADD Informática, en compromiso con la I+D[1], colabora con la Universidad Politécnica de Valencia con el principal objetivo de conocer y aplicar las últimas tecnologías.

El interés por las nuevas tecnologías y sobretodo por dispositivos de realidad aumentada han sido de gran motivación para llevar a cabo este proyecto. Por otro lado, gracias a la colaboración entre *ADD Informática* y la Universidad Politécnica de Valencia, ha sido posible adquirir un dispositivo de realidad aumentada como son las gafas *Vuzix Blade* para la realización del presente proyecto, las cuales detallaremos más adelante.

---

[1] I+D: Símbolo de *Investigación y Desarrollo*, que se aplica a los departamentos de investigación públicos o privados encaminados al desarrollo de nuevos productos o la mejora de los existentes por medio de la investigación científica.

1.2. Objetivos

A continuación, vamos a tratar los objetivos fundamentales para el desarrollo de este proyecto presentando las dificultades que podemos encontrarnos o las facilidades de las que disponemos para ello.

1.2.1. Desarrollo de aplicaciones para *Vuzix Blade*

Con el objetivo de estudiar el lanzamiento de un nuevo producto, la empresa ADD Informática está interesada en hacer una prueba de concepto previa y ver la viabilidad del mismo.

Esta prueba de concepto, de forma breve, consiste en el desarrollo de una aplicación que tiene como objetivo reconocer el rostro de una persona y sacar información relevante a ella, mencionado en el apartado de resumen.

Como se ha comentado anteriormente, para el desarrollo de la aplicación como prueba de concepto, se utilizan las gafas de realidad aumentada *Vuzix Blade*. El sistema operativo Android que se ejecuta en las *Blade* es una versión modificada de Android 5.1.1 (Lollipop[2]), adaptada a los componentes y capacidades del dispositivo.

Para llevar a cabo este desarrollo se utiliza Android Studio, entorno oficial de desarrollo integrado para el sistema operativo Android de Google.

Android Studio incorpora Android Emulator, que simula un dispositivo y lo muestra en el ordenador de desarrollo. Permite crear un prototipo de una app de Android, y también desarrollarla y probarla sin usar un dispositivo físico. Dado que el emulador es compatible solo con teléfonos y tablets Android, y con dispositivos Android Wear y Android TV, no podemos emular las gafas de realidad aumentada *Blade*. Esto supone la necesidad de utilizar el dispositivo físico para probar la aplicación a medida que avancemos en su desarrollo y, por tanto, la necesidad de familiarizase con el uso del dispositivo mediante los manuales de ayuda proporcionados en el sitio web oficial de Vuzix.

Por otro lado, para hacer la aplicación lo más fácil y rápida de usar, se han seguido las mejores prácticas de diseño de interfaces de usuario sugeridas por Vuzix, disponibles en su sitio web oficial. El seguimiento de estas prácticas implica desarrollar utilizando bibliotecas específicas desarrolladas por la compañía que, básicamente, son modificaciones de los componentes a nivel de *software* de las bibliotecas estándar de Android para *smartphones* con el objetivo de adaptar esos componentes de la mejor forma acorde a las características de las gafas *Blade*, sobre todo a las limitaciones que supone su tamaño de pantalla reducido.

A diferencia de las aplicaciones para *smartphones* u otros dispositivos que permiten una mejor visualización gracias a unas dimensiones de pantalla superiores, donde añadir funcionalidades enriquece y dota a la aplicación de más capacidad sin aumentar drásticamente su complejidad de uso, en aplicaciones para dispositivos de dimensiones de pantalla tan reducida el factor de complejidad es impactado de forma más notable provocando una restricción respecto al número de componentes visuales que se deberían incorporar a una misma vista (una pantalla concreta de la aplicación), y en general, a la aplicación entera.

---

[2] Lollipop: Android Lollipop es una versión del sistema operativo para dispositivos móviles Android. Fue dada a conocer el 25 de junio de 2014.

Por tanto, teniendo en cuenta el factor de complejidad y las restricciones que impone, un objetivo es el de utilizar el mínimo número de componentes visuales e interaccionales en las distintas vistas de forma que el usuario interactúe mínimamente con la aplicación para llevar a cabo la acción deseada y tener la posibilidad de visualizar la información de forma cómoda en cada vista.

Consideraremos los **componentes visuales** como aquellos que son utilizados para mostrar información al usuario y **componentes interaccionales** aquellos que se utilizan para navegar o efectuar una determinada acción de la aplicación. Un ejemplo de componente visual podría ser un contenedor con estructura de lista para mostrar una serie de elementos enumerados y uno de componente interaccinal podría ser un botón que efectúa la navegación a otra vista con funcionalidad diferente.

Con motivo de explicar mejor la problemática del impacto del factor de complejidad de uso, a continuación, se presentan las siguientes ilustraciones gráficas donde se puede observar dos ejemplos de diseños de navegación por una aplicación completos y opuestos:

La navegación por la aplicación es representada mediante árboles de nodos donde cada nodo hijo representa una posible vista distinta a la que el usuario puede navegar empezando éste por el nodo raíz.

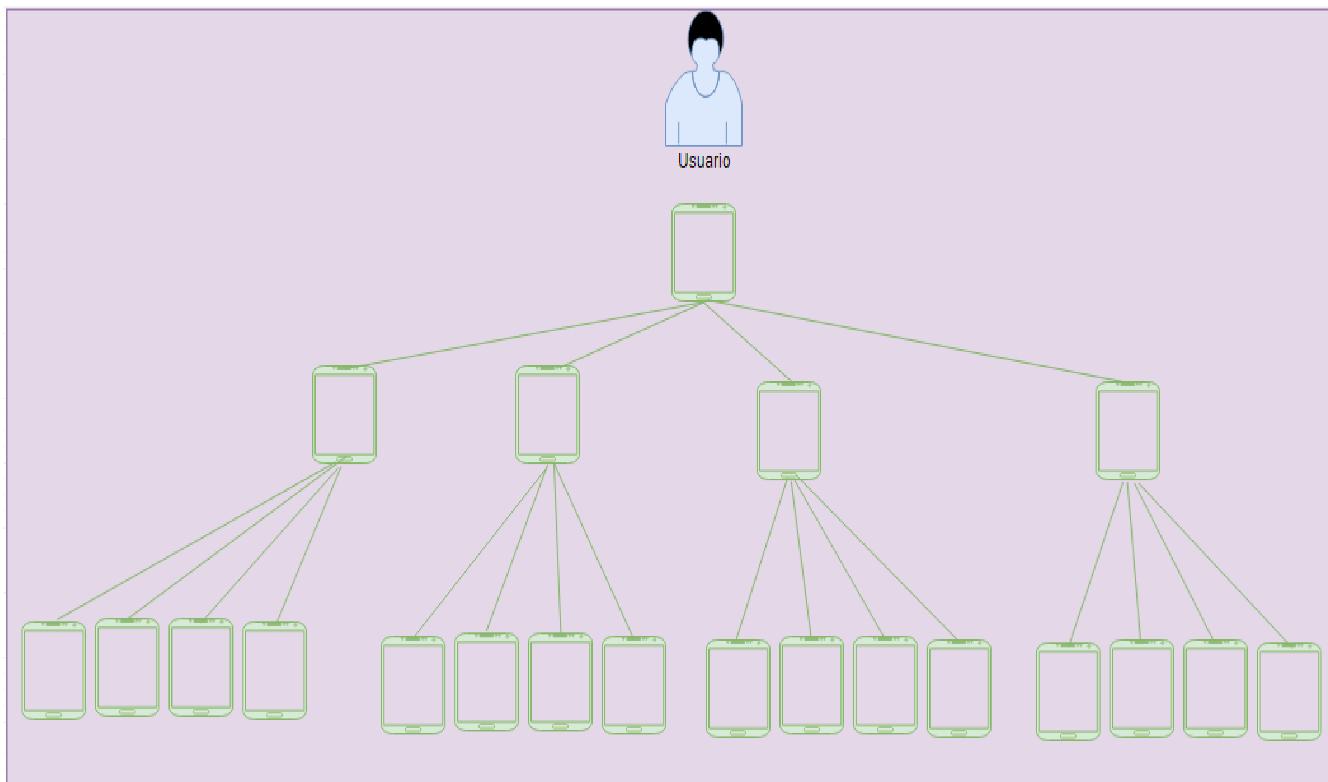


Figura 1. Diseño de navegación en anchura.

En este primer ejemplo observamos que el árbol tiene una altura de 3. Cada nodo no terminal tiene 4 hijos, es decir, cada vista tiene 4 componentes interaccionales distintos (supongamos que cada vista tiene el mismo número de componentes visuales).

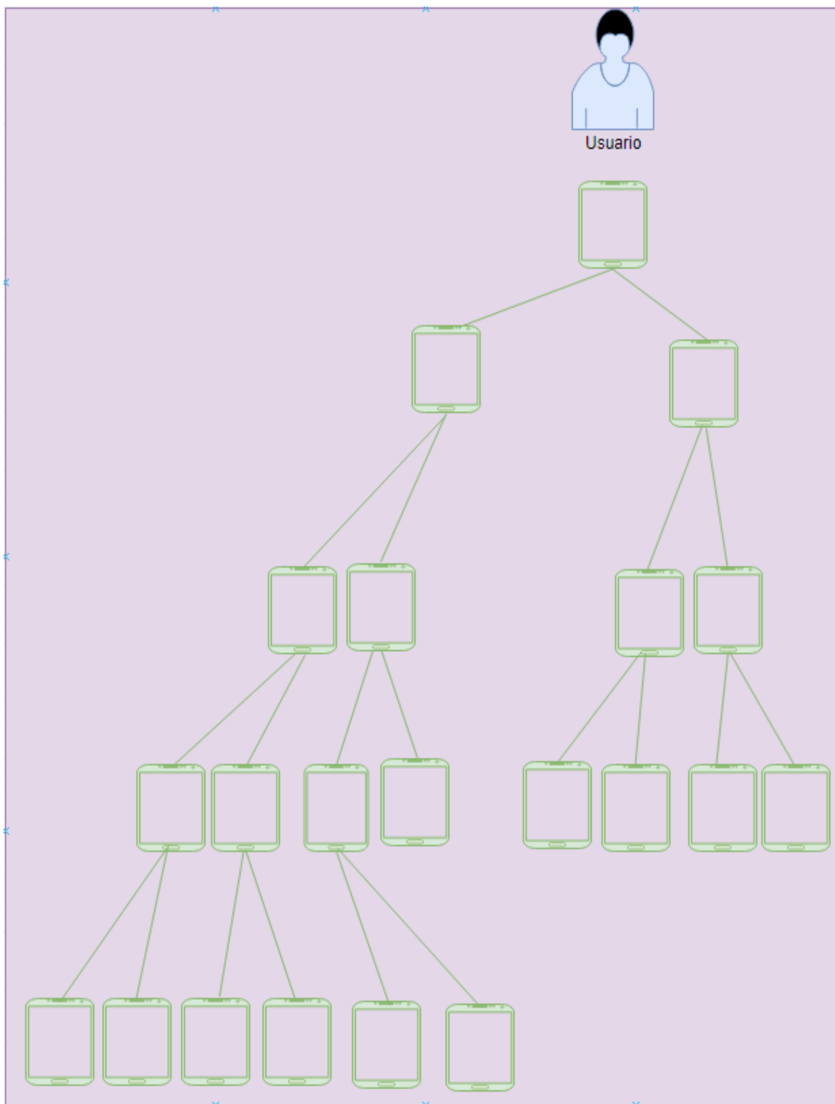


Figura 2. Diseño de navegación en profundidad.

En este segundo ejemplo podemos apreciar un árbol de una altura de 5, mayor que en el primer ejemplo pero con una anchura menor. Ambos árboles tienen el mismo número de nodos intentando representar la misma aplicación con un diseño de navegación distinto.

En términos de **complejidad visual**, observando los dos diseños, podemos darnos cuenta de que en el primer ejemplo el diseño es peor que en el segundo por tener más número de componentes. Pues al usuario le cuesta más visualizar la información en cada vista.

En términos de **complejidad interaccional**, podemos apreciar que el primer diseño es mejor que el segundo dado que las vistas más inaccesibles (los nodos terminales del árbol) son alcanzadas con un número menor de interacciones. El usuario navega más rápido a estas vistas.

Evaluados los dos ejemplos, podemos llegar a la conclusión de que cualquiera de los dos diseños ocasiona complejidad de uso al usuario, ya sea visual o interaccional. Sin embargo, teniendo en cuenta las limitaciones del dispositivo y de las tecnologías utilizadas tenemos que decidir las restricciones a aflojar y aproximar nuestro diseño a alguno de los dos ejemplos.

Dadas las características de las gafas *Blade,* que se detallarán más adelante, y de los componentes que proporciona la librería de Vuzix para Android, se opta como objetivo llevar a cabo un diseño aproximado al segundo ejemplo, es decir, se opta por dar preferencia a reducir la complejidad visual al coste de un aumento de complejidad interaccional.

En la sección de Conclusiones y trabajo futuro se explicará una alternativa para optimizar el diseño y reducir de forma drástica la complejidad de uso total.

1.2.2. Computación en la nube e inteligencia artificial

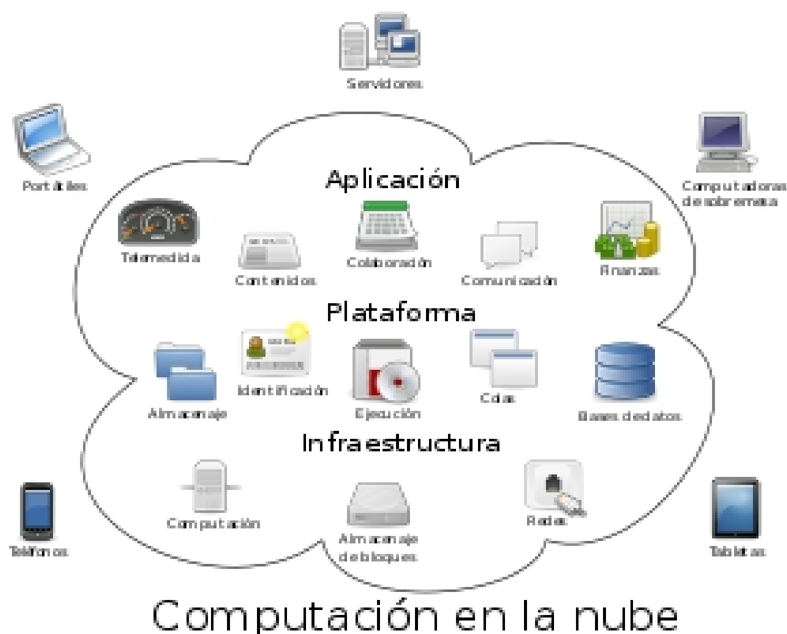


Figura 3. Computación en la nube.

La computación en la nube son servidores desde Internet encargados de atender las peticiones en cualquier momento. Se puede tener acceso a su información o servicio, mediante una conexión a internet desde cualquier dispositivo móvil o fijo ubicado en cualquier lugar. Sirven a sus usuarios desde varios proveedores de alojamiento repartidos frecuentemente por todo el mundo. Esta medida reduce los costos, garantiza un mejor tiempo de actividad y que los sitios web sean invulnerables a los delincuentes informáticos, a los gobiernos locales y a sus redadas policiales pertenecientes [2].

Cloud computing es un nuevo modelo de prestación de servicios de negocio y tecnología, que permite incluso al usuario acceder a un catálogo de servicios estandarizados y responder con ellos a las necesidades de su negocio, de forma flexible y adaptativa, en caso de demandas no previsibles o de picos de trabajo, pagando únicamente por el consumo efectuado, o incluso gratuitamente en caso de proveedores que se financian mediante publicidad o de organizaciones sin ánimo de lucro.

El software como servicio se encuentra en la capa más alta y caracteriza una aplicación completa ofrecida como un servicio, por-demanda. Las aplicaciones que suministran este modelo de servicio son accesibles a través de un navegador web —o de cualquier aplicación diseñada para tal efecto— y el usuario no tiene control sobre ellas, aunque en algunos casos se le permite realizar algunas configuraciones. Esto le elimina la necesidad al cliente de instalar la aplicación en sus propios computadores, evitando asumir los costos de soporte y el mantenimiento de hardware y software [2].

La Inteligencia artificial se ha convertido rápidamente en una de las tecnologías más importantes de la actualidad. Uno de los factores que ha impedido su desarrollo ha sido la capacidad de cálculo de los ordenadores, pues no todas las empresas pueden permitirse disponer de máquinas con capacidad de cálculo tan grande como lo requieren tareas que implican utilizar la inteligencia artificial. Ahí es donde entra la computación en la nube, que permite que cualquier organización tenga acceso a servicios de computación y cálculo para crear esos servicios de valor añadido.

Esta disponibilidad de servicios de inteligencia artificial en la nube brinda a cualquier desarrollador la oportunidad de desarrollar aplicaciones inteligentes sin tener conocimientos en inteligencia artificial.

Los servicios cognitivos de *Microsoft Azure* (*Azure Cognitive Services*, en inglés), una plataforma de computación en la nube abierta y de nivel empresarial, son APIs[3], SDKs[4] y servicios disponibles para ayudar a los desarrolladores a crear aplicaciones inteligentes sin tener conocimientos o habilidades de inteligencia de datos o ciencia directa. El objetivo de *Azure Cognitive Services* es ayudar a los desarrolladores a crear aplicaciones que puedan ver, escuchar, hablar, comprender e incluso razonar. El catálogo de servicios dentro de *Azure Cognitive Services* se puede clasificar en cinco pilares principales: visión, habla, idioma, búsqueda en la web y decisión [3].

Con el objetivo de dotar a nuestra aplicación de la capacidad de reconocer a una persona, de los cinco pilares de *Azure Cognitive Services*, utilizamos el pilar de visión. Éste contiene una API llamada *Face API* que nos permite construir y entrenar un modelo de aprendizaje que posteriormente nos sirve para identificar, mediante la *Face API*, a una persona. Se explica en detalle este proceso en secciones posteriores.

