Desarrollo de una aplicación en Android para la estimación automática de carbohidratos mediante un análisis de imágenes y técnicas de Inteligencia Artificial.

Laura Casas Torres
Milagros del Rocío Peña Quineche
José Antonio Bernal Pérez

GRADO EN INGENIERÍA DEL SOFTWARE
FACULTAD DE INFORMÁTICA
DEPARTAMENTO DE ARQUITECTURA DE COMPUTADORES Y AUTOMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

TRABAJO DE FIN DE GRADO EN INGENIERÍA DEL SOFTWARE

Director: **José Ignacio Hidalgo Pérez**
Codirectora: **María Guijarro Mata-García**

Dedicatoria

Dedicado a nuestras familias que siempre nos han apoyado a pesar de las dificultades.

Agradecimientos

Agradecimientos a todos los profesores que nos han ayudado a lo largo de nuestras vidas haciendo de sus asignaturas algo más que una mera forma de aprendizaje. Además agradecer a María e Ignacio por su dedicación y apoyo durante este trabajo.

Índice

Lista de figuras

Resumen

Este trabajo de fin de grado tiene como objetivo desarrollar una aplicación para dispositivos móviles, concretamente para aquellos dispositivos con sistema operativo Android. Esta aplicación, será capaz, mediante una fotografía tomada con el propio dispositivo, de reconocer los distintos alimentos presentes en la imagen a través de técnicas de visión por computador y de Inteligencia Artificial. La finalidad de esta aplicación es proporcionar una herramienta sencilla e intuitiva a las personas que sufren diabetes para ayudarles en la estimación y monitorización de nutrientes, consiguiendo así que puedan aplicar un plan de alimentación saludable que les ayude a controlar los síntomas de la enfermedad.

En primer lugar, la aplicación segmentará la imagen proporcionada por el usuario mediante el algoritmo de agrupamiento *KMeans*. Los resultados proporcionados por este algoritmo serán usados por el clasificador que, mediante el algoritmo *K-Nearest Neighbors,* etiquetará los distintos alimentos. Una vez los alimentos han sido etiquetados en la categoría correcta, la aplicación hace uso de una base de datos externa USDA para conseguir los valores nutricionales de cada alimento haciendo especial énfasis en los carbohidratos. Toda la información recopilada es mostrada al usuario y después es guardada en su propio historial. Cada usuario cuenta con un historial de comidas donde se guarda información de las fotos de las comidas que ha subido y todos los valores nutricionales de éstas.

De esta forma queda una aplicación capaz de identificar diferentes alimentos en una misma imagen usando técnicas de visión por computador y de Inteligencia Artificial facilitando la tarea a las personas que sufren de diabetes de estimar la cantidad de

insulina diaria necesaria, logrando así un mejor control sobre la enfermedad y sus síntomas.

Palabras clave

Diabetes, algoritmo, Knn, KMeans, USDA, procesamiento, segmentación, clasificación, activities, intents, informe, nutriente, bolo prandial, bolo corrector.

Abstract

This work is aimed to the development of an application for mobile devices, specifically for those with Android operating system. This app will be able to recognize different aliments inside a photograph, which has been taken with the device itself, using computer vision and Artificial Intelligence techniques.

First of all, the application segments the image, supplied by the user, using a clustering algorithm, *KMeans*. The classifier will use the results from the clustering algorithm to label the different aliments by means of the *K-Nearest Neighbors* algorithm. Once the aliments are labeled in the right category, the application uses an external data base to obtain the nutritional values of each element with further detail in the carbohydrates. All the collected information is shown to the user and then is saved in its own record. Each user counts with its own record of foods where the images they have uploaded and their nutritional values are saved.

That allows us to build an application that identifies different aliments in the same image using computer vision and Artificial Intelligence techniques facilitating the estimation of dairy insulin to people with diabetes, achieving a better control over the illness and the symptoms.

Keywords

Diabetes, algorithm, Knn, KMeans, processing, segmentation, classification, activities, intents, report, nutrient, prandial bolus, corrective bolus.

Introducción

Antecedentes

Diabetes

La Diabetes Mellitus [1] es una enfermedad que se produce cuando el páncreas no es capaz de crear insulina suficiente o cuando ésta no logra actuar en el organismo porque las células no responden a su estímulo. La insulina es esencial para el control de los niveles de glucosa en sangre, en su ausencia, las células no son capaces de asimilar el azúcar y como consecuencia se produce un aumento en los niveles de estos, lo que se conoce como hiperglucemia.

Existen distintos tipos de diabetes con distintas causas y tratamientos. La Fundación para la Diabetes distingue los siguientes tipos: [2]

- Diabetes tipo 1: ocurre cuando el páncreas no fabrica insulina suficiente. Como resultado la persona que padece diabetes tipo 1 necesita inyectarse insulina. Hoy en día aún se desconocen las causas que dan origen a este tipo de diabetes, aunque existen una serie de factores que combinados podrían fomentar su aparición, tales como el factor genético, la autoinmunidad o el daño ambiental.

- Diabetes tipo 2: este tipo de diabetes a diferencia del tipo 1 no se produce por la incapacidad de producir insulina sino por la resistencia que el cuerpo presenta a esta

hormona. Normalmente el páncreas de las personas que padecen este tipo de diabetes tiende a disminuir la producción de insulina paulatinamente. El 80% de las personas que desarrollan diabetes tipo 2 tienen obesidad y un estilo de vida muy sedentario. El 20% restante suelen tener un defecto hereditario que causa resistencia a la insulina.

- Diabetes gestacional: durante el embarazo se produce una intolerancia total a la glucosa que puede ser debida a múltiples causas. Este tipo de diabetes aparece en una de cada diez mujeres embarazadas.

- Otros tipos de diabetes: diabetes MODY (Maturity Onset Diabetes in the Young) que se produce por defectos de las células beta o Diabetes DRFQ (Diabetes Relacionada con Fibrosis Quística) que se produce por el impacto de la Fibrosis Quística en el páncreas.

En España los datos sobre la diabetes no son nada alentadores. Un estudio epidemiológico realizado por la Sociedad Europea de Diabetes muestra que el 13,8% de los españoles mayores de 18 años tiene diabetes tipo 2, lo que equivale a más de 5,3 millones de personas. De estos, el 43% desconocía que padecía la enfermedad. El retraso en descubrir que se padece diabetes implica que cuando se diagnostica la mitad de los casos ya presentan alguna complicación. Además, el estudio ha revelado más datos importantes relacionados con situaciones que se relacionan estrechamente con la diabetes. El 28,2% de la población es obesa y el 12,6% es intolerante a la glucosa. [3]

Análisis automático de imágenes

La visión por computador [4] trata de dotar a las máquinas del sentido de la vista con el objetivo de extraer información de las imágenes y poder utilizarla en distintas prácticas. Suele girar en torno al reconocimiento de formas, aunque hay muchas más características que se pueden utilizar de una imagen como, por ejemplo, el color. Cada píxel que conforma una imagen digital contiene un vector de tres valores que contiene el nivel de rojo, azul y verde, respectivamente.

Figura 1. Etapas del análisis automático de una imagen.

Para poder conseguir información útil de una imagen se siguen una serie de etapas y procesamientos orientados a mejorar la calidad de la información que se va a extraer de ella, tal y como se describe en la Figura 1. La primera etapa consistirá en la toma de la imagen. En este trabajo se realizará a través de la cámara del móvil donde esté instalada la aplicación.

En la etapa de pre-procesado se contemplan innumerables operaciones de pre-procesamiento de imágenes que ayuden a facilitar la tarea a las etapas sucesivas. Dentro de las técnicas de pre-procesamiento de imágenes existen dos grandes áreas:

procesamiento con observador humano o procesamiento sin observador humano. El procesamiento sin observador humano tiene mayor interés práctico en los sistemas de visión por computador. La técnica más común sería:

- **Transformación del histograma:** el histograma de una imagen es un gráfico que representa los niveles de gris en el eje de abscisas y el número de pixeles de cada nivel en el eje de ordenadas. Aunque en algunas ocasiones el histograma solo otorga la posibilidad de aumentar o disminuir los niveles de contraste de una imagen, en otras ocasiones es suficiente para separar objetos dentro de una imagen facilitando la etapa de interpretación.

Después de la etapa de pre-procesado la imagen pasa por la etapa de segmentación. La segmentación de una imagen separa las distintas zonas de interés de estudio, convirtiéndose así en una etapa decisiva por la importancia de los resultados que proporciona. Esta etapa difiere dependiendo del objetivo que se persiga con la interpretación de la imagen. En general se separan en zonas o en objetos individuales. Existen distintos tipos de técnicas de segmentación:

- **Agrupación por rasgos comunes:** segmenta las imágenes mediante algoritmos de agrupación de datos. Esta técnica segmenta las imágenes de forma automática y no supervisada sin exigir un conocimiento previo de las clases de objetos existentes.

- **Extracción de bordes:** separa los objetos a partir de sus bordes. Esta técnica se inspira en un principio muy intuitivo y simple puesto que los pixeles situados en los bordes de los distintos objetos presentan grandes variaciones en sus características con respecto a los pixeles vecinos. Por ejemplo, un objeto oscuro situado en un fondo claro.

Una vez la imagen ha sido segmentada se pasa a la fase de extracción de rasgos. Independientemente de la forma elegida de segmentación, los rasgos obtenidos han de ser suficientes para poder distinguir los distintos objetos. Las distintas características extraídas de cada objeto forman un vector que se usará en la etapa de clasificación.

Antes de pasar a la clasificación de los objetos es imprescindible haberlos catalogado previamente. Es decir, un sistema de visión artificial basa su conocimiento en una batería de datos cargados con anterioridad por un equipo de especialistas. La complejidad de la batería de datos depende del objetivo de la aplicación pudiendo ser muy sencilla o compleja. [5]

Inteligencia artificial

La inteligencia artificial [6] es aquella dirigida a las máquinas con el objetivo de "imitar las funciones «cognitivas» que los humanos asocian con otras mentes humanas, como por ejemplo aprender y resolver problemas".

Este concepto es conocido desde 1956, año en el que se determinaba si una máquina era capaz de imitar la inteligencia humana con el test de Alan Turing y los axiomas de la Ley de Moore. [6]

Hasta la actualidad, el desarrollo y la investigación de la Inteligencia Artificial ha tenido un crecimiento exponencial que gracias a los proyectos de innovación sigue aumentando. Por esto, en los últimos años ha crecido un nuevo concepto derivado de la IA, que es el Machine Learning [6], cuyo objetivo es construir programas que mejoren

automáticamente mediante la experiencia. Gracias a esta rama ha evolucionado el aprendizaje automático de las máquinas y con él, el concepto principal que estamos tratando en este TFG, que es la Visión por Computador o Visión Artificial. Del mismo modo que la IA pretende emular la inteligencia humana dotando a las máquinas de la capacidad de manipular datos sensoriales similares a los empleados por los seres vivos, la Visión Artificial sirve a este propósito con el objetivo último de generar información a partir de datos visuales mediante el estudio de los procesos de reconocimiento y localización de objetos por medio del procesamiento de imágenes. [7]

Objetivos y plan de trabajo

El objetivo es la construcción de una aplicación para dispositivos móviles con sistema operativo Android que sea capaz de estimar de forma automática, con un nivel de error bajo, los distintos alimentos que se le muestran a través de una imagen digital, extrayendo características de color de cada uno de los pixeles y usando éstas para segmentar y clasificar los alimentos. Además, la aplicación debe proporcionar y guardar información nutricional sobre cada alimento detectado.

Motivación

Varios han sido los motivos que nos han impulsado en el desarrollo de este proyecto.

El primero de todos ha sido el interés y motivación que nos proporcionaba trabajar en una aplicación que usa técnicas de inteligencia artificial con el objetivo de identificar objetos. El campo de la inteligencia artificial, que se encuentra lejos de ofrecer todas las ventajas que es capaz de proporcionar, supone un reto y un área de investigación fascinante donde poder aplicar muchos de los conocimientos que hemos adquirido a lo largo de nuestra carrera, el Grado en Ingeniería de Software, e incluso más que hemos adquirido a raíz de nuestro trabajo en este proyecto.

Además, para nosotros poder desarrollar esta aplicación en una plataforma como Android suponía un doble reto al vernos enfrentados a nuevas complicaciones y desafíos que no hemos tenido tan presentes a lo largo de nuestros estudios.

Por último, desarrollar una aplicación que supone una pequeña aportación al mundo de la salud y que, por lo tanto, podría suponer una mejora de la calidad de vida de un sector de la población nos ha motivado enormemente para perfeccionar aún más todo el trabajo que se ha realizado durante el proyecto.

Fundamentos teóricos

Sistema Operativo

Uno de los principales objetivos de este proyecto es mejorar la calidad de vida de la mayor cantidad de personas posible. Para conseguir esa meta realizamos una investigación sobre los terminales más vendidos en el año 2018 y el sistema operativo más utilizado, ambos a nivel mundial.

Figura 2. Venta de terminales en junio de 2018 [8]

Figura 3. Sistema Operativo más utilizado a nivel mundial. [9]

Como se puede apreciar en la Figuras 2 las tres marcas más vendidas en el mundo en junio de 2018 eran Samsung con el 30.66%, Apple con el 18.94% y Xiaomi con el 7.01%, de las cuales la primera y la tercera usan el sistema operativo Android. Además, en la Figura 3 existe un predominio de dicho sistema como el más usado también a nivel mundial.

Esto, añadido a la falta de recursos del equipo que imposibilitaba realizar una correcta fase de pruebas en un dispositivo con sistema operativo iOS y a la mayor experiencia en este terreno, debido a los conocimientos sobre Android adquiridos a lo largo de la carrera, nos hizo decantarnos por realizar el proyecto en esta plataforma.

Data-set y Base de datos

La implementación de la aplicación consta de dos partes a desarrollar. La primera parte es off-line y se encargará de generar un data-set, o base de conocimientos, para que el algoritmo encargado de la clasificación de los alimentos pueda obtener conocimientos y aprender.

Por otro lado, se tendrá una parte on-line en la que al tomar la decisión e identificar el alimento, deberá registrar esa información en una base de datos.

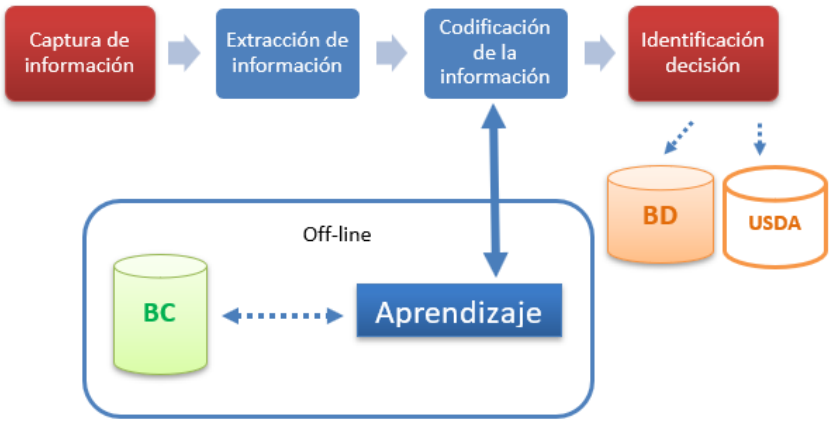

Figura 4. Esquema general funcionamiento aplicación.

Como se muestra en la Figura 4 la aplicación divide su comportamiento en dos partes principales. La parte off-line tiene como objetivo crear un data-set o conjunto de datos lo más extenso posible para el correcto funcionamiento de la parte on-line. Se tomó una batería de imágenes que fueron segmentadas automáticamente mediante el algoritmo KMeans [15] y etiquetadas manualmente por el equipo de desarrollo. Toda esta información fue subida manualmente a la base de datos proporcionando así el conjunto de datos de aprendizaje de la aplicación. La parte on-line sigue un flujo lineal dividido en cuatro tareas principales:

- **Captura de información**: es la primera etapa contenida en la parte on-line de la aplicación. En este caso, la captura de información es la toma en tiempo real de una fotografía por parte del usuario, o la elección de una que haya sido tomada previamente, del plato de comida que vaya a ingerir.

- **Extracción de la información**: una vez el usuario ha proporcionado una imagen pasamos a la etapa de extracción de la información. En esta etapa segmentaremos por color los distintos componentes, en este caso alimentos, que forman el plato.

Como resultado de la segmentación obtenemos los pixeles pertenecientes a cada alimento con los que calcularemos su tamaño y su media RGB.

- **Codificación de la información**: tras conseguir la información de cada alimento lo etiquetaremos mediante un algoritmo de clasificación. El algoritmo de clasificación Knn [20] hace uso de un data-set creado previamente en la parte off-line. Gracias a este conjunto de datos somos capaces mediante el cálculo de distancias euclídeas de predecir la categoría más posible a la que puede pertenecer el alimento. Después se muestra al usuario una lista de múltiple opción de las categorías más probables a las que pueden pertenecer donde podrá seleccionar la etiqueta final a la que se asignará cada uno, o crear una nueva en caso de que no existiese.

- **Identificación de la decisión**: los distintos alimentos ya han sido etiquetados correctamente, dicha información será subida a la base de datos para aumentar el data-set con el objetivo de que la aplicación continúe siempre aprendiendo. Además, se realizará una consulta a la base de datos estadounidense USDA [42], mediante la cual se consiguen los nutrientes, centrándonos sobre todo en los carbohidratos, de cada alimento.

Para estas bases de datos, el equipo decidió investigar qué bases de datos tendría mejor rendimiento y compatibilidad con Android. En un principio, se contempló la opción de utilizar el módulo INTERNAL STORAGE que ofrece Android, para poder guardar la información en el propio dispositivo. La gran ventaja de este módulo es que cada usuario tiene solo información con respecto a sí mismo siendo factible al esquema relacional de la BD. Sin embargo, la aplicación terminaría abarcando demasiado espacio en memoria y afectaría al terminal.

Por esta razón, se decidió utilizar el módulo de EXTERNAL STORAGE, el cual permite hacer uso de sistemas externos para la gestión de los datos. Gracias a este módulo obtenemos un funcionamiento normal y adecuado tanto para el dispositivo como para la aplicación. [11]

Las 3 opciones principales para la gestión de la BD fueron MySQL, SQLite y una API de Google que permitía gestionar una base de datos con la plataforma ya mencionada.

SQLite

Es un motor de base de datos SQL de código abierto, compatible con Android, ligero y que no necesita de la instalación en un servidor. [12]

Las ventajas que representa este sistema respecto a los demás es que se puede instalar mediante una librería en una aplicación para Android sin necesidad de tener que gestionarlo desde un servidor remoto. Por otro lado, no es un proceso independiente al programa, ya que genera un esquema persistente que se almacena en el dispositivo guardando toda la información en el propio terminal.

Debido, en primer lugar, a que no es recomendable guardar toda la información en el dispositivo y, en segundo lugar, a la falta de experiencia en este sistema se desestimó esta opción.

Cloud SQL

Es un sistema de bases de datos MySQL y PostgreSQL gestionado en la nube administrado y ofrecido por Google. [10]

Proporciona un alto rendimiento al hacer uso de la Cloud de Google y, por esta misa razón, una gestión total que permite dar escalabilidad y seguridad a los datos. Por otro lado, al ser un servicio de Google, mediante una API, es totalmente compatible con sistemas Android.

Se desestimó esta opción ya que a pesar de sus muchas ventajas habría supuesto un gasto extra al ser un servicio de pago.

MySQL y MariaDB

MySQL [13] es un servicio de bases de datos muy popular actualmente, aunque debido al cambio de licencia que tuvo desde la versión 5.7 se volvió un servicio privado ofrecido por Oracle. Por esto, es que MariaDB [14] es la mejor opción para sustituir a MySQL, ya que en definitiva es un remplazo aparecido desde un fork a MySQL.

En cuanto a motor, MySQL utiliza MyISAM e InnoDB, que es lo que mayormente utilizan los sistemas de bases de datos enfocados en MySQL, mientras que MariaDB utiliza Aria y XtraDB (plugin de InnoDB), que puede que provoque problemas de incompatibilidad con una aplicación MySQL, aunque la probabilidad es mínima. Por otro lado, MariaDB está por encima en cuanto a rendimiento y tablas gracias a que Aria tiene su caché en RAM.

En definitiva, ambos sistemas son muy similares y válidos para la aplicación. Debido a que el servidor ofrecido al equipo para realizar el proyecto ya disponía de este servicio se decidió gestionar con él la BD complementándolo con el servicio PhpMyAdmin que al proporcionar una interfaz permite trabajar más cómodamente.

Clasificación y segmentación

En la aplicación se recibe una imagen de un plato con alimentos que hay que extraer de la imagen para poder analizar, procesar y clasificar. Para poder realizar este proceso se ha hecho uso de algoritmos de segmentación y clasificación. A continuación, se explican detalladamente tanto los algoritmos utilizados en la aplicación como los estudiados que finalmente fueron descartados.

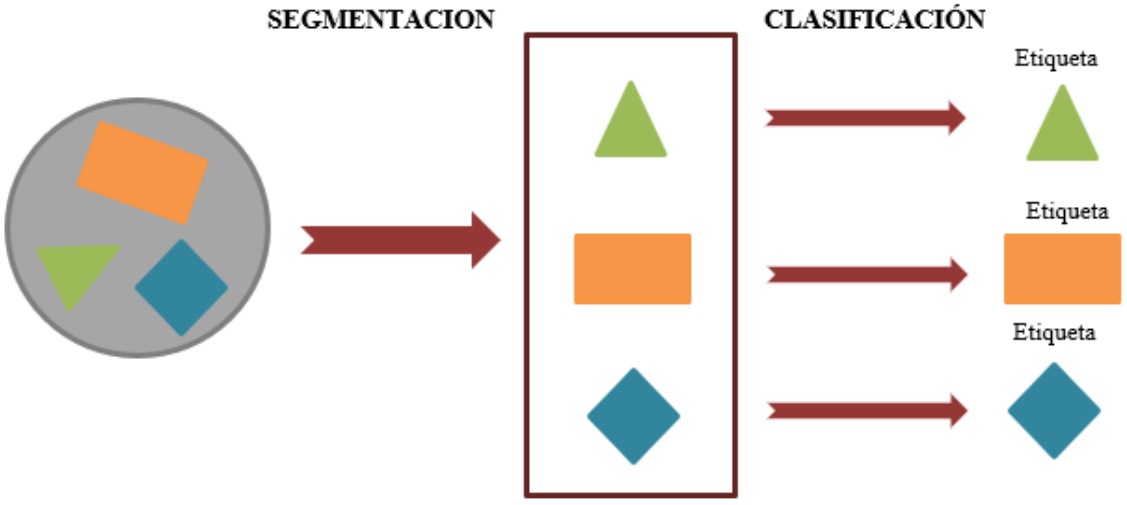

Figura 5. Esquema de funcionamiento segmentación y clasificación.

Algoritmos de segmentación

El algoritmo de segmentación será el encargado de separar aquellas componentes conexas por el color. Esta fase es clave para la correcta realización de la clasificación posterior.

Algoritmo iterativo

El algoritmo iterativo [15] es un algoritmo genérico utilizado para la extracción de las regiones que nos interesen en una imagen. El funcionamiento de este algoritmo es el siguiente:

- Paso 1: el algoritmo empieza con una matriz que representa la posición de cada píxel en la imagen, teniendo cada píxel inicializado a 1. Si el píxel no tiene una etiqueta asignada, es decir, que está a 1, se genera una nueva etiqueta para ese píxel.

- Paso 2: iteración de arriba-abajo. Recorre la matriz de arriba-abajo buscando etiquetas distintas de 0; una vez encuentra un píxel con esa etiqueta busca la mínima etiqueta de los vecinos etiquetados en esa misma línea, si la etiqueta es diferente a la del píxel actual, entonces se produce un cambio de etiqueta para este pixel.

- Paso 3: iteración de abajo-arriba. Realiza el mismo proceso que en el paso 2 pero recorriendo la matriz desde abajo.

Con esto, el algoritmo consigue obtener las componentes conexas que existen dentro de la imagen y así, por consiguiente, segmentar la imagen según los elementos que nos interese identificar.

En definitiva, para la extracción de objetos en una imagen este algoritmo es el básico para entender el concepto de etiquetado, pero a partir de éste derivará el clásico explicado en el siguiente apartado, los cuales ayudarán en esta aplicación en el proceso de segmentación de una imagen.

Algoritmo clásico

Este algoritmo se denomina así por estar basado en el algoritmo de componentes conexas para grafos descrito por Rosendfeld y Pfaltz (1966) [15]. El funcionamiento de este algoritmo consta de dos pasos:

- Paso 1: realiza la asignación de etiquetas como se ha descrito en el *Algoritmo iterativo* ya mencionado, pero contando también con una tabla de asignación pixel-etiqueta en la que se registran las equivalencias entre pixeles y la cual quedará completada en este paso. Si en algún momento de la propagación de etiquetas existe dos etiquetas para el mismo pixel siempre prevalecerá la más pequeña.

