Esquema estructura:

-Indice

-Indice de ilustraciones

-Resumen (explicación de que hemos desarrollado)

-Introduccion (reflexión de porque creemos necesario esta aplicación, tecnologías utilizadas etc)

-Objetivos (Objetivos del proyecto, objetivos docentes (decir que buscamos plasmar el mayor numero de conocimientos adquiridos en el grado + practicas))

-Tecnologias usadas (explicación de las tecnologías usadas (angular, Android, spring, azure, twiliio etc)

-Explicacion servidores http (por ejemplo, de manera introductoria)

-Explicacion en detalle de cada tenologia (Ej. decir que usamos Android y por que usamos el sistema operativo X para el desarrollo, porque usamos apache HTTP, porque Azure etc)

-Explicacion de la bbdd

-Explicacion en detalle del funcionamiento del aplicativo web / móvil

-Explicar en que puntos implementamos cada tecnología y breve recuerdo de en que consiste el uso de cada una.

-Definicion de requerimientos funcionales (pasos que hace el usuario desdem que abre la app hasta que la cierre (en orden)) (métodos que tenemos creados para el aplicativo)

-Definicion de requerimientos no funcionales (hardware, requisitos, software, escalabilidad…).

-¿Usuarios y roles?(no es lo mismo un jefe que un empleado por ejemplo, o nos basamos en que si?)

-Diseño del sistema

-Diseño de la base de datos (datos -> datos que vamos a utilizar tipo, nominas, facturas, fichajes, imágenes, nombres…, MODELO E-R de la bbdd (¿Tablas explicando cada tabla con su respectiva primary key y tipo de campos?, ¿diagrama de casos de uso?)

-Diseño del aplicativo móvil

-Diseño de la interfaz/interfaces visuales de vista al usuario

-Diseño de como funcionan las vistas (explicando el paso entre pantallas y que va haciendo cada una brevemente)

-Meter aquí imágenes exactas de la vista del alplicativo web y móvil, hasta ahora solo hacerlo mediante dibujos.

-Batería de pruebas

-Explicacion por ej de por que usamos el porcentaje tanto para verificar la identidad del usuario, mostrando una tabla con diferentes porcentajes de acierto y error, con las mismas y distintas implementaciones de las pruebas.

-Conclusiones: que hemos sacado de provecho y que hemos mostrado y que buscamos sacar de ello.

-Bibliografia

-Manual de usuario -> enseñar a un tercero como instalar cada aplciativo para poder levantar las aplicaciones desde su propio ordenador.

-¿Estudio económico? – demostrar con dinero hasta que punto hay que pagar/ es gratuito nuestro aplicativo haciendo un estudio económico sobre cuanto nos costará a largo plazo si tuviera éxito.

**Resumen**

Nuestro proyecto consiste en dos aplicativos, uno creado para Android y otro creado para plataformas web, pensado para empresas en la que los empleados tenga acceso desde su móvil a ver varios datos relacionados con su trabajo, así como les permita fichar mediante reconocimiento facial y geolocalización para poder implementar así varias de las tecnologías actuales que se están llevando a cabo como son los servicios cognitivos.

Por parte del aplicativo web, queremos llevar un registro cómodo e intuitivo para los empleados de Recursos Humanos, o para jefes que quieran tener constancia de sus empleados, así como modificar datos suyos en caso de que sean necesarios, donde puedan visualizar si los fichajes de los empleados son correctos, sus nóminas, etcétera.

**Introducción**

Dada la situación sanitaria a nivel mundial que estamos viviendo, así como el avance de las tecnologías, creemos que el uso de estos servicios tales como para fichar desde el propio dispositivo móvil de cada empleados, pueden ayudar un poco en las empresas a que, los trabajadores entren en contacto con un menor número de objetos físicos (como puede ser un terminal donde tengan que fichar, externo al su ordenador o su móvil) evitando así contactos de más con las instalaciones de la empresa.

Consideramos también que es una manera más “llamativa” y cómoda para los empleados de poder realizar sus fichajes, así como para ver sus datos personales dentro de un aplicativo seguro y eficaz donde quedan registrados sus datos.

**Objetivos**

Nuestro objetivo principal en este proyecto es plasmar los conocimientos adquiridos en estos años del grado, utilizando así los conocimientos adquiridos en desarrollo web y en Android, así como el uso de bases de datos.

También hemos querido plasmar las tecnologías utilizadas durante nuestras prácticas, ya que, a parte de ser muy útiles, consideramos también importante plasmar que este conocimiento que nos han permitido adquirir ha sido de provecho y podemos sacarle utilidad de cara a futuros proyectos.

**TECNOLOGIAS USADAS**

**Android**

**Angular**

**PosgresSql**

**API’s (Http)**

**AZURE**

¿Qué es y para qué sirve?

Azure es un servicio en nube en el que un usuario paga por su uso, es decir, dependiendo de la cantidad de recursos que consumas, pagaras más, o podrá llegar a ser gratuito.

Ofrece una gran cantidad de servicios, desde máquinas virtuales en nube de Linux y Windows con gran capacidad, como bases de datos securizadas y centralizadas. Nos otorga también un almacenamiento en nube de gran capacidad, diferentes recursos de inteligencia artificial y aprendizaje automático (en nuestro caso, usando servicios cognitivos), herramientas de análisis muy útiles en el ámbito empresarial y un largo etcétera que mostraremos a continuación mediante una serie de capturas del portal de Microsoft Azure.