[3] API: Un conjunto de funciones y procedimientos que permiten la creación de aplicaciones que acceden a las características o datos de un sistema operativo, aplicación u otro servicio.
[4] SDK: Es una colección de *software* utilizado para desarrollar aplicaciones para un dispositivo o sistema operativo específico.

1.3. Estructura de la memoria

En primer lugar, nos adentraremos en el estado del arte, un apartado donde nos detendremos a conocer las gafas *Blade* y haremos una comparativa con otras gafas de realidad aumentada en el mercado. En este mismo apartado, echaremos un vistazo a otras aplicaciones que incorporen tecnologías de reconocimiento facial en gafas de realidad aumentada.

En el siguiente apartado, repasaremos las tecnologías y herramientas utilizadas para llevar a cabo el proyecto. Esto proporcionará al lector un ligero concepto de ellas a la hora de adentrarse en los apartados siguientes donde aparecen mencionadas.

Continuaremos con el apartado de especificación, en el que podemos ver los requisitos funcionales, los no funcionales y los casos de uso.

En el siguiente apartado, diseño, veremos el esquema de componentes y la arquitectura del proyecto.

Una vez visto este apartado pasaremos a ver cómo se ha implementado la aplicación y sus diferentes partes, en el apartado de implementación.

A continuación, veremos el apartado de pruebas, donde se muestran los resultados de los tests de usabilidad con distintos usuarios y de respuesta de la aplicación.

Finalmente, llegamos al apartado de conclusión y trabajo futuro, donde comentaremos si se han cumplido los objetivos deseados de la prueba de concepto y las mejoras que podríamos aportar en un futuro para que la aplicación llegara a producción. Al final del documento podemos encontrar referencias bibliográficas.

2. Estado del arte

En este apartado se detallan las características del dispositivo *Blade* y las de otros dispositivos similiares en el mercado de gafas de realidad aumentada acabando por comentar por qué se han elegido las gafas *Blade*. También se contemplan otras aplicaciones de similitud con la que tenemos por objetivo desarrollar.

2.1. Gafas de realidad aumentada

Las gafas de realidad aumentada son un tipo de gafa virtual que permiten superponer imágenes digitales al entorno real de la persona que las usa. Por lo mismo, no cubren completamente los ojos y son transparentes porque es necesario ver el entorno y poder interactuar con él.

2.1.1. Vuzix Blade

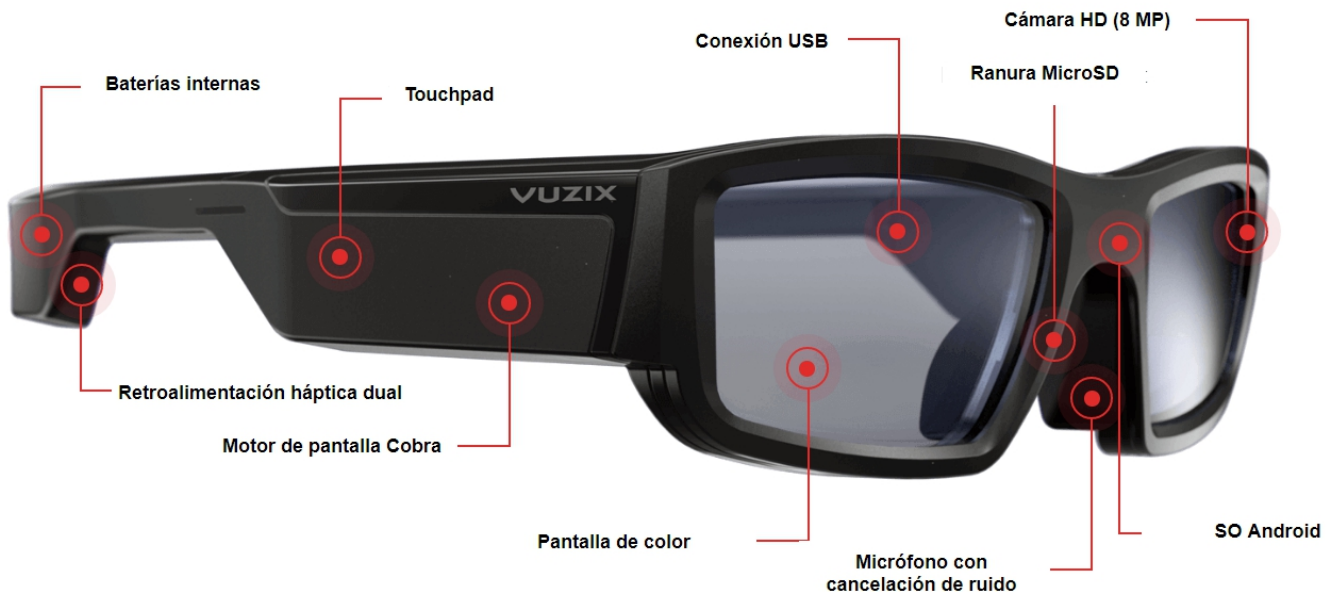


Figura 4. Características de Vuzix Blade.

Las gafas *Blade* funcionan de forma independiente y se pueden conectar a Internet a través de Wi-Fi, pero también se puede emparejar a través de Bluetooth con un dispositivo iPhone o Android para mostrar notificaciones, fotos y videos.

La compañía afirma que tiene una duración de la batería de entre dos y 12 horas, dependiendo de si lo está utilizando principalmente para notificaciones o para aplicaciones más intensivas como acceder a la web a través de Alexa, jugar juegos o usar la Cámara de 8 megapíxeles.

Para interactuar con la interfaz del dispositivo, se puede usar un sistema de control de voz interno que está separado de Alexa, o bien usar una serie de deslizamientos de múltiples dedos sobre el *touchpad* en el lado derecho del marco de las gafas [4].

Hoy en día, podemos encontrar cada vez más fabricantes de gafas de realidad aumentada competidores de Vuzix. Entre ellos, Google, con sus *Google Glass Enterprise Edition 2*, lanzadas a la venta este 2019. Otro competidor importante es Microsoft, con las *HoloLens 2,* también lanzadas en 2019. A parte de las gafas *Blade*, ambos dispositivos de estos grandes competidores están orientados a ser utilizados en un ámbito empresarial, es decir, están desarrolladas con la intención de ayudar a profesionales a llevar a cabo su trabajo de forma más ágil, inteligente y segura.

2.1.2. Microsoft Hololens 2

Figura 5. Gafas Hololens 2.

Microsoft HoloLens 2 son unas gafas inteligentes de realidad mixta desarrolladas y fabricadas por *Microsoft*. Son las sucesoras de las *Microsoft HoloLens*. El 24 de febrero de 2019 las *Hololens 2* debutaron como la primera variante del dispositivo, seguida de una edición para desarrolladores que se anunció el 2 de mayo de 2019, disponiendo estos de una tienda de aplicaciones abierta para que puedan crear sus propias aplicaciones en las gafas, un modelo de navegación web abierto y más APIs y controladores abiertos [5].

2.1.3. Google Glass Enterprise Edition 2

Figura 6. Gafas Google Glass Enterprise Edition 2.

Google Glass Enterprise Edition 2 es una marca de gafas inteligentes, una pantalla óptica montada en la cabeza diseñada en forma de gafas. Fueron lanzadas el 20 de mayo de 2019 y tienen la misión de producir un ordenador ubicuo. Las *Glass* muestran la información en un formato de teléfono inteligente y manos libres permitiendo a los usuarios acceder al navegador de Internet móvil, a la cámara, a los mapas, al calendario y a otras aplicaciones mediante comandos de voz [6].

2.1.4. Comparativa entre *Blade* y otros modelos de gafas de realidad aumentada

A continuación, se muestra una tabla comparativa entre las características técnicas de los dispositivos de los posibles competidores de Vuzix en el mercado de las gafas de realidad aumentada.

Modelo	Blade [7][8]	HoloLens 2 [9]	Glass Enterprise Edition 2 [10]
Pantalla	- Monóculo en el ojo derecho. - Pantalla a color.	-Lentes holográficas transparentes. - Optimización de pantalla para la posición del ojo en 3D.	Módulo de pantalla óptica
Resolución	480x480	2k. relación 3:2	640x360
Compatibilidad	SO Android	SO holográfico de Windows	SO Android Oreo
Procesador	Quad Core ARM CPU	Qualcomm Snapdragon 850	Qualcomm Quad Core, 1.7GHz, 10nm
Peso	93.6 g	566 g	46 g
Audio	Conector micro USB para auriculares	Sonido espacial incorporado	- Un altavoz. - Audio USB. - Audio BT.
Micrófono	Micrófono con cancelación de ruido	Conjunto de cinco canales de micrófonos	3 micrófonos incorporados
Cámara	8 Megapíxeles, soporta 720p 30fps o 1080p 24fps	Imágenes fijas de 8 MP, video 1080p30	8Mp, 80 DFOV
Batería	Baterías internas recargables.	- Duración de 2 a 3 horas de uso activo. - Carga rápida por USB.	820mAh con carga rápida
Controladores	- *Touchpad* - Rastreador de movimiento de cabeza. - Alerta de vibración háptica. - Aplicación de control remoto para dispositivos Android e iOS.	- Modelo totalmente articulado para manipulación directa con dos manos. - Seguimiento visual en tiempo real. - Control por voz; lenguaje natural con conectividad a internet.	*Touchpad* con gestos multitáctiles
Conexiones	Bluetooth, Wifi	- Wifi 802.11ac 2x2. - Bluetooth 5.0. - USB tipo C.	- Wifi 802.11ac, doble banda con antena única. - Bluetooth 5.x AoA.
Almacenamiento	- 5.91 GB expandible con tarjeta microSD. - 1 GB RAM.	- 64 GB UFS 2.1. - 4 GB LPDDR4x.	- 32 GB eMMC Flash. - 3 GB LPDDR4.
Precio	665,00 € sin iva	3.143,00 €	897,00 €

Tabla 1. Comparativa técnica entre gafas de realidad aumentada.

A la hora de comprar unas gafas de realidad aumentada habrá que prestar atención a ciertos requerimientos mínimos generales.

Uno de los puntos más importantes es la resolución de pantalla, que definirá la nitidez con la que vemos las imágenes virtuales en la pantalla óptica transparente.

Otra aspecto a considerar es el campo de visión. Hay que tener claro las gafas de realidad aumentada no están pensadas para aislar a la persona por lo que su campo visual no puede ser completo. Muchos usuarios se han quejado de que las Microsoft Hololens tienen un reducido campo visual pero no entienden que su función es la de aumentar nuestra realidad, no taparla por completo con imágenes digitales. En ese caso se transformarían en gafas de realidad virtual cien por cien inmersivas.

Observando las características de los tres dispositivos en la tabla presentada podemos darnos cuenta de que, en términos de calidad y potencia, las *HoloLens 2* son mejores que los otros dos dispositivos aun que su precio sea más disparatado. Actualmente, Microsoft solo ofrece la posibilidad de adquirir las *HoloLens 2* en Estados Unidos, Francia, Alemania, Irlanda, Nueva Zelanda, Australia y Reino Unido. Las siguen las *Google Glass Enterprise Edition 2*, cuyo precio es orientativo y depende de las necesidades del cliente. Google ofrece diferentes planes, que según estas necesidades, incluyen unos servicios u otros como por ejemplo asistencia técnica. Por último tenemos las *Blade,* que son las más económicas y están a la venta a cualquiera, ya sean empresas o usuarios individuales.

Para llevar a cabo la prueba de concepto deseada, ADD Informática ha decidido adquirir unas gafas de realidad aumentada económicas y, fijándonos en la tabla de carácterísticas, podemos ver que las *Google Glass Enterprise Edition 2* ofrecen mejores prestaciones a cambio de un coste no muy superior. Recordemos que la aplicación está destinada a ser utilizada en residencias geriátricas, donde el personal de asistencia utilizará el dispositivo para identificar a una persona residente. Esto implica que el personal de asistencia tiene que llevar las gafas puestas y hacer una captura al rostro del residente para llevar a cabo la identificación. Teniendo en cuenta este proceso, la razón por la que se han elegido las gafas *Blade* es diferente de los requerimientos generales mínimos descritos arriba. A gran diferencia de las *HoloLens 2* o las *Google Glass Enterprise Edition 2*, el diseño de las *Blade* es de una similitud muy superior a unas gafas no computarizadas y de uso diario. Este es el detalle que desmarca a las *Vuzix Blade* de sus competidores y es de gran interés para ADD Informática. Pues los residentes pueden sentirse nerviosos o molestos al ver a un personal de asistencia llevando un dispositivo tecnológico en la cabeza y apuntando a los mismos.

2.2. Aplicaciones de reconocimiento facial

La tecnología de reconocimiento facial está teniendo un efecto impactante y positivo en nuestra sociedad. Los casos de uso son de gran alcance. Las aplicaciones de reconocimiento facial se están utilizando actualmente para proteger la información personal de los ciberataques, minimizar los arrestos falsos e incluso ayudar a diagnosticar a los pacientes con afecciones genéticas [11]. También se utiliza en redes sociales como Facebook para identificar y etiquetar caras en una imagen.

Figura 7. Face2Gene.

Un ejemplo de aplicación similar a la nuestra prueba de concepto es *Face2Gene*, una aplicación de salud potencialmente innovadora que utiliza reconocimiento facial para ayudar a los médicos y a la bioinformática a priorizar y determinar ciertos trastornos y variantes para sus pacientes. Funciona utilizando un algoritmo patentado que compara las caras de los individuos con aquellos con síndromes que presentan una morfología similar. Como resultado, los pacientes posiblemente pueden ser diagnosticados con síndromes genéticos más rápido y de manera más eficiente [11].

La gran diferencia respecto a nuestra aplicación es que está disponible solo para *smartphones*.

Actualmente, aun es difícil encontrar aplicaciones para gafas de realidad aumentada que utilicen la tecnología del reconocimiento facial.

En la tienda de aplicaciones oficial de Vuzix no se encuentra ninguna aplicación con dichas características, sin embargo, podemos encontrar información en la Internet sobre una aplicación desarrollada para ser integrada en las *Vuzix Blade* cuyo uso está restringido a personal específico: *iFalcon*.

Figura 8. iFalcon.

iFalcon Face Control Mobile, la nueva solución desarrollada por las compañías *Vuzix* y *NNTC*, enfocada para su uso por parte de los responsables de mantener el orden en los espacios públicos, permitiendo una rápida identificación de las caras de los asistentes a dichos espacios sin necesidad de conexión a Internet. Básicamente, la solución ha consistido en integrar capacidades de reconocimiento facial totalmente autónomas impulsada por tecnología de Inteligencia Artificial a las gafas inteligentes *Vuzix Blade*. La cámara es la encargada de escanear los rostros de las personas, enviando inalámbricamente toda la información a una base portátil, encargada de cotejar los rostros escaneados con los de una base de datos local. Una vez haya coincidencias, se ofrecerá toda la información relevante a las propias gafas a través de una pantalla transparente [12].

A diferencia de *Face2Gene*, *iFalcon* sí se utiliza en las *Blade* aunque sea destinada a otro sector. A simple vista, podemos encontrar varias diferencias respecto a nuestra aplicación. La primera es que *iFalcon* procesa todas las caras que detecta a su paso sin excepción. Esto significa un procesamiento de datos significativamente mayor. Nuestra aplicación identifica a un objetivo que nosotros elijamos y en el momento que decidamos. El procesamiento de datos es notablemente menor y por tanto el consumo de batería también, lo que alarga la duración de la misma. Otra gran diferencia es la integración de las capacidades de reconocimiento facial a las *Blade* por parte de *iFalcon*. Esto supone la no necesidad de una conexión a Internet lo que es valorable. En cambio, en nuestra aplicación, necesitamos conexión a internet para poder utilizar el servicio en la nube de Azure.

3. Tecnologías y herramientas utilizadas

En este apartado se describen las herramientas y tecnologías utilizadas para el desarrollo del proyecto y de qué forma es útil su uso.

3.1. Android Studio

Figura 9. Android Studio IDE.

Android Studio es el entorno de desarrollo integrado oficial para la plataforma Android. Está basado en el software *IntelliJ IDEA* de *JetBrains* y ha sido publicado de forma gratuita a través de la Licencia Apache 2.0. Está disponible para las plataformas *Microsoft Windows, macOS y GNU/Linux*. Ha sido diseñado específicamente para el desarrollo de Android. La primera versión estable fue publicada en diciembre de 2014 [13].

Utilizamos *Android Studio* para desarrollar la interfaz gráfica y la lógica relacionada con hacer capturas con la cámara, enviar solicitudes para detectar e identificar a la persona de la captura y consultar al servicio donde está alojada la base de datos para recuperar la informacón de la persona y mostrarla por pantalla.

3.2. Microsoft Visual Studio 2017

Figura 10. Visual Studio IDE.

Microsoft Visual Studio es un entorno de desarrollo integrado (IDE) de Microsoft. Se utiliza para desarrollar programas informáticos, así como sitios web, aplicaciones web, servicios web y aplicaciones móviles. Visual Studio utiliza plataformas de desarrollo de software de Microsoft como Windows API, Windows Forms, Windows Presentation Foundation, Windows Store y Microsoft Silverlight. Puede producir tanto código nativo como código administrado [14].

Se utiliza *Visual Studio 2017* con el objetivo de desarrollar nuestra API web[5] en el lado del servidor. También utilizaremos este entorno para desarrollar un microservicio el cual permite recuperar información de una persona. Este microservicio es expuesto a la Internet gracias a nuesrta API web.

[5]API web: Una API web del lado del servidor es una interfaz programática que consta de uno o más puntos finales expuestos públicamente a un sistema de mensaje de solicitud-respuesta definido, generalmente expresado en JSON o XML, que se expone a través de la web, más comúnmente a través de un Servidor web HTTP.

3.3. Microsoft Azure

Figura 11. Azure.