- Paso 2: en este paso se obtienen las diferentes clases a partir de la tabla de equivalencia generada en el paso anterior. Por lo que a partir de estas clases se empiezan a reasignar etiquetas a cada pixel gracias a un algoritmo estándar

que busca a los vecinos conexos que contengan la misma etiqueta y finalmente, segmenta la imagen en una lista de componentes según la etiqueta correspondiente a cada pixel.

Al igual que el algoritmo iterativo, estos son algoritmos genéricos útiles para la extracción de regiones en una imagen, por eso, el algoritmo que se ha utilizado en este trabajo para segmentar la imagen se basa finalmente en este algoritmo y en el anterior.

Algoritmo KMeans

KMeans [17] es un algoritmo de agrupación de datos no supervisado, es decir, que no dispone de un conocimiento previo sobre las clases en las que se pueden distribuir los distintos objetos. No obstante, este algoritmo también resulta de gran utilidad como herramienta de verificación y testeo de la calidad del diseño de un reconocedor automático.

El algoritmo de agrupamiento propuesto por Mac Queen en 1967 [17] divide los datos en *k* predeterminadas clases siendo necesario el conocimiento previo del número de clases.

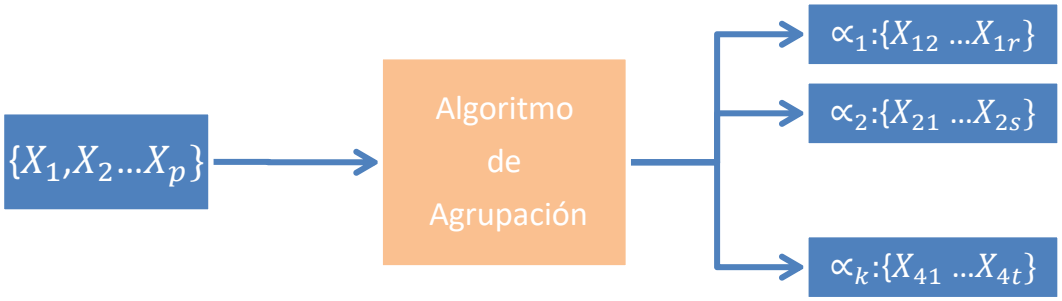

Figura 6. Representación del funcionamiento de un algoritmo de agrupación. Referenciado en Maravall, D. (1993)

El funcionamiento del algoritmo sería el siguiente partiendo de un conjunto de objetos a clasificar: [18]

- Paso 1: selecciona aleatoriamente *k* puntos dentro del conjunto de objetos o data set. Estos puntos serán considerados los centros de las *k* clases.

- Paso 2: calcula respectivamente la distancia de cada punto con cada uno de los centros. Los puntos pertenecerán al centro con menor distancia.

$$\boldsymbol{J = \sum_{n=1}^{N} \sum_{k=1}^{K} ||x_n - \mu_k||^2}$$

  - Donde *k* es el número de centros y *n* el número de observaciones, siendo k ≤ n.
  - Siendo $(x_1, x_2, \ldots, x_n)$ el conjunto de observaciones, en nuestro caso, siendo el conjunto de pixeles.
  - Donde $\boldsymbol{\mu_k}$ es la media de puntos en el centro *k*.

- Paso 3: una vez se han agrupado los puntos en sus respectivos centros, es preciso recalcular los centros de las clases. El objetivo de recalcular los centros es minimizar *J*. En cada iteración J irá disminuyendo o al menos no sufrirá ningún cambio, pero nunca aumenta su valor, lo que garantiza que eventualmente alcanzará su mínimo.

- Paso 4: de acuerdo con los nuevos centros se vuelve a calcular la distancia de cada punto con cada centro. Si la distancia con el nuevo centro es menor que la distancia con el centro anterior el punto pasa a formar parte del nuevo grupo. Este paso se repite hasta que el algoritmo sea estable, es decir, ninguno de los puntos cambia de centro.

El algoritmo K-means depende en gran medida de los centros que se eligen aleatoriamente en la primera iteración. El problema principal es que si la clasificación inicial que se hace entorno a estos centros se desvía mucho de la clasificación óptima lleva a resultados erróneos. Cuanto mayor es el conjunto de datos, por ejemplo, en el caso de una imagen, mayor es la desviación hacia la clasificación óptima. Debido a esto la imagen debe procesarse en varias rondas de agrupamiento para poder lograr mejores resultados. Utilizando K-means para segmentar una misma imagen varias veces, los resultados conseguidos son diferentes al cambiar los centros de agrupación iniciales. El algoritmo suele repetirse varias veces consiguiendo así mejores resultados al haber contemplado más posibilidades.

En conclusión, K-means es un algoritmo que nos permitirá segmentar las imágenes sin un conocimiento previo de las clases en las que tenía que dividir los distintos objetos ofreciéndonos así la posibilidad de automatizar la segmentación de las imágenes para poder conseguir los datos de cada uno de los alimentos automáticamente.

Algoritmos de clasificación

Una vez las imágenes han sido segmentadas el siguiente paso es clasificar las distintas clases, en el caso de este proyecto los distintos alimentos, que forman parte de la imagen. Para esto se han investigado distintos algoritmos de clasificación que usan distintos métodos para clasificar datos.

Bayes

El clasificador bayesiano [19] es un clasificador probabilístico que se fundamenta en el Teorema de Bayes. Este clasificador se basa en que toma como hipótesis que cada variable predictora es independiente, es decir, ninguna variable está influida por la existencia o desaparición de otra. Debido a la independencia de las variables el algoritmo recibe el apelativo de ingenuo.

El funcionamiento de este clasificador se basa en encontrar la probabilidad de que, conociendo los valores que describen a una muestra, esta pertenezca a una clase u otra. De esta manera, cada característica ayuda independientemente a la probabilidad de que pertenezca a una clase u otra.

Un hecho importante y positivo de este tipo de clasificador, es que el "peso" o "probabilidad" de una variable en la clasificación puede ser muy importante en relación con la causa, pero ser además muy común en otras, con lo cual el "peso" final disminuye.

$$f(x_i, w) = p\left(\frac{x_i}{m}, C\right)$$

- Donde *w = (w1, w2)* es el vector de parámetros a estimar.
- Siendo $p(x_i/m)$ la función de probabilidad condicional de la clase x_i para m.
- Donde *C* es la matriz de covarianza.

El clasificador bayesiano ingenuo suele entrenarse de manera muy eficaz en un entorno de aprendizaje supervisado, es decir, estima los parámetros a partir de un conjunto de datos de entrenamiento. Este conjunto de datos no tiene por qué ser grande ya que como se asumen las variables independientes solo es necesario calcular las varianzas de las variables de cada clase.

K-Nearest Neighbors (KNN)

K-Nearest Neighbors o K-vecinos más cercanos [20] forma parte de la familia de algoritmos de aprendizaje basado en instancias o Instance-Based Learning. Este tipo de algoritmos no tienen un modelo global asociado a los conceptos a aprender, en su lugar las predicciones se realizan basándose en los ejemplos más parecidos a la instancia a predecir. Es decir, el aprendizaje en este tipo de algoritmos consiste simplemente en almacenar los datos de entrenamiento y cuando existe un nuevo dato se recupera de memoria un conjunto de datos similares que serán usados para clasificarlo. La ventaja de este tipo de algoritmos es que son capaces de usar una representación de los datos más compleja. Por otro lado, puede llegar a ser muy costoso clasificar nuevas instancias.

Knn asume que todas las instancias o datos corresponden a puntos en un plano de n dimensiones. Los vecinos más cercanos de cada instancia son calculados mediante

distancias euclidianas. De esta manera la distancia entre dos instancias x_i y x_j se define como:

$$d(x_i, x_j) = \sqrt{\sum^{n} (x_i - x_j)^2}$$

- Donde x_i y x_j son los puntos considerados para calcular la distancia euclidea.
- Siendo *n* el número de atributos a considerar.

Después de calcular la distantica con cada muestra de entrenamiento nos quedaremos con K muestras más cercanas. Esta implementación puede ser optimizada si pesamos la contribución que hace cada uno de los k vecinos más cercanos dándole, de esta forma, mayor peso al vecino más cercano. La mayoría de las variaciones del algoritmo KNN consideran sólo los k vecinos más cercanos para clasificar la nueva instancia, pero al añadirles peso a los datos del conjunto de entrenamiento el algoritmo considerará todas las muestras. Permitir que todas las muestras tengan influencia en el clasificador no supone un peligro para el algoritmo ya que las muestras más distantes apenas tendrán peso. Esta modificación hace que el algoritmo sea más robusto y efectivo cuando el conjunto de datos de entrenamiento es muy grande, además puede suavizar el impacto que tienen los datos de entrenamiento más aislados. El problema de esta modificación es que al considerar todas las muestras el algoritmo se vuelve más lento.

El equipo de desarrollo decidió elegir como algoritmo de clasificación el K-Nearest Neighbors ya que es un algoritmo cuyo coste de aprendizaje es nulo, no necesita hacer ninguna suposición sobre los conceptos a aprender y es muy tolerante al ruido.

Implementación

Teniendo en cuenta la investigación y las decisiones tomadas a lo largo del desarrollo, descritas en el apartado anterior, veremos a lo largo de este apartado en mayor detalle cómo es la implementación de la aplicación.

Nuestra aplicación basa su funcionamiento en una arquitectura cliente-servidor, REST [43], siendo el cliente nuestro *Smartphone* y, el servidor, el proporcionado por la Universidad Complutense de Madrid.

Esta arquitectura consiste en el envío de peticiones HTTP desde el cliente hacia el servidor y, en consecuencia, en el envío de respuestas del servidor hacia el cliente. Esta arquitectura permite tener varios clientes conectados al mismo servicio al unísono y un mantenimiento de la consistencia de la base de datos ya que, debido a estar situada en el propio servidor, es común a todos los clientes; por tanto, permite procesar la información de un modo distribuido. [20]

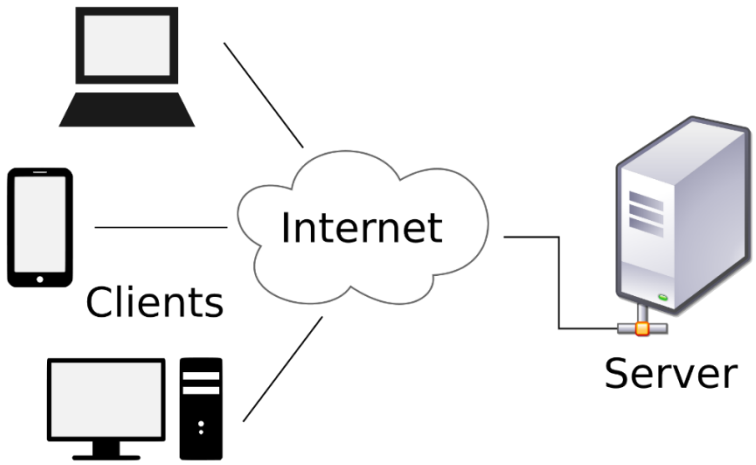

Figura 7. Esquema de funcionamiento de la arquitectura cliente-servidor.

Tecnologías y SO

- *Debian GNU/Linux 8 (jessie)*
- *Windows 10 10.0*
- *mysql Ver. 14.14 Distrib 5.6.28, for Linux (x86_64) using EditLine wrapper*
- *Matlab R2018b Update 2(9.5.0.1033004)*
- *Android Studio 3.4.1*
- *JRE: 1.8.0_152-release-1343-b01 amd64*
- *JVM: OpenJDK 64-Bit Server VM by JetBrains s.r.o*
- *Eclipse IDE for Enterprise Java Developers. Version: 2018-12 (4.10.0)*
- *Postman v7.1.1*
- *Sublime Text 3.2.1*
- *Github v2.17.0*

Cliente

La parte cliente del sistema corresponde a la aplicación Android en sí misma. El código fuente de ésta consta, prácticamente en su totalidad, de código Java. También contiene archivos xml para configuración y *layouts*, incluido el AndroidManifest.

Java

La arquitectura elegida para desarrollar el código en Java ha sido una arquitectura multicapa basada en el patrón Modelo-Vista-Controlador (MVC) [21]. La ventaja de este

tipo de arquitecturas es el aumento de la mantenibilidad, la modularidad y el desacoplamiento del software, así como proporcionar una base para el desarrollo.

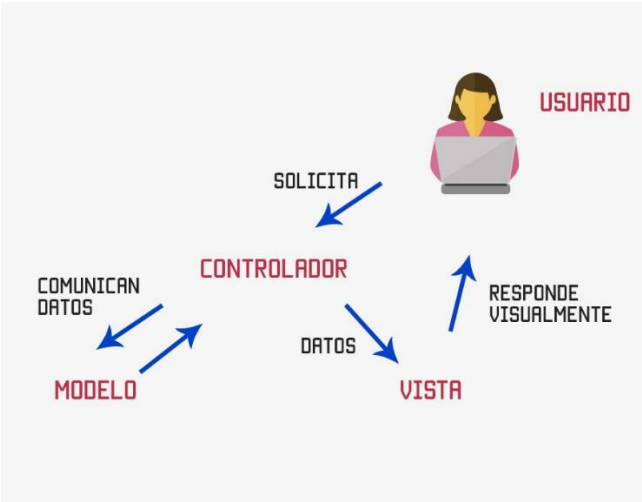

Figura 8. Esquema de funcionamiento de la arquitectura MVC

Esta arquitectura separa y desacopla el funcionamiento de las partes de la aplicación en tres módulos:

- **Modelo:** sus componentes contienen la representación de los datos del sistema, la lógica de negocio, los mecanismos de acceso y persistencia de dichos datos.
- **Vista:** la vista tiene como objetivo principal manejar las interacciones entre la aplicación y el usuario. Las vistas contienen toda la información de valor para el usuario, por lo tanto, su principal cometido es mostrar de forma intuitiva y clara dicha información.
- **Controlador:** se encarga de manejar las comunicaciones entre el modelo y la vista; actúa como un *middleware* entre los dos.

En cuanto a la separación física de los componentes en paquetes, la estructura consiste en tres capas diferenciadas:

- **Presentación:** todo lo relacionado con los elementos gráficos que el usuario ve en la pantalla de su móvil, así como las interacciones entre dichos elementos y con el usuario de la aplicación. El funcionamiento de la interfaz gráfica en Android se basa en *activities*, y las interacciones se realizan mediante *intents*. Sobre este funcionamiento hablaremos más adelante. Esta capa contiene la interfaz gráfica y el controlador, que comunica el modelo con la vista.

  Dentro de esta capa encontramos varias divisiones:

  - *Activities*, componentes de la interfaz gráfica; hablaremos de ellos más adelante.
  - Comandos, que son usados por el controlador para llevar a cabo un comportamiento u otro.
  - Controlador, que recibe las peticiones del usuario y llama al *dispatcher* con el resultado para actualizar la vista.
  - *Dispatcher*, se encarga de actualizar la vista según el resultado del comando que ha ejecutado previamente el controlador.

  La comunicación entre estas divisiones se puede ver en el diagrama de clases descrito en la Figura 9.

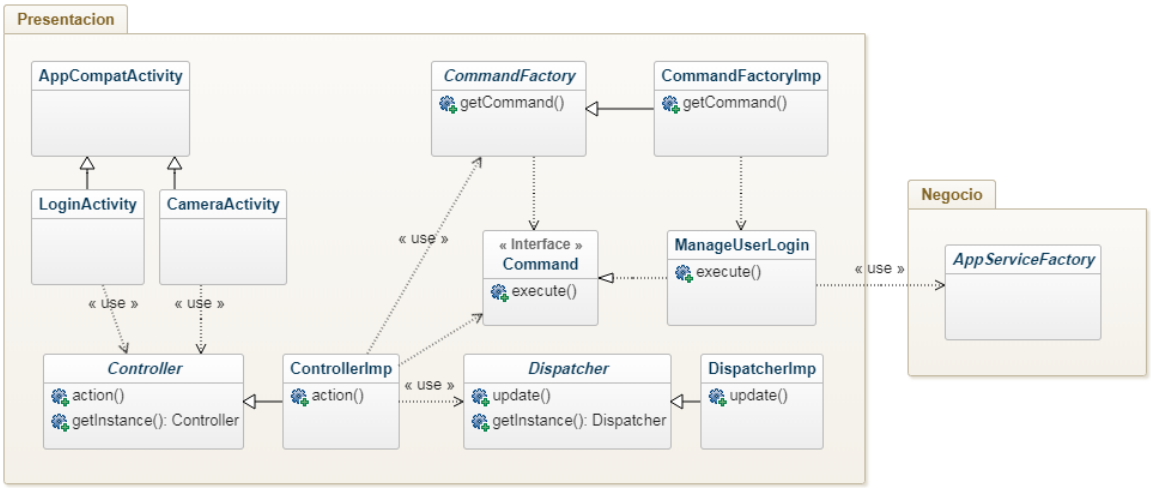

Figura 9. Diagrama de clases simplificado de la capa de presentación.

- **Negocio:** esta capa se encarga de toda la lógica de negocio de la aplicación y la representación de los datos del modelo. La capa de presentación se comunica con esta capa de negocio a través del controlador, dependiendo de la acción del usuario en la interfaz gráfica, el controlador realiza una operación u otra y, con la respuesta, se comunica de nuevo con la vista para actualizarla como proceda.

  Dentro de esta capa encontraríamos los distintos servicios de aplicación de las distintas funcionalidades de la aplicación. El uso de servicios de aplicación nos permite centralizar la lógica del negocio y proporciona una capa de servicio uniforme. Se han implementado los siguientes servicios de aplicación:

  - UsersAppService: realiza todas las operaciones relacionadas con el usuario como el registro/inicio de sesión, actualización de datos del usuario como su peso y contraseña o la bajada de la lista de alimentos consumidos por el usuario, así como sus dosis de insulina registradas.

- ImageAppService: realiza todas las operaciones relacionadas con la segmentación y clasificación de las imágenes ejecutando los algoritmos kmeans y knn previamente descritos.

- FoodAppService: realiza todas las operaciones relacionadas con la información nutricional de cada foto subida proporcionando los informes necesarios para que la vista muestre la información que precisa. También se encarga de la búsqueda de alimentos en la base de datos de USDA y su posterior inserción en nuestra propia base de datos.

Dentro de la capa de usuario, también se ha añadido una clase estática UserSession que permite consultar, sin acceder a base de datos, los datos del usuario que ha iniciado sesión.

Además, se ha implementado la clase *AppServiceFactory* aplicando el patrón de creación *Singleton* que garantiza que cada clase anteriormente mencionada tenga únicamente una instancia proporcionando así un punto de acceso global a ella.

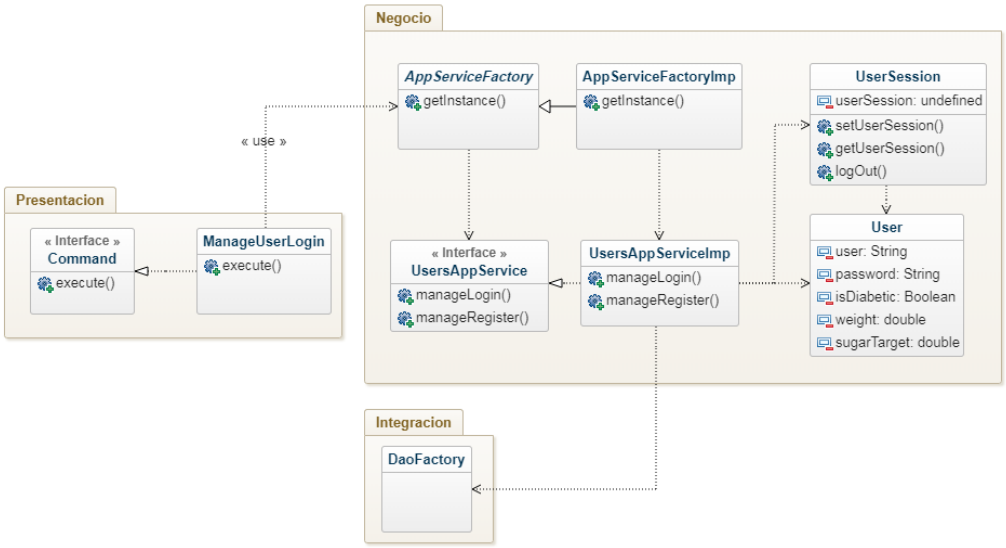

Figura 10. Diagrama de clases simplificado de la capa de negocio.

- **Integración:** contiene todos los componentes que se encargan de acceder y/o persistir los datos en una base de datos. En el caso de nuestra aplicación, esta capa se encarga de mandar peticiones al servidor para almacenar o pedir datos de la base de datos externa. La capa de negocio se comunica con la de integración a través del *AppService* después de haber aplicado la lógica de negocio necesaria para dicha comunicación.

  Para encapsular la manipulación y acceso a los datos en una capa separada se ha usado el patrón DAO (*Data Access Object*). Se han implementado distintos DAOs con distintos objetivos:

  - UserDao: accede y almacena los datos relacionados con el usuario como su email o contraseña. También añade y consulta información sobre las comidas y dosis de insulina recientes del usuario.

- ImageDao: accede y almacena los datos relacionados con las imágenes como las medias identificativas de rojo, verde y azul de cada centro encontrado en una imagen.

- FoodDao: accede a los datos relacionados con los informes nutricionales de los alimentos buscados. Además, permite añadir datos sobre nuevos alimentos y consultas a la base de datos de USDA.

Adicionalmente, esta capa contiene una clase HttpConnection que permite a los diferentes DAOs mandar peticiones GET/POST a las diferentes APIs utilizadas en esta aplicación.

Del mismo modo que en la capa de negocio se ha implementado en la capa de integración la clase *DaoFactory* aplicando el patrón de creación *Singleton*.

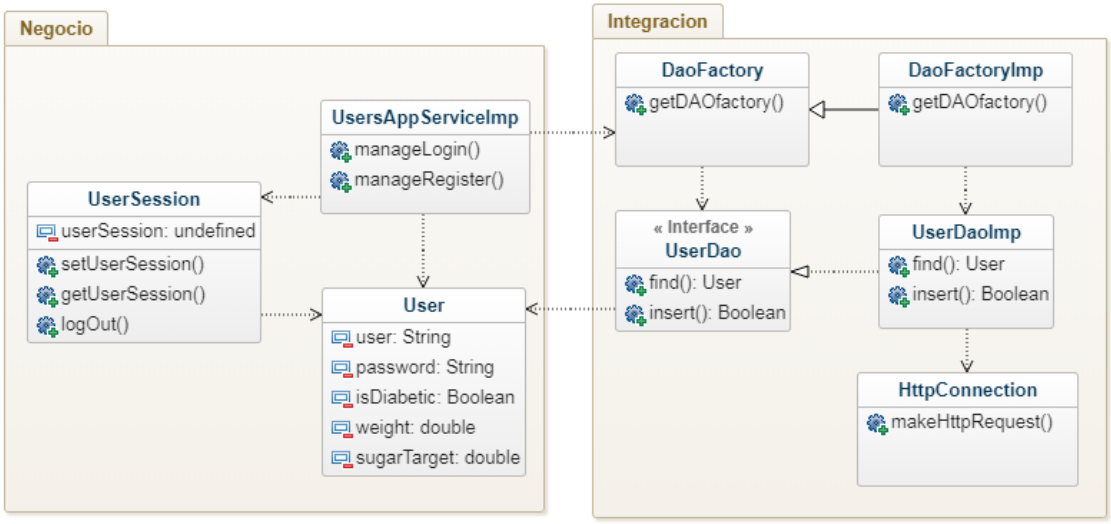

Figura 11. Diagrama de clases simplificado de la capa de integración.

Android Manifest

AndroidManifest.xml [22] es un archivo de formato xml que contiene la configuración del funcionamiento de la aplicación; esto incluye: el nombre de la aplicación, el nombre del paquete Java para la aplicación que sirve como identificador único para la aplicación, componentes de la aplicación, como las actividades, los servicios, los receptores de mensajes, y los proveedores de contenido, declara los permisos que debe tener la aplicación a las partes protegidas de una API o con el resto de los componentes de la aplicación, el nivel mínimo de API que requiere la aplicación y las bibliotecas con las que debe estar vinculada.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="es.ucm.carbocuenta">

    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.GET_ACCOUNTS" />
    <uses-permission android:name="android.permission.READ_PROFILE" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />

    <uses-feature
        android:name="android.hardware.camera"
        android:required="true" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="Carbocuenta"
        android:networkSecurityConfig="@xml/network_security_config"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".presentacion.activities.user.InsulinHistoryActivity"></activity>
        <activity android:name=".presentacion.activities.user.UpdateUserActivity" />
        <activity android:name=".presentacion.activities.user.UserActivity" />
        <activity android:name=".presentacion.activities.food.FoodExplorerActivity" />
        <activity android:name=".presentacion.activities.food.FoodReportActivity" />
        <activity android:name=".presentacion.activities.food.FoodSelectorActivity" />
```

Figura 12. Extracto del archivo AndroidManifest.xml

Gradle

Es un plugin que permite compilar y actualizar un proyecto en función de sus dependencias y librerías. Este sistema "está basado en *Java Virtual Machine* y contiene las mejores prestaciones de otros sistemas de compilación, por lo que permite facilitar este proceso y exportar el proyecto de una forma sencilla, con el fin de poder distribuirlo" [23].

En este caso el *script de Gradle* generará un APK que contendrá la siguiente información:

- ***Plugin:*** este proporciona todo lo necesario para construir y probar una aplicación.