Puede considerarse una herramienta de gran ayuda para la transformación digital en empresas, haciendo intuitivo y más veloz el paso de ciertos mecanismos a un formato digital actualizado.

Cabe destacar también la importancia que tiene Microsoft en el ámbito de la privacidad y seguridad de sus clientes, creando así una mayor confianza en el uso de sus recursos.

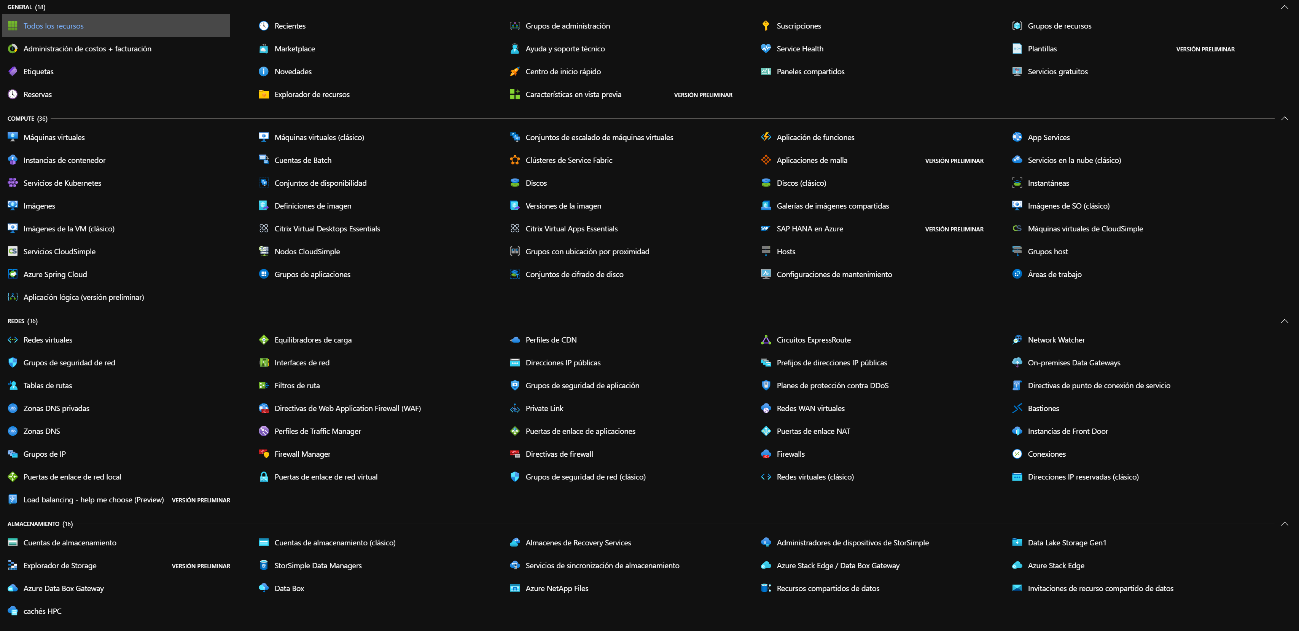


Ilustración 1 Servicios de Microsoft Azure

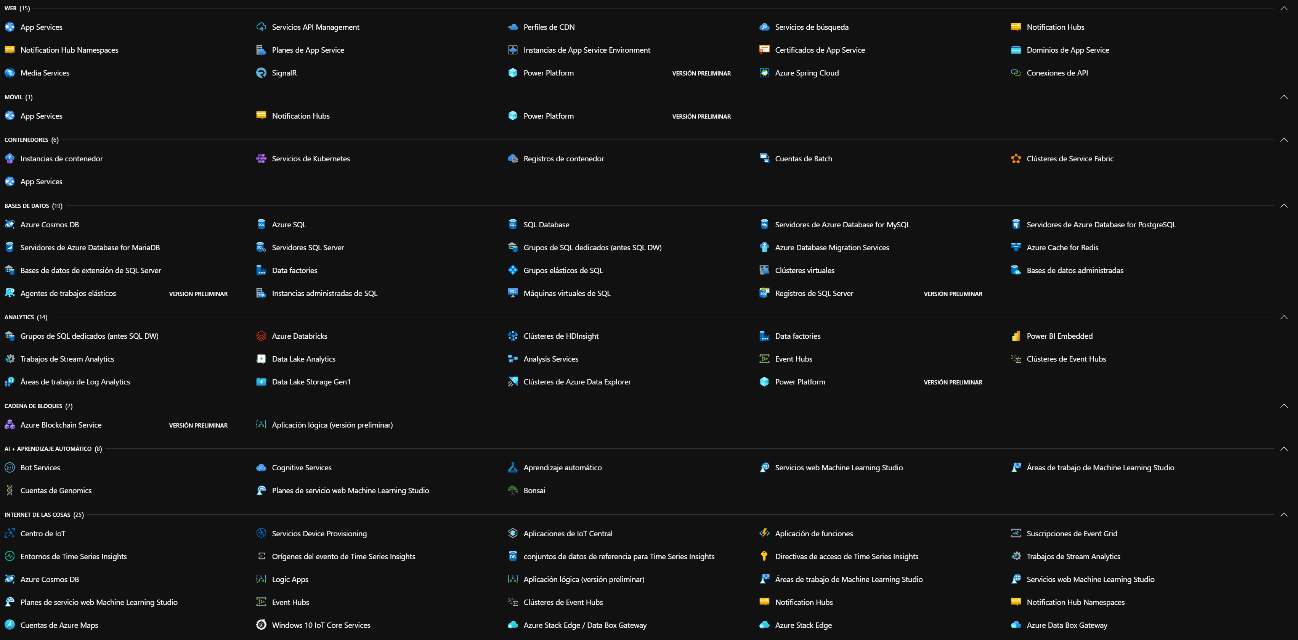


Ilustración 2 Servicios de Microsoft Azure

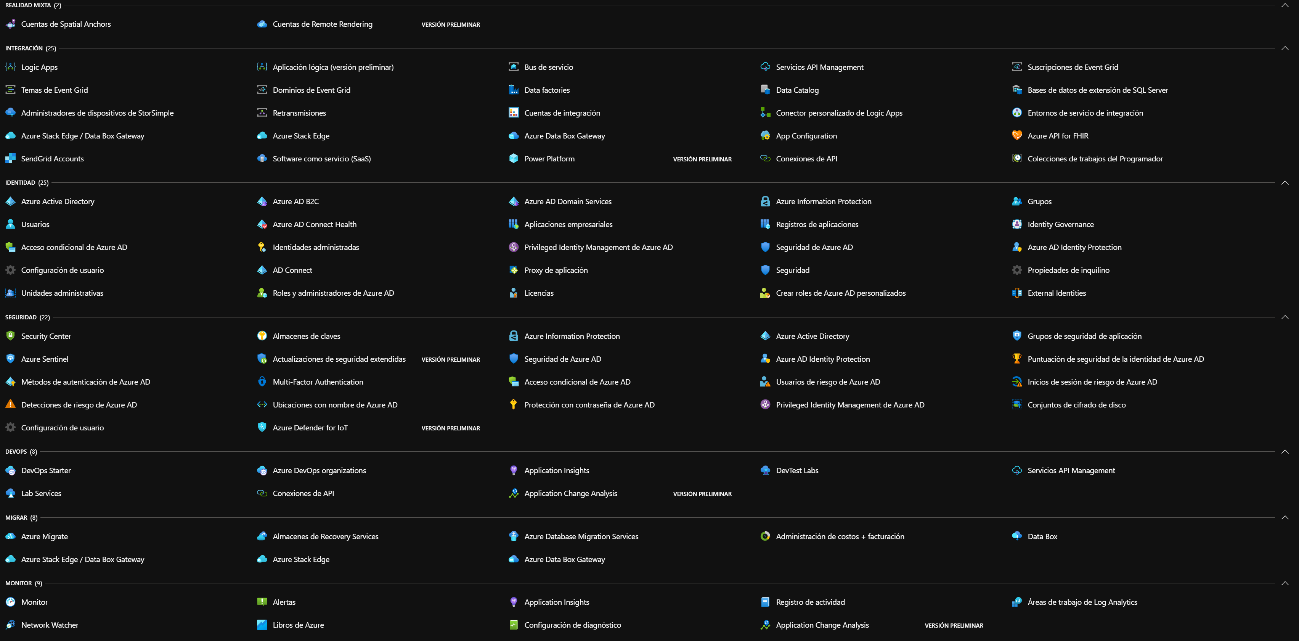


Ilustración 3 Servicios de Microsoft Azure

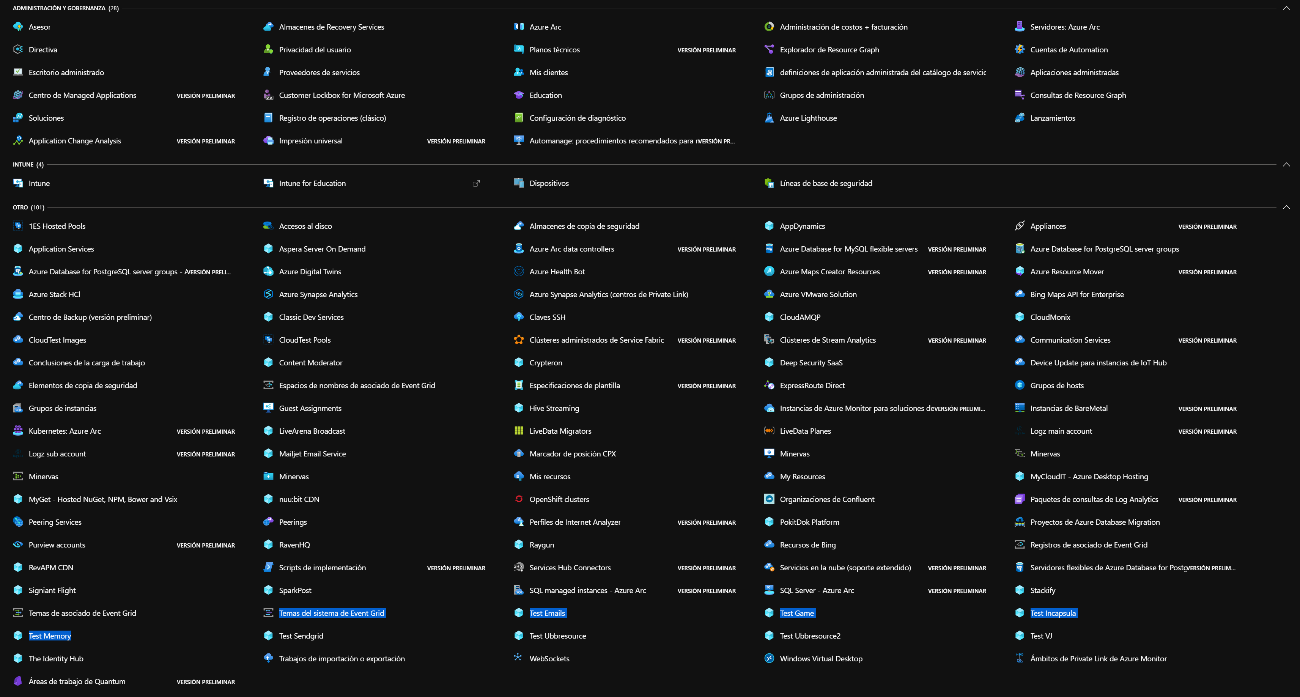


Ilustración 4 Servicios de Microsoft Azure

A continuación vamos a entrar más en detalle en los servicios usados para nuestro proyecto: los Servicios Cognitivos y las cuentas de almacenamiento.

SERVICIOS COGNITIVOS (<https://azure.microsoft.com/en-us/services/cognitive-services/> )

En nuestro caso utilizaremos los servicios cognitivos de Microsoft Azure para el reconocimiento facial de nuestros usuarios, no obstante, consideramos importante el comentar, aunque sea de manera fugaz, las posibilidades que estos servicios nos otorgan.