Microsoft Azure es un servicio de computación en la nube creado por Microsoft para crear, probar, desplegar y administrar aplicaciones y servicios a través de centros de datos administrados por Microsoft. Proporciona software como servicio (SaaS), plataforma como servicio (PaaS) e infraestructura como servicio (IaaS) y admite muchos lenguajes, herramientas y marcos de programación diferentes, incluidos software y sistemas específicos de Microsoft y de terceros [15].

Azure es necesario para desplegar nuestro microservicio en un servidor, haciéndolo accesible desde la Intranet a cualquier aplicación. Así podemos acceder a él desde nuestra aplicación Android, instalada en las gafas *Blade*. Otra utilidad que necesitamos de Azure es el servicio de reconocimiento facial (*Face API*) utilizado por nuestra aplicación Android, cuyo plan se puede crear y administrar desde el portal web de Azure.

3.4. SQLite

Figura 12. SQLite.

SQLite es un sistema de gestión de bases de datos relacionales (RDBMS) contenido en una biblioteca C. A diferencia de muchos otros sistemas de administración de bases de datos, SQLite no es un motor de base de datos cliente-servidor. Más bien, está incrustado en el programa final [16].

SQLite nos es de utilidad para crear nuestra base de datos y almacenar toda la información de las personas que es después consultada. Esta base de datos será incrustada dentro de nuestro microservicio.

3.5. Postman

Figura 13. Postman.

Postman es una aplicación de Google Chrome para interactuar con las API HTTP. Presenta una interfaz de usuario gráfica amigable para construir solicitudes y leer respuestas [17].

Postman es de gran utilidad para probar el correcto funcionamiento de nuestra API web que expone nuestro microservicio. Para probar este funcionamiento hacemos llamadas HTTP a la API y verificamos si la respuesta a estas llamadas es correcta. Gracias a su interfaz gráfica podemos comprobar el correcto formato de los datos recibidos y el contenido de los mismos entre otros factores. También utilizamos Postman para comprender y probar el funcionamiento de la *Face API* de Azure y entrenar nuestro modelo de aprendizaje para el reconocimiento facial.

3.6. GitLab

Figura 14. GitLab.

Gitlab es un servicio web de control de versiones y desarrollo de software colaborativo basado en Git[6] . Además de gestor de repositorios, el servicio ofrece también alojamiento de wikis y un sistema de seguimiento de errores, todo ello publicado bajo una licencia de código abierto [18].

El principal motivo de utilizar este servicio es tener una copia de seguridad de tener nuestro código en la nube. De esta forma, si hubiera algún contratiempo con nuestro equipo local, dispondríamos de nuestro código en la nube y podríamos acceder a él desde cualquier parte. También nos sirve para controlar las versiones de la aplicación, pudiendo volver a una versión anterior del código implementado o comparar entre versiones. Un ejemplo común es cuando tenemos una versión de la aplicación conteniendo esta implementados una serie de componentes determinados funcionando correctamente. Si quisiéramos empezar a desarrollar un nuevo componente, primero subiríamos la versión actual a gitlab, comúnmente hablando, y después, empezaríamos desarrollando la nueva funcionalidad deseada. Esto permite volver a la versión subida si ocurriera algún problema que afectara al correcto funcionamiento de los componentes ya desarrollados a la hora de alterar el código implementando esta nueva funcionalidad.

3.7. Bibliotecas

Aquí se expondrán algunas de las bibliotecas más relevantes para nuestros proyectos.

3.7.1. Bibliotecas en el proyecto Android

```
implementation 'com.vuzix:hud-actionmenu:1.1'                    1
implementation 'com.vuzix:hud-resources:1.1'
implementation 'com.microsoft.projectoxford:face:1.4.4'           2
implementation 'com.google.code.gson:gson:2.8.5'                  3
implementation 'com.android.volley:volley:1.1.1'                  4
```

Figura 15. Bibliotecas del proyecto Android.

Entre las bibliotecas utilizadas para nuestro proyecto Android podemos destacar las siguientes:

- El HUD[7] de Vuzix Blade (1) nos permite utilizar los componentes visuales propios de Vuzix para desarrollar nuestra aplicación.
- La biblioteca de la *Face API* (2) nos permite utilizar un cliente HTTP para realizar llamadas a la misma y así, utilizar el servicio de reconocimiento facial desde nuestra aplicación.
- Utilizamos Gson (3) para convertir objetos Java en su representación JSON.
- Volley (4) es la biblioteca que utilizamos para hacer peticiones HTTP desde nuestra aplicación Android a nuestra API web.

[6] Git: un software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente. Su propósito es llevar registro de los cambios en archivos de computadora y coordinar el trabajo que varias personas realizan sobre archivos compartidos [19].

[7] HUD: método por el cual se representa visualmente la información.

3.7.2. Bibliotecas en el proyecto .Net

Figura 16. .Net Core.

.NET Core es un *framework* de software informático gratuito y de código abierto para los sistemas operativos Windows, Linux y macOS. Es un sucesor multiplataforma de código abierto para .NET Framework. El proyecto es desarrollado principalmente por Microsoft y lanzado bajo la Licencia MIT [20].

Nuestro proyecto .Net, donde tenemos desarrollado nuestro microservicio, se basa en la versión 2.1. de .Net Core.

Entre las tecnologías utilizadas para llevar a cabo su desarrollo podemos destacar:

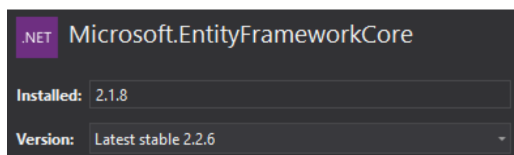


Figura 17. EntityFrameworkCore.

- Entity Framework Core: una versión ligera y extensible de la popular tecnología de acceso a datos Entity Framework.

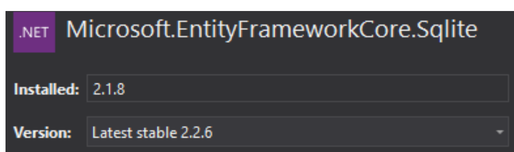


Figura 18. EntityFrameworkCore.Sqlite.

- SQLite para Entity Framework Core: con esta librería podemos utilizar Entity Framework Core para acceder a bases de datos desarrolladas con SQLite.

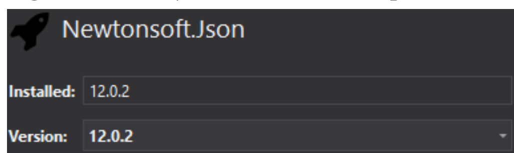


Figura 19. Newtonsoft.Json.

- Newtonsoft.Json: se utiliza esta librería para para convertir objetos C# en su representación JSON.

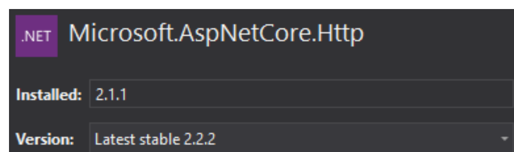


Figura 20. AspNetCore.Http.

- AspNetCore Http: es la librería que permite la capacidad de recibir y resolver peticiones HTTP.

4. Especificación

La especificación de *software* es una descripción del comportamiento del sistema que se va a desarrollar. Este apartado es dedicado a explicar este comportamiento, presentando los requisitos funcionales y no funcionales, los casos de uso, los bocetos de la interfaz de usuario y el modelo de datos utilizado.

4.1. Requisitos funcionales

Los requisitos funcionales son las características que debe incorporar nuestra aplicación. En esta primera versión de nuestra aplicación podemos especificar los siguientes:

- RF1. El usuario tendrá la opción de "identificar" en un menú de la pantalla principal de la aplicación.
- RF2. El usuario es capaz de hacer una captura con la cámara después de seleccionar la opción de "identificar".
- RF3. Una vez hecha la captura al rostro de una persona, se permitirá al usuario elegir si utilizar esta captura o tener la posibilidad de hacer otra diferente.
- RF4. Cuando el usuario elige la captura que utilizar para la identificación, se le proporciona un menú con la opción de elegir sobre qué grupo de personas realizar la búsqueda y reconocimiento.
- RF5. Cuando el proceso de identificación termina, los datos de la persona identificada se representan mediante un componente desplazable verticalmente para poder facilitar la visualización.

4.2. Requisitos no funcionales

- RNF1. Cuando el usuario selecciona el grupo de personas sobre el que realizar la búsqueda y reconocimiento, se inicia el proceso de identificación y, mientras dure, la pantalla es bloqueada mediante una barra de carga impidiendo al usuario interactuar con la aplicación.
- RNF2. Durante todo el uso de la aplicación, el usuario será notificado si ocurren errores.
- RNF3. La aplicación debe estar diseñada siguiendo las guías de diseño de Vuzix.
- RNF4. El tiempo de realizar un flujo de uso completo para un usuario que no había utilizado una aplicación similar anteriormente no debe ser superior a 4 minutos.
- RNF5. El tiempo de realizar un flujo de uso completo por un usuario que ya había utilizado la aplicación anteriormente no debe ser superior a 1 minuto.
- RNF6. Cuando la aplicación identifica a una persona la seguridad de que sea esa persona es de más del 60 %.

4.3. Casos de uso

Después de identificar los requisitos vamos a representar, mediante un diagrama de casos de uso, el comportamiento que debe tener la aplicación. Esta representación es mediante la notación UML.

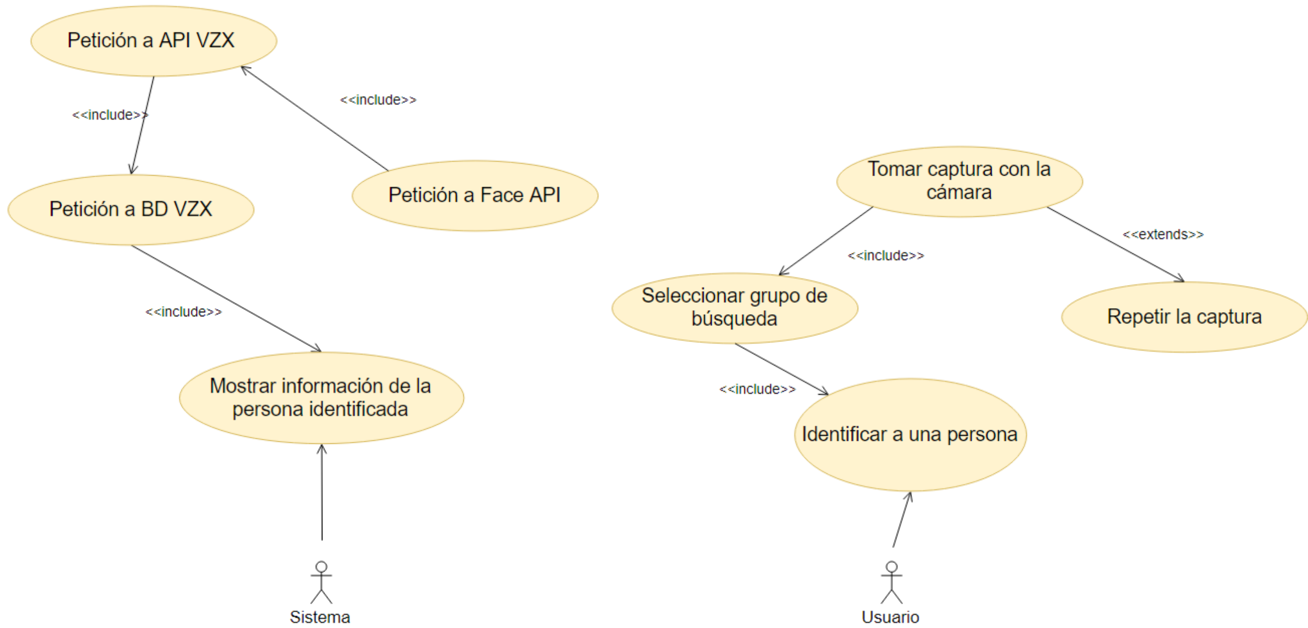


Figura 21. Diagrama de casos de uso.

En este diagrama de casos de uso podemos diferenciar a los dos actores, el sistema y el usuario. El usuario tiene un único caso de uso: la acción de identificar. El caso de uso por parte del sistema es también uno: mostrar información de una persona por pantalla, que es la parte visible del proceso cara al usuario. El resto de operaciones por parte del sistema son transparentes al mismo.

A continuación vamos a especificar cada caso de uso.

Acción	Identificar
Actor	Usuario
Descripción	Ver información de una persona por pantalla.
Pre-condición	- Tomar captura de la persona con la cámara del dispositivo. - Seleccionar un grupo de búsqueda.
Flujo general	- El usuario selecciona la opción de identificar de la pantalla principal. - El usuario toma una captura con la cámara a la persona objetivo de identificar. - El usuario selecciona el grupo de personas en el que quiere realizar la búsqueda e identificación.
Flujo alternativo	- El usuario selecciona la opción de hacer la captura de la persona de nuevo.
Post-condición	El usuario es capaz de ver en la pantalla la información de la persona capturada con la cámara del dispositivo.

Tabla 2. Caso de uso Identificar.

Acción	Mostrar información
Actor	Sistema
Descripción	Mostrar información de una persona por pantalla mediante un componente deslizable.
Pre-condición	- Conexión a Internet. - Haber seleccionado una imagen capturada con la cámara del dispositivo. - Haber elegido un grupo de búsqueda.
Flujo general	- El sistema realiza una petición a la *Face API* y obtiene un identificador asociado a la persona identificada. - El sistema realiza una petición a la API VZX y obtiene la información asociada a una persona. - El sistema muestra la información obtenida.
Flujo alternativo	- No hay conexión a Internet, el sistema muestra un mensaje de aviso. - No se detecta ninguna persona en la imagen capturada. El sistema lanza un mensaje de advertencia. - No se identifica a ninguna persona detectada en la imagen capturada. El sistema lanza un mensaje de advertencia. - No hay ninguna información asociada a la persona identificada en la base de datos. El sistema lanza una advertencia.
Post-condición	El usuario es capaz de ver en la pantalla la información de la persona capturada con la cámara del dispositivo.

Tabla 3. Caso de uso Mostrar información.

4.4. Mock-ups

Para facilitar el diseño de la interfaz gráfica de una aplicación se suelen utilizar los mock-ups, que son bocetos de las diferentes partes de la interfaz.

A continuación, vamos a presentar mediante mock-ups las distintas pantallas de la aplicación.

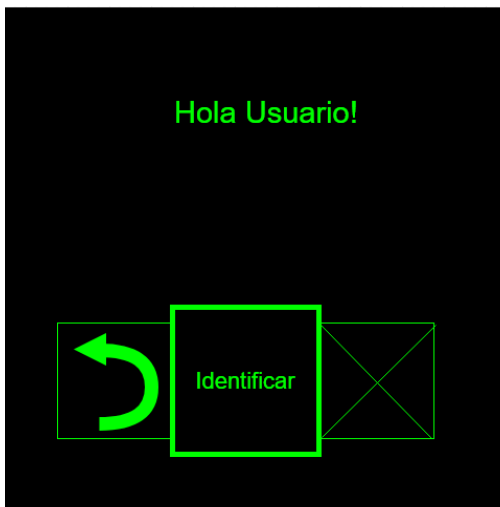


Figura 22. Pantalla principal.

Pantalla principal

Esta pantalla es la principal de la aplicación. Cuando el usuario arranca la misma se encuentra con un menú donde tendrá la opción de empezar la identificación o salir de la aplicación.

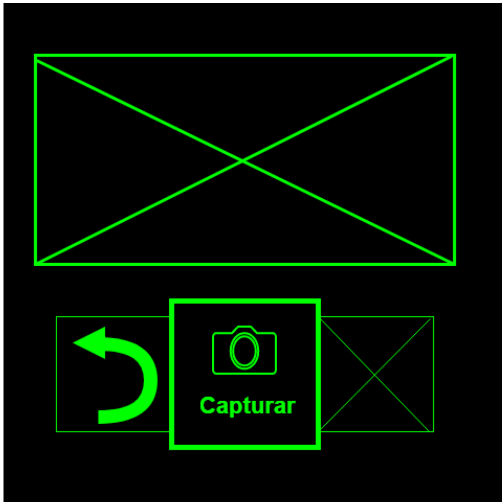


Figura 23. Pantalla de tomar captura.

Pantalla de tomar captura

En esta pantalla el usuario es solicitado tomar una captura con la cámara del dispositivo a la persona que quiere identificar. El usuario tiene las opciones de hacer la captura o volver a la pantalla anterior y, en el recuadro superior, previsualizar lo que está enfocando el objetivo de la cámara antes de hacer la captura. Esta pantalla se muestra después de que el usuario haya seleccionado la opción de identificar de la pantalla principal.

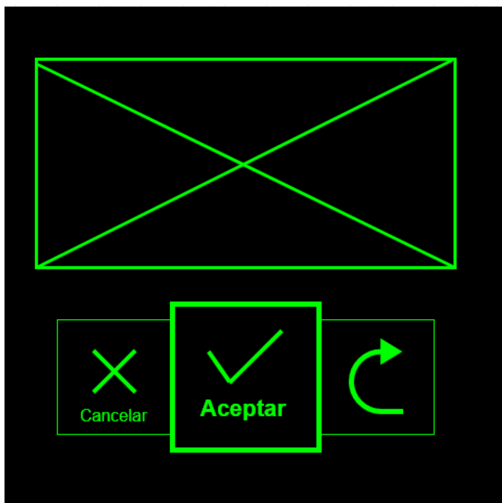


Figura 24. Pantalla de aceptar captura.

Pantalla de aceptar captura

Esta pantalla se muestra después de que el usuario haya hecho una captura. En el recuadro superior puede visualizar el resultado de la captura hecha. El menú de opciones da la posibilidad de cancelar volviendo a la pantalla principal, aceptar la captura y utilizarla, o volver a la pantalla de realizar una nueva captura.

Figura 25. Pantalla de selección de grupo.

Pantalla de selección de grupo

Esta pantalla aparece después de que el usuario haya seleccionado una imagen capturada de la pantalla previa. El menú de opciones da la posibilidad de elegir el grupo de personas según el género de las mismas. Esto permite reducir la búsqueda. El menú también ofrece la posibilidad de volver a la pantalla principal.