```
apply plugin: 'com.android.application'
```

Figura 13. Plugin del archivo build.gradle

- ***Android:*** configura los parámetros para la construcción de Android. Es primordial que incluya la propiedad compileSdkVersion para poder realizar la compilación, en este caso se ha utilizado la versión 28. Además, incluye un apartado para la configuración de pruebas y metadatos.

```
android {
    compileSdkVersion 28
    defaultConfig {
        applicationId "es.ucm.carbocuenta"
        minSdkVersion 24
        targetSdkVersion 28
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
        multiDexEnabled true
    }
    packagingOptions {
        exclude 'META-INF/DEPENDENCIES'
        exclude 'META-INF/LICENSE'
        exclude 'META-INF/LICENSE.txt'
        exclude 'META-INF/license.txt'
        exclude 'META-INF/INDEX.LIST'
        exclude 'META-INF/NOTICE'
        exclude 'META-INF/NOTICE.txt'
        exclude 'META-INF/notice.txt'
        exclude 'META-INF/ASL2.0'
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }
}
```

Figura 14. Android del archivo build.gradle

- ***Dependencies:*** incluye todas las librerías que va a necesitar la aplicación para poder compilar

```
dependencies {
    androidTestImplementation 'com.android.support.test.espresso:espresso-core:3.0.2'
    implementation 'com.android.support.constraint:constraint-layout:1.1.3'
    implementation 'com.android.support:appcompat-v7:28.0.0'
    implementation 'com.android.support:support-v4:28.0.0'
    implementation 'com.android.support:design:28.0.0'
    implementation 'com.android.volley:volley:1.1.1'
    implementation 'com.jjoe64:graphview:4.2.2'
    implementation fileTree(include: ['*.jar'], dir: 'libs')
    testImplementation 'junit:junit:4.12'
    api 'com.google.guava:guava:27.1-android'
}
```

Figura 15. Dependencies del archivo build.gradle

Activities, Intents y *Layouts*

Hablábamos antes de las *activity* [24] como una parte importante del funcionamiento de la interfaz gráfica de Android. Las *activities* son componentes de la aplicación que contienen una representación gráfica con la que los usuarios pueden interactuar para realizar una acción concreta

Dependiendo de la acción que se dispare con la interacción del usuario, puede ser que se necesite iniciar otra *activity*, mandar un mensaje o iniciar un servicio concreto. Para este fin se utilizan objetos *intent* [25], que permite comunicar y hacer peticiones desde un componente de la aplicación hacia otro.

En nuestro caso, se han implementado varias *activities:*

- ***AddAlimentActivity:*** permite al usuario añadir alimentos de la base de datos internacional USDA, pidiendo un nombre y una cantidad (gramos) para poder añadir un ingrediente al alimento que pretende registrar. Una vez añadidos los ingredientes, le ofrece la opción de registrar el plato o cancelar el proceso.

- ***CameraActivity:*** maneja las peticiones relacionadas con la cámara: tomar fotos, ver la *preview* de la foto tomada y realizar la petición de procesamiento de la foto. Esta *activity*, a su vez, está dividida en dos *fragments*. Un *fragment* [26] es un fragmento del comportamiento y/o interfaz de usuario de una *activity*.
  En nuestro caso, dividimos la interfaz de usuario y el comportamiento en dos: *CameraFragment*, encargado de mostrar lo que ve la cámara en tiempo real y de tomar la foto, y *UploadFragment*, encargado de visualizar la foto tomada y mostrar botones para que el usuario interactúe con ella.

- ***CorrectiveBolusActivity:*** da la opción al usuario diabético de calcular el bolo corrector, permitiéndole ingresar su dosis actual, ver y registrar la nueva dosis correctora.

- ***CropActivity:*** permite al usuario hacer un recorte personalizado de una comida en una imagen tomada o seleccionada previamente para posteriormente añadir ese plato a la base de datos si es que no existe.

- ***CropImageActivity:*** muestra al usuario el recorte que ha realizado a una imagen previamente y le permite añadir un nombre personalizado al plato que pretende subir, dándole la opción de continuar con la subida si está conforme con el recorte.

- ***FoodExplorerActivity:*** encargada de mostrar en una lista de selección múltiple las predicciones de los distintos alimentos encontrados en el plato.

- ***FoodSelectorActivity:*** encargada de mostrar en una nueva lista los alimentos seleccionados en la *activity* anterior mostrando su nombre, la cantidad en gramos y los carbohidratos correspondientes. Esta *activity* además amplía su funcionamiento mediante el *fragment FoodItemFragment* que usa un adaptador, *FoodAdapter*, para poder personalizar visualmente la lista.

- ***FoodReportActivity:*** se encarga de recoger y mostrar los informes de los alimentos mostrados en la *activity* anterior mostrando sus nutrientes indicando la cantidad aportada y la cantidad diaria recomendada.

- ***FoodUserActivity:*** encargada de recoger y mostrar las últimas 30 comidas del usuario, mostrando un breve resumen de los componentes del plato y la información de estos componentes como los gramos, los carbohidratos y la insulina que haya decidido el usuario calcular en esa comida.

- ***InsulinHistoryActivity:*** muestra un resumen de las dosis de insulina del día, una media de dosis diaria de los últimos días y una gráfica que muestra la cantidad de dosis en los días anteriores.

- ***InsulinParametersActivity:*** permite al usuario ingresar sus datos para poder calcular el bolo prandial de la comida que haya procesado previamente.

- ***LoginActivity*****:** maneja todas las peticiones del usuario relacionadas con el inicio de sesión o el registro de usuario. También se encarga de mostrar errores en caso de inicio de sesión incorrecto o fallo de conexión.

- ***MenuActivity:*** ofrece un menú al usuario que le permite añadir una comida ya procesada a sus comidas, obtener un reporte general y/o específico de los nutrientes de cada alimento o calcular la insulina de esa comida.

- ***UpdateUserActivity:*** gestiona la recogida, validez y modificación de los datos que el usuario quiera cambiar sobre su cuenta, es decir, si es diabético, su peso y su azúcar en sangre, hipoglucemia, glucosas objetivo, además, de la contraseña y la fecha de nacimiento, parámetro común de todos los usuarios.

- ***SelectorAlimentActivity:*** muestra una lista de los alimentos que componen el plato que ha procesado el usuario, informando de los gramos y carbohidratos que componen cada alimento y dando la opción de seleccionar alguno para obtener un reporte nutricional más específico sobre el alimento.

- ***UserActivity:*** maneja las peticiones del usuario relacionadas con el cierre de sesión.

La representación gráfica de una *activity* se define mediante un archivo *layout*. Los *layouts* son archivos con formato xml que describen una representación gráfica concreta, entendiendo por esto los componentes de una interfaz gráfica, sus identificadores, su composición y su posición dentro del marco de la interfaz.

Generación de la Base de Conocimientos

La base de conocimientos es aquella con la cual el algoritmo, en este caso el Knn [20], utiliza unos datos registrados en esta base para modelar y entrenar un conjunto de soluciones con las que, finalmente, se obtenga un conjunto de resultados semejantes al que se ha solicitado. Para este proceso se ha utilizado *Google Drive*, *Matlab*, *Eclipse* (*Java*) y la BD.