Uno de los procutos que Microsoft Azure nos facilita, es el uso de servicios cognitivos, lo que nos permite tener Inteligencia Artificial en nuestra aplicación sin requerir unos conocimientos muy extensos respecto a la AI, y en nuestro caso, sin ni siquiera necesitar conocimientos en *machine learning*. El único requisito que necesitamos, es el llamar a nuestra API para que podamos uarla.

Los servicios cognitivos que nos proporciona Microsoft Azure son:

-Decisiones: permite tomar decisiones de forma más rápida:

*Anomaly Detector:* identificar probables problemas de una forma más temprana.

*Content Moderator:*  identifica contenido ofensivo o no deseado.

*Metrics Advisor:* supervison de métricas y diagnosticar problemas (es una versión preliminar)

Permite también crear experiencias personalizadas para cada usuario.

-Lenguaje: consigue significados de texto mal estructurados:

*Inmersive Reader:*  ayuda a los lectores a comprender el texto mediante el uso de anotaciones por audio y visuales.

*Language Understanding:* desarrollar en un lenguaje natural y comprensible en aplicaciones, bots y servicios IoT.

*QnA Maker:* cree una conversación de preguntas y respuestas sobre sus datos

*Text Analytics:* detecta sentimientos, frases clave y entidades con nombre.

*Translator:* detecta y traduce mas de 60 lenguajes.

-Habla (*speech):* integra procesos de habla en aplicaciones y servicios:

*Speech to text:* transcribe un audio a texto legible.

*Text to Speech:* convierte texto a audio para interfaces naturales.