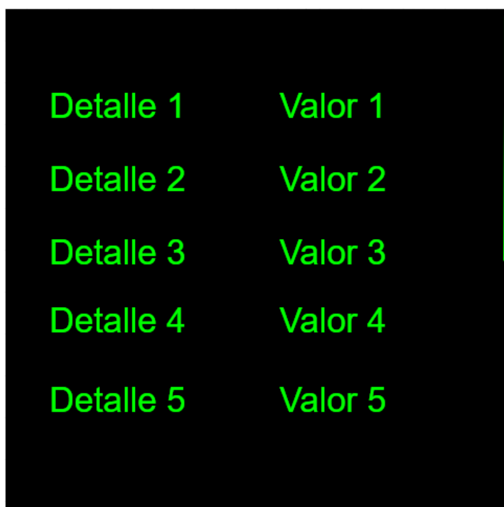


Figura 26. Pantalla de muestra de resultado.

Pantalla de muestra de resultado

Esta pantalla muestra los resultados de la identificación de una persona. El usuario puede deslizar la pantalla verticalmente con el objetivo de visualizar el resto de los datos si hubiera. Con objetivo de una mejor visualización, se ha quitado el menú impidiendo al usuario volver a la pantalla principal. En lugar de utilizar las opciones de menú para tal fin, el usuario deberá utilizar las opciones que proporciona el dispositivo (tocar el *touchpad* con los dos dedos).

4.5. Modelo de datos

Por último, en la siguiente imagen, se muestra el modelo de la base de datos formado por una única tabla llamada "Person" que contiene la información de las personas identificadas.

Person		
Id	text	PK
Codigo	text	
Nif	text	N
Nie	text	N
NumSS	integer	
NumSip	integer	
Nombre	text	
Apellidos	text	
Tipologia	text	N
EstadoCivil	text	
Habitacion	integer	
LugarNacimiento	text	N
Sexo	text	
Cama	integer	
FechaNacimient	text	
FechaIngreso	text	
Dependencia	integer	

Figura 27. Tabla "Person" de la base de datos VZX.

5. Diseño

En este apartado se definen los distintos componentes de la aplicación de forma que podamos tener una idea del diseño de la interacción entre ellos y se presta atención a su arquitectura *software* a nivel de proyecto.

5.1. Esquema de componentes

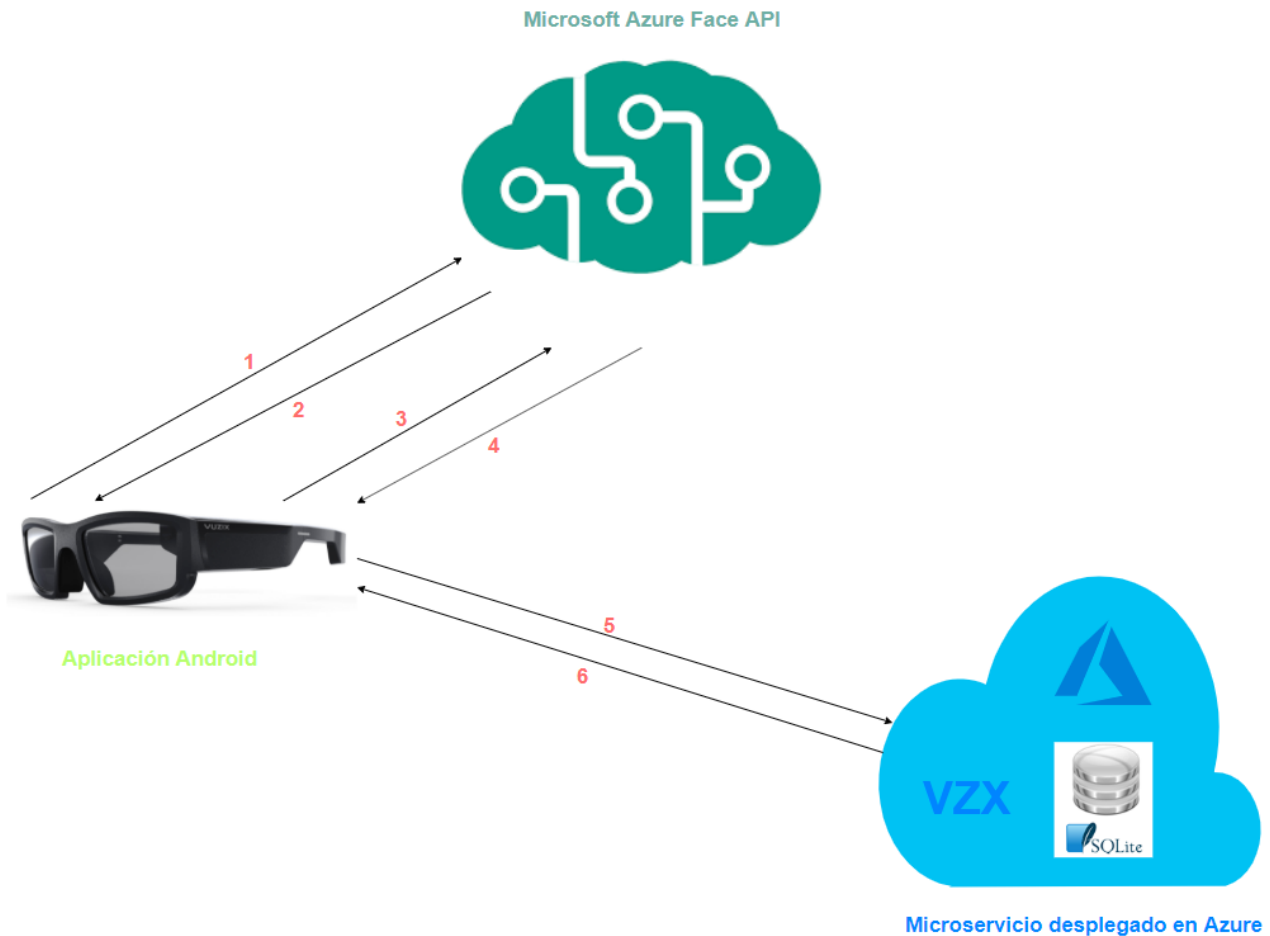


Figura 28. Esquema de componentes.

En este esquema podemos apreciar de forma global la interacción de los distintos componentes de la aplicación para el funcionamiento del flujo básico de esta.

Generalmente analizando, desde el punto de vista de la aplicación Android, podemos diferenciar estas interacciónes separándolas por pasos. En el paso 1 y 2, la aplicación Android llama a los servicios de la Face API de Azure haciendo uso del proceso de detección de caras de la imagen proporcionada en la llamada al servicio. En segundo lugar, los pasos 3 y 4 consisten en otra llamada a la Face API para identificar a la persona asociada a las caras detectadas de la imagen. Por último, en los pasos 5 y 6, se llama al microservicio desplegado en Azure, que tiene una base de datos SQLite incrustada, para obtener la información de la persona identificada. En el siguiente apartado, detallaremos el diseño de los mensajes que intercambian estos componentes entre sí.

5.2. Transferencia de información entre componentes

Siguiendo el esquema de componentes del apartado anterior, a continuación vamos a mostrar el diseño estructural de los mensajes intercambiados entre estos componentes a la hora de comunicarse.

Hacemos uso de Postman para efectuar está comunicación entre componentes mediante llamadas REST[8] . Mostraremos de forma gráfica, mediante capturas de la interfaz gráfica de Postman, la forma de cada llamada efectuada entre componentes y su respuesta. Esta muestra es en formato JSON[9] donde es posible.

5.2.1. Detección

En el paso 1, la aplicación envía una petición al servicio de detección de la Face API de Azure.

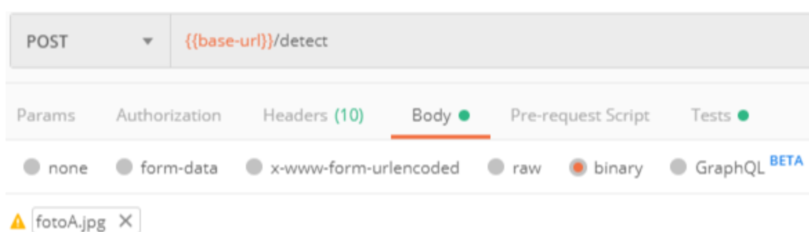


Figura 29. Petición REST Detección.

Como se observa en la captura, el cuerpo de esta petición contiene una imagen ("fotoA.jpg") donde se detecta la cara de una persona en el caso ideal de que la hubiera. Esta es la llamada básica del servicio de detección. En el apartado de implementación de este documento veremos que podemos especificar más parámetros para esta llamada dependiendo de la respuesta que queremos de vuelta.

En el paso 2, la Face API devuelve los resultados del proceso de detección obtenidos tras efectuar la petición del paso 1.

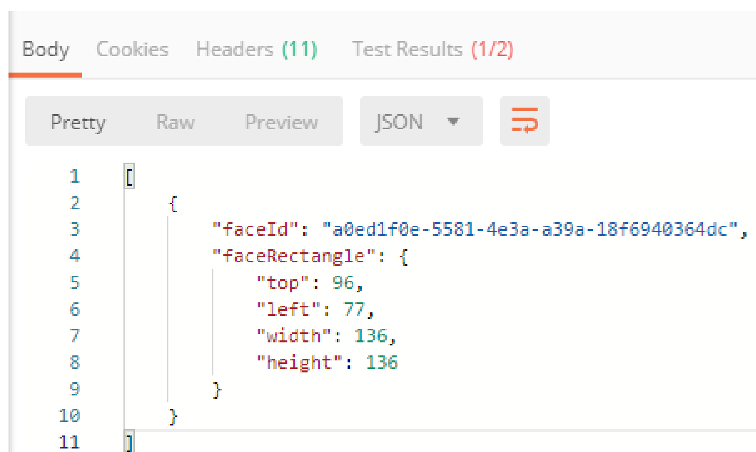


Figura 30. Respuesta REST Detección.

Esta respuesta consiste en una lista de objetos. Estos objetos representan las caras detectadas en la imagen. En la captura podemos ver que la respuesta de la petición del paso anterior consiste en una lista que contiene la información de una cara detectada. Los detalles que proporciona la respuesta sobre una cara son un identificador ("faceId") de la cara detectada y las medidas del rectángulo que envuelve esta. Estas características relativas al rectángulo son irrelevantes para nuestra aplicación. La información necesaria de esta respuesta es el identificador de la cara detectada, que utilizaremos en el paso siguiente.

---

[8]REST: un estilo de arquitectura de software que define un conjunto de restricciones que se utilizarán para crear servicios Web.

[9]JSON: un formato de texto sencillo para el intercambio de datos.

5.2.2. Identificación

Continuando en el paso 3, después del proceso de detección, se efectúa una llamada a la Face API para llevar a cabo el proceso de identificación.

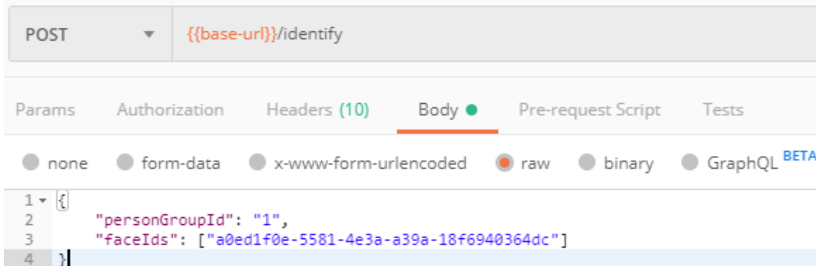


Figura 31. Petición REST Identificación.

En el cuerpo de esta petición se especifica la lista de identificadores obtenidos mediante el proceso de detección. En el caso del proceso anterior, un solo identificador. Otro parámetro especificado es el del grupo de personas sobre el que hacer el proceso de identificación. Se explica este parámetro más en detalle en el apartado de implementación.

Después del paso 3, en respuesta a la petición efectuada, el paso 4 consiste en la respuesta devuelta por la Face API conteniendo esta los resultados de la identificación.

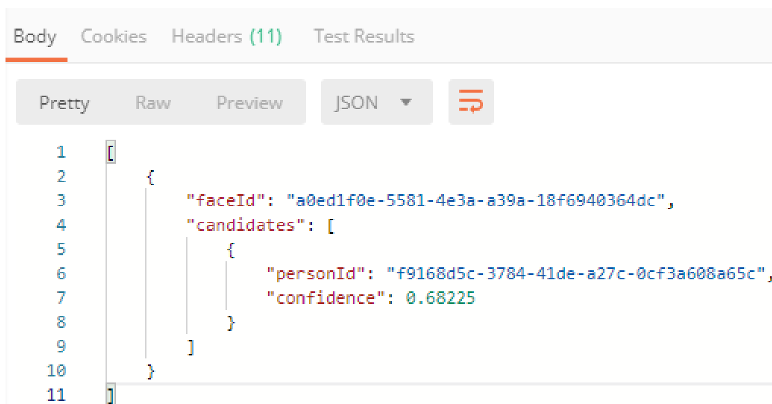


Figura 32. Respuesta REST Identificación.

Los resultados de la identificación contenidos en la respuesta se representan mediante una lista de objetos. Cada objeto representa la información relativa a una cara identificada. Esta consiste en el identificador de la cara ("faceId") y una lista de posibles candidatos ("candidates") a ser la persona detectada e identificada de la imagen. Cada objeto que representa a un candidato contiene un identificador de la persona ("personId") y un valor ("confidence") que representa, entre 0 y 1, la certeza de que sea la persona identificada en la imagen.

De esta lista necesitamos recuperar, de todos los candidatos de todas las caras, el candidato con más certeza de ser la persona de la imagen. Se utiliza el identificador de este candidato en el siguiente paso.

5.2.3. Recuperación de información

Después de obtener el identificador de la persona identificada, en el paso 5, se realiza una petición al microservicio desplegado en Azure para obtener la información de esta persona.

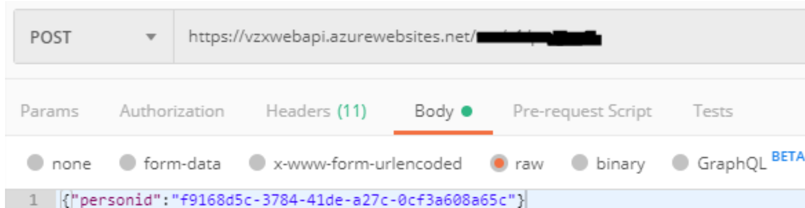


Figura 33. Petición VZX web API.

El cuerpo de esta petición está formado por un objeto que contiene el identificador de la persona de la cual recuperar la información. Este identificador es el obtenido en el paso anterior.

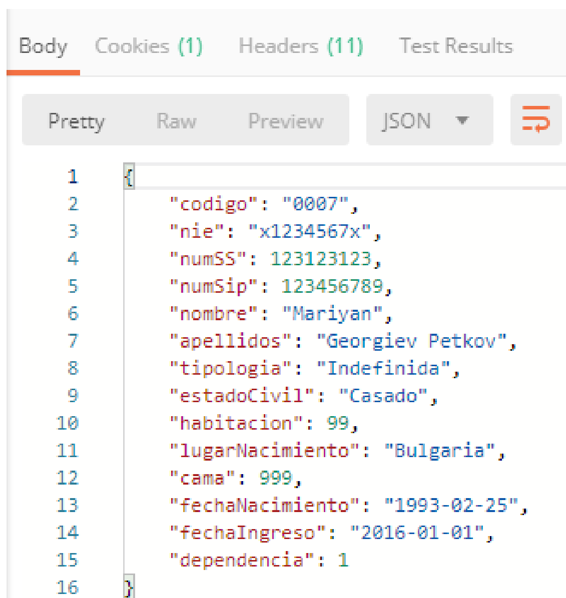


Figura 34. Respuesta REST VZX web API.

En el paso 6, se recibe la respuesta desde el microservicio con la información relevante a la persona identificada. Esta respuesta consiste en un objeto con los detalles de la persona.

5.3. Arquitectura

La arquitectura de software es el diseño de más alto nivel de la estructura de un sistema.

Una arquitectura de software, también denominada arquitectura lógica, consiste en un conjunto de patrones y abstracciones coherentes que proporcionan un marco definido y claro para interactuar con el código fuente del software. Esta se selecciona y diseña con base en objetivos (requisitos) y restricciones. Los objetivos son aquellos prefijados para el sistema de información, pero no solamente los de tipo funcional, también otros objetivos como la mantenibilidad, auditabilidad, flexibilidad e interacción con otros sistemas de información. Las restricciones son aquellas limitaciones derivadas de las tecnologías disponibles para implementar sistemas de información.

La arquitectura de software define, de manera abstracta, los componentes que llevan a cabo alguna tarea de computación, sus interfaces y la comunicación entre ellos [21].

5.3.1. Arquitectura de la aplicación Android

El Model-view-presenter (MVP) es una variante del patrón arquitectónico model-view-controller (MVC). Está diseñado para facilitar las pruebas unitarias automatizadas y mejorar la separación de componentes en la lógica de presentación.

En MVP, el presentador asume la funcionalidad del "hombre intermedio", toda la lógica de presentación es empujada a este [22].

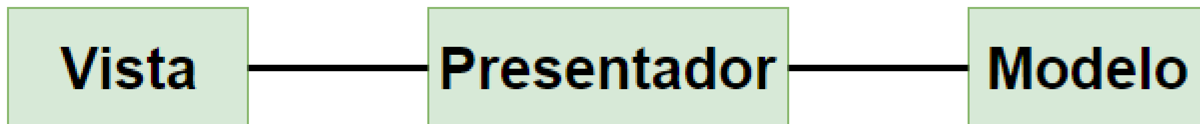


Figura 35. Arquitectura MVP.

El **modelo** es una interfaz que define los datos que se mostrarán o sobre los que se actuará en la interfaz de usuario.