Esta base de conocimientos ha sido generada gracias a un repositorio de imágenes, tanto de la cafetería como personales, creado en *Google Drive* con un peso total de 726MB.

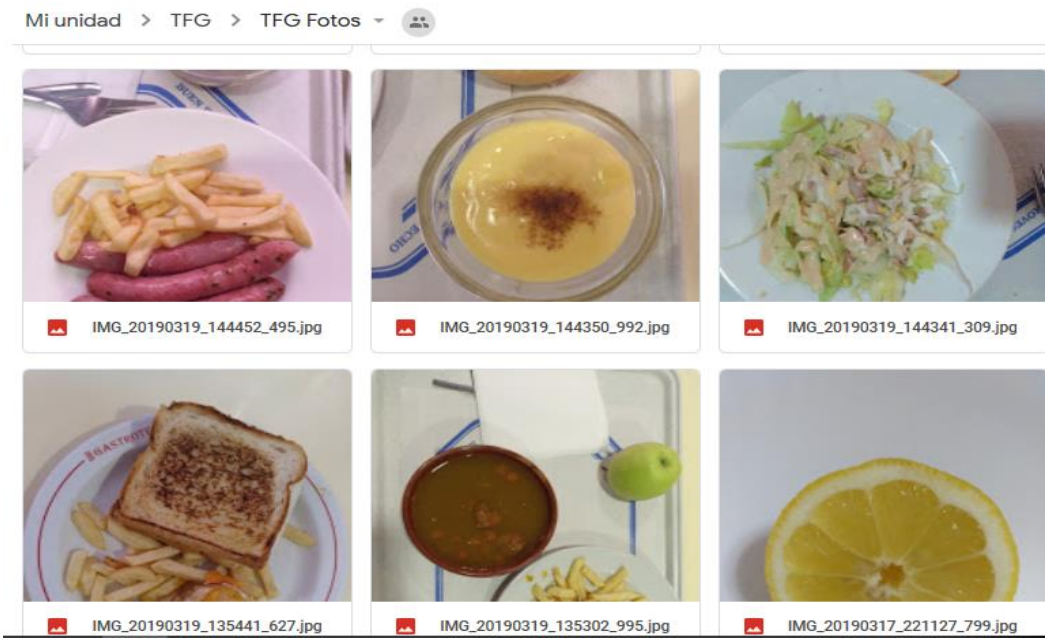

Figura 16. Imágenes de Google Drive

En un principio, se utilizó para la segmentación de la imagen el algoritmo *KMeans* de *Matlab* [27], con el que separábamos los alimentos del plato según los *clusters* generados por el algoritmo y representábamos esos alimentos en una imagen aparte generando también un documento CSV en el que se registraban las medias de rojo, verde y azul junto a los pixeles totales de cada *cluster*.

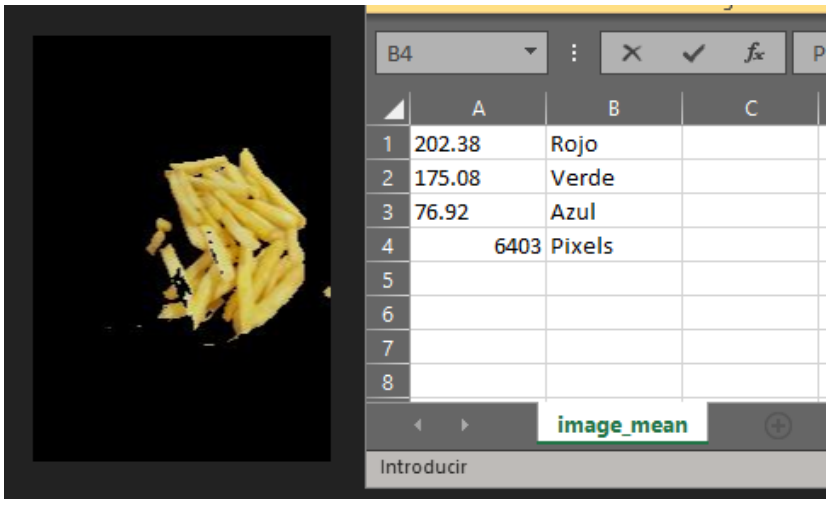

Figura 17. Alimento para base de conocimientos

Como se puede apreciar en la Figura 11, para obtener las medias, este equipo ha segmentado la imagen utilizando el algoritmo *KMeans* de *Matlab* y creando una capa para cada parte, en función del número de *cluster* y así, posteriormente, calcular las medias RGB de cada sección y corroborar que se asemeja a las medias obtenidas por nuestro algoritmo *KMeans*, implementado en *Java*, con el fin de poder insertarlas en la base de datos.

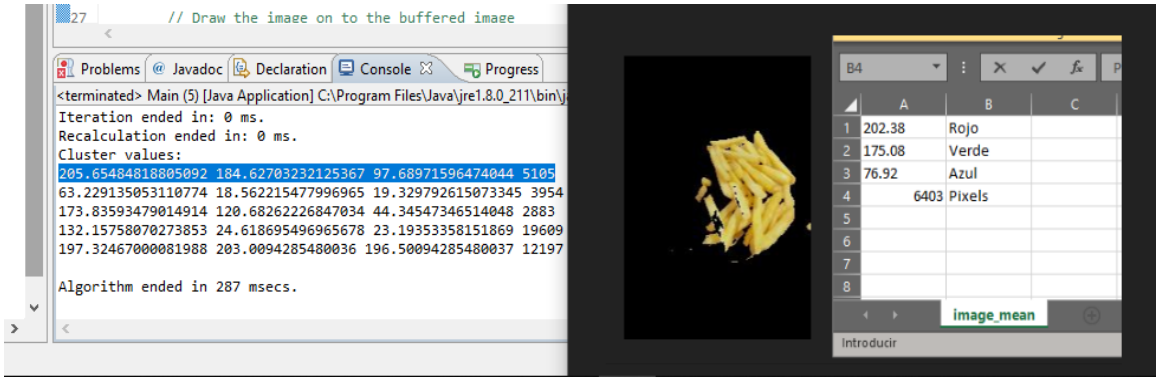

Figura 18. Comparación de KMeans de Java con Matlab

Como anotación, mencionar que las medias no tienen que ser exactamente iguales sino parecidas, ya que el algoritmo de *Java*, ha sido implementado por el equipo realizando una segmentación parecida a la de *Matlab*, pero no idéntica, debido a que la de *Matlab* es mucho más precisa y tiene ventaja en cuanto al avance algorítmico y al procesamiento de imágenes; aun así, la diferencia entre un algoritmo y otro ha sido de unos 25 puntos de media por encima con respecto a la implementación de *Matlab*.

id_aliment ▲ 1	aliment	red_mean	green_mean	blue_mean	pixels
1	Carne en salsa de tomate	132.6	24.5	21.5	19646
2	Filete de ternera a la plancha	135.011	64.256	68.6733	5513
3	Pescadilla	30.552	27.097	18.878	7672
4	Huevo frito	15.351	53.853	12.145	3322
5	Zanahoria	128.238	23.159	21.239	5348
6	Croquetas	151.1124	80.157	37.125	5180
7	Entrecotte	13.726	92.039	71.924	4786
8	Pisto	24.373	14.79	65.601	7239
9	Patatas fritas	220.548	187.48	108.124	6403
10	Pollo asado	129.5	88.312	43.634	10104

Figura 19. Base de conocimientos para Knn

Esta aplicación no guarda las imágenes en la BD, ya que abarcaría demasiado espacio para información irrelevante, debido a que cada usuario puede tener imágenes de la misma comida con los mismos ingredientes y, por tanto, se repetiría la misma fotografía para muchas personas. Por eso, se queda con las medias RGB para la base de conocimientos y con los pixeles para saber la cantidad de comida que hay en el plato. Esto es posible ya que la foto sigue siempre el mismo "estándar", es decir, una imagen tomada desde arriba al plato con los componentes y el menor fondo posible, como se puede observar en la Figura 20. Si la imagen que recibe la aplicación se ha hecho desde esta perspectiva, da igual la cámara del dispositivo o la escala a la que se haya producido la fotografía que siempre se realizará un *resize* a nivel de sistema, para que la variación del número de pixeles sea mínima y poder concretar mejor la cantidad de gramos que existe de cada alimento, asignando los gramos en función de los pixeles recogidos y los pixeles de los que consta ese alimento en la BD.

Figura 20. Base de conocimientos para Knn

Finalmente, cabe destacar que para que el usuario pueda añadir alimentos a la base de conocimientos que puedan compartir plato con otros que sí que están añadidos, facilitamos una vista para que se saltara el paso de segmentación y pudiera recortar la sección de manera personalizada y facilitar la obtención de medias sin mezclar información con otros elementos de la imagen.

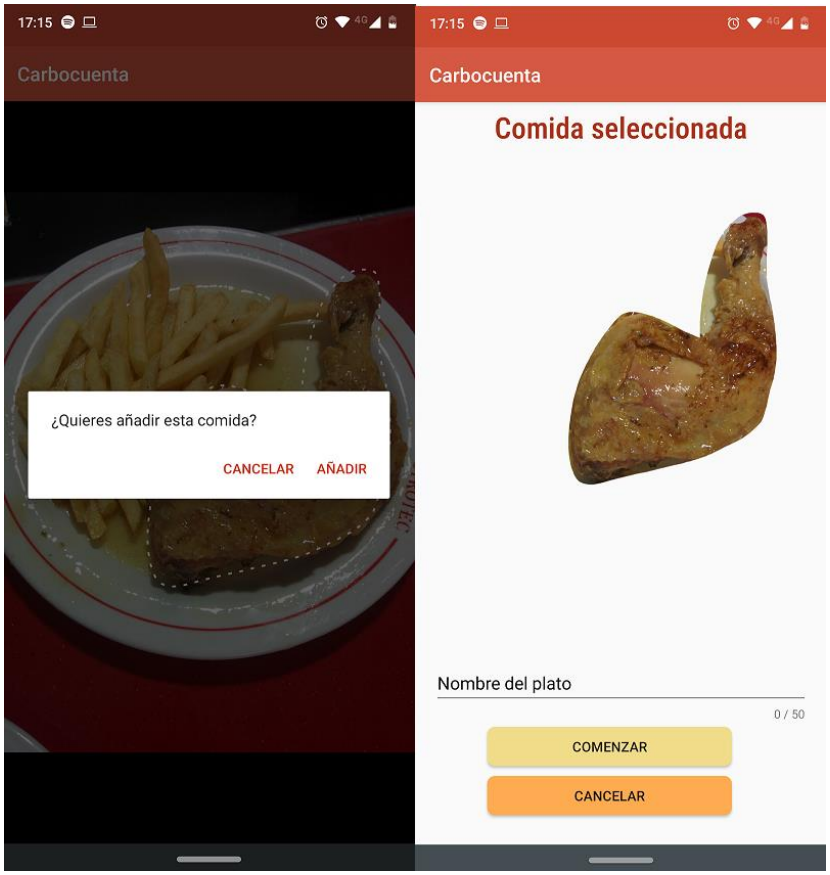

Figura 21. CropActivity, recortar alimento de una imagen

K – Nearest Neighbor

La correcta generación de la base de conocimiento es indispensable para un funcionamiento adecuado del clasificador. La implementación de este es extensa y hace uso de distintas clases que serán explicadas detalladamente a continuación.

La clase principal *KnnClassifier* es la encargada de llevar a cabo la clasificación, cuenta con varios parámetros, los cuales son necesarios explicar para comprender el funcionamiento de la clase:

- *Knn*: es la constante usada para definir el número de *k* vecinos.
- *Threshold*: representa el porcentaje de umbral permitido pudiendo valer entre 0 y 1. Nos ayudará a decidir si un punto en el lado límite pertenece a una clase u a otra.

Figura 22. Código de la clase KnnClassifier

El primer paso que realiza el clasificador es calcular las distancias euclídeas entre los puntos a clasificar, en este caso los clústeres, y las distintas categorías. Para ordenar los resultados se hace uso de otra clase, *KnnQueue,* en la que se ha implementado una cola que ordena las posibles categorías a las que podría pertenecer el *cluster* por prioridad, es decir, de menor a mayor distancia euclídea.

Figura 23. Código de la clase KnnQueue

Para guardar los resultados en la cola se hace uso de la clase *KnnQueueElement* donde se guarda el índice, la distancia y el peso de cada *cluster*.

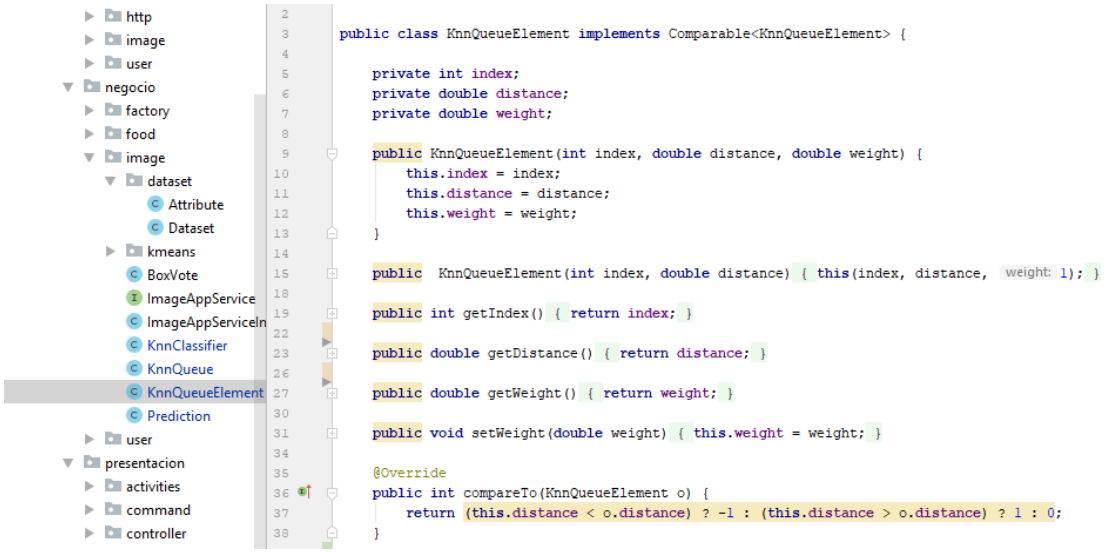

Figura 24. Código de la clase KnnQueueElement

Una vez se ha llenado la cola de prioridad adjudicamos un peso a cada categoría iterando sobre esta. El peso es calculado a proporción de la distancia. Si una de las distancias calculadas es demasiado grande se le adjudicará un peso igual a $1.8 * 10^{308}$, es decir, el mayor valor que se puede guardar en un doublé sin perder precisión. De esta forma dicha categoría será colocada al final de la cola de prioridad evitando así que el clasificador la tenga en cuenta.

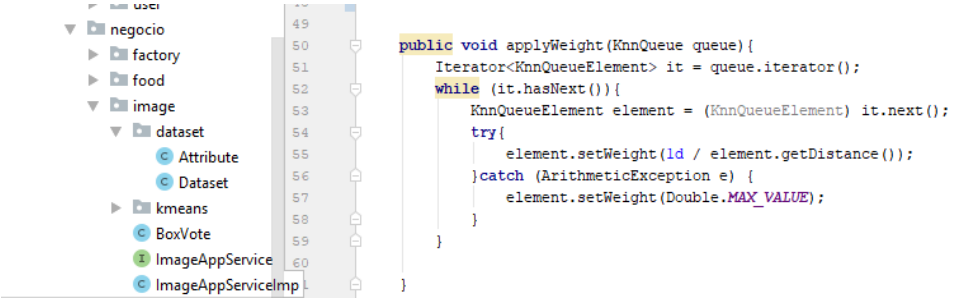

Figura 25. Método de cálculo de pesos

Por último, usaremos la clase *BoxVote*, dicha clase usa un sistema de votos para predecir la categoría correcta. Adjudica una mayor cantidad de votos según sea la distancia, según su posición en la cola y teniendo también en cuenta el umbral para adjudicar una mayor cantidad de votos a una categoría u otra.

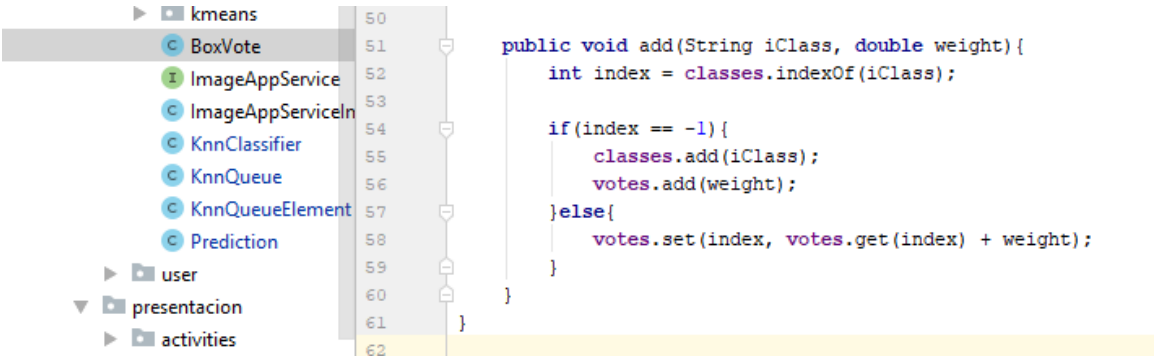

Figura 26. Método de votos de la clase BoxVote

De esta forma el clasificador es capaz de predecir las categorías más probables. Debido a que se muestran varias opciones para cada uno de los centros se crea un array de *Predictions*, cada elemento lleva el número de pixeles que forman el cluster y un array de *Strings* con las distintas predicciones para este.

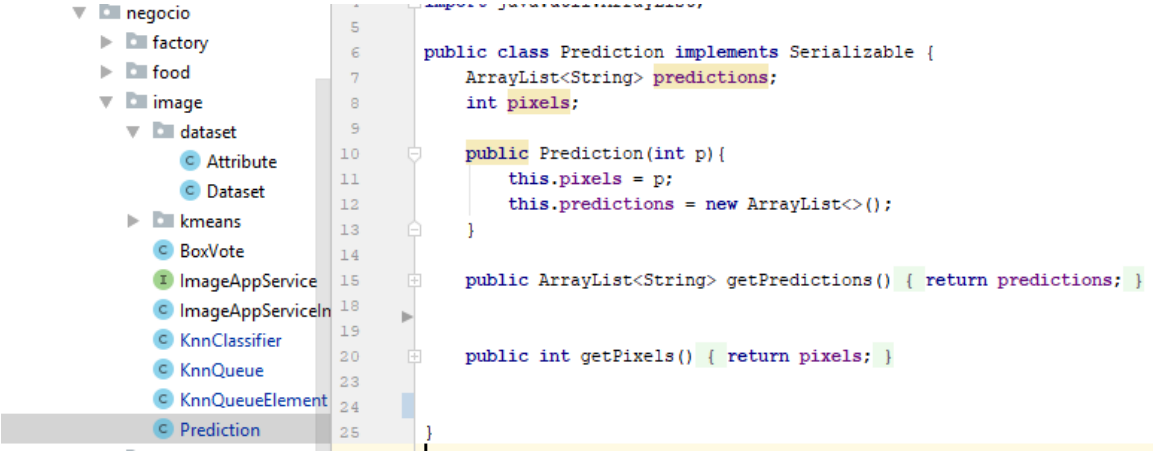

Figura 27. Código de la clase Prediction

API de USDA

La *National Agricultural Library* [28] del Departamento de Agricultura de los Estados Unidos (*United States Department of Agriculture*) ofrece variadas y extensas bases de datos, concretamente las *USDA Food Composition Databases*, que contienen información nutricional de gran cantidad de alimentos, ya sean naturales o fabricados, procesados o sin procesar.