*Speech Translation:* integra traducciones en tiempo real en las aplicaciones.

*Speaker Recognition:* identifica y verifica a personas hablando mediante el audio.

-Vision: identifica y analiza contenido de imágenes y videos:

*Computer Vision:* analiza contenido de imágenes y videos.

*Custom Vision:* personaliza el reconocimiento de imágenes adaptable a las necesidades de negocio.

*Face:* detecta e identifica personas y emociones en imágenes.

*From Recognizer:* extrae texto y pares clave-valor de documentos.

*Video Indexer:* analiza tanto la aprte visual como de audio y nunte su contenido.

Vamos a pasar como funciona *Face*, que es el servicio de Vision que usamos nosotros para la detección y comparación de caras.

*Vision: Face*

<https://azure.microsoft.com/en-us/services/cognitive-services/face/#demo>

Desde este servicio, usamos dos de sus funciones:

-*Detection*: en esta primera llamada al servicio, nosotros enviamos una imagen al servidor de Azure, para que la analice y detecte una cara, devolviéndonos cierta información de esta (como pueden ser, posiciones de los ojos, nariz boca, si la persona lleva gafas, si tiene pelo o no, etcétera) así como un Id único para cada cara. Este Id, lo reconocerá Azure durante 24h hasta eliminarlo de su base de datos.

Para nuestro aplicativo, el único valor que necesitamos es el Id de la cara detectada en la imagen.