La **vista** es una interfaz pasiva que muestra los datos (el modelo) y envía los comandos de usuario (eventos) al presentador para que actúe sobre ellos.

El **presentador** actúa sobre el modelo y la vista. Recupera datos de los repositorios (el modelo) y los formatea para mostrarlos en la vista [22].

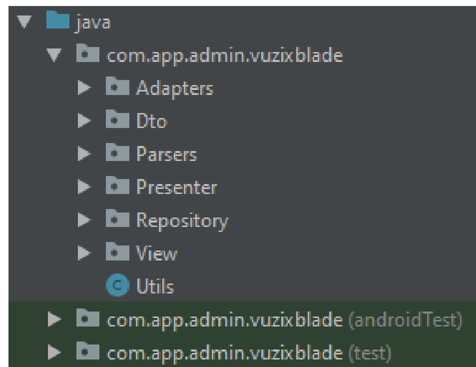


Figura 36. Estructura del proyecto Android.

En esta figura podemos observar la estructura general de nuestro proyecto Android de acuerdo con el patrón MVP.

La vista se compone del directorio "View", que contiene las clases correspondientes a cada pantalla de la aplicación. En estas clases se definen los componentes de esa misma pantalla de forma lógica. Por otro lado, en el directorio "res" podemos encontrar los componentes XML[10] correspondientes a estos componentes lógicos.

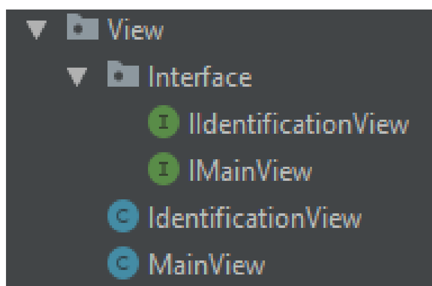


Figura 37. Vista lógica del proyecto Android (MVP).

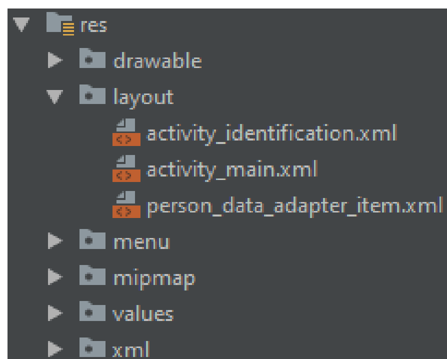


Figura 38. Vista XML del proyecto Android (MVP).

El modelo está compuesto por el directorio "Repository". Este directorio tiene dos subdirectorios, "LocalRepository" y "RemoteRepository", para diferenciar el acceso local y el acceso remoto a los recursos. Podemos ver que el directorio "LocalRepository" no tiene contenido de momento. Sin embargo, este se ha creado con el objetivo de preparar al proyecto con la capacidad de albergar esta funcionalidad, si la hubiera, en un futuro. De esta forma, si un nuevo desarrollador continúa este proyecto, puede ser más ágil a la hora de desarrollar nueva funcionalidad al tener la arquitectura clara a simple vista. En el directorio "RemoteRepository" podemos encontrar los servicios de la Face API y del microservicio que hemos llamado VZX. Estos servicios utilizan tareas en segundo plano para no sobrecargar el hilo principal de ejecución que podemos encontrar en el directorio de "Tasks".

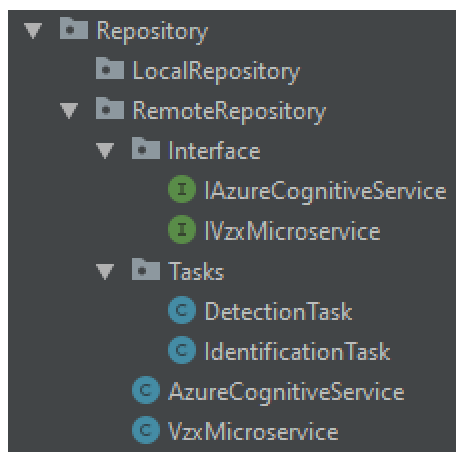


Figura 39. Modelo del proyecto Android (MVP).

XML[10]: es un meta-lenguaje que permite definir lenguajes de marcas utilizado para almacenar datos en forma legible.

La parte del presentador está compuesta por el directorio “Presenter”. Desde aquí se reciben las órdenes del usuario al interactuar con las vistas, se modifican estas, o se consultan los recursos necesarios al modelo.

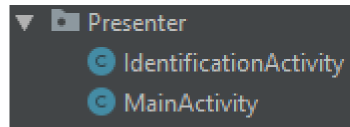


Figura 40. Presentador del proyecto Android (MVP).

Gracias a esta arquitectura con sus respectivos componentes diferenciados, podemos concluir que este proyecto está preparado para soportar cambios sin suponer grandes alteraciones de forma general. Esto quiere decir que, al tener los componentes de la vista, el presentador y el modelo desacoplados el uno del otro, se reducen las dependencias entre estos permitiendo que al modificar o cambiar cualquiera de ellos el resto no se vea afectado significativamente.

El buen desacople entre componentes se puede notar al cambiar un componente y ver cómo este cambio afecta al resto.

Un ejemplo real, relativo a nuestro proyecto, podría ser el cambio de acceso a recursos de forma remota a una forma local. Esto se podría dar si en vez de acceder a la información de la persona identificada mediante el microservicio VZX, quisiéramos acceder a esta información albergada en una base de datos local o en el sistema de ficheros.

El cambio para el resto de componentes sería casi insignificante. En el presentador, que es el que se encarga de interactuar con el modelo, habría que modificar la llamada y el servicio al que se llama. Este cambio no supondría más de cinco líneas de código. Por otro lado, la vista no se vería afectada de ninguna manera ya que no tiene interacción con el modelo.

5.3.2. Arquitectura de la aplicación .Net Core

La arquitectura de microservicios es una aproximación para el desarrollo de software que consiste en construir una aplicación como un conjunto de pequeños servicios, los cuales se ejecutan en su propio proceso y se comunican mediante mecanismos ligeros. Cada servicio se encarga de implementar una funcionalidad completa del negocio. Cada servicio es desplegado de forma independiente y puede estar programado en distintos lenguajes y usar diferentes tecnologías de almacenamiento de datos [23].

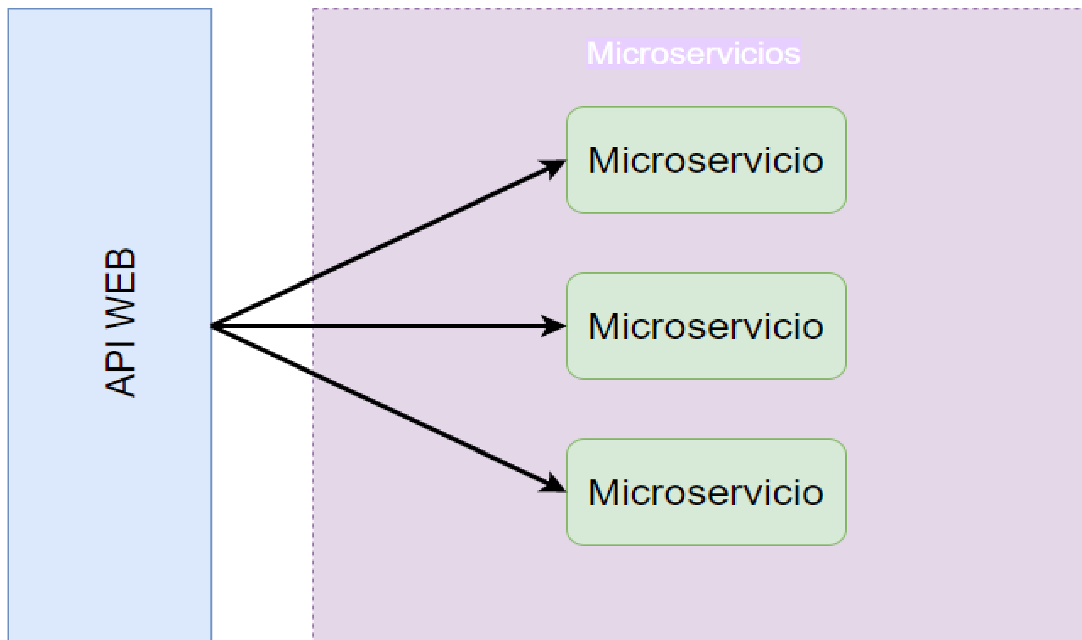


Figura 41. Arquitectura de microservicios.

Como hemos visto en el apartado anterior, nuestra aplicación Android consulta a un servicio llamado VZX para recuperar información de una persona. Nos referimos a este servicio como microservicio porque el proyecto al que pertenece está diseñado siguiendo una arquitectura de microservicios, donde este servicio está comprendido como microservicio de dicha arquitectura. A continuación expondremos el diseño de sus distintos componentes.

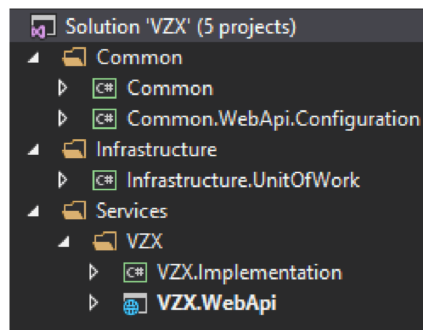


Figura 42. Estructura del proyecto .Net Core

La aplicación desarrollada en .Net Core se encarga de la lógica de negocio y del almacenamiento de datos, es decir, el usuario no interactúa con ella (al menos de forma directa), y por tanto, el proyecto no tiene componentes gráficos ni interfaces de usuario. Este proyecto expone una web API con puntos de acceso mediante los cuales se accede a los recursos disponibles.

En la figura de arriba podemos diferenciar tres directorios lógicos del proyecto .Net Core.

En el directorio de "Services" se encuentran los microservicios de la aplicación, cada uno con su módulo de implementación ("Implementation") y su módulo de configuración ("WebApi"). Los directorios de "Infrastructure" y "Common" contienen módulos que pueden ser comunes a varios microservicios. Esto significa que tienen recursos, funciones o configuraciones globales que son utilizados por numerosos microservicios y, para no replicar estos recursos y funciones en todos los microservicios por separado, se ha optado por generalizarlos de forma que sean accesibles por todos.

A continuación, vamos a detallar el diseño de la estructura del microservicio VZX empezando por el módulo de implementación.

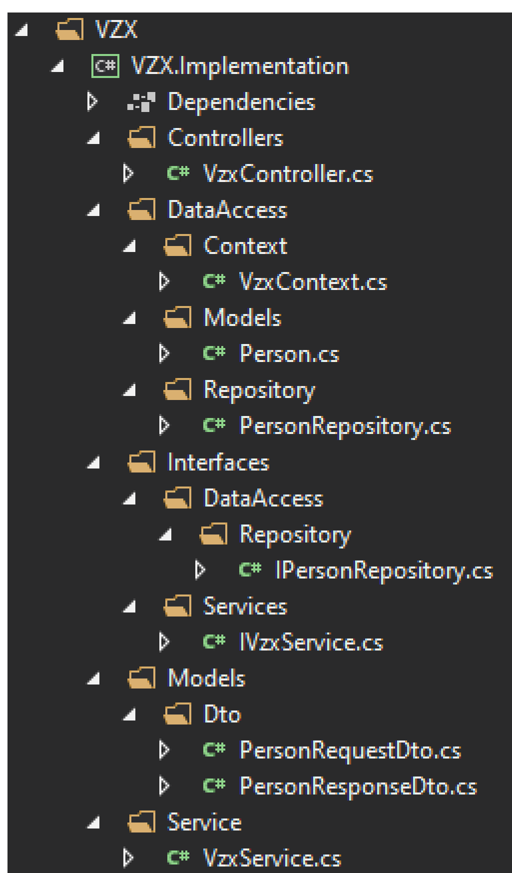
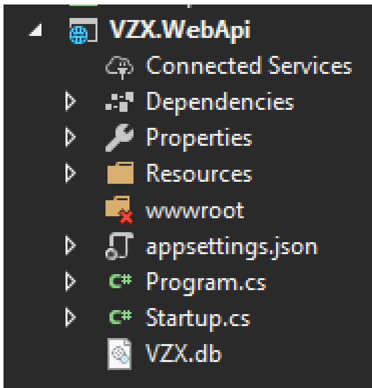


Figura 43. Estructura de la implementación del microservicio VZX.

Empezando por arriba, en el directorio "Controllers" podemos encontrar las clases donde se definen los puntos de acceso de la web API. Seguidamente, en "DataAccess" se puede encontrar el contexto ligado a la base de datos, las entidades representadas mediante clases y el repositorio para operar con ellas. Idealmente hay un repositorio por cada entidad, definiendo las operaciones realizables sobre ellas. En nuestro caso solo tenemos una entidad llamada "Person". Después tenemos el directorio de interfaces, donde se definen las interfaces de los distintos repositorios y de los distintos servicios del microservicio. En el directorio de "Models" podemos encontrar los DTOs[11] utilizados por el microservicio. Por último, en el directorio "Service" se alberga la lógica del microservicio. En el apartado de implementación veremos más en detalle estos componentes de la aplicación.

[11]DTO: Un objeto de transferencia de datos (en inglés, data transfer object, abreviado DTO) es un objeto que transporta datos entre procesos. La motivación de su uso tiene relación con el hecho que la comunicación entre procesos se realiza generalmente mediante interfaces remotas (por ejemplo, servicios web), donde cada llamada es una operación costosa.



En el módulo de configuración se puede destacar el archivo "appsettings.json", donde se pueden configurar ciertas características del microservicio. Este archivo es leído desde las clases "Program.cs" y "Startup.cs" para aplicar las configuraciones especificadas. También cabe destacar el archivo "VZX.db" que es la base de datos incrustada en el mismo microservicio.

Figura 44. Estructura de la configuración del microservicio VZX.

5.4. Estructura del modelo de aprendizaje entrenado

Para acabar el apartado de diseño, expondremos la estructura del modelo de aprendizaje de la *Face* API, necesario para el servicio de identificación utilizado.

En aprendizaje automático y minería de datos, el aprendizaje supervisado es una técnica para deducir una función a partir de datos de entrenamiento. La salida de la función puede ser un valor numérico (como en los problemas de regresión) o una etiqueta de clase (como en los de clasificación). El objetivo del aprendizaje supervisado es que esa función sea capaz de predecir el valor correspondiente a cualquier objeto de entrada válida después de haber visto una serie de ejemplos, los datos de entrenamiento. Para ello, tiene que generalizar a partir de los datos presentados a las situaciones no vistas previamente [24].

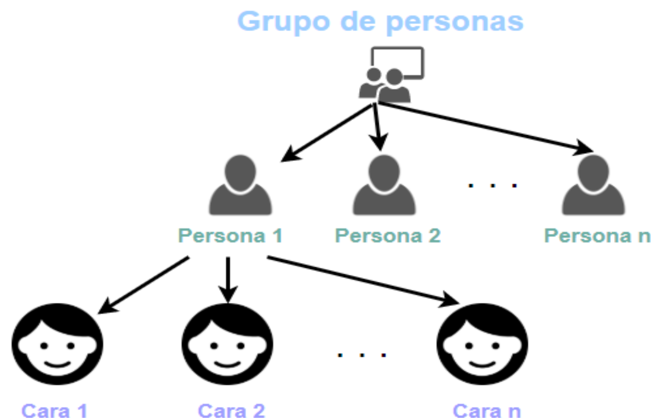


Figura 45. Estructura de modelo de aprendizaje de la Face API.

Para llegar a identificar personas mediante la *Face* API de Azure, primero se tiene que crear un modelo de aprendizaje y entrenarlo mediante esta técnica. Para ello, tenemos que crear un grupo de personas y, por cada persona, múltiples caras las cuales se proporcionarán mediante imágenes. Después entrenamos el modelo. Veremos este proceso en el apartado de implementación, donde utilizamos Postman para llevarlo a cabo.

6. Implementación

Este apartado está dedicado a explicar la implementación de la aplicación en detalle, desde una perspectiva técnica. Empezaremos por el proyecto de Android, pasando por el proyecto de .Net Core y acabaremos explicando cómo se ha entrenado el modelo de aprendizaje de la *Face* API.

6.1. Implementación del proyecto Android

Teniendo en cuenta la estructura del proyecto Android vista en el apartado de diseño, vamos a dividir la explicación en las tres partes correspondientes al patrón MVP.

6.1.1. Vista del proyecto Android

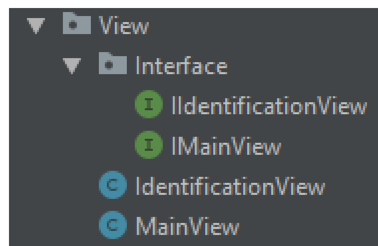


Figura 46. Vista lógica del proyecto Android (MVP).

Recordando el directorio "View" podemos ver dos clases, "MainView.java" y "IdentificationView.java". La primera corresponde a la pantalla principal de la aplicación y la segunda a la pantalla de seleccionar el grupo de búsqueda y la de mostrar la información de una persona. Cada una dispone de su interfaz en el directorio de "Interface", donde se exponen sus métodos de visibilidad pública.

```
public class MainView implements IMainView {

    private MenuItem cameraMenuItem;
    private TextView mainText;
    private Menu mainMenu;
    private Activity mainActivity;

    public MainView(Activity mainActivity) {
        this.mainActivity = mainActivity;

        setMainText(this.mainActivity.findViewById(R.id.mainTextView));
    }
```

Figura 47. Clase MainView.java.

En la clase “MainView.java” podemos encontrar definidos de forma lógica los componentes visuales de la pantalla principal. Además, en el constructor de la clase se recibe un objeto “Activity”. Este objeto es necesario para poder asociar estos componentes lógicos a sus respectivos componentes XML, disponibles solo en el contexto de dicho objeto. El objeto “Activity” forma parte del alcance del presentador, que se encarga de crear una instancia de esta vista. Lo veremos más adelante.