A estas bases de datos se puede acceder mediante un servicio REST [43] y una clave API gratuita, que sirve simplemente para identificar al usuario que realiza las peticiones al servicio y para evitar que colapse el servicio si realiza una inundación de llamadas (actualmente, cada clave API puede realizar hasta 3600 peticiones por hora y por dirección IP).

Esta API permite varios tipos de consulta:

- ***Food Reports*:** son informes de alimentos específicos en los que se lista los valores nutritivos de éstos. Para consultar un *report* necesitas un identificador de *report* que puede ser obtenido mediante consultas de búsqueda.

- ***Nutrient Reports*:** estos informes proporcionan una lista de alimentos y sus valores nutritivos para un conjunto específico de nutrientes. Para obtener uno de estos *reports* necesitas un identificador que se obtiene mediante consultas de búsqueda.

- ***Nutrient List*:** permite buscar nutrientes en la base de datos para obtener el identificador correspondiente de cada *nutrient report*.

- ***Food Search*:** permite buscar alimentos en la base de datos usando parámetros como el nombre, el identificador de grupo de comida, etc. Esta búsqueda permite obtener los identificadores de *food reports* para obtener información más detallada de la comida deseada.

De estas consultas, se utiliza el *food search* y la obtención de *food reports* en base a su identificador.

Como hemos mencionado anteriormente, esta base de datos, al ser estadounidense, está en inglés. Por tanto, para poder coincidir con los datos de búsqueda que el usuario genera en español, se utiliza la API de traducción de Google.

API de traducción de Google

Google ofrece un servicio de traducción a los desarrolladores en su *Google Cloud Platform*. Este servicio permite traducir las cadenas que quieras a cualquiera de los idiomas compatibles [29], incluso sin saber su lenguaje de origen. También permite crear modelos de lenguajes específicos de dominio.

En cuanto a nuestra aplicación, el uso de esta API se basa en traducir los platos de comida que el usuario añade a la base de datos para buscar sus nutrientes en una base de datos externa de Estados Unidos (de la que hablaremos más adelante). Debido a que esta base de datos está en inglés y el input/output del usuario en español, la API debe encargarse de traducir las cadenas.

Referente al uso, esta API actualmente no ofrece servicio directamente a Android, por lo que debe utilizarse haciendo llamadas GET al servicio REST que ofrece. Para ello, hace falta una clave de API que se puede obtener desde la web de *Google Cloud Platform* abonando la tasa correspondiente si se requiere.

Esta llamada GET, además de la clave API, necesita la palabra o palabras que tiene que traducir y el lenguaje al que se quiere traducir. Se puede indicar también el lenguaje origen, pero si no se indica la propia API lo deduce.

Concurrencia, paralelismo e hilos

Estos tres conceptos son determinantes para poder realizar una aplicación en *Android* que funcione de manera rápida y eficiente, por lo que, para la mejora de rendimiento de muchos procesos, este equipo ha hecho uso de la clase *AsyncTask* [30], una clase que proporciona *Android*, para poder realizar hilos que trabajen paralelamente con otros procesos de la aplicación sin que se crucen entre sí, esperando los eventos y respuestas del otro para poder seguir ejecutando la aplicación sin errores. Además, sin ella no sé pueden realizar peticiones a un servidor y/o base de datos.

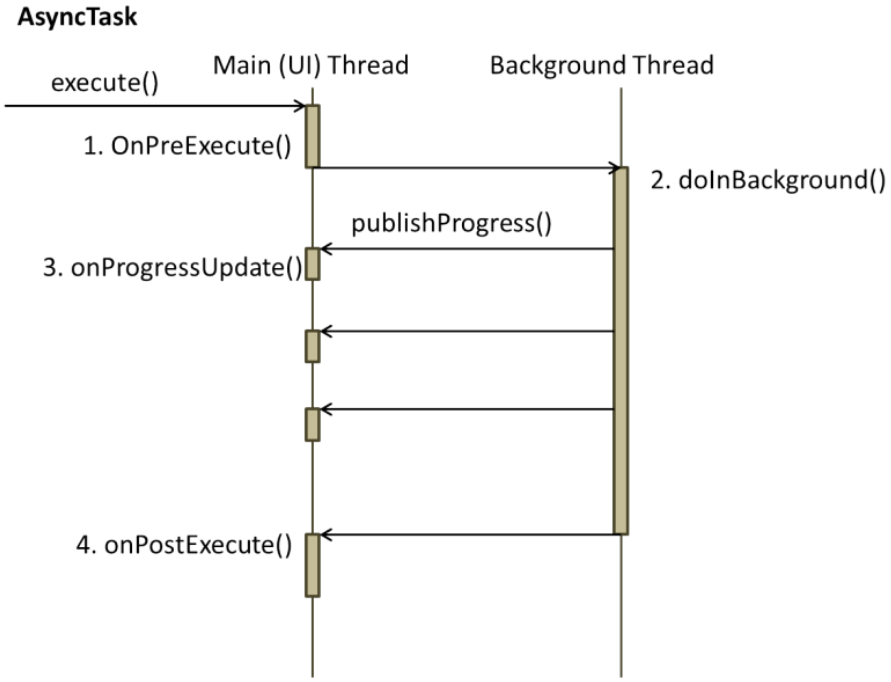

Figura 28. Esquema de ejecución de la clase AsyncTask [30]

Esta clase es utilizada en varias ocasiones en la aplicación, ya que permite crear una concurrencia que nos ayuda a recoger y/o procesar datos que se necesitan o necesitarán para poder ejecutar distintas funcionalidades de la aplicación.

Casos en los que se utiliza *AsyncTask*:

- ***Peticiones al servidor:*** se realizan peticiones HTTP de tipo GET y POST para la inserción o selección de datos de la BD.

- ***Procesamiento de una imagen:*** se procesa la imagen en otro hilo con el algoritmo KMeans y Knn para la obtención de la lista de comidas predecidas.

- ***Comunicación con APIs:*** para la traducción de los términos y la comunicación con las BD USDA.

- ***Cambio de activity con información:*** existen diferentes *activities* en los que se necesita calcular, procesar u obtener datos ya sea de una comida, de una imagen o de un usuario, para poder realizar esos procesos se hace usos de estos hilos.

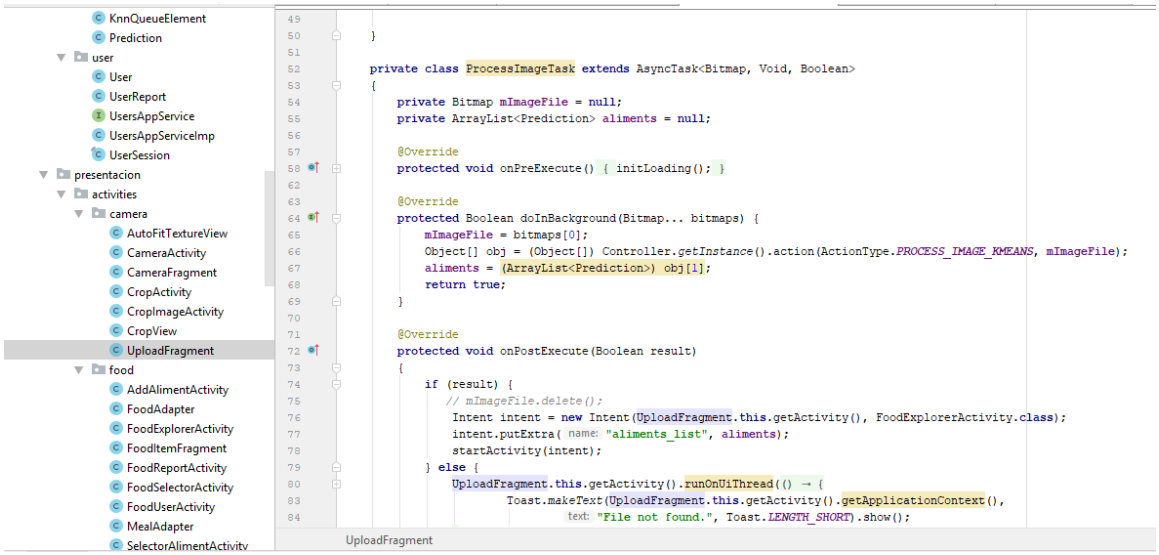

Figura 29. Uso de AsyncTask en la aplicación

Funcionalidades destacadas

En esta aplicación hay que destacar tres módulos principales, los cuales ofrecen al usuario diversas funcionalidades de las que se compone la aplicación.

- ***User:*** este módulo ofrece al usuario las funcionalidades mostradas en la Figura 30.

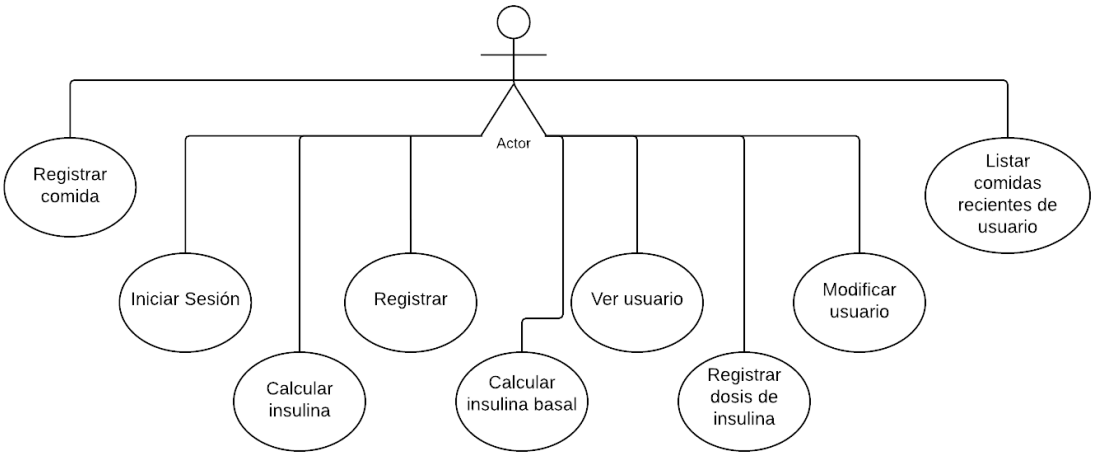

Figura 30. Casos de uso el usuario

Este es el módulo que más casos de uso contiene ya que está relacionado con el módulo *Food* y el equipo ha decidido implementar las operaciones en *User*, ya que al final es quien se considera que está más relacionado con la comida y la insulina, conceptos muy importantes en esta aplicación.

Para el cálculo de la insulina tenemos que diferenciar entre la dosis de insulina basal, que sirve para reemplazar entre el 40% y 50% de la insulina total durante la noche o durante los periodos de ayuno, y la insulina de bolo que representa el tanto por ciento

restante y sirve como cobertura de carbohidratos. A continuación, se explica detalladamente el cálculo de la dosis de bolo prandial, *Ir(t):* [31]

$$\mathrm{Ir(t)} = \mathrm{Ii(t)} + \mathrm{Ic(t)}$$

Donde *Ii(t)* es la insulina necesaria para metabolizar la ingesta de carbohidratos y *Ic(t)* es el valor de la insulina correctora que puede ser positiva o negativa.

Para empezar el cálculo hay que hablar de los siguientes datos:

- *G(t)*: cantidad de glucosa en sangre en el instante t.
- *C(t)*: carbohidratos que se van a ingerir en el instante t.
- *Gpre*: glucosa objetivo antes de la comida.
- *Ghipo:* valor de glucosa considerado como hipoglucemia.
- *R:* ratio insulina-carbohidratos. Es la insulina necesaria para metabolizar una ración de CH.
- *S:* factor de sensibilidad a insulina.
- *P:* pero del paciente.
- Edad del paciente.

Como en este proceso se utilizará el ratio y lo ideal es obtener el histórico de dosis del paciente, se pueden calcular también a partir de las siguientes fórmulas:

$$\text{Requisito diario de insulina} = 0.55 * \text{Peso corporal en kg}$$

$$\text{Cobertura de CHO} = \frac{500}{\text{Requisito diario de insulina}}$$

$$R = \frac{unidades\ de\ insulina}{carbohidratos}$$

Existen 2 condiciones a considerar para poder calcular el bolo prandial correctamente:

- Si *Ghipo* < *G(t)* < *Gpre* la recomendación de insulina para t será:

$$\mathrm{Ir(t)} = \mathrm{Ii} = \mathrm{C(t)} * \mathrm{R}$$

- Si *G(t)* < *Gpre* ó *G(t)* > *Gpre* se calculará tanto *Ii(t)* como *Ic(t)*. En ambos casos *Ii(t)* no varía y vale lo siguiente:

$$\mathrm{Ii(t)} = \mathrm{C(t)} * R$$

Para calcular *Ic*, se necesita el factor de sensibilidad a la insulina *S*, el cual se puede estimar de una forma fiable si la dosis total de insulina diaria está entre 0,4 y 0,8 u/kg/día a partir de la regla del "1800":