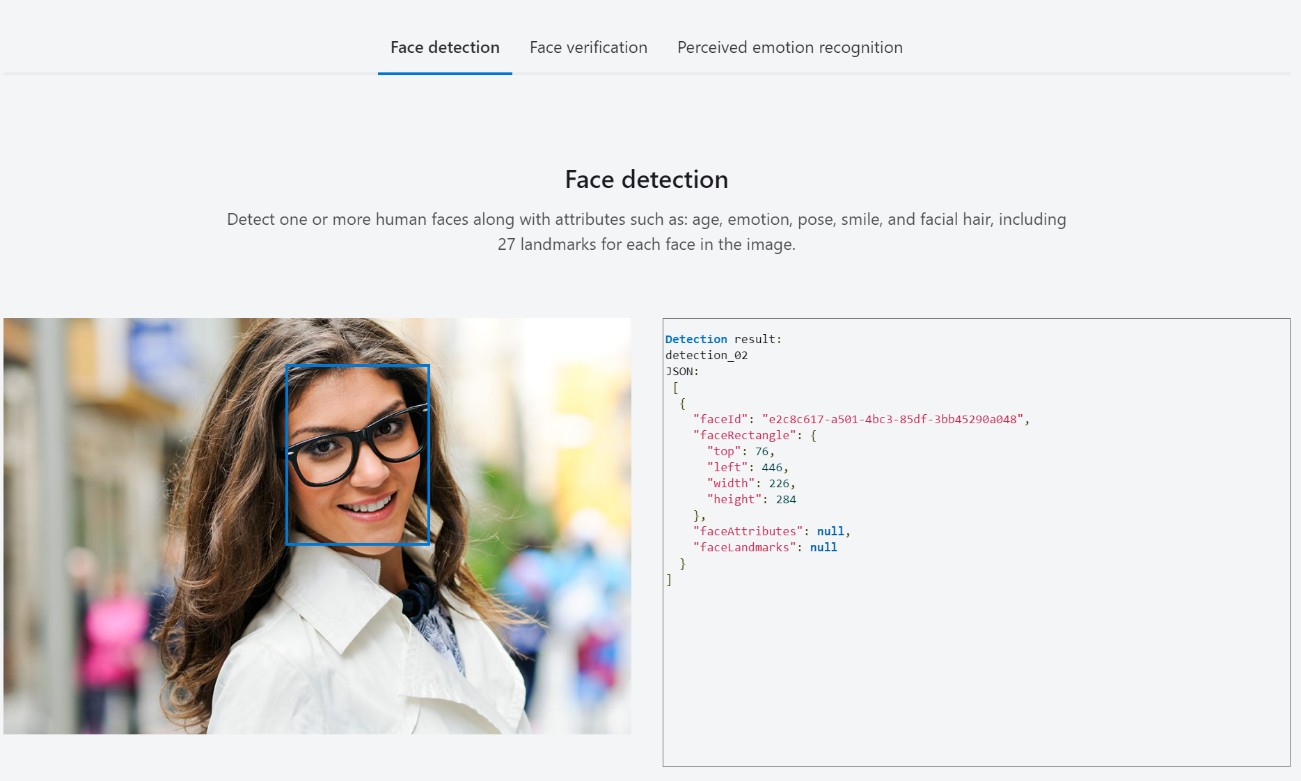


Ilustración 5 Ejemplo del servicio Face-Detection

-*Verification (* [*https://azure.microsoft.com/en-us/services/cognitive-services/face/#demo*](https://azure.microsoft.com/en-us/services/cognitive-services/face/#demo) *) :* a continuación, usando el servicio de verificación, lo que hacemos es leer dos, imágenes de las cuales obtenemos un Id para cada una, y enviando estos Id’s al servicio de Microsoft Azure *verification* para que se encarge de comparar ambas imágenes, devolviéndonos un valor comprendido entre 0 y 1 sobre su porcentaje de similitud que ha encontrado entre ambas imágenes.

Captura de pantalla de un celular con la imagen de un hombre

Descripción generada automáticamente

Ilustración 6 Ejemplo del servicio Face-Verification

*Nota: cabe destacar que lo comentado simplemente es a modo introductorio para entender como funcionan estos servicios, ya que para nuestro aplicativo, nosotros llamamos a los servicios mediante la API que nos proporciona Microsoft Azure, la cual entraremos más en detalle a la hora de explicar nuestro código, explicando correctamente como pasamos las imágenes para que las reconozca, así como explicaremos como llamamos a los servicios cognitivos de Azure y como tratamos sus valores de respuesta, dados en formato JSON (añadir el punto de donde lo explicaremos)*

BIBLIOTECA DE AZURE STORAGE

En este apartado vamos a comentar brevemente como funciona el Storage de Azure en la nube, e igual que en el apartado anterior, redactaremos de forma extensa su implementación en nuestro aplicativo.

Primero de todo, dejaremos un enlace explicando como crear una cuenta de almacenamiento en el portal de Microsoft Azure.

<https://docs.microsoft.com/es-es/azure/storage/common/storage-account-create?tabs=azure-portal>

Una vez hemos creado el recurso, dentro de este, crearemos un contenedor, que será la “carpeta” donde almacenaremos los archivos que enviemos desde nuestro aplicativo.

La facilidad que nos aporta el Storage proporcionado por Azure, es la gran cantidad de objetos de datos que almacena (blobs, archivos, colas, tablas y discos) con una fácil accesibilidad a estos a través de peticiones HTTP o HTTPS, así como seguros y con una gran escalabilidad.

EXPLICACIONES A INCRUSTAR LUEGO SOBRE EL BACK DE COMO DETECTAMOS LAS CARAS Y COMO ALMACENAMOS EN EL STORAGE

En nuestro back-end, vamos a tener una carpeta con todas las clases realcioandas con Azure, tanto para la detección y reconocimiento facial, como para los datos que almacenemos en el storage.