El resto de la clase contiene métodos para acceder o asignar valores a los componentes. Las demás clases los utilizan mediante la interfaz “IMainView”.

```
public class IdentificationView implements IIdentificationView {

    private Menu genderMenu;
    private MenuItem femaleGroupMenuItem;
    private MenuItem maleGroupMenuItem;
    private ProgressBar progressBar;
    private TextView selectPersonGroupText;
    private ListView identificationResultListView;
    private Activity identificationActivity;

    public IdentificationView(Activity identificationActivity) {
        this.identificationActivity = identificationActivity;
        setProgressBar(identificationActivity.findViewById(R.id.progressBarIdentification));
        setSelectPersonGroupText(identificationActivity.findViewById(R.id.selectPersonGroupText));
        setIdentificationResultListView(identificationActivity.findViewById(R.id.personDataList));
        loadFirstTimeView();
    }
    private void loadFirstTimeView(){
        getProgressBar().setVisibility(View.GONE);
        getSelectPersonGroupText().setText(R.string.select_person_group_text);
    }
```

Figura 48. Clase IdentificationView.java.

La clase “IdentificationView.java” sigue la misma idea que la clase anterior. De esta clase podemos destacar el componente de la barra de progreso, que se muestra cuando la aplicación procesa tareas costosas y se oculta al terminarlas. Otro componente a destacar es el componente de lista donde se muestran los resultados de la identificación una vez obtenidos. Además, esta vista dispone de un método que configura algunos componentes nada más construirla, dejando a esta en un estado que podemos llamar inicial.

6.1.2. Modelo del proyecto Android

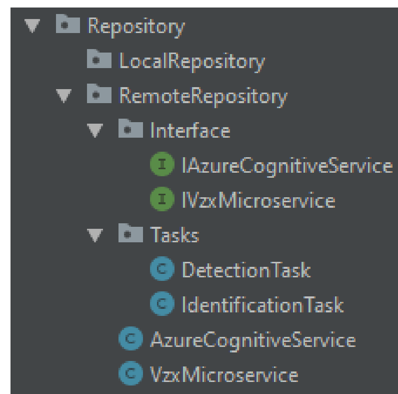


Figura 49. Modelo del proyecto Android (MVP).

Como se explicó en el apartado de diseño, el modelo del proyecto consiste en un repositorio remoto y otro local (el cual no se utiliza por ahora) desde los cuales se accede a los recursos. En este subapartado vamos a comentar el acceso a los recursos remotos como son el uso de la Face API comprendida en la clase “AzureCognitiveService.java” y el uso del microservicio VZX accesible desde la clase “VzxMicroservice.java”. También se comentarán las clases “DetectionTask.java” y “IdentificationTask.java” que abarcan las tareas lanzadas en segundo plano para no sobrecargar el hilo principal de ejecución.

```
@Override
public void detectAndIdentify(Bitmap bitmap) {
    ByteArrayOutputStream output = new ByteArrayOutputStream();
    bitmap.compress(Bitmap.CompressFormat.JPEG, quality: 100, output);
    ByteArrayInputStream inputStream = new ByteArrayInputStream(output.toByteArray());
    try {
        output.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
    // Start a background task to detect
    new DetectionTask(endpoint, subKey, service: this).execute(inputStream);
    try {
        inputStream.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

@Override
public void DetectPostExecution(Face[] result){
    identificationActivityWeakReference.get().DetectPostExecution(result);
    if (result != null && result.length > 0) {
        faces = Arrays.asList(result);
        detected = true;
        identify();
    } else {
        detected = false;
    }
}
```

Figura 50. Detección en la clase AzureCognitiveServices.java

En el esquema de componentes del apartado de diseño se diferenciaron dos pasos para identificar a una persona mediante los servicios de la *Face* API. Esto quiere decir dos procesos diferentes, los cuales son detectar e identificar. La clase "AzureCognitiveService.java" contiene estos dos pasos comprendidos en diferentes métodos. En la figura de arriba podemos ver la parte de detección, formada por dos métodos. El método "detectAndIdentify" lanza una tarea en segundo plano para llevar a cabo la detección de la imagen recibida como parametro de entrada. Tras acabar la tarea, se llama al método "DetectPostExecution" el cual, en caso de detectar alguna cara, inicia el proceso de identificación. Hay que destacar que este método también llama al método con el mismo nombre del presentador para que este actualice la vista según los resultados de la detección. Esto es posible gracias a lo que se llama una referencia débil a la clase correspondiente del presentador, la cual llama a este servicio ("IdentificationActivity.java").

```java
// Called when the detection is done.
private void identify() {
    if (detected && mPersonGroupId != null) {
        // Start a background task to identify
        List<UUID> faceIds = new ArrayList<>();
        for (Face face:  faces) {
            faceIds.add(face.faceId);
        }
        new IdentificationTask(mPersonGroupId, endpoint, subKey,  service: this).execute(
                faceIds.toArray(new UUID[faceIds.size()]));
    }
}

@Override
public void IdentifyProgressUpdate(){
    identificationActivityWeakReference.get().IdentifyProgressUpdate();
}

@Override
public void IdentifyPostExecution(IdentifyResult[] result){

    if(result != null && result.length > 0 && result[0].candidates != null && result[0].candidates.size() > 0) {
        Candidate mostRelevantCandidate = result[0].candidates.get(0);
        for (IdentifyResult identifyResult : result) {
            for (Candidate candidate : identifyResult.candidates) {
                if (candidate.confidence > mostRelevantCandidate.confidence) {
                    mostRelevantCandidate = candidate;
                }
            }
        }
        identificationActivityWeakReference.get().IdentifyPostExecution(mostRelevantCandidate.personId.toString());
    }else{
        identificationActivityWeakReference.get().IdentifyPostExecution( personId: null);
    }
}
```

Figura 51. Detección en la clase AzureCognitiveServices.java

En esta figura se detalla el proceso de identificación, iniciado justo después del proceso de detección. Este proceso tiene la misma estructura que el anterior lanzando un tarea en segundo plano para no sobrecargar el hilo principal de ejecución y recibiendo los resultados de la tarea en el respectivo método. Este último, mediante la referencia débil, devuelve el resultado al presentador ejecutando su método de post-ejecución.

```java
public class DetectionTask extends AsyncTask<InputStream, String, Face[]> {

    private final String endpoint;
    private final String subKey;
    private WeakReference<IAzureCognitiveService> azureCognitiveServiceWeakReference;
    public DetectionTask(String endpoint, String subKey, IAzureCognitiveService service){
        this.endpoint = endpoint;
        this.subKey = subKey;
        azureCognitiveServiceWeakReference = new WeakReference<>(service);
    }

    @Override
    protected Face[] doInBackground(InputStream... params) {
        // Get an instance of face service client to detect faces in image.
        FaceServiceClient faceServiceClient = new FaceServiceRestClient(endpoint, subKey);
        InputStream inputStream = params[0];
        try{
            // Start detection.
            return faceServiceClient.detect(
                    inputStream,  /* Input stream of image to detectAndIdentify */
                    b: true,       /* Whether to return face ID */
                    b1: false,       /* Whether to return face landmarks */
                        /* Which face attributes to analyze */
                    faceAttributeTypes: null);
        }  catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }

    @Override
    protected void onPostExecute(Face[] result) {

            azureCognitiveServiceWeakReference.get().DetectPostExecution(result);
    }
}
```

Figura 52. Tarea de detección en la clase DetectionTask.java.

El lanzamiento en segundo plano de la tarea de detección nos lleva a la clase "DetectionTask.java". Esta clase está formada por un método llamado "doInBackground" que comprende la parte que se ejecutará en segundo plano. El método "onPostExecute" recibe el resultado cuando la tarea acaba y llama al método de post-detección de la clase que ha lanzado la tarea.

Podemos observar que esta tarea consiste en utilizar un cliente REST y utilizar el servicio de detección. Este cliente necesita un enlace y una clave de subscripción para acceder al servicio remoto de la *Face* API.

```java
@Override
protected IdentifyResult[] doInBackground(UUID... params) {
    // Get an instance of face service client to detectAndIdentify faces in image.
    FaceServiceClient faceServiceClient =  new FaceServiceRestClient(endpoint, subKey);
    try{
        TrainingStatus trainingStatus = faceServiceClient.getPersonGroupTrainingStatus(
                this.mPersonGroupId);     /* personGroupId */
        if (trainingStatus.status != TrainingStatus.Status.Succeeded) {
            return null;
        }
        return faceServiceClient.identityInPersonGroup(
                this.mPersonGroupId,   /* personGroupId */
                params,                  /* faceIds */
                v: 0.6F,                 /*confidence threshold */
                i: 1);  /* maxNumOfCandidatesReturned */
    }  catch (Exception e) {
        e.printStackTrace();
        return null;
    }
}

@Override
protected void onProgressUpdate(String... values) {
    azureCognitiveServiceWeakReference.get().IdentifyProgressUpdate();
}

@Override
protected void onPostExecute(IdentifyResult[] result) {
    azureCognitiveServiceWeakReference.get().IdentifyPostExecution(result);
}
```

Figura 53. Tarea de identificación en la clase IdentificationTask.java.

La tarea de identificación tiene la misma forma que la tarea de detección. En la clase "IdentificationTask.java" podemos destacar la llamada de identificar del cliente REST. En esta llamada hemos especificado que la respuesta que nos devuelva el servicio contenga un resultado de identificación de tamaño máximo de una persona como candidato. También hemos especificado que el candidato devuelto tenga una certeza de ser la persona de la imagen de un valor mínimo de 0.6 (entre 0 y 1).

Por último de este subapartado, vamos a mostrar la implementación de la clase "VzxMicroservice.java". Esta clase contiene la llamada al microservicio de donde se obtiene la información de la persona identificada.

La parte destacable consiste en un método que recibe un identificador correspondiente a una persona, construye una petición que envía al microservicio y recibe la respuesta de este con la información de la persona identificada mediante el identificador. Como se explicó en apartados anteriores, se utiliza la biblioteca de Volley para llevar a cabo el proceso de hacer la petición y recibir la respuesta. Esta clase también tiene una referencia débil a la clase, del dominio del presentador, desde la que es construida su instancia. Cuando se recibe la respuesta, los resultados son devueltos al presentador llamando a un método de la clase del dominio de este.

```java
@Override
public void getPersonById(String personId){
    PersonRequestDto personRequestDto = new PersonRequestDto(personId);
    try {
        JSONObject jsonObject = new JSONObject(gson.toJson(personRequestDto));
        jsonObject.put( name: "Content-Type",  value: "application/json");
        String url = identificationActivityWeakReference.get().getResources().getString(R.string.microservice_endpoint);
        Log.i( tag: "Microservice",  msg: "starting VZX microservice request");
        JsonObjectRequest objReq = new JsonObjectRequest(Request.Method.POST, url, jsonObject,
                response -> {
                    if (response != null) {
                        identificationActivityWeakReference.get()
                                .showPersonData(gson.fromJson(response.toString(), PersonResponseDto.class));
                    } else {
                        Log.e( tag: "JsonObjectRequest",  msg: "Response is null");
                        identificationActivityWeakReference.get().showPersonNotFound();
                    }
                },
                error -> {
                    // If there is a HTTP error then notify.
                    Log.e( tag: "Volley", error.toString());
                    error.printStackTrace();
                }
        );
        requestQueue.add(objReq);
    } catch (JSONException e) {
        Log.e( tag: "JSONException",  msg: "Invalid JSON Object");
        e.printStackTrace();
    }
}
```
Figura 54. Clase VzxMicroservice.java.

6.1.3. Presentador del proyecto Android

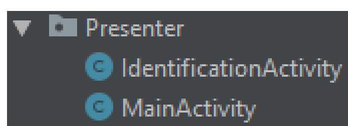


Figura 55. Presentador del proyecto Android (MVP).

En este último subapartado nos queda comentar las clases correspondientes al dominio del presentador (concepto del patrón MVP). Estas clases son “MainActivity.java” y “IdentificationActivity.java”. En la primera se reciben las ordenes efectuadas por el usuario interactuando mediante la vista de la pantalla principal. Esta clase se encarga de iniciar y recoger los resultados del proceso de tomar una captura con la cámara del dispositivo y de iniciar la clase de “IdentificationActivity.java” una vez tomada la captura. Esta segunda clase, se encarga de recibir las órdenes del usuario mediante la vista de la pantalla de selección de grupo. Cuando se da esta orden y se obtiene el grupo preferido, desde aquí se llama a los servicios de la *Face* API y al microservicio obteniendo la información de una persona y actualizando la vista para mostrar esta información por pantalla.

A continuación, vamos a ver cada una de estas clases en detalle.

```java
//views
private IMainView mainView;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    mainView = new MainView( mainActivity: this);
}
```

Figura 56. Construcción de la vista en MainActivity.java.

Cuando se abre la aplicación, la primera pantalla a la que se accede es a la pantalla principal. La vista de esta pantalla se inicializa desde la clase "MainActivity.java" construyendo una instancia de la vista lógica y asociando sus componentes XML. A partir de ese momento se puede manipular la vista mediante esta instancia.

```java
//Action Menu Click event
public void identify(MenuItem item) {

    takePhoto(IDENTIFY_CODE);
}

private void takePhoto(int requestCode){
    Intent takeSnapIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
    if (takeSnapIntent.resolveActivity(getPackageManager()) != null) {
        File photoFile = null;  // The file where the captured image will go
        try {
            photoFile = Utils.createImageFile();
        } catch (IOException e) {
            e.printStackTrace();
        }

        if (photoFile != null) {
            mPhotoPath = photoFile.getAbsolutePath();   // Save the path for future use
            mPhotoURI = FileProvider
                    .getUriForFile( context: this,  authority: "com.app.admin.vuzixblade.fileprovider", photoFile);
            takeSnapIntent.putExtra(MediaStore.EXTRA_OUTPUT, mPhotoURI);
            startActivityForResult(takeSnapIntent, requestCode);

            Log.i( tag: "IMAGE_CAPTURE",  msg: "Photo taken");
        }
    }
}
```

Figura 57. Métodos de tomar captura en MainActivity.java

Cuando el usuario selecciona la opción de identificar del menú de opciones se abre la pantalla de tomar una captura con la cámara del dispositivo. En el fragmento de código de arriba se puede apreciar el método "identify", que se ejecuta al seleccionar dicha opción del menú. Este método a su vez llama al método "takePhoto" que se encarga de crear un archivo en el sistema de archivos para almacenar la captura e inicia la actividad de hacer la captura. La funcionalidad y las vistas respecto a las pantallas de tomar la captura están disponibles por defecto en Android y no han sido implementadas, solo utilizadas. Por este motivo no mostraremos su implementación.

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode == IDENTIFY_CODE && resultCode == RESULT_OK) {
        Utils.notifyGalleryAboutPic( context: MainActivity.this, mPhotoPath);
        startIdentification();
    }
}

private void startIdentification() {
    Intent identificationIntent = new Intent( packageContext: this, IdentificationActivity.class);
    identificationIntent.putExtra( name: "imageUri", mPhotoURI.toString());
    startActivity(identificationIntent);
}
```

Figura 58. Proceso post-captura.

Después de hacer la captura se inicia la actividad de identificar, esto es inicializando la clase "IdentificationActivity.java". Esta clase recibe la ubicación de la imagen capturada a utilizar en el proceso de identificación.

```
public class IdentificationActivity extends ActionMenuActivity {

    // services
    private IVzxMicroservice vzxMicroservice;
    private IAzureCognitiveService azureCognitiveService;

    //views
    private IIdentificationView identificationView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_identification);
        identificationView = new IdentificationView( identificationActivity: this);
        vzxMicroservice = new VzxMicroservice( identificationActivity: this);
        azureCognitiveService = new AzureCognitiveService( identificationActivity: this);
    }
```

Figura 59. Inicio de instancias de vista y servicios de IdentificationActivity.java

Al iniciarse la clase "IdentificationActivity.java", como en la clase anterior, se crea la instancia de la vista asociada. A diferencia de la clase anterior, desde esta clase se llama a nuestro repositorio remoto detallado anteriormente para utilizar los servicios de la *Face* API y el microservicio VZX. Por este motivo se crean las instancias de los dos servicios.

Por otro lado, nótese también que esta clase extiende de la clase "ActionMenuActivity.java" al igual que la clase anterior. Esta clase padre no es propia de las bibliotecas básicas de Android sino que forma parte de las bibliotecas de Vuzix, utilizadas para poder desarrollar aplicaciones en el dispositivo *Blade*.

```java
public void searchInFemalePersonGroup(MenuItem item) {
    azureCognitiveService.setmPersonGroupId(FEMALE_GROUP);
    startIdentification();
}

public void searchInMalePersonGroup(MenuItem item) {
    azureCognitiveService.setmPersonGroupId(MALE_GROUP);
    startIdentification();
}

private void startIdentification(){
    hideComponentsShowProgressBar();
    Uri imageUri = Uri.parse(getIntent().getStringExtra( name: "imageUri"));
    Log.i( tag: "PHOTO URI",  msg: "photo uri " + imageUri);
    Bitmap mBitmap = null;
    try {
        mBitmap = MediaStore.Images.Media.getBitmap(this.getContentResolver(), imageUri);
    } catch (IOException e) {
        e.printStackTrace();
    }

    if(mBitmap != null){
        azureCognitiveService.detectAndIdentify(mBitmap);
    }else {
        Log.i( tag: "PHOTO",  msg: "Image not loaded");
    }
}
```

Figura 60. Opciones del menú de seleccionar grupo de búsqueda.

Después de inicializar la vista se muestra la pantalla de seleccionar el grupo de búsqueda desde el menú de opciones. En la figura 60 se pueden apreciar los dos métodos correspondientes a los dos botones del menú. Cualquiera de estos dos métodos llama al método “startIdentification”, que oculta los componentes visuales de la pantalla para mostrar el componente de la barra de progreso seguidamente de recuperar la imagen del sistema de archivos, capturada mediante la pantalla anterior, e iniciar el proceso de detección e identificación llamando al servicio de la *Face* API.