$$\mathrm{S} = \frac{1800}{\text{Insulina diaria}}$$

Por tanto, si *G(t)* < *Gpre*, entonces:

$$\mathrm{Ic(t)} = \frac{\mathrm{Gpre} - \mathrm{G(t)}}{\mathrm{S}}$$

Y si *G(t) > Gpre*, entonces:

$$\mathrm{Ic(t)} = \frac{\mathrm{G(t)-Gpre}}{\mathrm{S}}$$

Para que finalmente el bolo prandial sea la suma de la insulina cobertura de CH y la insulina correctora, como se ha mencionado anteriormente.

Por último, especificamos como calcular el bolo corrector, del cual también hay que conocer otros conceptos además de los que ya se han explicado.

- *Gesp(t + tc)*: glucosa esperada en el instante de tiempo t
- *DIA*: duración de la acción de la insulina (*Duration of the Insulin Action)*, su valor por defecto es 0.0182
- *IOB*: la insulina activa (*Insulin On Board)*
- Δ*post*: recomendación de glucosa pasado tc tiempo.
- *u(t)*: unidades de insulina en el instante t

Para empezar a calcular el bolo corrector se empieza por:

$$Gesp(t + tc) = G(t) + \Delta post$$
$$\Delta post = G(tc) - G(t)$$

Se puede simplificar el cálculo de $\Delta post$ de la siguiente manera:

- Si tc < 90 => $\Delta post$ = *G(tc) – G(t)*
- Si 90 ≤ tc < 105 => $\Delta post$ = 60
- Si 105 ≤ tc < 125 => $\Delta post$ = 80
- Si 125 ≤ tc ≤ 140 => $\Delta post$ = 60
- Si tc > 140 => $\Delta post$ = *G(tc) – G(t)*

La insulina activa se calcula a partir de las siguientes ecuaciones:

$$C1(t+1) = u(t) - DIA * C1(t) + C1(t)$$
$$C2(t+1) = DIA * \big(C1(t) - C2(t)\big) + C2(t)$$
$$IOB = C1(t) + C2(t)$$

Tomando como casos base *C1(0) = 0* y *C2(0) = 0*. Así esto permitirá calcular *IOB* de forma iterativa para obtener el *IOB* que nos interesa en el instante t. Con esto, podremos calcular el bolo corrector *Irc*.

$$Irc(t+tc) = \frac{G(t+tc) - Gesp(t+tc)}{S} - IOB$$

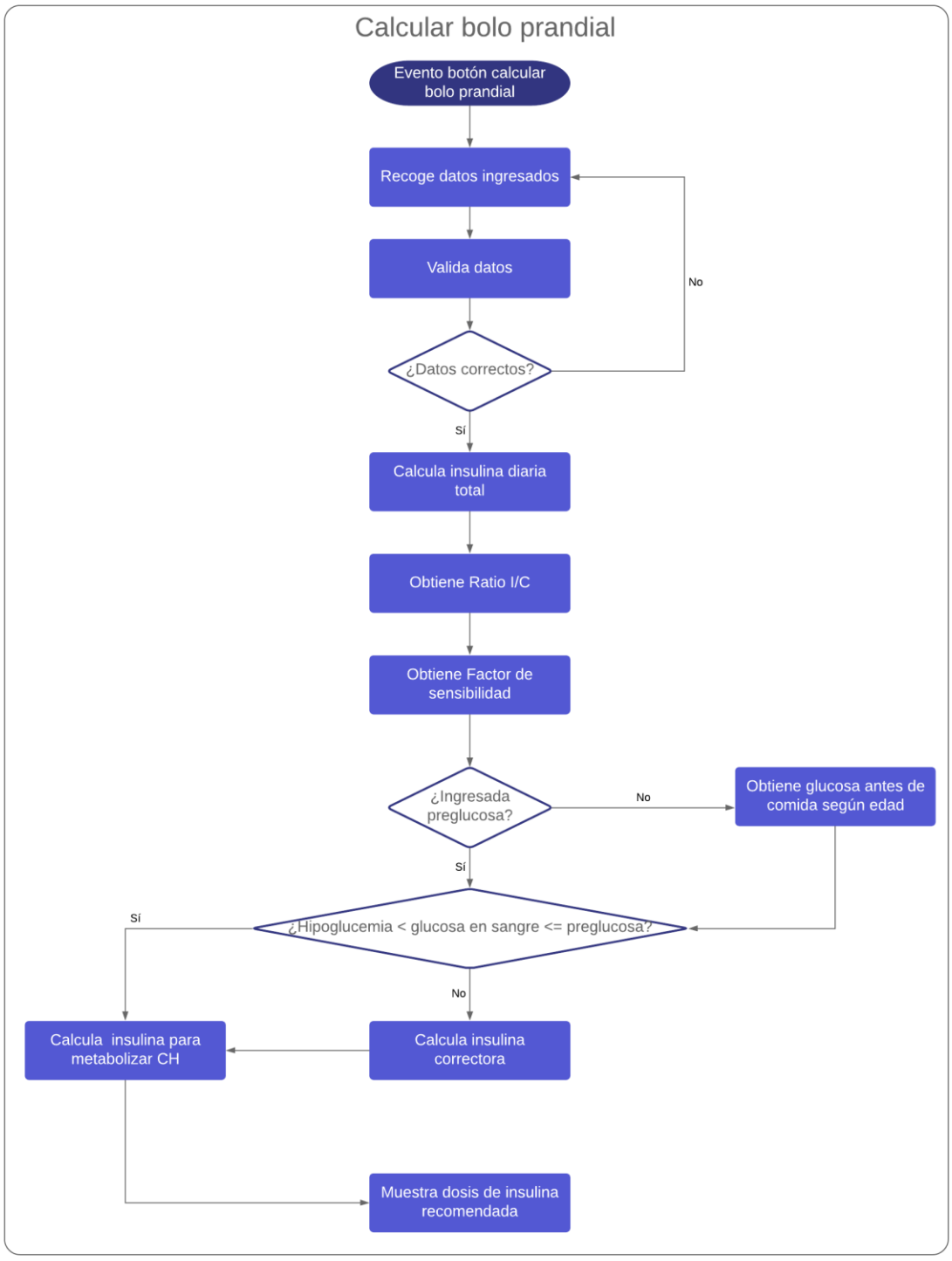

Figura 31. Diagrama de actividad de calcular bolo prandial

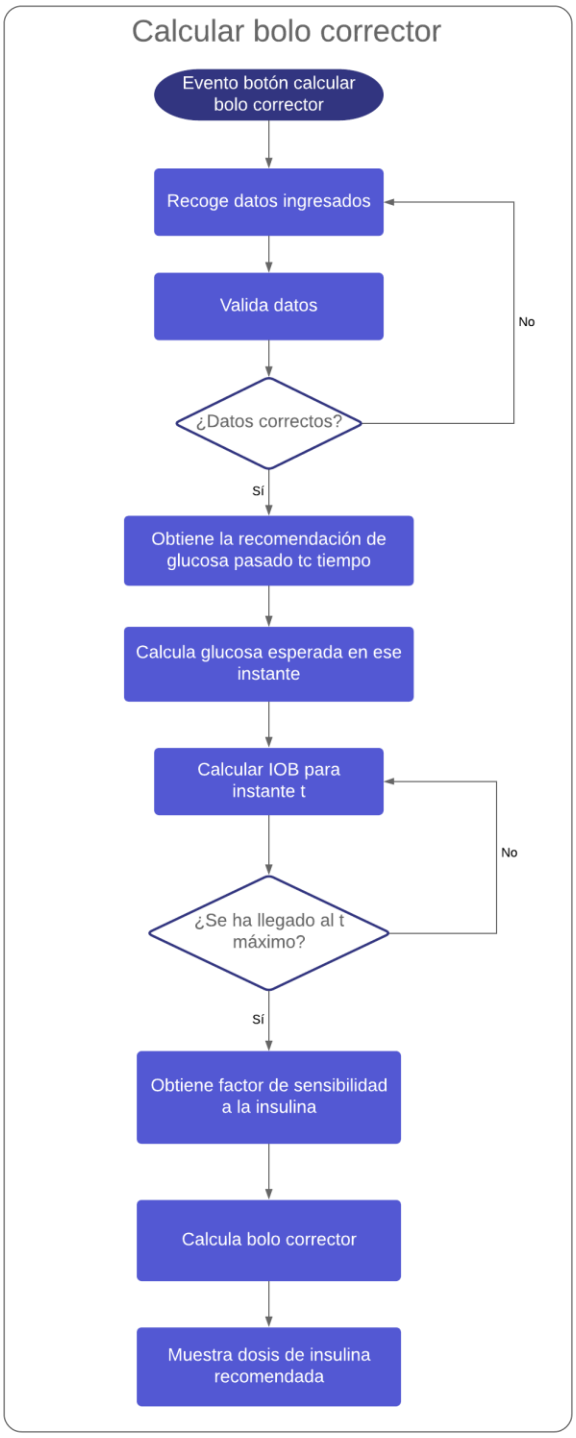

Figura 32. Diagrama de actividad de calcular bolo corrector

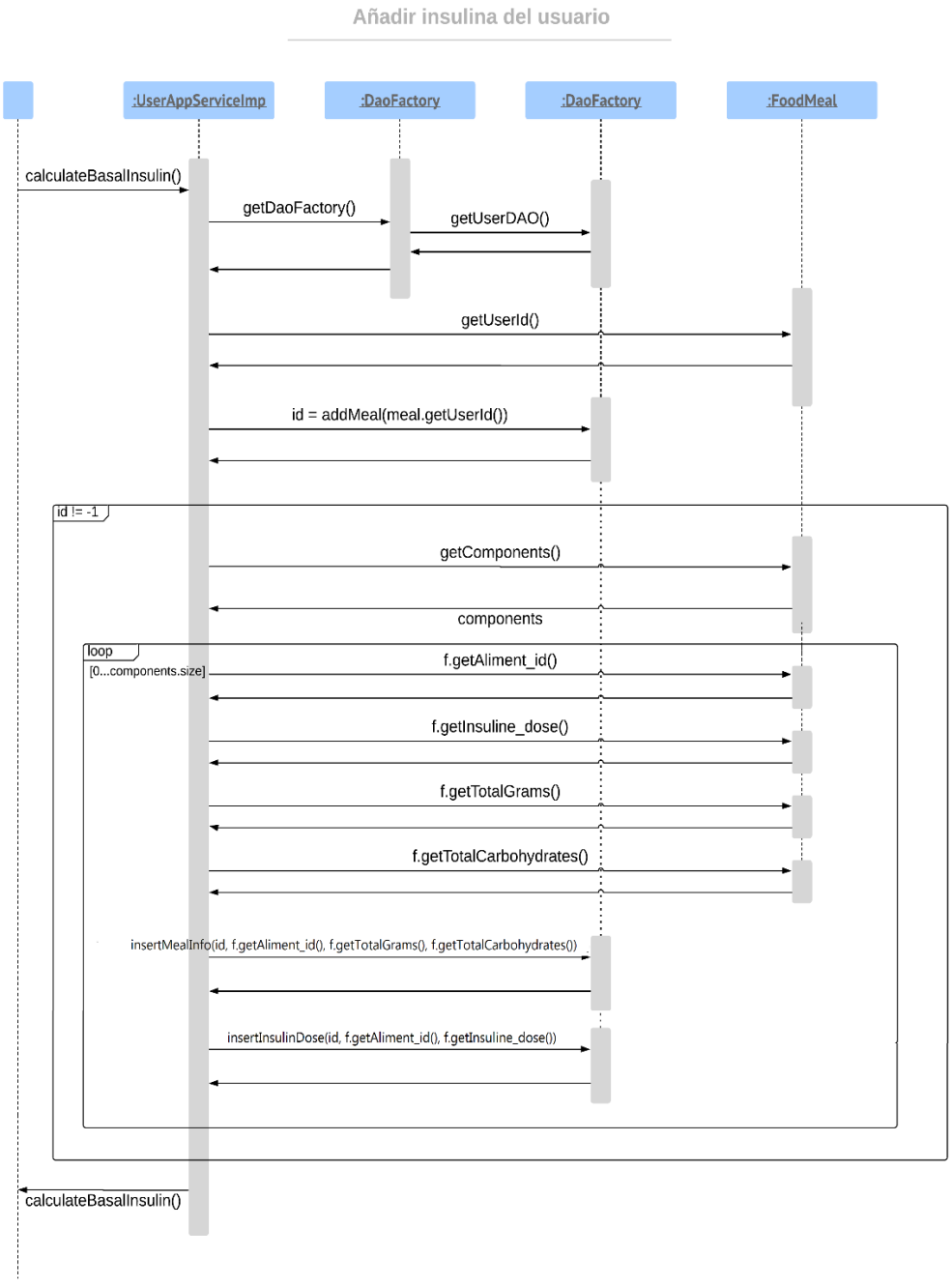

Figura 33. Diagrama de secuencia de añadir insulina

El usuario tiene dos opciones a la hora de registrar una comida, una de ellas es añadir un plato de comida a su histórico de comidas. Para ello se le ofrece al usuario la posibilidad de registrar la comida, que ha procesado previamente la aplicación, con la opción de registrarla con insulina si el usuario es diabético; este proceso se puede apreciar en la Figura 27.

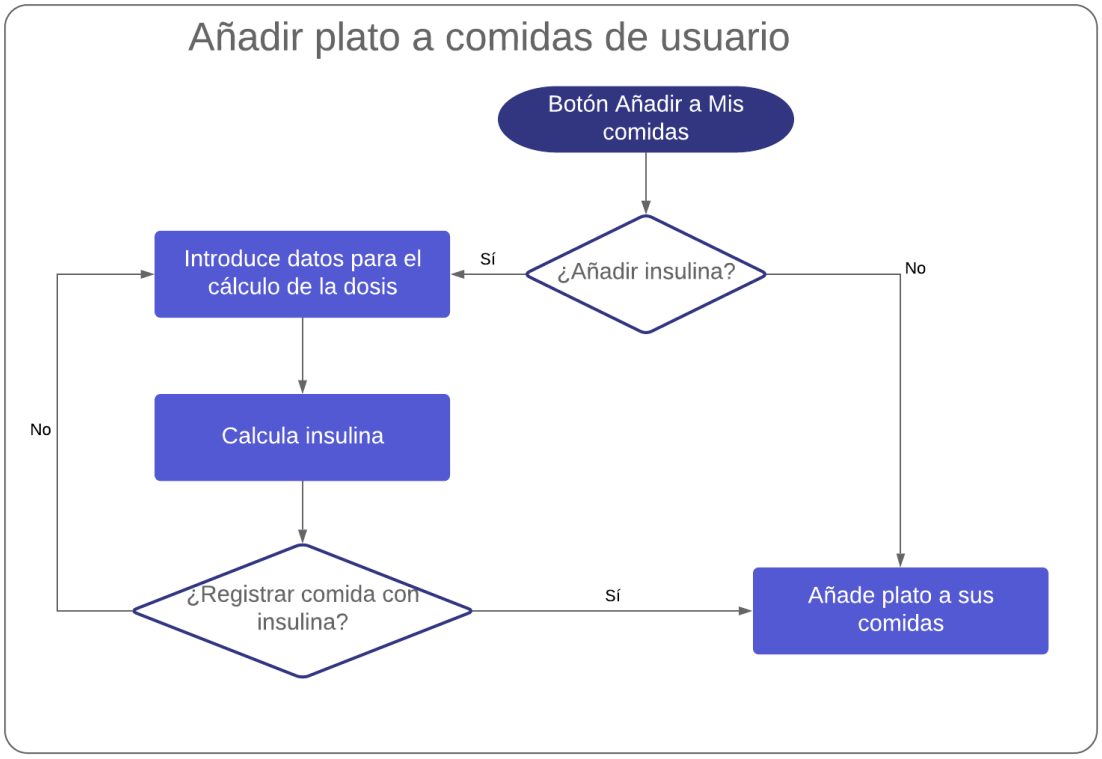

Figura 34. Diagrama de flujo de añadir plato en usuarios diabéticos

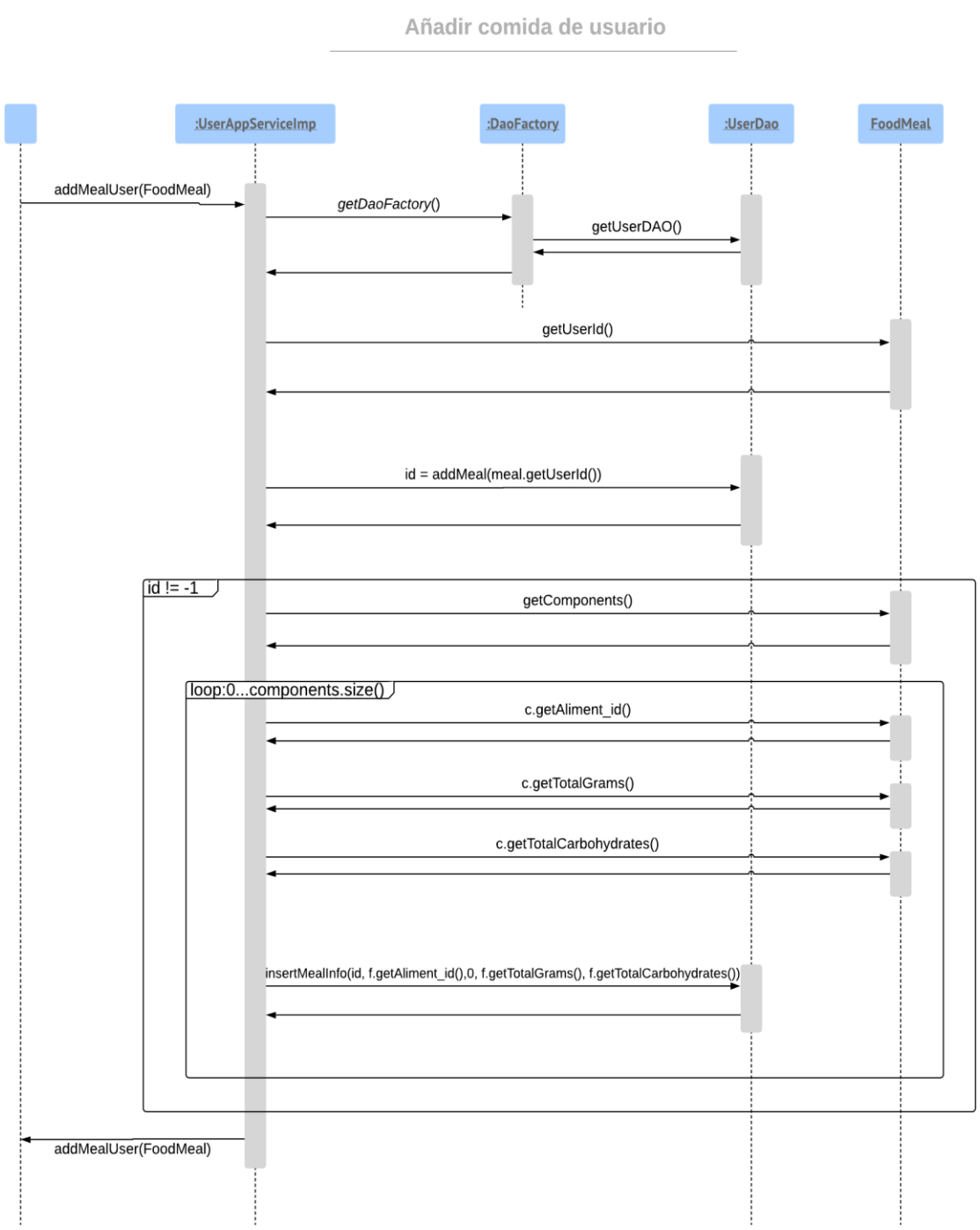

Figura 35. Diagrama de secuencia de añadir plato a sus comidas

- ***Image:*** este módulo ofrece al usuario las funcionalidades mostradas en la Figura 36.

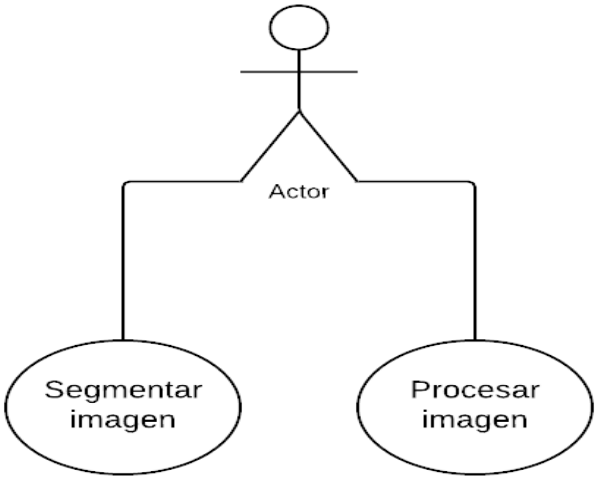

Figura 36. Casos de uso para la imagen

Este módulo contiene los algoritmos *KMeans* y *Knn*, los cuales son el "cerebro" de la aplicación, ya que se encargarán de obtener la mayor cantidad de información posible para dar al usuario los alimentos que componen la imagen y su información nutricional.

En el caso de Procesar Imagen, para el usuario es, básicamente, la opción que le ofrece la aplicación de subir una imagen ya sea desde la galería o desde la cámara y procesarla con el fin de obtener los alimentos que componen el plato de comida.

Por otro lado, procesar la imagen a nivel de sistema, se refiere al hecho de obtener las medias RGB y los pixeles totales de cada sección de comida existente en una imagen.

Finalmente, está la opción de segmentar y clasificar la imagen a nivel de sistema, esta opción se muestra en la Figura 37, que explica el proceso de segmentar la imagen

para obtener las partes de comida y clasifica esas secciones creando una lista de predicciones sobre ella.

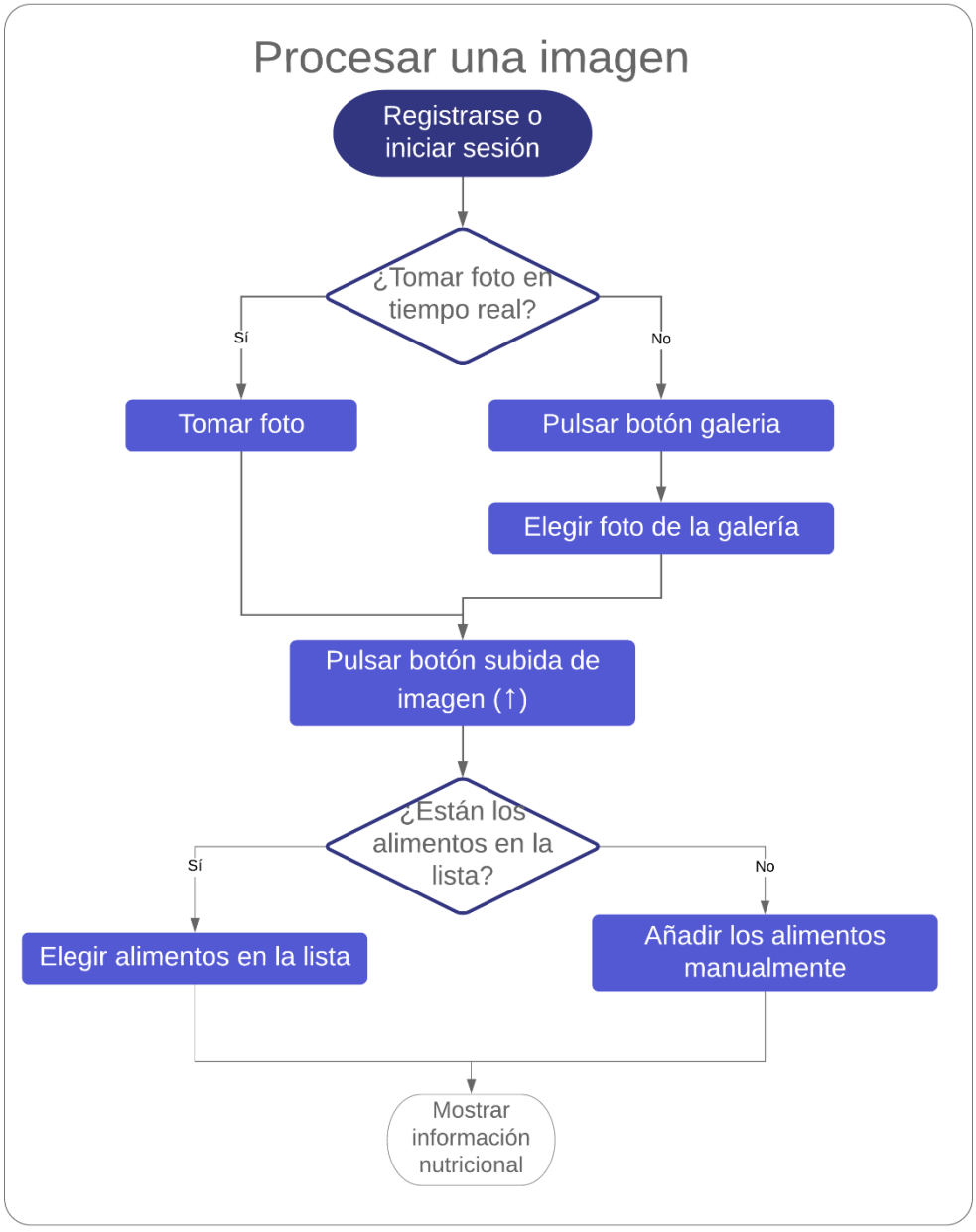

Figura 37. Diagrama de flujo de procesar imagen

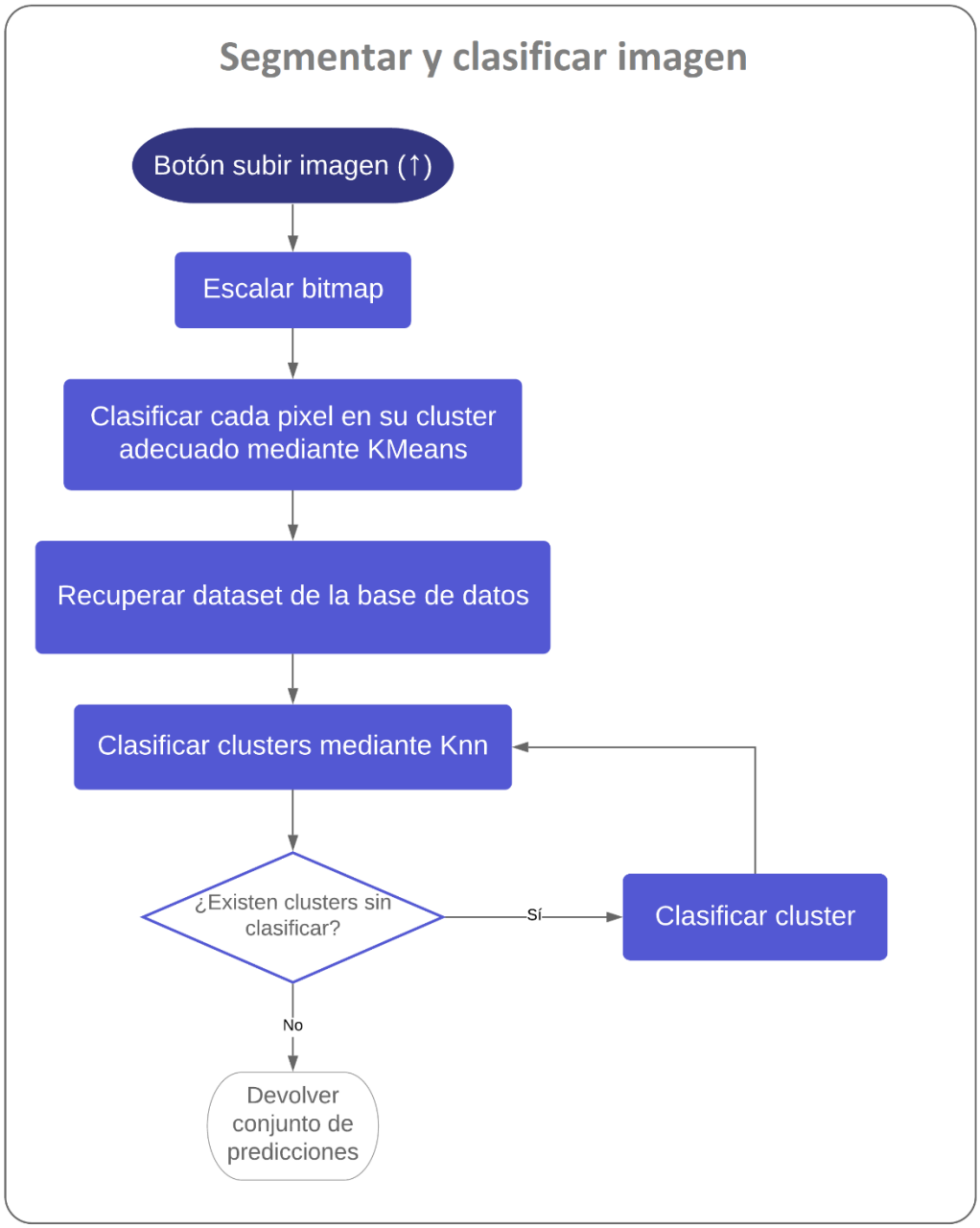

Figura 38. Diagrama de actividad de segmentar y clasificar imagen

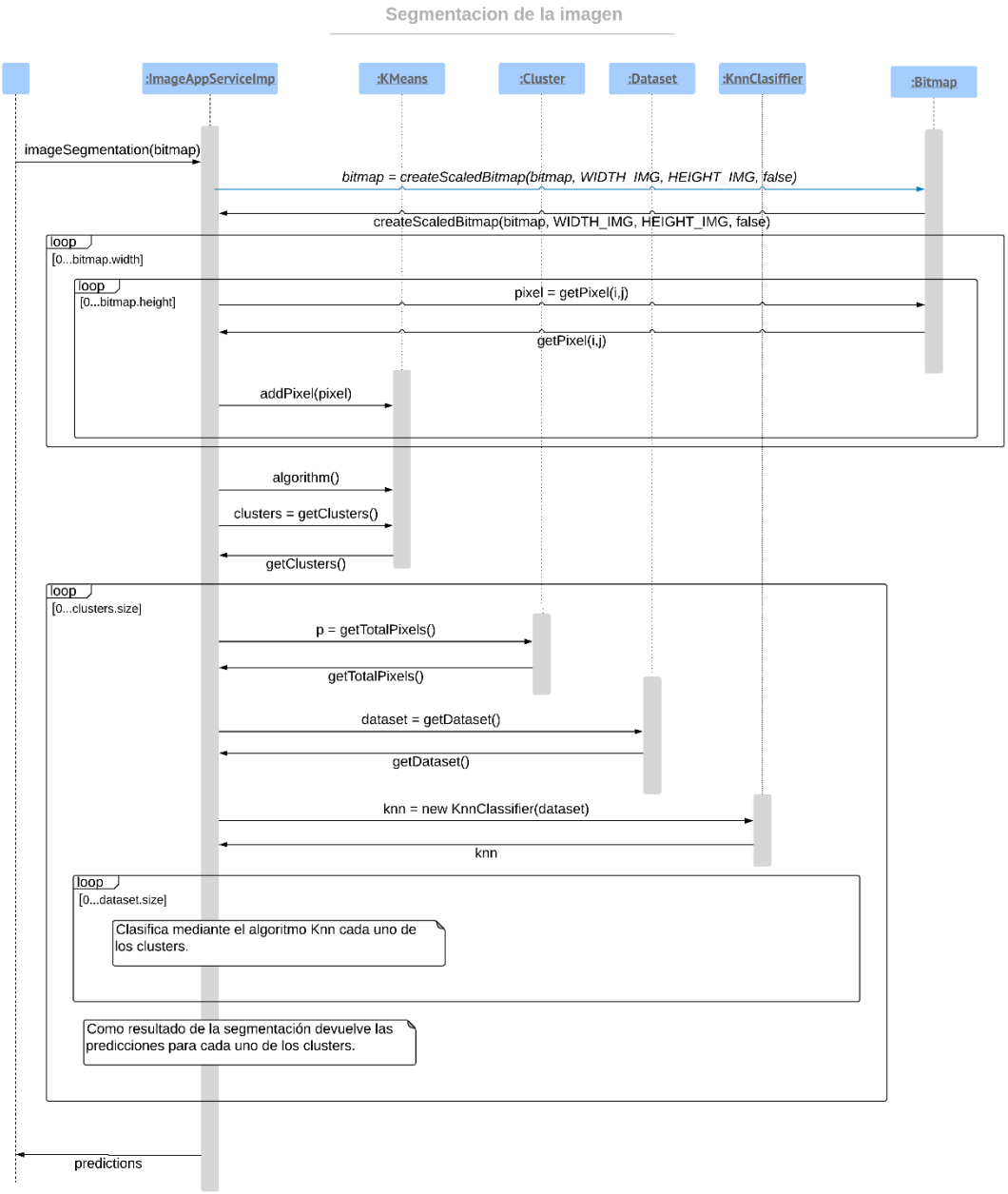

Figura 39. Diagrama de secuencia de segmentar imagen

Añadimos la Figura 39, ya que forma parte del proceso para añadir una comida inexistente en la base de datos. Esta parte es primordial debido a que es necesario el cálculo de las medias para la base de conocimiento. El resto del proceso se especifica más adelante.

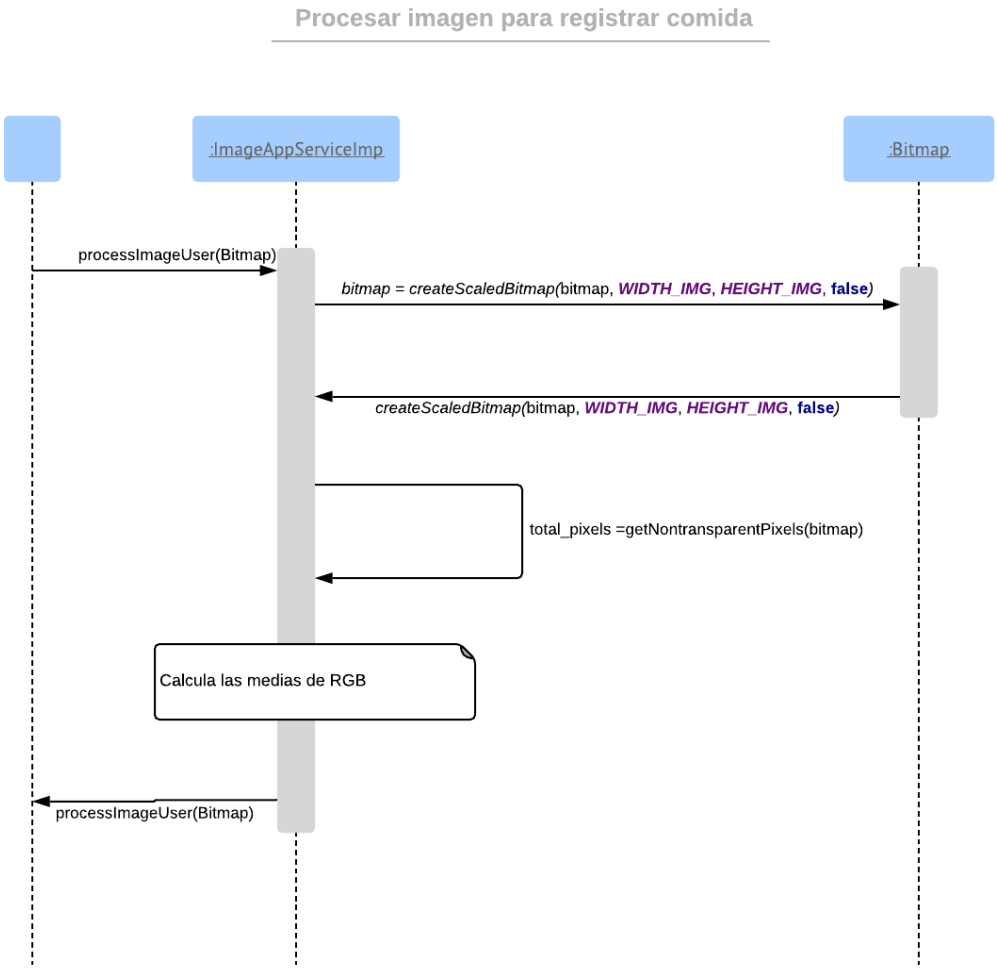

Figura 40. Diagrama de secuencia de procesar imagen de usuario

- ***Food:*** este módulo ofrece al usuario las funcionalidades mostradas en la Figura 41.

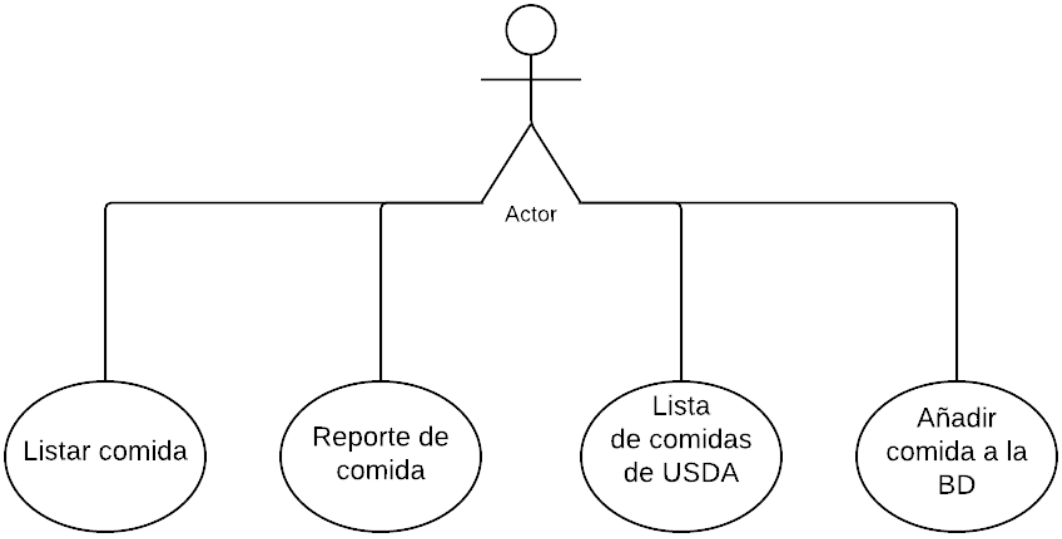

Figura 41. Diagrama de casos de uso para la comida

Este módulo consta de las operaciones principales que se pueden realizar con las comidas. Además, es el encargado de comunicarse con las APIs para la traducción de los datos y la obtención de los reportes nutricionales de cada alimento que contenga cada comida, pero estas operaciones ya están implícitas en todos los casos de uso de este módulo.

Aun así, hay que destacar una funcionalidad, añadir comidas a la BD, debido a que gracias a esta funcionalidad se puede ampliar la base de conocimientos de los algoritmos y, por tanto, ampliar la lista de predicciones.

A continuación, se puede observar cómo actúan las funcionalidades más relevantes de este módulo.

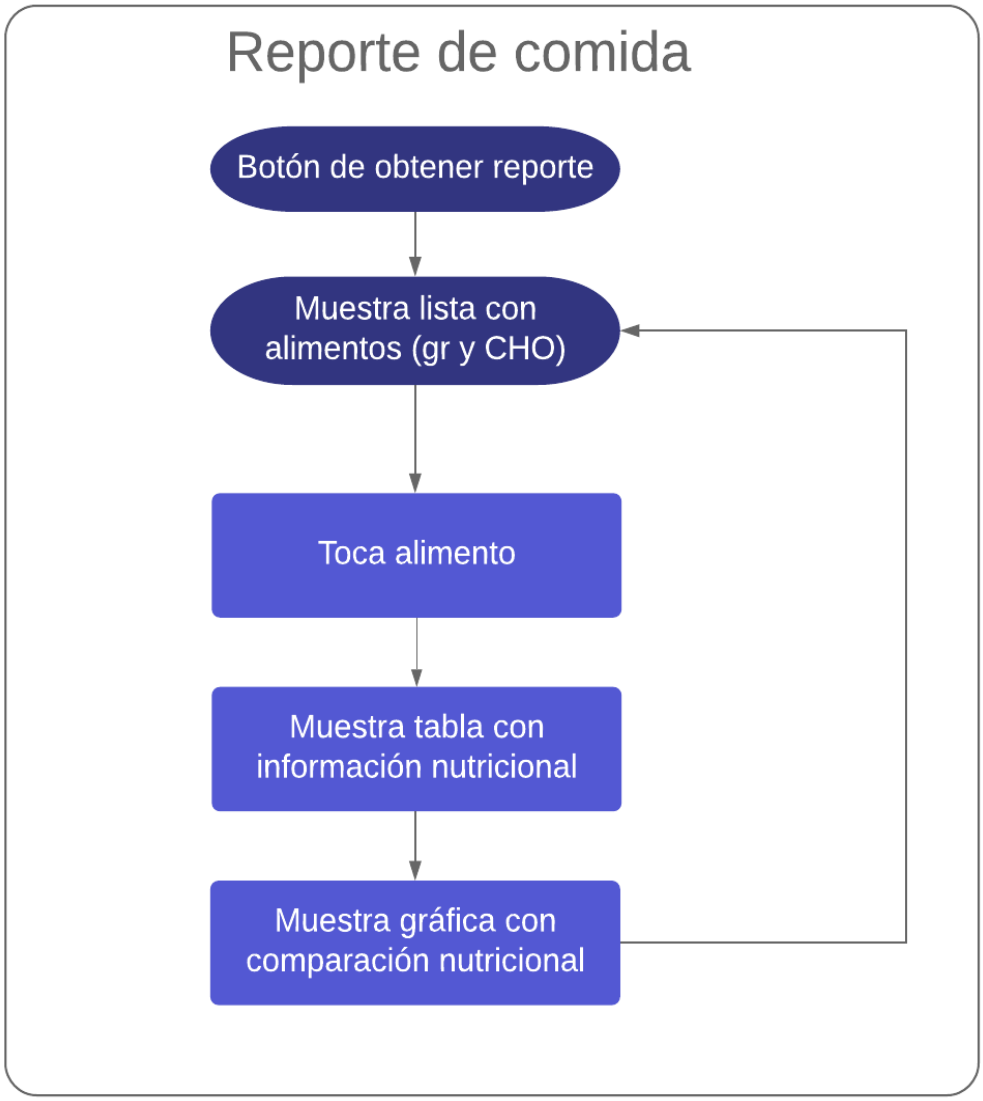

Figura 42. Diagrama de flujo para reporte de alimentos

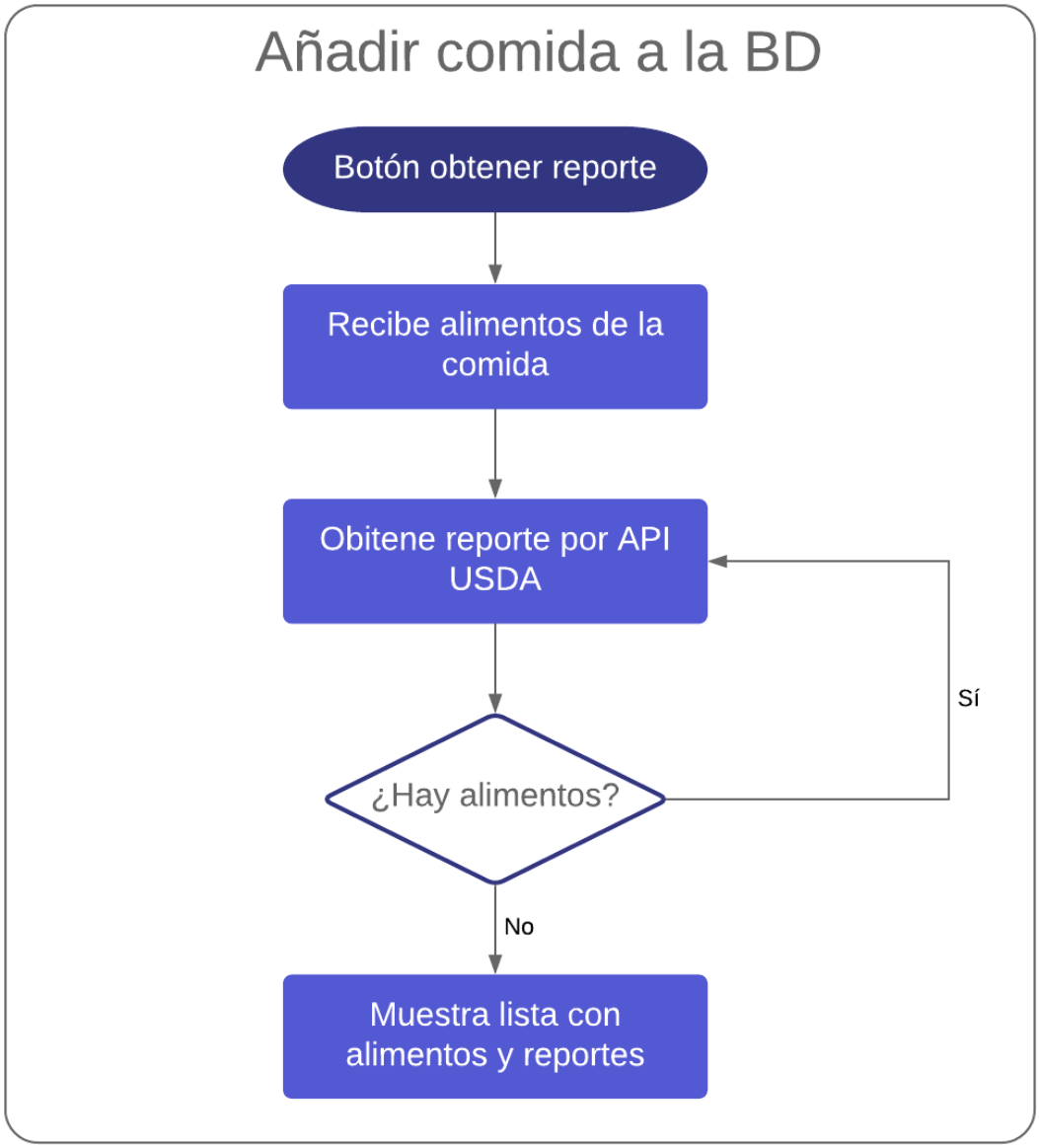

Figura 43. Diagrama de actividad para reporte de alimentos

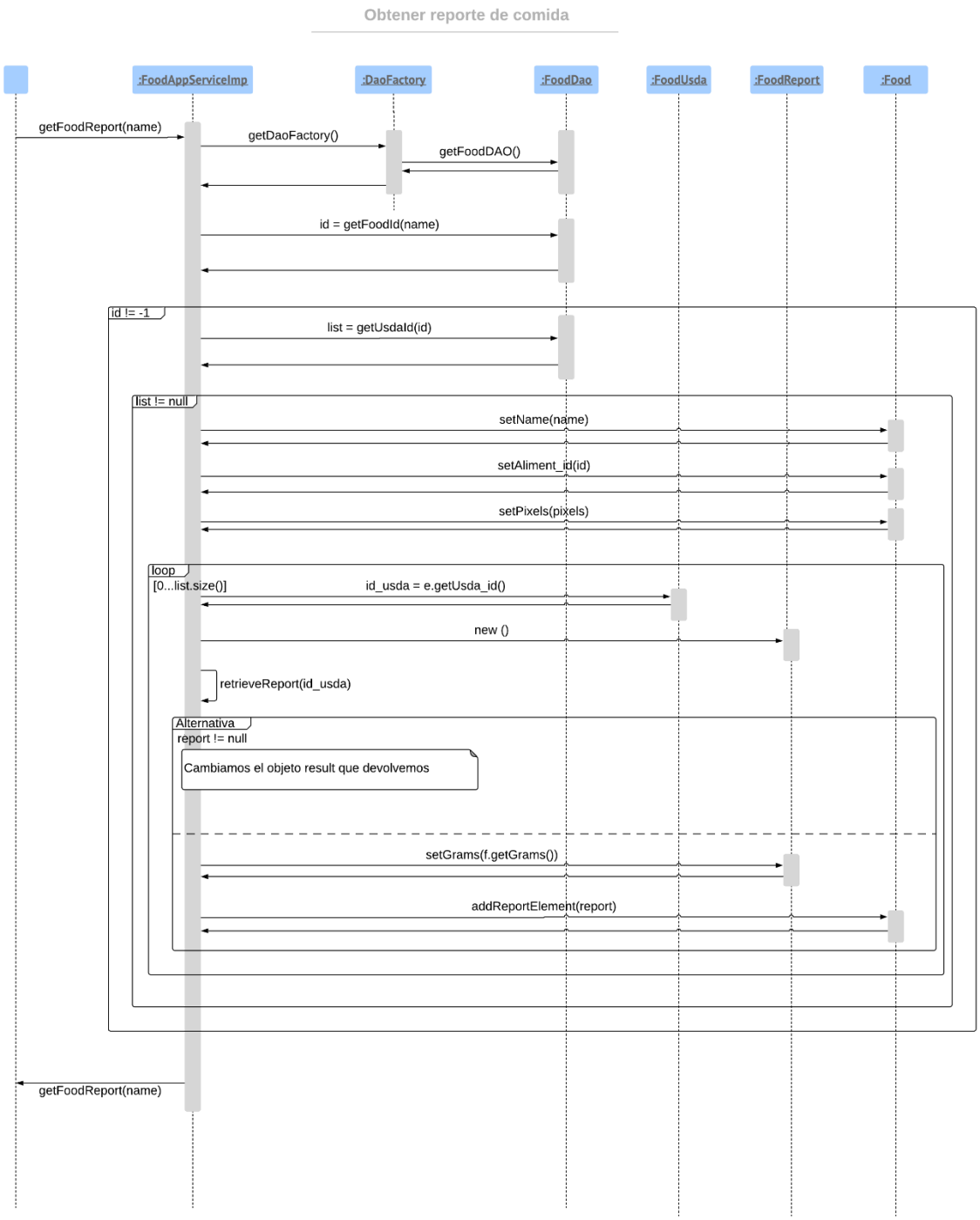

Figura 44. Diagrama de secuencia para reporte de alimentos

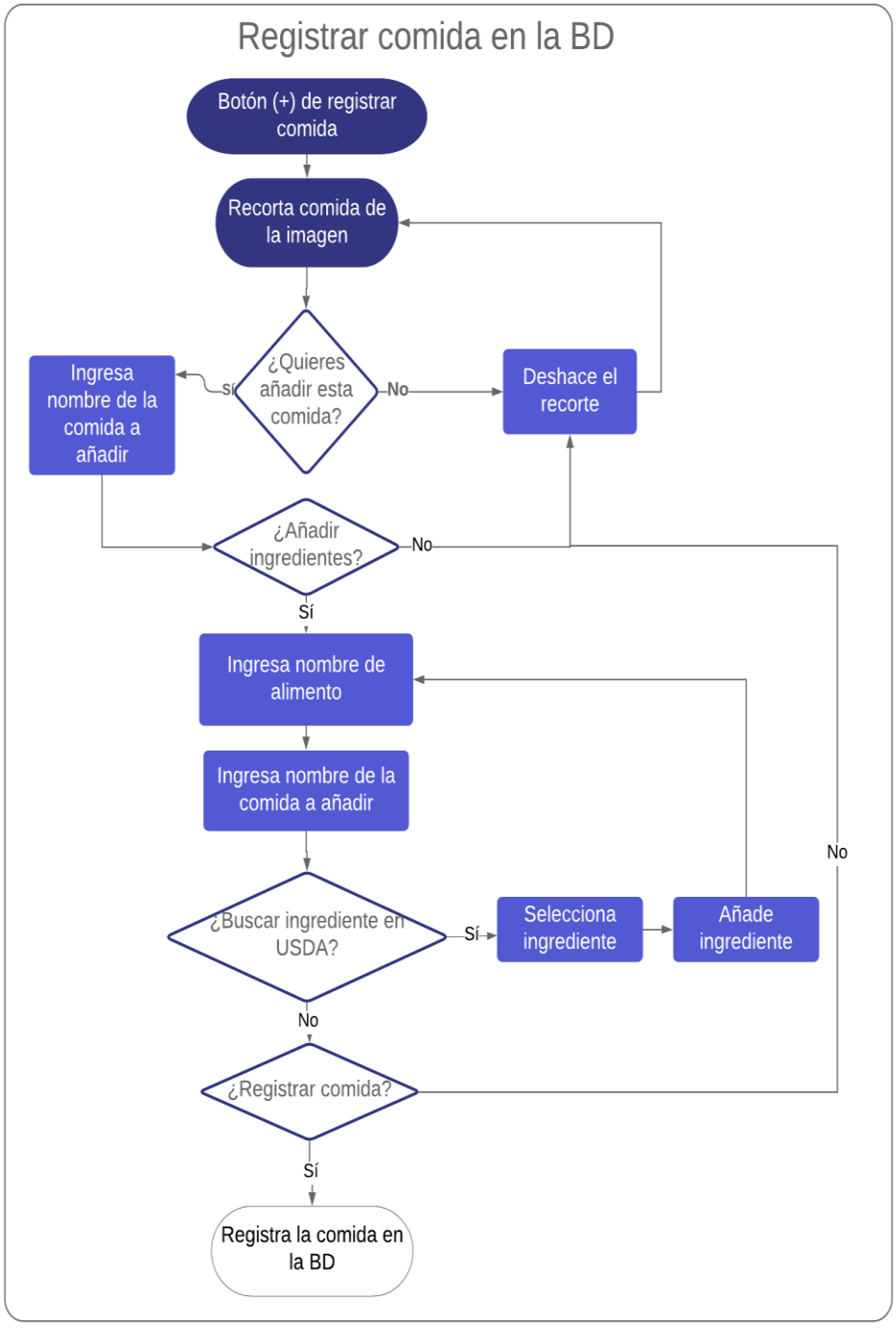

Figura 45. Diagrama de flujo para añadir comida nueva a la BD

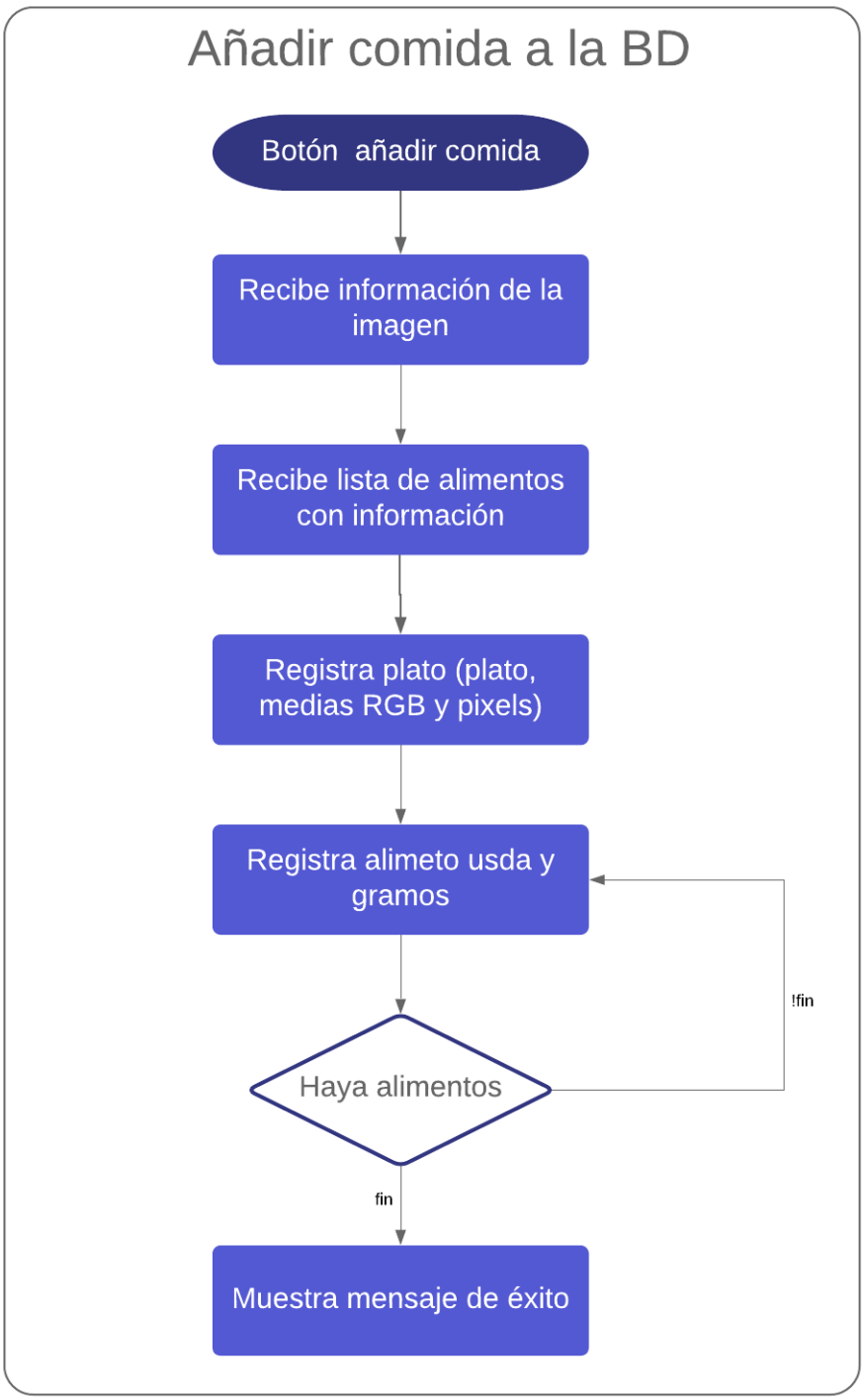

Figura 46. Diagrama de actividad para añadir comida nueva a la BD

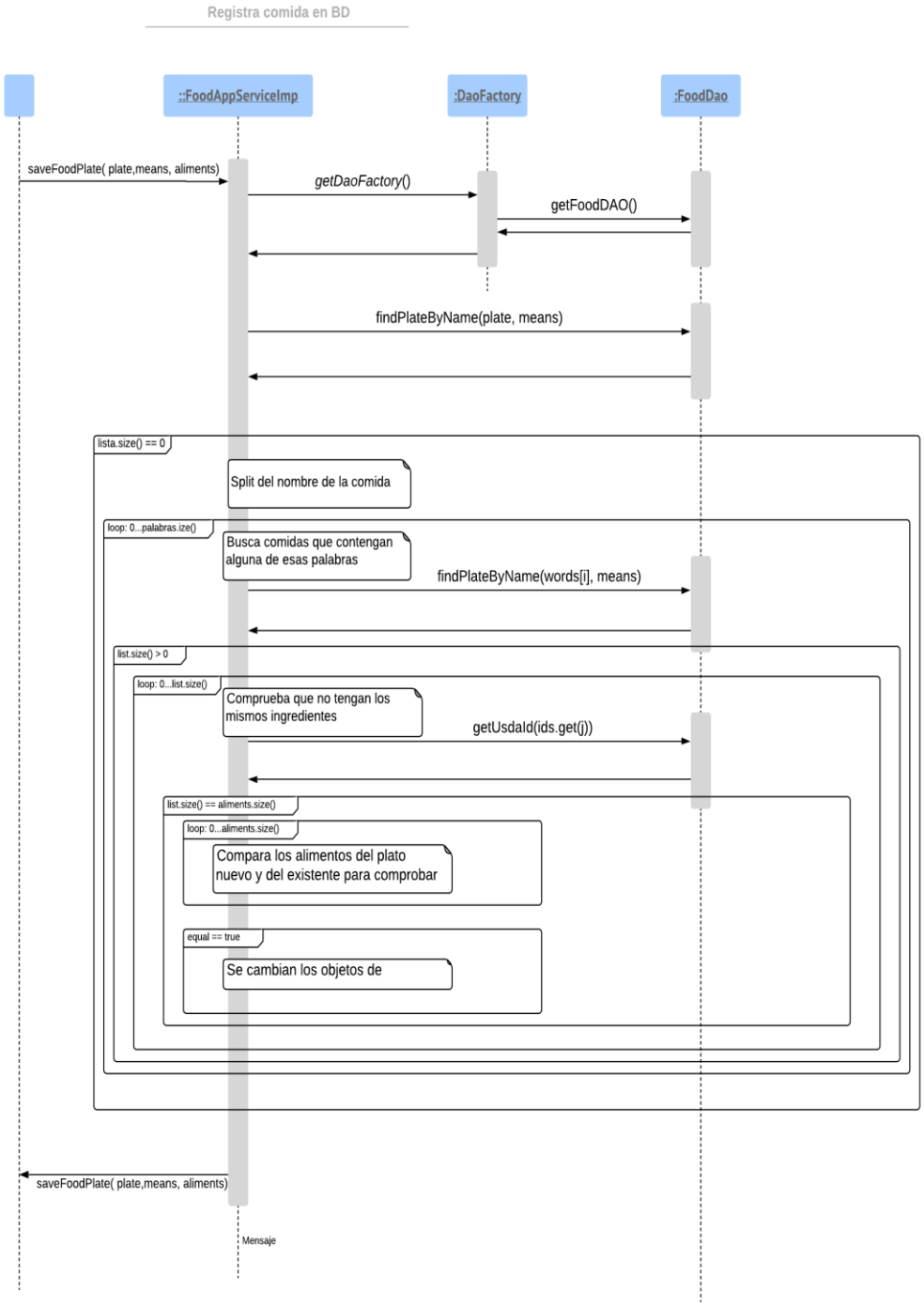

Figura 47. Diagrama de secuencia para añadir comida nueva a la BD

Servidor

La parte servidor consiste en un servidor que utiliza Apache para manejar las peticiones y *phpMyAdmin* para administrar la base de datos, como mencionamos en apartados anteriores.

Apache

Apache [31] es el servicio que se ejecuta en el servidor para que éste actúe precisamente como un servidor web. Este software se encarga de manejar las peticiones que lleguen desde la parte cliente y de responder dichas peticiones con los datos o información que correspondan.

Los datos que componen la respuesta del servidor a las peticiones del cliente se extraen de la base de datos utilizando scripts escritos en PHP. Estos scripts están alojados en el servidor y hacen una petición HTTP a la base de datos utilizando el servicio de MySQL. Al recibir los datos, componen la respuesta en formato JSON y devuelven los datos al cliente o devuelven un error en caso de haber ocurrido alguno. Todo este proceso es transparente tanto a la parte cliente como al usuario como tal.

Además, a este navegador se le ha añadido el servicio *phpMyAdmin*, con el cual se puede gestionar la base de datos utilizada desde cualquier navegador, este servicio trabaja de la mano con *Apache*, por lo que se ha tenido que crear otro archivo de configuración e instalar todos los paquetes relacionados con PHP para que el servicio funcione correctamente.