Encontraremos las siguientes clases dentro de nuestra carpeta:

-GuardarImagenStorage: esta clase, la vamos a dejar a modo ejemplo para mostrar como podríamos subir la imagen al storage desde la aplicación Android. Pese a ello, hemos decidido que la primera foto de nuestro *empleado* será almacenada directamente su URL en la base de datos, ya que creemos conveniente que esta primera imagen no la maneje el usuario, para mejorar la veracidad de que cada empleado es quien dice ser.

-DetectarCaraArray: es la clase encargada de detectar la cara desde un array de bytes. Esta clase, la usamos, tanto para comprobar al hacer cada foto, que se está reconociendo al individuo asegurándonos así que siempre se pueda detectar una cara, así como, a la hora de comparar las caras, la que nos analice la segunda imagen.

-DetectarCaraUrl: es la clase encargada de detectar la cara desde una URL, URL que pertenecerá a la imagen del usuario que hemos almacenado en el storage.

-CompararCaras: esta clase, se encarga de comparar ambas caras y devolviéndonos el resultado de si son idénticas, o no.

Entremos mas en detalle en la explicación de cada clase:

**detectarCaraArray**

En esta clase, tendremos un método llamado *DetectarCara*, público, que recibe como parámetro un *String* con la imagen recibida desde el *front-end* de la aplicación (Android en este caso).

Texto

Descripción generada automáticamente Primero de todo, vamos a mostrar las librerías usadas para la clase, pertenecientes a **apache** para la realización de las peticiones *http* y también a **JSON** para tratar la respuesta de Azure, que viene dada en este formato.

Ilustración detectarCaraArray.class - Librerías usadas

Dentro de nuestro método *DetectarCara*, transformaremos la imagen recibida como *String* a un array de bytes (*byte[])*, que es el formato que reconoce el servicio de Azure para leer la imagen. A continuación, crearemos unas variables necesarias para la realización del método, las cuales son:

-*LlaveAzure*: en ella recogemos la clave privada del servicio creado de Azure para el reconocimiento facial (este será del tipo *String*). (METER NOMBRE DE LA ILUSTRACION)

-*EndpointAzure*: recogemos también el endpoint de nuestro servicio, que será una URL de llamada a este (este será del tipo *String*).

-*jsonString:* creamos una variable del tipo *String* donde recogeremos el JSON de Azure en este formato para poder “tratarlo” y obtener así el id.

-*valorIdImagen1*: variable del tipo *String* que retornará el método para devolvernos el id que ha asignado el servicio de Azure a la cara reconocida en nuestra imagen.

-httpClient: creamos esta variable del tipo HttpClient donde creamos un cliente que nos permita enviar la solicitud al servidor de Azure y obtener respuesta de este, en formato http.

Texto

Descripción generada automáticamente

Ilustración detectarCaraArray.class - Método DetectarCara y sus variables

Una vez creadas, dentro de un try, comenzamos nuestra llamada a la API de Azure. Crearemos una variable llamada *builder* del tipo *UriBuilder* donde construimos la URL para la petición del servicio, pasando a esta variable nuestra variable *EndPointAzure* y añadiéndole al final de esta la notación *“/face/v1.0/detect”*, la cual le indica que del recurso *face* de Azure, lo que necesitamos usar en este caso es su método *detect* para que detecte la cara de la imagen.

Una vez creada la variable, establecemos los parámetros de la API, configurando, el modelo de detección que usaremos de Azure (en nuestro caso, es irrelevante usar uno que otro ya que solo necesitamos el id y no obtener información de las características del usuario, asique cogemos el modelo *detection\_01)*, establecemos también como true que retorne un Id y por último establecemos el modelo de reconocimiento, para así poder usar luego la comparación de caras (*recognition\_03)*.

Ya establecidos los parámetros, creamos la URI y, creamos una nueva variable llamada *request*, la cual será la respuesta http que vamos a hacer mediante un método *POST.* Falta también establecer las cabeceras de esta respuesta, indicando que el objeto que vamos a enviar es del tipo *octect-stream* (este objeto será nuestro array de bytes) y pasando también nuestra clave de acceso al servicio de Azure. De esta forma, hemos creado la petición API REST para llamar a Azure.

Texto

Descripción generada automáticamente

Ilustración detectarCaraArray.class - Creando la petición API REST

Ahora, tenemos que pasar la imagen que hemos guardado en el array de bytes a Azure, para ello, tenemos que guardar la imagen en una variable del tipo *Entity* de las librerías de apache donde la guardamos, y pasamos mediante un método de la respuesta donde creamos esta *Entity.*

Texto

Descripción generada automáticamente Ahora hemos de ejecutar la API, guardando su respuesta en una variable del tipo *HttpResponse* y acto seguido obtenemos la *Entity* de esta respuesta.

Ilustración detectarCaraArray.class - Obteniendo la respuesta de la API de Azure

Cuando por fin obtenemos respuesta, procedemos a tratar el JSON que nos habrá devuelto, lo haremos dentro de una sentencia *if* para comprobar que esta sea distinta de *null*. Lo primero que realizaremos será almacenar esta respuesta en la variable *jsonString* que habíamos creado previamente, pudiéndola así hacer legible en java. Al tratarse de un array nuestro JSON, creamos una variable del tipo *JSONARRAY* donde pasamos este String para que quede como un objeto JSON del tipo array. De él, tras hacer varias pruebas con el modelo de detección facial, sabemos que el primer campo que siempre muestra será el id de la cara, luego, seleccionamos el campo 0 del array para guardar el valor de este, siendo el valor que retornamos después.