```java
public void IdentifyPostExecution(String personId){
    if(personId != null) {
        vzxMicroservice.getPersonById(personId);
    }else{
        showToast( text: "identification result is not found");
    }
}

public void showPersonData(PersonResponseDto personData) {
    identificationView.getProgressBar().setVisibility(View.GONE);
    invalidateOptionsMenu();
    identificationView.getMaleGroupMenuItem().setEnabled(false);
    PersonDataParser parser = new PersonDataParser();
    Map map = parser.parsePersonData( activity: this,personData);
    PersonDataAdapter personDataAdapter = new PersonDataAdapter(map);
    identificationView.getIdentificationResultListView().setAdapter(personDataAdapter);
}
```

Figura 61. Proceso post identificación.

Por último, cuando el proceso de detección e identificación termina, se utiliza el microservicio para recuperar la información de la persona identificada de la imagen. El proceso termina con el método “showPersonData” que básicamente oculta la barra de progreso y muestra los datos de la persona mediante el componente en forma de lista de la vista.

6.2. Implementación del proyecto .Net Core

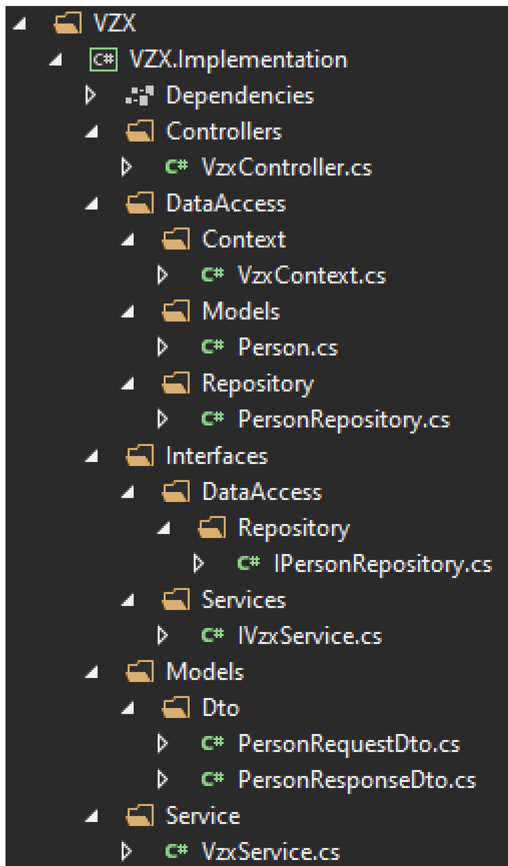


Figura 62. Estructura de la implementación del microservicio VZX.

Recordando la estructura del proyecto .Net Core, podemos detallar la implementación del microservicio VZX. Este microservicio se puede separar en tres partes diferentes. Siguiendo el flujo de trabajo de este, la parte donde se definen los puntos de acceso de la API web se encuentra en el directorio de "Controllers". Cuando se accede a estos puntos, desde allí, se llama a los servicios ubicados en el directorio "Service", que alberga la lógica del microservicio. Por último está el directorio de "DataAccess", que contiene la parte de acceso a recursos utilizada por la la parte de la lógica. Por tanto, empezaremos explicando la clase del directorio "Controllers", después seguiremos por la clase del directorio de "Service" y, finalmente, acabaremos contemplando el directorio de "DataAccess".

6.2.1. Puntos de acceso de la API web

En la figura 63 se muestra el fragmento de código fuente correspondiente al punto de acceso (llamado *endpoint*, en inglés) a la API web que permite la utilización del microservicio. Este código se ejecuta cuando desde la aplicación Android se manda la petición mediante la biblioteca de Volley. Podemos observar que el *endpoint* produce tres posibles códigos de respuesta HTTP. Un código de respuesta 200 significa que la petición se ha resuelto correctamente. Un código de respuesta 400 significa que la petición recibida del cliente, en este caso nuestra aplicación Android, no está bien formada y, por tanto, no es válida. Un código 500 significa que se ha producido un error inesperado a la hora de procesar la petición. Este tipo de errores suelen estar relacionados con problemas respecto a la conexión a algún recurso externo, estando el servidor que lo proporciona caído o fuera de servicio por razones que no están al alcance del conocimiento del cliente que hace la petición.

Desde aquí se llama a la parte del servicio, la cual veremos a continuación.

```
[HttpPost]
[Route("/[redacted]")]
[ProducesResponseType(typeof(PersonResponseDto), StatusCodes.Status200OK)]
[ProducesResponseType(400)]
[ProducesResponseType(StatusCodes.Status500InternalServerError)]
public async Task<IActionResult> GetPersonInfo([FromBody]PersonRequestDto personInfo)
{
    try
    {
        if (!ModelState.IsValid)
        {
            // bad request
            return StatusCode(StatusCodes.Status400BadRequest);
        }
        var result = await _vzxService.GetPersonInfo(personInfo.PersonId).ConfigureAwait(false);
        return Ok(result);
    }
    catch (Exception ex)
    {
        _logger.LogError($"{ex.Message} : {ex.StackTrace}");
    }

    return StatusCode(StatusCodes.Status500InternalServerError);
}
```

Figura 63. Clase VzxController.cs

6.2.2. Lógica del microservicio VZX

```
public async Task<PersonResponseDto> GetPersonInfo(string personId)
{
    // get person info
    Person person = await PersonRepository.GetFirstOrDefault(x => x.Id == personId).ConfigureAwait(false);
    return BuildPersonDto(person);
}

private PersonResponseDto BuildPersonDto(Person person)
{
    if (person == null) return null;
    return new PersonResponseDto
    {
        Apellidos = person.Apellidos,
        Cama = person.Cama,
        Codigo = person.Codigo,
        Dependencia = person.Dependencia,
        EstadoCivil = person.EstadoCivil,
        FechaIngreso = person.FechaIngreso,
        FechaNacimiento = person.FechaNacimiento,
        Habitacion = person.Habitacion,
        LugarNacimiento = person.LugarNacimiento,
        Nie = person.Nie,
        Nif = person.Nif,
        Nombre = person.Nombre,
        NumSip = person.NumSip,
        NumSS = person.NumSs,
        Tipologia = person.Tipologia
    };
}
```

Figura 64. Clase VzxService.cs

Al llamar la parte del servicio, en la figura 64, podemos ver que este se encarga de consultar al repositorio para obtener la información de la persona asociada a un identificador y devolverla. Nótese que el repositorio devuelve un objeto entidad con la información. Este objeto entidad refleja la información tal cual está almacenada en la tabla de la base de datos y, en ocasiones, no es necesario devolver toda la información de la tabla. Ahí es donde los objetos tipo DTO son la solución. En nuestro caso no queremos devolver toda la información de la tabla relativa a la persona y, por esto, se construye un DTO a devolver. Entre los beneficios que ofrece utilizar DTOs están la reducción del tamaño del mensaje de la respuesta devuelta y tener la posibilidad de no enviar datos sensibles.

6.2.3. Acceso a recursos en el microservicio VZX

```
public class PersonRepository : Repository<Person>, IPersonRepository
{
    public PersonRepository(VzxContext context) : base(context)
    {
    }
}
```

Figura 65. Clase PersonRepository.cs

La clase "PersonRepository.cs" es utilizada desde la parte del servicio para recuperar datos de una persona. Si nos fijamos en la figura del fragmento de código correspondiente, vemos que está compuesta solo por un constructor. Esto es porque esta clase actúa como una capa por encima del repositorio general. Normalmente, para cada clase entidad se crea un repositorio dedicado a operar con objetos de ese tipo. Las operaciones implementadas en estos repositorios específicos suelen ser de más complejidad que cualquier operación CRUD[12] , dejando esta definición de operaciones básicas en el repositorio general. La clase "PersonRepository.cs" hereda de la clase que comprende el repositorio general, "Repository.cs".

```
public Task<T> GetFirstOrDefault(Expression<Func<T, bool>> predicate = null)
{
    return GetQueryFromPredicate(predicate).FirstOrDefaultAsync();
}
```

Figura 66. Operación tipo CRUD en la calse Repository.cs

El método utilizado desde el servicio, mostrado en la figura 66, consiste en una operación básica de lectura y está definido en "Repository.cs".

[12]CRUD: En informática, CRUD es el acrónimo de "Crear, Leer, Actualizar y Borrar" (del original en inglés: Create, Read, Update and Delete), que se usa para referirse a las funciones básicas en bases de datos o la capa de persistencia en un *software*.

6.3. Entrenamiento del modelo de aprendizaje para *Face API*

En el apartado de diseño comentamos la estructura que tiene el modelo de aprendizaje de la *Face API*. En este subapartado se explica el proceso de entrenar el modelo y dejarlo listo para resolver los procesos de identificación a petición de la aplicación.

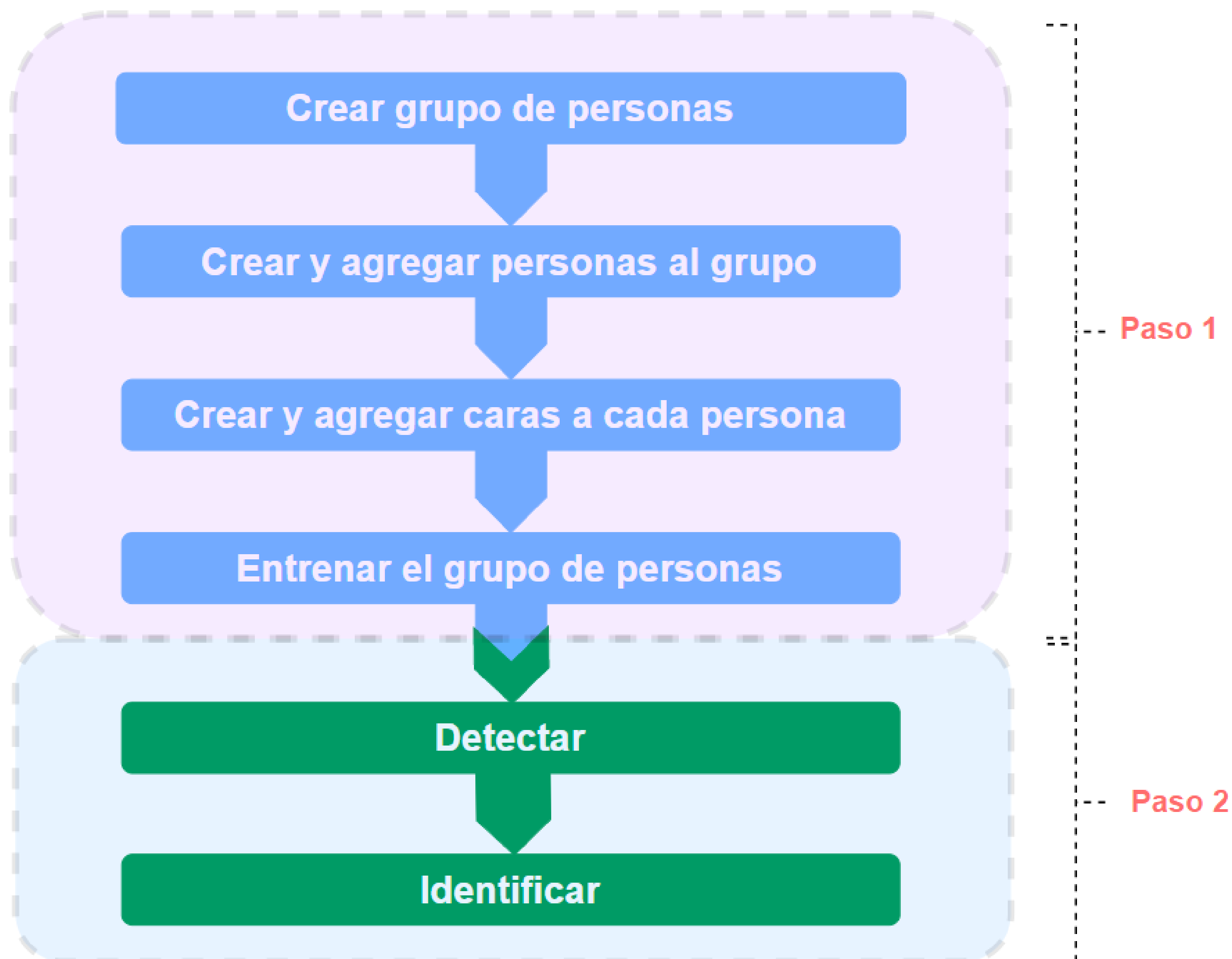


Figura 67.Flujo completo de identificación.

Hasta este momento, en apartados anteriores, se ha mencionado el proceso de identificación sin especificar el preproceso necesario para llevarlo a cabo. En la figura 67 se muestra el flujo completo del funcionamiento del servicio de identificación proporcionado por la *Face* API. El paso 2 del diagrama es la funcionalidad que utilizamos desde la aplicación Android y, el paso 1, es el preproceso necesario para que funcione.

A continuación, se demuestra cómo se ha procedido a entrenar nuestro modelo de aprendizaje mediante Postman para cumplir con este preproceso necesario (el paso 2 del diagrama se explicó en el apartado de diseño de la misma forma en la que se hará a continuación).

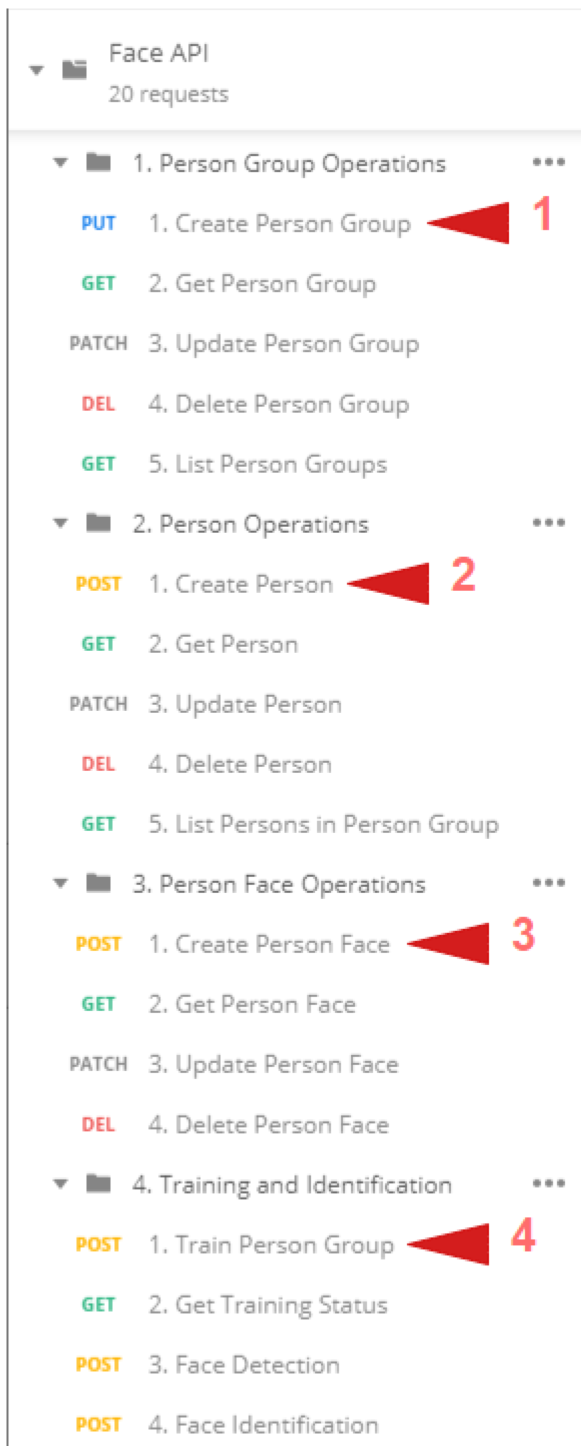


Figura 68. Colección de llamadas REST de Postman.

Como se ha comentado en apartados anteriores, Postman es una herramienta que da la posibilidad de realizar peticiones de tipo REST mediante su interfaz gráfica. También permite crear colecciones de peticiones REST, importarlas y exportarlas para su uso en diferentes entornos.

El modelo de aprendizaje a entrenar se hace mediante llamadas REST a la *Face* API, al igual que los procesos de detección e identificación.

En la figura 68 se muestra la colección de llamadas REST de Postman para entrenar este modelo. Detallaremos el flujo básico necesario para disponer del modelo listo para llevar a cabo las fases de detección e identificación. Este flujo comprende los cuatro pasos señalados en la imagen.

Primero (1) crearemos un grupo de personas. Acto seguido, crearemos una persona (2) perteneciente al grupo creado. La persona se asocia a un grupo en el momento de la creación de esta. En tercer lugar, crearemos una cara (3) para la persona que acabamos de crear. La cara se asocia a la persona en el momento de la creación. Por último, entrenaremos el grupo de personas que acabamos de crear.

A continuación, detallaremos las llamadas REST efectuadas para completar el flujo que se acaba de describir gracias a la interfaz gráfica de Postman.

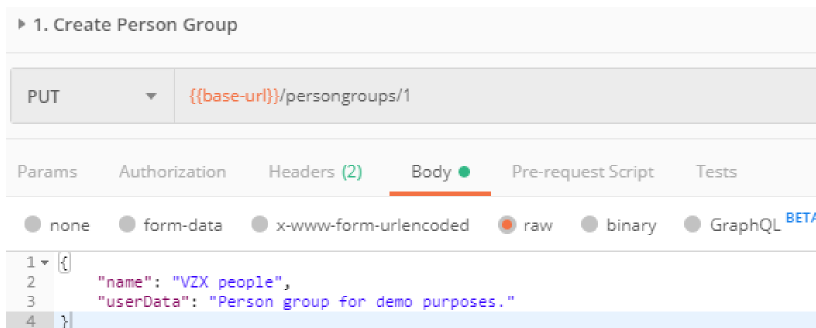


Figura 69. Llamada REST para crear grupo de personas.

Esta es la llamada a la *Face* API correspondiente a crear un grupo de personas. Podemos ver que en la dirección del enlace, *URL* desde ahora, se especifica el identificador del grupo, en este caso, 1. A parte de este, también se especifica un nombre y una descripción para el grupo.

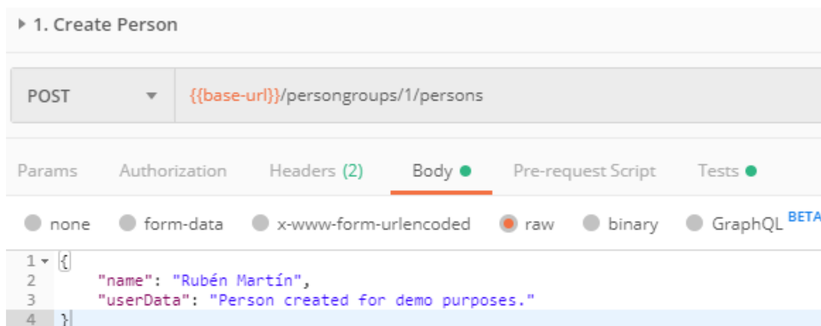


Figura 70. Llamada REST para crear una persona.

El siguiente paso es el de crear una persona asociada al grupo que se acaba de crear. En la *URL* se especifica el identificador del grupo en el que se quiere agregar. También se especifica el nombre de la persona y una descripción asociada a ella.

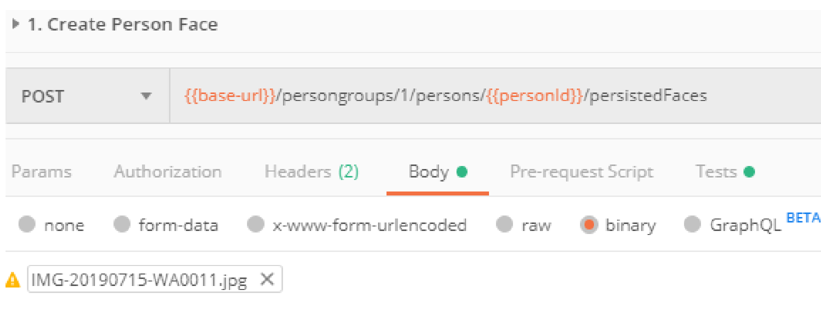


Figura 71. Llamada REST para crear una cara.

Una vez creamos a la persona, tenemos que asociar una cara a esta. Para crear la cara, en el cuerpo de la petición se adjunta una imagen. En la *URL* de la petición se especifica el identificador del grupo y el identificador de la persona a la que se asocia la cara. Este último identificador se obtiene como respuesta de la petición anterior.

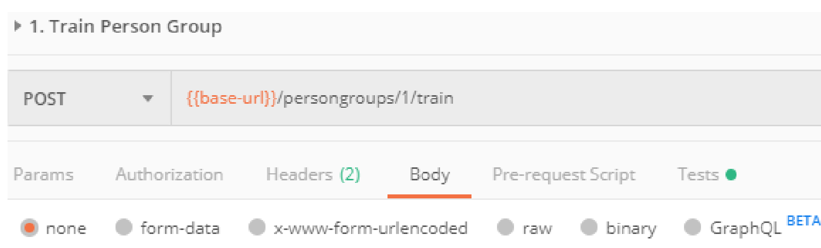


Figura 72. Llamada REST para entrenar un grupo.

Por último, se lleva a cabo la petición de entrenar al modelo con el grupo creado. Esta petición no requiere contenido en su cuerpo, solo requiere indicar el identificador del grupo objeto del entrenamiento en la *URL*.

7. Pruebas

En este apartado prestaremos atención al cumplimiento de algunos requisitos no funcionales. Concretamente a los que implican el uso de la aplicación por el usuario. Por otro lado más técnico, se presentará la implementación de pruebas unitaras que sirven para comprobar el funcionamiento de un trozo de código fuente relacionado a una funcionalidad.

7.1. Pruebas unitarias en el proyecto .Net Core

Las pruebas unitarias en este proyecto consistirán en probar la correcta funcionalidad de la lógica del microservicio. En la parte de la lógica, se accede a servicios externos para obtener recursos. Esto es posible accediendo a los repositorios implementados. Como las pruebas unitarias consisten en probar la funcionalidad solo de la lógica, el acceso a estos repositorios queda fuera del alcance de pruebas. Para poder de alguna manera saltarnos o simular la parte de acceso a los repositorios en nuestras pruebas unitarias se utilizan los *mocks*. Estos son objetos que imitan el comportamiento de objetos reales de una forma controlada.

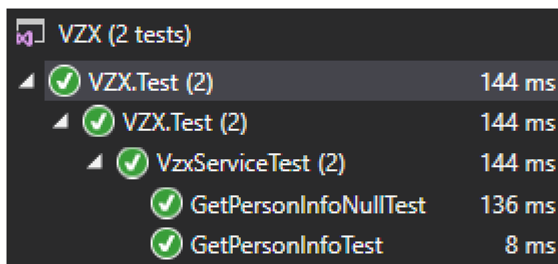


Figura 73. Pruebas unitarias del microservicio.