```
root@carbocuenta:~# cd /var/www/html/carbocuenta/
root@carbocuenta:/var/www/html/carbocuenta# ls -lart
total 92
-rw-r--r-- 1 root root   151 nov 26 21:21 db_config.php
-rw-r--r-- 1 root root   623 mar 17 19:12 add_user.php
-rw-r--r-- 1 root root   474 abr  4 16:26 db_connect.php
-rw-r--r-- 1 root root  1703 abr  4 17:58 get_aliments.php
drwxrwxrwx 4 root root  4096 abr 10 23:08 ..
-rw-r--r-- 1 root root  1276 abr 25 22:50 get_usda_id.php
-rw-r--r-- 1 root root  1471 abr 29 21:09 get_aliment_id.php
-rw-r--r-- 1 root root   706 may  4 21:26 add_aliment_usda.php
-rw-r--r-- 1 root root   963 may  4 21:26 add_plate_food.php
-rw-r--r-- 1 root root  2006 may  8 23:59 get_plate_food.php
-rw-r--r-- 1 root root 12288 may 15 12:53 .get_user.php.swo
-rw-r--r-- 1 root root   713 may 15 21:07 add_insulin_dose.php
-rw-r--r-- 1 root root   917 may 16 21:19 update_user.php
-rw-r--r-- 1 root root   720 may 17 11:07 get_food_name.php
-rw-r--r-- 1 root root   636 may 18 09:33 add_meal.php
-rw-r--r-- 1 root root   824 may 18 10:26 add_meal_info.php
-rw-r--r-- 1 root root  1637 may 18 11:16 get_user.php
drwxr-xr-x 2 root root  4096 may 18 11:53 .
-rw-r--r-- 1 root root   911 may 18 13:18 get_user_meals.php
-rw-r--r-- 1 root root   896 may 18 13:45 get_meals_info.php
-rw-r--r-- 1 root root   880 may 18 14:37 get_recents_user_meals.php
```

Figura 48. Scripts PHP del servidor

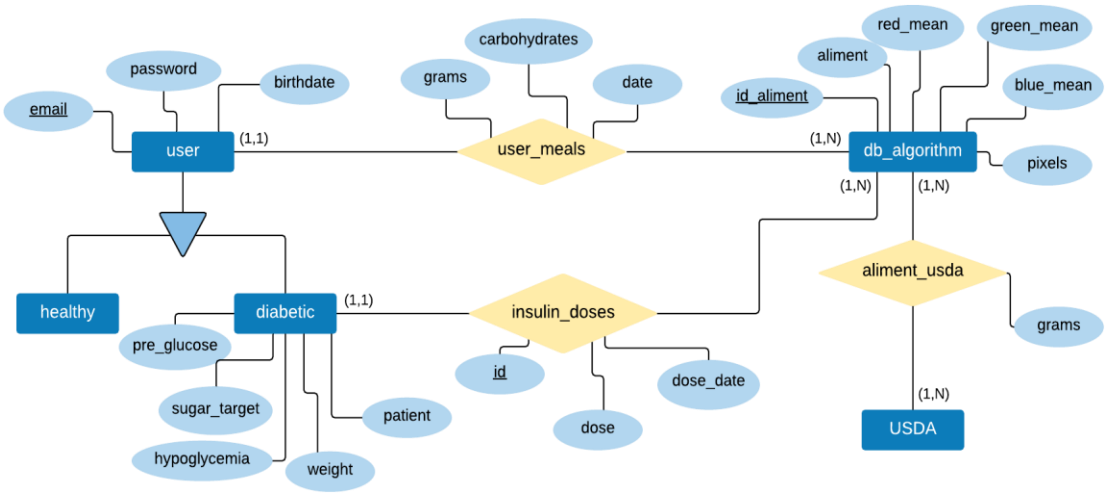

Figura 49. Esquema Entidad-Relación de la BD

Seguridad

Una vez realizada la implementación de la aplicación y haber comprobado que cumple todas las funcionalidades requeridas, se realizó una investigación para mejorar la seguridad tanto en la aplicación como en el servidor para evitar vulnerabilidades con los recursos que tenía el equipo.

La comunicación entre el Cliente y el Servidor se realiza mediante peticiones HTTP que hay que controlar por ambas partes, ya sea realizando las peticiones en Android con *HttpsURLConnection* y *SSLSocket,* habilitando la conexión HTTPS en el servidor, cambiando así la configuración de Apache y habilitando el puerto deseado donde se comunicarán cliente y servidor, es decir, hay que realizar las siguientes acciones para mejorar la seguridad del servidor:

- Utilizar certificados de confianza generados por el equipo con *openssl* y/u otorgados por una entidad de confianza, en este caso, la universidad.
- Añadir esos certificados en la configuración *ssl.conf*
- Añadir los puertos necesarios a las *iptables* o a la zona de interés del *firewall*.

Por esto, es recomendable, por ahora no hacer uso de redes no protegidas, ya que actualmente las peticiones son HTTP.

La seguridad en el lado del cliente se basa principalmente en la encriptación de datos sensibles, para evitar posibles robos de sesión y de APIKeys de las que disponemos. Para ello, se utilizan los siguientes métodos de encriptado y desencriptado.

Figura 50. Métodos de encriptación y desencriptación

Para el encriptado de datos se ha usado el estándar recomendado AES (*Advanced Encryption Standard*) [33] o también conocido por su nombre original *Rijndael* que fue elegido por el NIST (*National Institute of Standards and Technology*) para sustituir el estándar DES (*Data Encryption Standard*), el cual fue considerado inseguro debido al corto tamaño de las claves utilizadas (56 bits). El estándar AES es un *block Cipher* [34], es decir, no cifra los datos bit por bit sino por bloques, en concreto encripta bloques de 128 bit soportando claves de 128, 192 y 256 bits. AES cuenta con distintos modos de cifrado como ECB [35] (*Electronic Code-book)* que nos permite encriptar en un mismo *Cipher* varios bloques de 128 bits usando la misma clave.

Su funcionamiento se resume en coger bloques de texto plano y producir un bloque del mismo tamaño de texto cifrado, esta transformación es controlada mediante una clave.

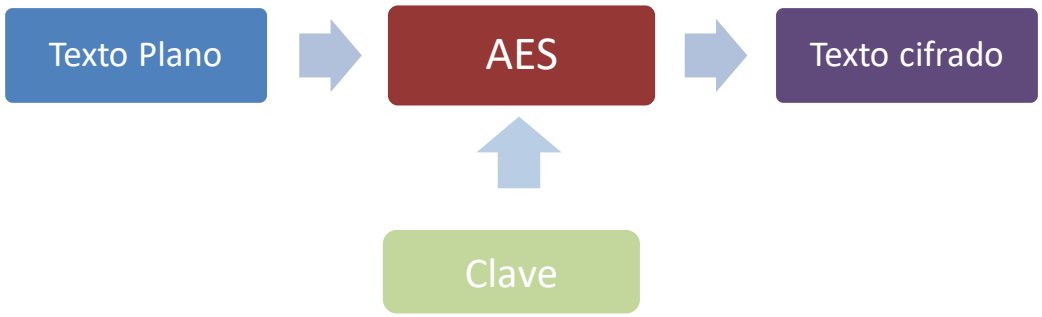

Figura 51. Funcionamiento del estándar AES para encriptado de texto

Como se muestra en la Figura 50 se ha usado para la encriptación la clase *Cipher* [36] que forma el núcleo de la extensión JCE (*Java Cryptographic Extension*) la cual proporciona un cifrado criptográfico. Para la creación del objeto *Cipher* se hace uso de su método *getInstance()* al que se le pasa como parámetro el estándar deseado, AES en este caso. La clase *SecretKeySpec* [37] es la encargada de construir la clave secreta en el estándar adecuado para el *Cipher*.

Por otro, Android recomienda una serie de consejos para tener una aplicación segura [38], con estas recomendaciones además de asegurar la red, también se proporciona seguridad al almacenamiento de datos, tanto interno como externo. Este problema no existe, ya que solo se accede a la memoria para leer imágenes. También, recomienda no otorgar tantos permisos a la aplicación; en esta los permisos son mínimos, acceso a la memoria en modo lectura, a la cámara y a la red a la que se encuentre conectado el dispositivo en ese momento.

Por último, hacer mención del concepto IPC [39], el cual se refiere al protocolo que suelen seguir las aplicaciones para comunicarse entre ellas y presentes en un mismo dispositivo. Un invasor de la aplicación suele aprovechar este protocolo en momentos en

los que se hace una transición entre *activities* para obtener información de la aplicación que ha invadido mediante la suya. Pero en este caso, se ha revisado el proyecto y no se permite que otra aplicación tenga acceso a través de *bundles*, *intents* y *binders*.

Mantenimiento

La aplicación depende totalmente del servidor para poder recoger información de la base de datos que se encuentra almacenada en este, por lo que es necesario tener un control de monitorización y alertado para poder recibir notificaciones sobre estado del servidor.

En un principio, se ha creado un sistema de *backup* que realiza una copia de la base de datos todos los días a las 3 a.m. de tal manera que cuando finalice de realizarse el *backup* se notifique a la/el responsable del servicio con el fin de controlar la BD. Todo esto está configurado en el crontab.

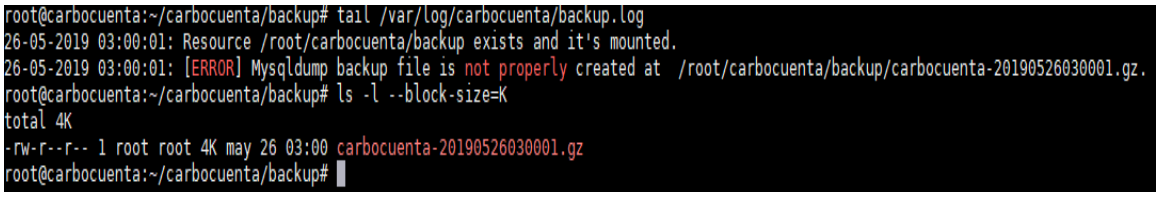

Figura 52. Funcionamiento del script de backup en caso de error

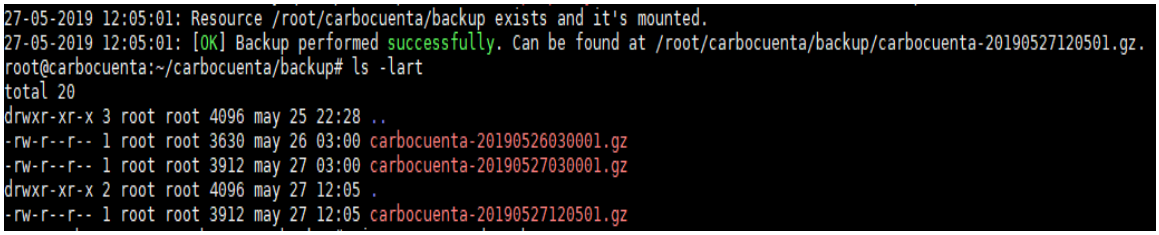

Figura 53. Backup correcto

Además, es importante el buen mantenimiento del código de la aplicación tanto de la parte *front-end*, enfocada en el usuario, como *back-end*, enfocada en todo se encuentra detrás para el correcto funcionamiento de la aplicación. De esta forma es importante seguir actualizando la parte visual de la aplicación para que sea capaz de adaptarse a los nuevos dispositivos en el mercado y a las guías de estilo de los nuevos sistemas operativos. Asimismo, es indispensable para el mantenimiento de una aplicación la corrección de errores (*bugs*). El desarrollo software se encuentra en constante cambio por lo que también es importante, a parte de la corrección, la actualización del código. Cuando hablamos de actualización no solo nos referimos al propio código de la aplicación sino también a las librerías, paquetes o APIs que use para su funcionamiento. De esta forma se mantienen los buenos resultados alcanzados y, además, se consigue mejorarlos.

Análisis de Ingeniería del Software

A pesar de ser un equipo compuesto por 3 miembros, se ha intentado seguir el sistema de *Sprints* que ofrece la metodología ágil, *Scrum*. Con esto, desde que se empezó el proyecto se ha ido generando y actualizando un Backlog con el que se repartían las tareas los miembros del equipo. Además, se ha intentado hacer una estimación de tiempo-esfuerzo que se iba a invertir en cada *Sprint*, aunque estas normalmente han sido incorrectas debido a que se ha tenido que invertir demasiado tiempo y esfuerzo en la fase de investigación del proyecto y en la fase de pruebas.

Riesgos

Cuando se habla de riesgos en IS se refiere a todo aquello que puede afectar de forma negativa al desarrollo del software, por eso se realizar un plan de gestión de riesgos en el cual se clasifican los riesgos en función de su probabilidad de aparición y del esfuerzo necesario para mitigarlo y la prioridad que hay que darle a cada uno.

Existen 3 tipos de riesgos, pero en este caso al ser un TFG solo se consideraron los riesgos del proyecto:

A. ***Entrega tardía.***
B. ***Abandono de un miembro del equipo.***
C. ***Falta de comunicación entre el equipo.***
D. ***Falta de implicación por parte de algún miembro.***
E. ***Inexperiencia con las tecnologías.***
F. ***Cambios en los requisitos.***
G. ***Ocupaciones extra laborables.***
H. ***Baja de algún miembro.***

Tras esto, se realizó un análisis de basado en el SQAS-SET [41] donde la probabilidad podía ser **frecuente, probable, ocasional, remota e improbable** y el grado de gravedad podía ser **catastrófico, crítico, serio, tolerable e insignificante.**

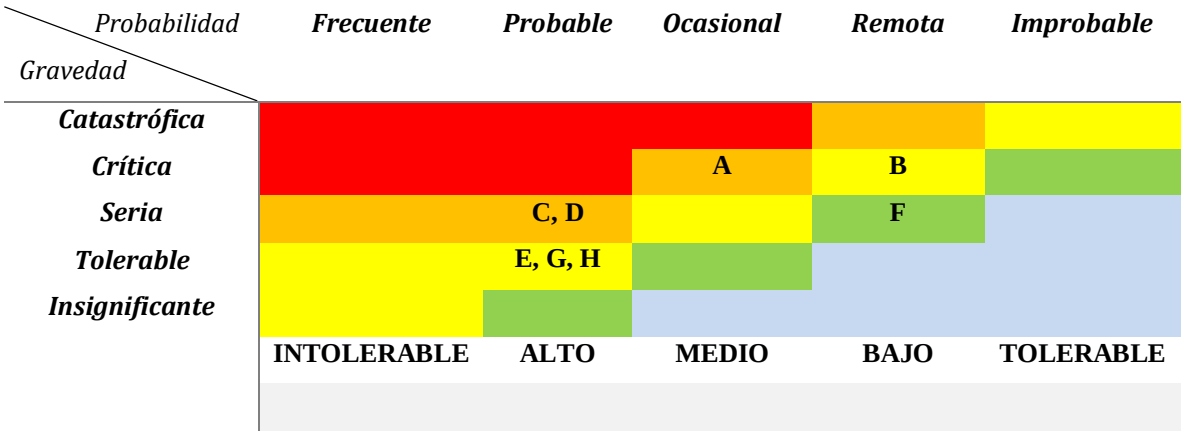

Figura 54. Tabla de priorización del riesgo

Para filtrar estos riesgos se utilizó la Regla de Pareto, escogiendo el 20% de los riesgos identificados, con esto, se han elegido 2 riesgos con mayor prioridad, es decir, A y B. Así, se aplicaron las técnicas de **reducción, supervisión y gestión** del riesgo.

A. Entrega tardía

1. **Reducción:** llevar una organización del equipo basado en alguna metodología.
2. **Supervisión:** comprobar si con lo que se ha hecho hasta el momento se llega a la entrega final.
3. **Gestión:** implementar lo máximo posible, aunque en poca cantidad y comunicárselo a los directores.

B. Abandono de un miembro del equipo

1. **Reducción:** maximizar la comunicación en el equipo para conocer las circunstancias por las que pasa cada miembro.
2. **Supervisión:** ayudar en todo lo posible a ese miembro para que siga trabajando.
3. **Gestión:** comunicárselo a los directores y establecer un nuevo plan para la organización del proyecto.

Costes y negocio

Existen diferentes maneras de estimar costes a nivel de hardware y software y costes a nivel de esfuerzo humano y tiempo, para este proyecto el análisis de costes fue el siguiente:

- ***Hardware:*** se utilizaron los portátiles y *smartphones* de los miembros del equipo, de los que se constaba antes de este proyecto. Un servidor cuyo nodo físico se encuentra en las instalaciones de la UCM.

- ***Software:*** herramientas de uso libre, aunque también se incluye la API de Google, que está en periodo de prueba y que hay que pagar la licencia tras un año.

- ***Persona:*** se ha desarrollado por 3 estudiantes de la UCM y bajo acuerdo de confidencialidad con los directores del proyecto.

Con respecto al negocio, existen bastantes aplicaciones relacionadas con el cálculo de los bolos y la creación de dietas para comer diariamente en función de las dosis diarias de insulina que suelen inyectarse [40], al igual que existen algunas para clasificar elementos mediante fotografías o la propia cámara, por ejemplo, Google Lens, pero, por ahora, no existe ninguna que procese la imagen de un plato completo y de información sobre todos los componentes existentes en él, además, de ofrecer un calculador de bolo prandial y corrector. Aun así, no se puede descartar que en poco tiempo aparezca una parecida, ya que el campo de la informática está en constante cambio e innovación, pero no se descarta que a corto plazo la aplicación pueda ser incluida en plataformas de servicios para Android como PlayStore de Google.

Calidad y rendimiento

En la fase de pruebas es cuando se garantiza la calidad del software y se realiza un análisis de rendimiento con el fin de mejorar el proyecto.

Aunque no se han podido realizar demasiadas pruebas la calidad, el software ha superado las siguientes pruebas:

- ***Pruebas de unidad:*** ya que a cada método ha dado el resultado esperado según una entrada concreta que el equipo ha ido ingresando.

- ***Pruebas de integración:*** para cada funcionalidad se ha utilizado un conjunto de clases relacionadas entre sí, y por el mismo método que en el de las pruebas de unidad, todas han dado los resultados esperados según la entrada.

- ***Pruebas de sistema:*** el sistema completo ha sido probado con un nivel de opacidad denominado “de caja negra”, el cual ha funcionado correctamente y ha respondido adecuadamente.

- ***Pruebas de aceptación:*** aquí solo se han podido realizar pruebas Alpha, es decir, llevadas a cabo por los desarrolladores, al superarlas se podrán pasar a los usuarios finales para seguir con las pruebas beta y terminar la verificación y validación del software.

Por otro lado, se han realizado pruebas de rendimiento en los cuales se ha monitorizado el funcionamiento de la RAM, la CPU, la red y la energía del dispositivo gracias a la herramienta *Profile* que ofrece *Android Studio* para ver en qué cantidad afecta la aplicación al hardware disponible en los dispositivos de prueba.

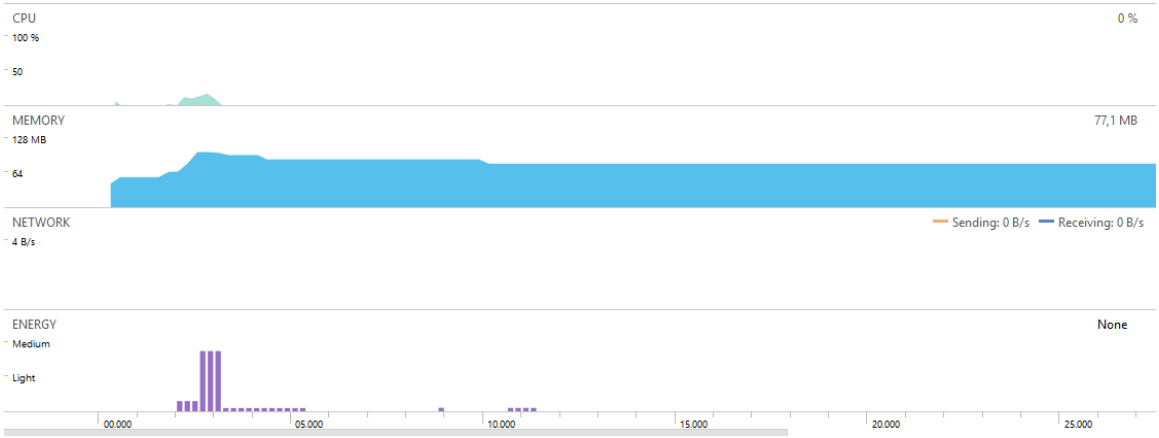

Figura 55. Gráfica de monitorización al iniciar la aplicación

La parte en la que más consumía recursos era durante el procesamiento de la imagen para la obtención de los alimentos, como se puede observar en la Figura 56.

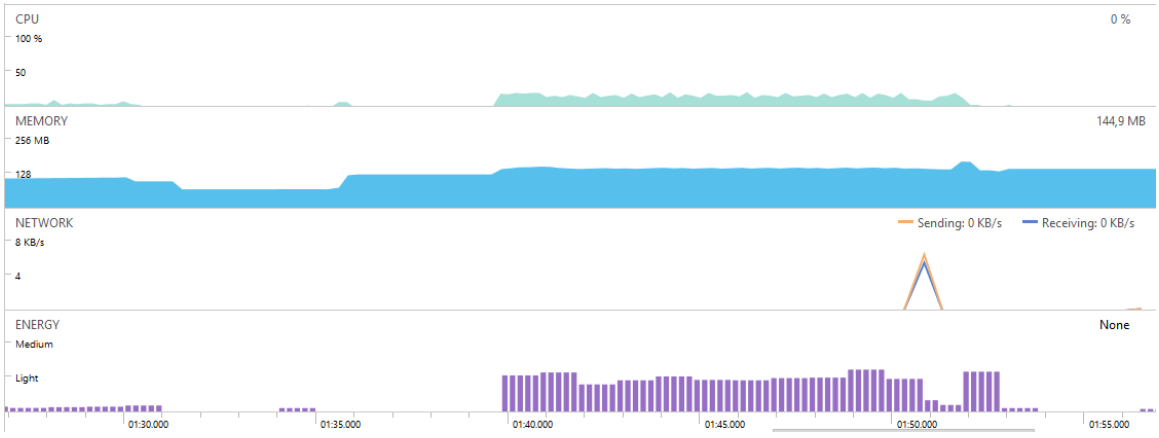

Figura 56. Gráfica de monitorización durante el procesamiento de la imagen

En el resto de las operaciones la aplicación mantenía datos parecidos a los que se muestran en la Figura 55, con muy poca varianza entre transiciones.

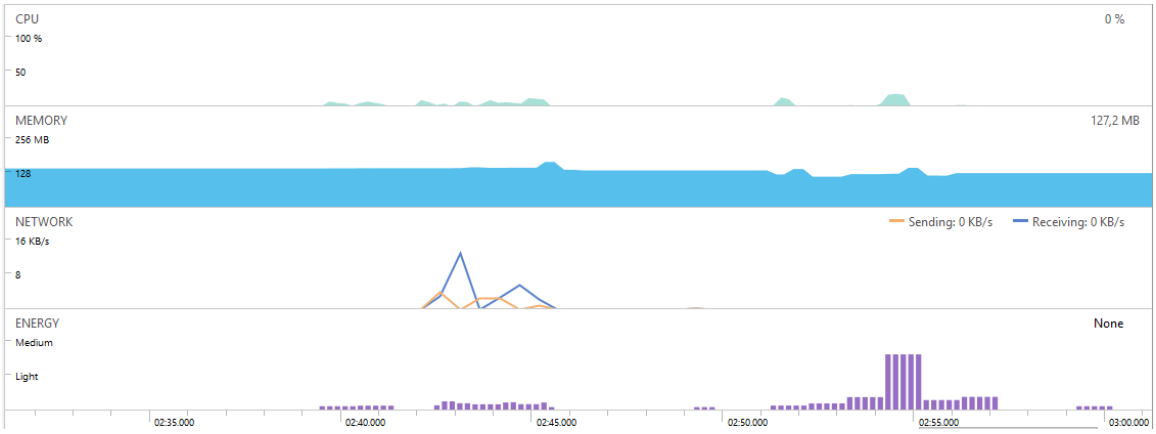

Figura 57. Gráfica de monitorización durante el reporte nutricional de los alimentos

Manual de usuario

Inicio de sesión

Para poder acceder a la aplicación hay que estar registrado/a e iniciar sesión.

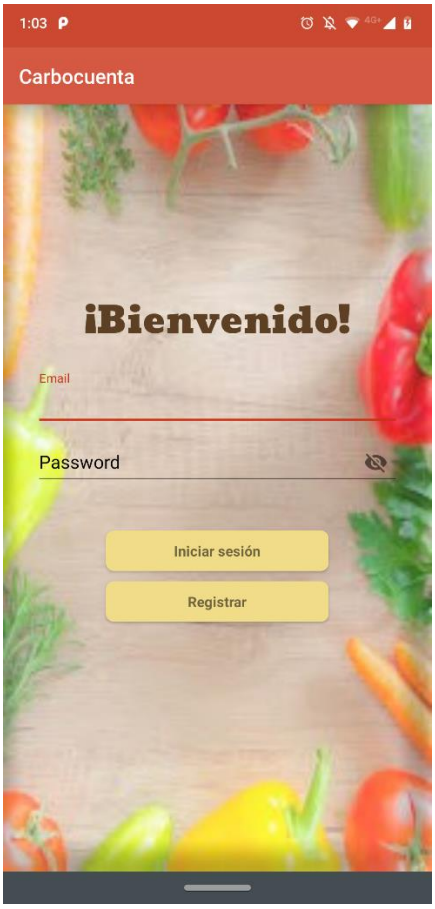

Figura 58. Vista de acceso a la aplicación

El registro se realiza en esta misma vista, para ello hay que ingresar un email válido y una contraseña de mínimo seis caracteres. Una vez hecho el registro correctamente la aplicación llevará al usuario al menú principal, este ofrece tres opciones.

Figura 59. Vista de menú principal

Acceder a la **galería** y elegir una foto para procesarla.

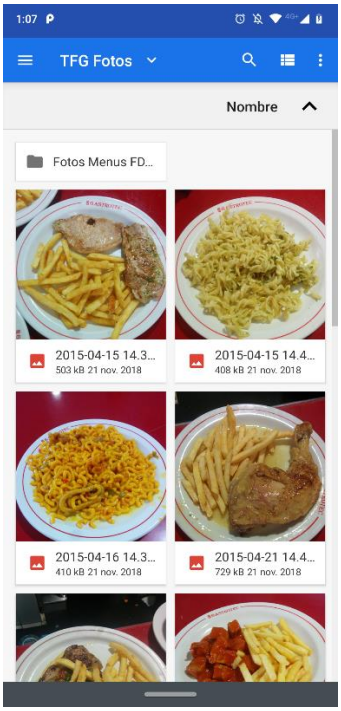

Figura 60. Vista de galería

Realizar una foto con la **cámara** y procesarla.

Figura 61. Vista de cámara

Menú de usuario.

Figura 62. Vista de menú de usuario sano

Figura 63. Vista de menú de usuario diabético

Opciones de imágenes

Una vez elegida o realizada la fotografía, se da la opción al usuario de **cancelar** la operación, **procesar** la imagen o **añadir** una nueva comida inexistente en la BD.

Figura 64. Vista para procesar imagen

El usuario quiere procesar la imagen

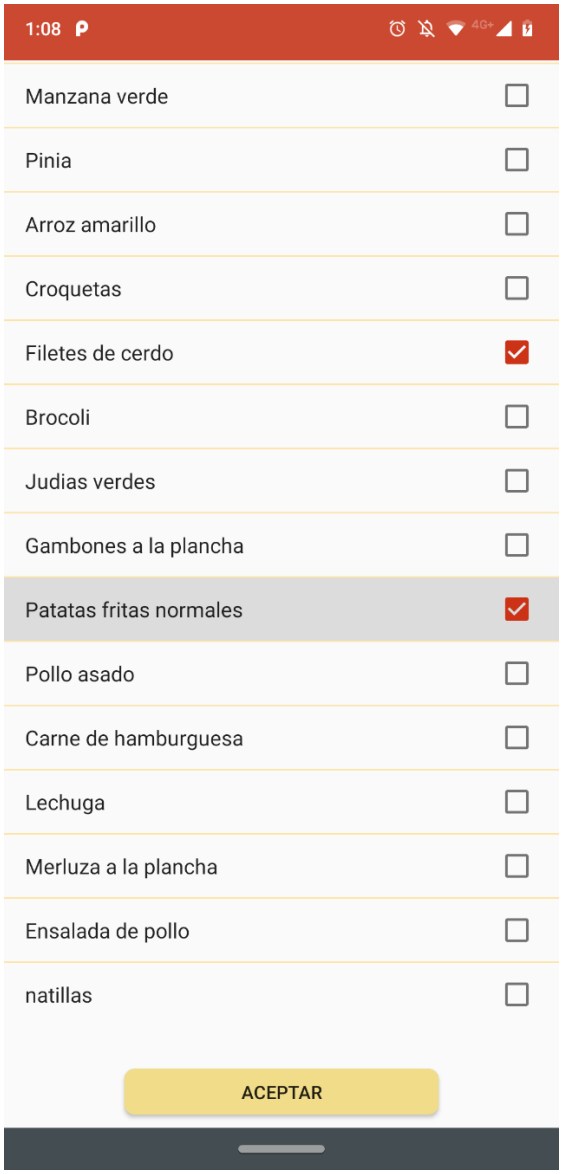

Figura 65. Vista con lista de posible/s comidas en la imagen

El usuario tiene que seleccionar el o los alimentos de su plato según la lista de la Figura 65, siempre puede retroceder y volver a procesar la imagen si por un casual su alimento no aparece.

Figura 66. Menú secundario

El usuario quiere añadir nueva comida.

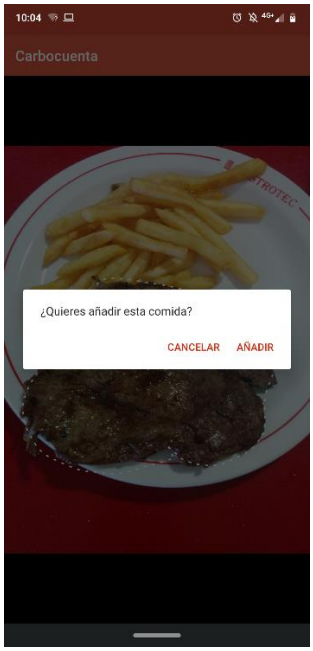

Figura 67. Vista para recortar imagen

Figura 68. Vista para añadir nombre a la nueva comida

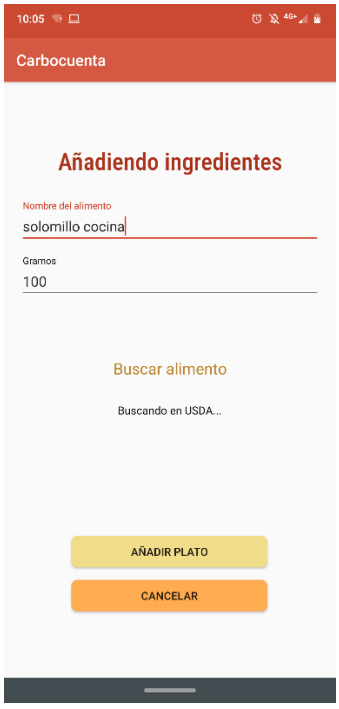

Figura 69. Vista para añadir nueva comida con nombre e ingredientes

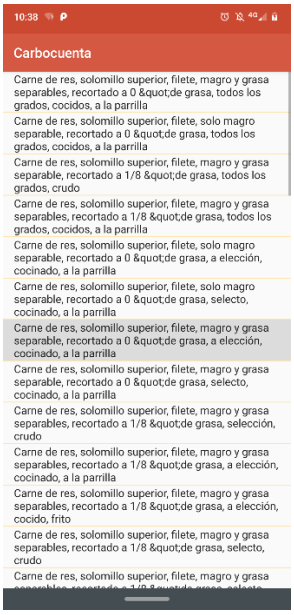

Figura 70. Vista para añadir ingredientes de la nueva comida

Menú secundario

Este menú ofrece al usuario **añadir** plato de comida a historial de comidas, **obtener reporte nutricional** del plato de comida y **calcular bolo prandial**.

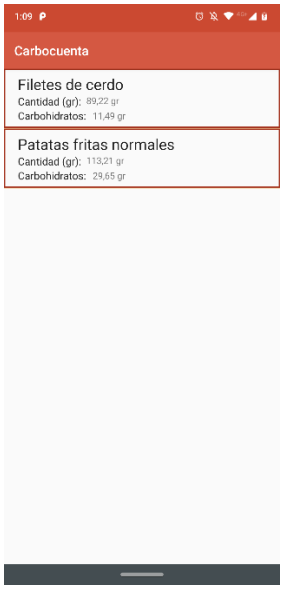

Figura 71. Vista con resumen nutricional de la/las comidas

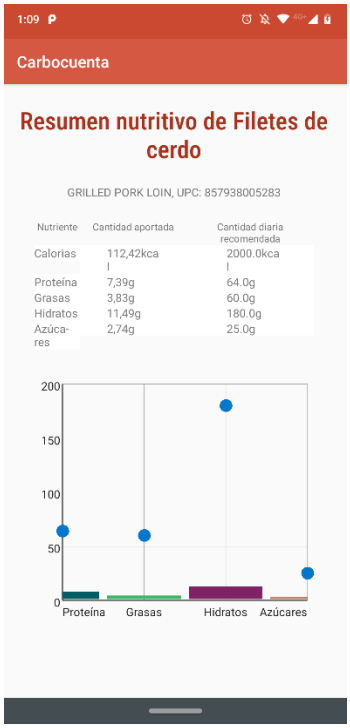

Figura 72. Vista con resumen nutricional extendido de la/las comidas

Figura 73. Vista para calcular bolo prandial

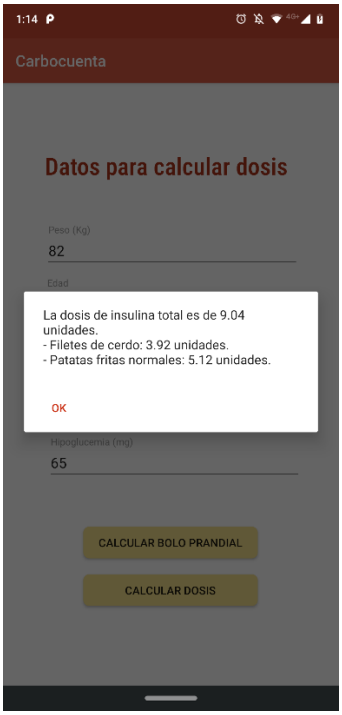

Figura 74. Mensaje de bolo prandial para usuario sano

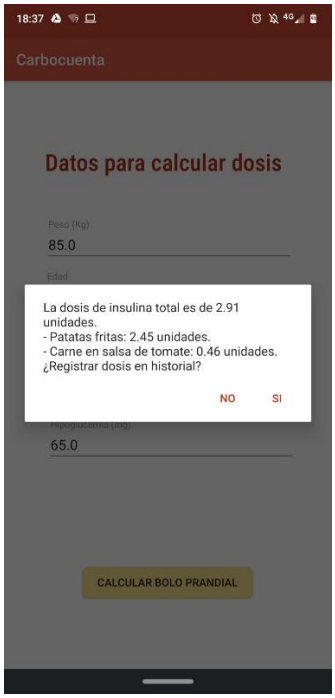

Figura 75. Mensaje de bolo prandial para usuario diabético

Menú de usuario

En este menú se le ofrece al usuario poder **modificar** su usuario, ver sus **comidas recientes**, ver **histograma** de dosis de insulina (en el caso de ser diabético), **calcular bolo corrector** y **cerrar sesión**.

Figura 76. Vista para modificar usuario

Figura 77. Vista de comidas recientes

Figura 78. Vista de histograma de dosis de insulina

Figura 79. Vista para calcular bolo corrector

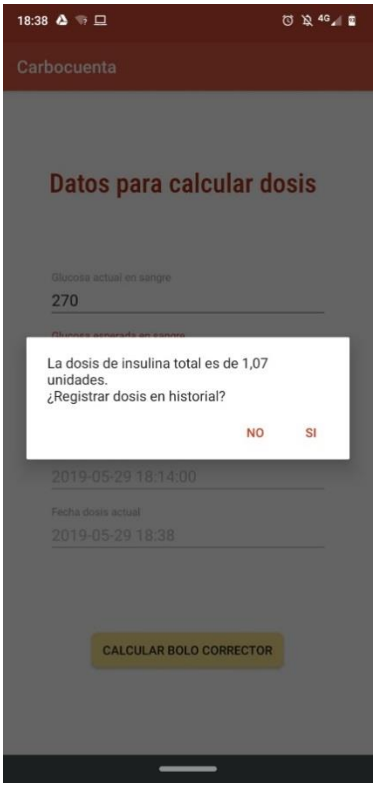

Figura 80. Mensaje sobre dosis correctora

Resultados

Durante la fase de pruebas se han obtenido diferentes resultados que garantizan la mantenibilidad y usabilidad de la aplicación. En este apartado las pruebas están enfocadas al rendimiento y porcentajes de fallo y acierto que proporcionaron los algoritmos utilizados en el proyecto.

Por una parte, el tiempo de espera en el algoritmo *KMeans* era en un principio de casi 5 minutos, dato que se consiguió reducir al 94,4% de lo que duraba en un principio, es decir, que actualmente el tiempo medio de espera es de 2,3 segundos.