Texto

Descripción generada automáticamente Tanto dentro de nuestro catch como fuera del try y este, retornamos *null* para indicar que no se ha detectado ninguna cara en la imagen.

Ilustración detectarCaraArray.class - Recogiendo la id de la imagen del JSON

**detectarCaraURL**

Esta clase, actúa prácticamente igual que *DetectarCaraArray*, cambiando solamente, la imagen, que esta vez la pasamos directamente la URL de la imagen que tenemos almacenada en el storage convertida en formato JSON, y también cambiamos la cabecera de la respuesta, que en lugar de ser del tipo *octet-stream*, indicamos que es del tipo JSON (*“application/json”*). Cabe destacar que también usaremos las mismas librerías.

Texto

Descripción generada automáticamente

Ilustración detectarCaraURL.class - Librerías, método y variables de la clase detectarCaraURL

Texto

Descripción generada automáticamente

Ilustración detectarCaraURL.class - Petición API REST para detectar la cara de la imagen y resultado.

**CompararCaras**

En esta clase, vamos a realizar la comparación entre ambas caras mediante un método que hemos llamado *returnIdentical*, al cual le pasamos dos parámetros del tipo *String*, que serán las imágenes, siendo la primera la url del storage de azure que obtenemos de la base de datos, y la segunda, la imagen que capturamos en Android.

Volveremos a usar las mismas librerías que hemos usado en las dos clases anteriores, ya que el método para comparar ambas caras mediante la petición API REST es similar.

Como variables, vamos a tener dos del tipo String que serán los Id’s de las imágenes, llamadas *id1 e id2* respectivamente. También, tendremos las variables *imagen1 e imagen2* que son del tipo *detectarCaraURL*  y *detectarCaraArray* respectivamente, para, desde los parámetros recibidos, llamar a sus métodos para retornar los ids.

Esta vez, cuando creemos la URI, debemos indicar al final de ella que es del tipo *“/face/v1.0/verify”* en lugar de detect como en las clases anteriores. Nuestras cabeceras, seguirán siendo del tipo *“application/json”* ya que vamos a pasar un JSON con ambos Id’s.

A continuación, recorremos el array del JSON para recoger el valor que queremos obtener, ya que en dicho array tendremos un valor del tipo *boolean* llamado *itsIdentical* que nos comunica si las caras comparadas, las considera iguales o no, y a parte, el valor que nosotros queremos coger, *confidence*, que será del tipo *double* y nos remite un valor entre 0 y 1 del “porcentaje” de parecido que encuentra entre las caras, siendo 0 el 0% y 1 el 100%.

Texto

Descripción generada automáticamente Retornamos este valor para reconocer si son la misma persona o no. Esta verificación, la hacemos desde la clase *EmpleadosServiceImp*, la cual comentaremos más adelante con cada uno de sus métodos. A continuación, mostramos el código de la clase, así como un pequeño fragmento de la consola para ver la respuesta.

Ilustración CompararCaras.class - Librerías y variables

Texto

Descripción generada automáticamente

Ilustración CompararCaras.class - Método returnIdentical

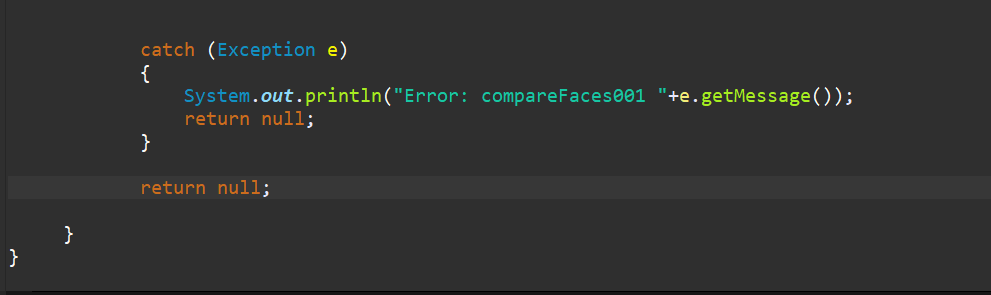


Ilustración CompararCaras.class - Tratando el error y retorno en caso de fallo

Veamos a continuación un ejemplo en la consola al comparar dos caras y mostrando los Id’s de cada una y el valor confidence.

**Texto

Descripción generada automáticamente**

Ilustración Resultado de comparar las caras

*(Nota: el valor de la url a la imagen se muestra por consola ya que estamos haciendo la llamada mediante un controlador y lo usamos para comprobar que la url sea correcta)*

**GuardarImagenStorage**

*Nota: previo a poder usar el storage de Azure, hemos de añadir una dependencia en nuestro archivo pom.xml para que puedan usarse las librerías propias de Azure en esta clase.*

Texto

Descripción generada automáticamente

Ilustración 18 Dependencia de Microsoft Azure Storage en el pom.xml

**MANUAL DE CREACION DE LOS RECURSOS *FACE* Y *ALMACENAMIENTO* EN AZURE**

Primero de todo, necesitamos tener creada una cuenta en el portal de Microsoft Azure, en mi caso, accederé con la cuenta ya creada.