```
[Fact]
public async void GetPersonInfoNullTest()
{
    personRepository.Setup(p => p.GetFirstOrDefault(It.IsAny<Expression<Func<Person, bool>>>()))
        .Returns(Task.FromResult<Person>(null));

    PersonResponseDto responseDto = await vzxService.
        GetPersonInfo("ee6ac1bd-65fa-4dda-9777-5bdd0e44faae").ConfigureAwait(false);

    Assert.True(responseDto == null);
}
```

Figura 74. Prueba unitaria de objeto nulo.

En esta primera prueba se comprueba que el servicio devuelve un objeto nulo cuando el repositorio devuelve un objeto nulo al no encontrar a la persona del identificador proporcionado. Primero se hace el *mock* a la operación básica del repositorio haciendo que devuelva un objeto nulo y después se llama al servicio. Por último se comprueba que el servicio también devuelve un objeto nulo.

```
[Fact]
public async void GetPersonInfoTest()
{
    Person person = new Person()
    {
        Codigo = "00001",
        Nombre = "NombreDemo",
        Apellidos = "ApellidosDemo",
        Nif = "012334567A",
        FechaNacimiento = "1980-09-08",
        LugarNacimiento = "LugarDemo",
        EstadoCivil = "EstadoCivilDemo",
        Habitacion = 9,
        Cama = 9,
        Dependencia = 1,
        FechaIngreso = "2019-09-09",
        NumSip = 123456789,
        NumSs = 321321321,
        Sexo = "SexoDemo",
        Tipologia = "TopologiaDemo"
    };
    personRepository.Setup(p => p.
    GetFirstOrDefault(It.IsAny<Expression<Func<Person, bool>>>()))
        .ReturnsAsync(person);

    PersonResponseDto responseDto = await vzxService.
        GetPersonInfo("ee6ac1bd-65fa-4dda-9777-5bdd0e44faae")
        .ConfigureAwait(false);

    Assert.True(personResponseDto.Equals(responseDto));
}
```

Figura 75. Prueba unitaria de funcionamiento base.

```
personResponseDto = new PersonResponseDto()
{
    Codigo = "00001",
    Nombre = "NombreDemo",
    Apellidos = "ApellidosDemo",
    Nif = "012334567A",
    FechaNacimiento = "1980-09-08",
    LugarNacimiento = "LugarDemo",
    EstadoCivil = "EstadoCivilDemo",
    Habitacion = 9,
    Cama = 9,
    Dependencia = 1,
    FechaIngreso = "2019-09-09",
    NumSip = 123456789,
    Tipologia = "TopologiaDemo",
    NumSS = 321321321,
};
```

Figura 76. Objeto esperado del servicio.

En esta segunda prueba unitaria se comprueba que el servicio devuelve el objeto correspondiente al recuperar del repositorio un objeto existente. Para ello, se construye un objeto del tipo que devuelve el repositorio y se hace el *mock* de la llamada de este, usada por el servicio, para que devuelva el objeto que se acaba de construir. Seguidamente, se llama al servicio y se comprueba que el objeto devuelto es el esperado (figura 76).

7.2. Pruebas de usuario

En este apartado vamos a mostrar mediante tablas los resultados de los requisitos no funcionales relativos al tiempo que tarda el usuario en utilizar la aplicación para hacer un flujo básico.

La dificultad principal a la que se han enfrentado los usuarios, sujetos a estas pruebas y que no habían utilizado el dispositivo, ha sido el uso de este. Esto se refleja gracias a los tiempos medidos comenzando desde que un usuario se pone las gafas hasta que encuentra la aplicación desarrollada objeto de este proyecto.

En esta primera tabla vemos los resultados de usuarios que utilizan el dispositivo por primera vez.

Sujeto	Tiempo en encontrar la aplicación en el dispositivo	Tiempo en hacer un flujo básico de la aplicación
Sujeto 1	1 minuto y 45 segundos	1 minuto
Sujeto 2	1 minuto y 16 segundos	40 segundos
Sujeto 3	45 segundos	50 segundos
Sujeto 4	2 minutos	1 minuto y 30 segundos
Media de tiempos	1 minuto y 26 segundos	1 minuto

Tabla 4. Tiempos de uso por primera vez.

En esta segunda tabla se muestran los resultados de usuarios que ya habían utilizado el dispositivo y la aplicación anteriormente.

Sujeto	Tiempo en encontrar la aplicación en el dispositivo	Tiempo en hacer un flujo básico de la aplicación
Sujeto 1	15 segundos	30 segundos
Sujeto 2	11 segundos	25 segundos
Sujeto 3	10 segundos	20 segundos
Sujeto 4	15 segundos	37 segundos
Media de tiempos	12.75 segundos	28 segundos

Tabla 5. Tiempos de uso por usuarios conocedores del dispositivo y la aplicación.

Observando las dos tablas, podemos notar la gran diferencia de tiempos entre usuarios que no habían utilizado el dispositivo y los que sí.

Otras adversidades significativas encontradas por los usuarios son la dificultad de leer el texto en los componentes visuales de texto y la dificultad de navegar a las funcionalidades tanto de la aplicación como en general del dispositivo.

8. Conclusión

El objetivo principal de esto proyecto desde el punto de vista comercial es cumplir con la prueba de concepto requerida por la empresa ADD Informática. Esto se refleja en el cumplimiento de los requisitos funcionales y no funcionales de esta primera versión de la aplicación.

Entre las dificultades que se han encontrado destacan las de enfrentarse a nuevas tecnologías. Hay que decir que no se ha encontrado toda la documentación relativa al desarrollo en dispositivos *Blade* necesaria para desarrollar ciertas funcionalidades adicionales y mejorar algunos aspectos de la aplicación. Un ejemplo de ello es el no poder hacer pruebas de instrumentación (para probar el flujo total de la aplicación Android) porque los componentes visuales proporcionados por las bibliotecas de Vuzix no son reconocibles por el *framework* de pruebas Espresso. Esto resta calidad a la aplicación como proyecto *software*.

Gracias a las dificultades de este tipo hemos aprendido que existen algunos límites a la hora de desarrollar aplicaciones con estas nuevas tecnologías y nos hemos enfrentado a retos que nos han motivado a seguir aprendiendo y desarrollando este proyecto.

8.1. Relación del trabajo desarrollado con los estudios cursados

Con relación a las asignaturas cursadas en el grado, tenemos que destacar algunas que nos han sido de gran ayuda, y de base, para poder llevar a cabo este proyecto. En primer lugar, la asignatura de la que más conocimientos hemos aprovechado ha sido Desarrollo de aplicaciones para dispositivos móviles, donde adquirimos los conocimientos para el desarrollo para dispositivos Android. El aprovechamiento de esta asignatura al completo no huebiera sido posible sin asignaturas como Programación, Estructuras de Datos y Algoritmos e Interfaces Persona Computador, ya que sirven como base para aprender ciertas competencias como por ejemplo el lenguaje de programación Java. Por otro lado, asignaturas como Ingeniería del Software nos han ayudado en la comprensión del ciclo de vida del *software*, el lenguaje C# y la utilización de Visual Studio.

8.2. Trabajo futuro

La problemática de la complejidad de navegación en aplicaciones de gafas de relidad aumentada como *Blade* expuesta en el apartado de objetivos supone un replanteamiento en cuanto a la forma que tiene un usuario de interactuar con estas aplicaciones. En la última reunón con la empresa ADD informática y vistas estas complejidades se concluyó que, para que la aplicación saliera a producción, como mínimo ha de proporcionar una navegación por comandos de voz incluyendo verificación del usuario a la hora de dar estos comandos. Esto simplificaría significativamente la navegación por la aplicación por el motivo de que en vez de tener un posible flujo en forma de árbol de esta navegación, como se explicaba en el apartado de objetivos, se accedería a las funciones de la aplicación de forma directa creando accesos directos a cada funcionalidad y evitando flujos complejos de navegación para llegar a la funcionalidad deseada.

Otros requisitos deseables por parte de la empresa son las funcionalidades descritas en el apartado 6.3, que explicaba cómo entrenar el modelo de aprendizaje. Se requiere que las operaciones previas al proceso de detección e identificación vistas en ese apartado puedan hacerse desde la aplicación Android, en vez de desde Postman. Por último, se requiere que la aplicación funcione sin necesitar conexión a internet permanente.

Como trabajo futuro, se pretende desarrollar una segunda versión de la aplicación incluyendo los requisitos funcionales que se acaban de describir. Con esta segunda versión, se concluirá si la aplicación es viable para ser utilizada como producto a comercializar por parte de ADD Informática.

9. Bibliografía

[1] «Realidad Aumentada» [En línea]. Disponible: http://realidadaumentada.info/realidad-aumentada/ . [Último acceso: 7 septiembre 2019].

[2] «Computación en la nube» [En línea]. Disponible: https://es.wikipedia.org/wiki/Computaci%C3%B3n_en_la_nube#Software_como_servicio . [Último acceso: 7 septiembre 2019].

[3] Microsoft, «What are Azure Cognitive Services?» [En línea]. Disponible: https://docs.microsoft.com/en-us/azure/cognitive-services/welcome . [Último acceso: 7 septiembre 2019].

[4] Nick Statt, «Vuzix Blade AR glasses» 9 enero 2018 [En línea]. Disponible: https://www.theverge.com/2018/1/9/16869174/vuzix-blade-ar-glasses-augmented-reality-amazon-alexa-ai-ces-2018 . [Último acceso: 7 septiembre 2019].

[5] «HoloLens 2» [En línea]. Disponible: https://en.wikipedia.org/wiki/HoloLens_2. [Último acceso: 7 septiembre 2019].

[6] «Google Glass» [En línea].Disponible: https://en.wikipedia.org/wiki/Google_Glass. [Último acceso: 7 septiembre 2019].

[7] Vuzix «Feature-Packed Vuzix Blade Smart Glasses» [En línea].Disponible: https://www.vuzix.com/products/blade-smart-glasses. [Último acceso: 7 septiembre 2019].

[8] Vuzix «Getting to Know the Vuzix Blade» [En línea].Disponible: https://www.vuzix.com/Developer/KnowledgeBase/Detail/63. [Último acceso: 7 septiembre 2019].

[9] Microseoft «HoloLens 2» [En línea].Disponible: https://www.microsoft.com/en-us/hololens/hardware . [Último acceso: 7 septiembre 2019].

[10] Google «Glass Enterprise Edition 2» [En línea].Disponible: https://www.google.com/glass/tech-specs/ . [Último acceso: 7 septiembre 2019].

[11] Greer Williams «21 mejores aplicaciones de reconocimiento facial para 2018» 29 agosto 2018 [En línea].Disponible: https://www.facefirst.com/blog/21-mejores-aplicaciones-de-reconocimiento-facial-para-2018/ . [Último acceso: 7 septiembre 2019].

[12] José Hidalgo «Llegan nuevas gafas con reconocimiento facial para IA para la vigilancia de espacios públicos» 10 junio 2019 [En línea].Disponible: https://wwwhatsnew.com/2019/06/10/llegan-nuevas-gafas-con-reconocimiento-facial-por-i-a-para-la-vigilancia-de-espacios-publicos/ . [Último acceso: 7 septiembre 2019].

[13] «Android Studio» [En línea].Disponible: https://es.wikipedia.org/wiki/Android_Studio . [Último acceso: 7 septiembre 2019].

[14] «Microsoft Visual Studio» [En línea].Disponible: https://en.wikipedia.org/wiki/Microsoft_Visual_Studio . [Último acceso: 7 septiembre 2019].

[15] «Microsoft Azure» [En línea].Disponible: https://en.wikipedia.org/wiki/Microsoft_Azure . [Último acceso: 7 septiembre 2019].

[16] «SQLite» [En línea].Disponible: https://en.wikipedia.org/wiki/SQLite. [Último acceso: 7 septiembre 2019].

[17] Mike Yockey «API Testing with Postman» 20 de abril de 2015 [En línea].Disponible: https://seesparkbox.com/foundry/api_testing_with_postman . [Último acceso: 7 septiembre 2019].

[18] «GitLab» [En línea].Disponible: https://es.wikipedia.org/wiki/GitLab. [Último acceso: 7 septiembre 2019].

[19] «Git» [En línea].Disponible: https://es.wikipedia.org/wiki/GitLab. [Último acceso: 7 septiembre 2019].

[20] «.Net Core» [En línea].Disponible: https://en.wikipedia.org/wiki/.NET_Core. [Último acceso: 7 septiembre 2019].

[21] «Arquitectura de software» [En línea].Disponible: https://es.wikipedia.org/wiki/Arquitectura_de_software . [Último acceso: 7 septiembre 2019].

[22] «Model-view-presenter» [En línea].Disponible: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93presenter . [Último acceso: 7 septiembre 2019].

[23] «Arquitectura de microservicios» [En línea].Disponible: https://es.wikipedia.org/wiki/Arquitectura_de_microservicios. [Último acceso: 7 septiembre 2019].

[24] «Aprendizaje supervisado» [En línea].Disponible: https://es.wikipedia.org/wiki/Aprendizaje_supervisado . [Último acceso: 7 septiembre 2019].