Interfaz de usuario gráfica, Aplicación, PowerPoint

Descripción generada automáticamente

Ilustración 19 Inicio de sesión en el portal de Azure

**RECURSO FACE**

Una vez nos hemos loggeado, iremos a la pantalla principal del portal, donde tendremos un esquema de nuestros recursos utilizados, así una ventana de botones con recursos en la parte superior. Seleccionaremos el botón de crear un nuevo recurso.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Ilustración 20 Creando un nuevo recurso en Azure

Al crear un nuevo recurso, nos saldrá una barra de búsqueda donde pondremos como palabra clave *Face*, y acto seguido, nos abrirá la ventana del recurso *Face* para crearlo.

Captura de pantalla de computadora

Descripción generada automáticamente

Ilustración 21 Buscando el recurso Face

Captura de pantalla de un celular

Descripción generada automáticamente

Ilustración 22 Creando un recurso Face en el portal de Azure

Captura de pantalla de un celular

Descripción generada automáticamente Una vez le demos a crear, nos pedirá crear un nombre de recurso, seleccionar la zona donde lo estamos creando, y la tarifa que vamos a usar (en nuestro caso gratuita). Una vez establecidos los campos, daremos a crear recurso.

Ilustración 23Creando el recurso

Una vez creado el recurso, ya podremos acceder a él desde nuestra petición. Para ello, como comentamos a la hora de explicar como funciona el reconocimiento facial, necesitaremos de la clave, y del endPoint de nuestro recurso. Estos datos podemos obtenerlos desde la pantalla principal del recurso, yendo al apartado de *Claves y punto de conexión.*

Texto

Descripción generada automáticamente

Ilustración 24 Vision general del recurso Face.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Ilustración 25 Claves y punto de conexión

Bastará entonces con, a la hora de llamar al recurso, copiar y pegar estas credenciales en nuestro código para poder tener acceso a ellas.

**RECURSO ALMACENAMIENTO**

Al igual que en el apartado del recurso fase, comenzaremos creando un nuevo recurso, o mejor dicho en este caso, una cuenta de almacenamiento.

Cabe destacar que a la hora de crear el contenedor, nos dejara introducir un nombre en minúsculas y con números, pero no acepta mayúsculas ni otros caracteres.

Texto

Descripción generada automáticamente

Ilustración 26 Creando cuenta de almacenamiento

Una vez hemos creado una cuenta de almacenamiento, dentro, tenemos que crear un contenedor. Podemos observar en la pagina principal del recurso, la gran cantidad de información que nos aporta de la cuenta de almacenamiento, así como en Herramientas y SDK, una ayuda en distintos lenguajes para poder implementarlo.

Captura de pantalla de computadora

Descripción generada automáticamente

Ilustración 27 Ventana principal de la cuenta de almacenamiento

Desde aquí, pulsamos en contenedores y desde ahí creamos un nuevo contenedor. En mi caso, ya tengo creado el contenedor caras, asique desde este, pasaré a explicar un factor muy importante para poder hacer uso del contenedor desde una aplicación, así como poder visualizar los archivos que haya en este. El nivel de acceso de nuestro contenedor, será privado por defecto, lo que bloquea el poder visualizar las imágenes teniendo las rutas, o el añadirle imágenes desde nuestro código si no lo cambiamos. Para ello, una vez hayas creado el contenedor, pincha en él y entra dentro, y ve al apartado con un candado que pone *Cambiar nivel de acceso*. Para darle un nivel de acceso poublico, en mi caso le de hado un nivel de Contenedor, que nos permite acceder a sus archivos.

Texto

Descripción generada automáticamente

Ilustración 28 Cambiando el nivel de acceso del contenedor

Volviendo a la ventana principal de la cuenta de almacenamiento, desde ella, en las claves de acceso, al igual que con el recurso de los servicios cognitivos, encontramos el apartado de claves de acceso, la cual nos hará falta para poder usarla a la hora de llamar al recurso.

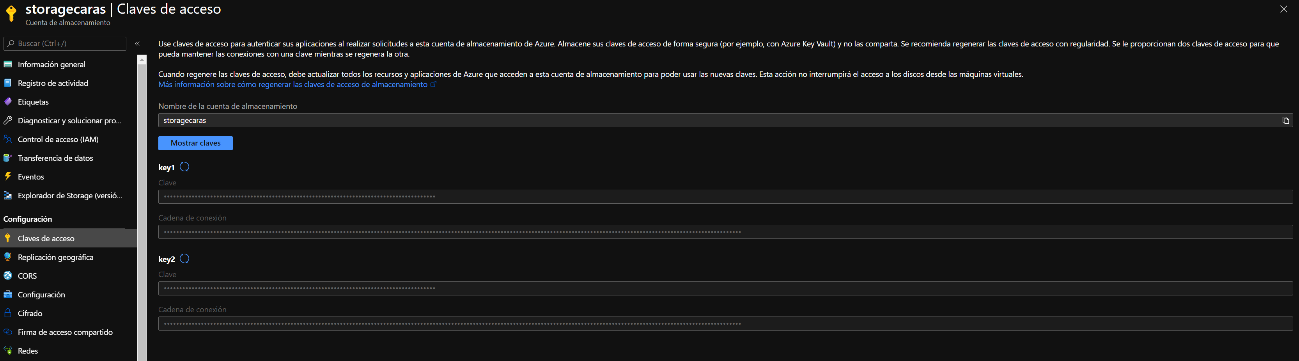


Ilustración 29 Claves de acceso a la cuenta de almacenamiento