```
D/Recalculation ended in:: 5 ms.
D/Iteration ended in:: 88 ms.
D/Recalculation ended in:: 6 ms.
D/Iteration ended in:: 87 ms.
D/Recalculation ended in:: 7 ms.
D/Iteration ended in:: 74 ms.
D/Recalculation ended in:: 6 ms.
D/Iteration ended in:: 69 ms.
D/Recalculation ended in:: 5 ms.
D/Iteration ended in:: 76 ms.
D/Recalculation ended in:: 6 ms.
D/Iteration ended in:: 66 ms.
D/Recalculation ended in:: 6 ms.
D/Iteration ended in:: 66 ms.
D/Recalculation ended in:: 6 ms.
D/Iteration ended in:: 74 ms.
D/Recalculation ended in:: 5 ms.
D/Iteration ended in:: 65 ms.
D/Recalculation ended in:: 6 ms.
D/Iteration ended in:: 64 ms.
D/Recalculation ended in:: 5 ms.
D/Iteration ended in:: 72 ms.
D/Recalculation ended in:: 6 ms.
D/Iteration ended in:: 65 ms.
D/Recalculation ended in:: 7 ms.
D/Iteration ended in:: 67 ms.
D/Recalculation ended in:: 9 ms.
I/System.out: KMeans algorithm ended in: 2518 ms.
```

Figura 81. Tiempo de espera actual del kMeans

Para la obtención de las tasas de acierto y fallo, se realizaron 50 pruebas con imágenes diferentes en las que algunas tenían alimentos en común y se fueron registrando los intentos que eran necesarios para obtener los alimentos correctos, los alimentos encontrados y los no encontrados. Gracias a esto se obtuvieron los siguientes resultados:

- ***Tasa de acierto:*** 91,5% para 50 imágenes con un total 75 alimentos encontrados.
- ***Tasa de fallo:*** 8,5% para 50 imágenes con un total de 6 alimentos no encontrados.
- ***Más intentos:*** se obtuvo un 21,17% más de intentos para conseguir obtener todos los alimentos de la imagen.

Con estos datos se puede deducir que el algoritmo obtiene buenos resultados, aunque en algunas ocasiones va a necesitar procesar la imagen más de una vez para obtener los alimentos correctos.

Finalmente, se realizaron pruebas para comprobar los tiempos de espera para la conexión con la base de datos, en este caso es útil volver al apartado de Calidad y Rendimiento con el fin de observar las gráficas mostradas en él y poder observar el comportamiento de la CPU, ya que en todas las figuras se realizan peticiones a la BD y esto ayuda a decidir si se debe optimizar la base de datos.

Para obtener estos tiempos se hizo uso de PostMan, una herramienta que ayuda a realizar pruebas con scripts php y que devuelve la respuesta de la petición realizada junto con parámetros de interés como es el tiempo de espera de respuesta, es decir, lo que se tarda en realizar la conexión la base de datos, realizar una petición a esta, obtener un resultado y devolverlo. Con esto, los tiempos quedaron como se muestra en la Figura 81 y la media de tiempo que se obtuvo fue de 44 ms.

get_user.php	GET	57
add_plate_food.php	POST	50
add_insulin_dose.php	POST	52
add_meal.php	POST	51
add_meal_info.php	POST	50
add_user.php	POST	50
get_aliment_id.php	GET	47
get_aliments.php	GET	52
get_food_name.php	GET	49
get_insulin_dose_with_meal.php	GET	53
get_last_insulin_dose.php	GET	52
get_meals_info.php	GET	50
get_plate_food.php	GET	50
get_recent_insulin_doses.php	GET	51
get_recent_user_meals.php	GET	52
get_usda_id.php	GET	49
get_user_meals.php	GET	52
update_user.php	POST	50

Figura 82. Tiempos de espera en la conexión con la BD

Contribuciones

En el comienzo del proyecto se realizó una reunión, no solo para asentar las bases y la estructura del trabajo que se realizaría posteriormente, sino para definir claramente los requisitos necesarios de la aplicación. Definimos que era de máxima importancia tanto el trabajo de investigación y codificación como su correcta documentación. La unión de estas tres partes era indispensable para conseguir el resultado buscado, por lo que todos los miembros del equipo han formado parte de todas ellas en mayor o menor medida. A continuación, se procede a explicar más detalladamente la contribución individual de cada uno de los miembros del equipo:

LAURA

Como se menciona anteriormente una de las bases sobre las que asentamos nuestro proyecto fue la investigación. Era de vital importancia informarse de los distintos métodos y opciones que teníamos para realizar la aplicación para poder elegir cuál de todos ellos encajaba mejor en el proyecto. Por lo tanto, desde un inicio me centré más en la investigación, en primer lugar, sobre las formas de segmentar la imagen, y en segundo lugar, sobre como etiquetarlas y clasificarlas correctamente. Ambos campos son muy extensos por lo que recibí la ayuda de mi compañera Milagros para poder avanzar más rápido.

Dentro de la segmentación de imágenes existen multitud de métodos ya desarrollados por lo que mi trabajo consistió en investigar todos ellos, con sus ventajas y

desventajas, valorando si su implementación era viable y encajaba con la aplicación. Toda esta información nos ayudó a decidir qué método elegiríamos para la segmentación. Una vez decidimos el algoritmo *Kmeans* comenzó la etapa de codificación del algoritmo donde me centré en que arrojara resultados eficientes, pero también en implementarlo rápidamente ya que era necesario para seguir el curso de la aplicación. Mis compañeros contribuyeron también a esta tarea aportando código e ideas.

Después de realizar la segmentación de las imágenes era necesario ser capaces de clasificarlas automáticamente y que, además, la aplicación continuase aprendiendo constantemente. Debido a esto comencé la etapa de investigación sobre algoritmo de segmentación y sobre *machine learning*. Para tomar la decisión sobre que algoritmo usaríamos para la clasificación era necesario encontrar un algoritmo que fuese compatible con los resultados arrojados por el Kmeans. Esta etapa me resultó muy complicada por lo que tuvimos que acudir a varias reuniones con nuestros tutores. Una vez se decidió que sería el algoritmo K-nearest neightbors para la clasificación comencé a formarme sobre el tema. Este algoritmo nos permitía tratar cada uno de los pixeles como puntos en un plano lo que me permitió encajarlo perfectamente al Knn.

Además era necesario dotar al algoritmo *Knn* de un conjunto de datos de entrenamiento con los que fuese capaz de entrenar y aprender. Con este objetivo creamos una tabla en la base de datos en la que almacenaríamos todos los datos referentes al conjunto de entrenamiento. Dicho algoritmo toma mejores decisiones cuanto mayor es el conjunto por lo que comencé a calcular los datos necesarios para completar la BBDD que utilizaría posteriormente el *Knn*. Los datos obtenidos provienen no solo de la propia aplicación sino que además utilicé Matlab como herramienta auxiliar para contrastar que los datos obtenidos eran correctos. Este código fue configurado junto con mi compañera Milagros con el objetivo de conseguir datos más fiables disminuyendo de esta forma la tasa de errores de la aplicación.

Mientras yo realizaba el trabajo anteriormente mencionado mis compañeros se dedicaron a establecer la base de datos necesaria para guardar los resultados obtenidos por los algoritmos y a implantar la base de la aplicación. Además, era necesario comenzar con el trabajo de documentación ya que debe ser paralelo a la investigación, por esta razón comencé a escribir la memoria del proyecto planteando la estructura que debía seguir. Me he encargado de la redacción de la parte principal incluyendo no solo la información necesaria sobre los algoritmos anteriormente mencionados y su investigación previa sino además sobre algunos de los conocimientos que nos han sido necesarios para comprender el objetivo del proyecto como la diabetes o la misma inteligencia artificial. Además, he ido corrigiendo las distintas versiones que íbamos redactando, teniendo en cuenta las anotaciones y correcciones que nuestra directora María nos iba proporcionando.

MILAGROS

Desde un inicio y una vez sabidos los requisitos para la creación de la aplicación, todos contribuimos a la identificación de los módulos principales de los que se componía la aplicación, es decir, el usuario, las imágenes y la comida, por lo que propuse utilizar una arquitectura Modelo-Vista-Controlador (MVC) de tipo activo, ya que teníamos experiencia previa con ese patrón y por la forma de actuar de la aplicación era también coherente elegir un tipo de arquitectura así. Una vez aceptada esta propuesta por el equipo procedí a la creación del modelado de software de la aplicación, ya habían sido especificados y validados los requisitos funcionales y aunque durante la implementación aparecieron requisitos emergentes, esto fue suficiente para empezar a crear los diagramas que componen el modelo. Empecé por la creación de los casos de uso, de flujo del usuario y de actividad, para así tener aún más idea de cómo debía actuar cada

funcionalidad de la aplicación, esto me proporcionó más idea sobre la descomposición y la jerarquía entre clases y objetos que tenía el sistema por lo procedí a la creación de los diagramas de clases y de secuencia en las siguientes capas:

- Presentación: donde iba a tener todo lo relacionado con la vista, es decir, los activities, los comandos, el controlador y el dispatcher.
- Negocio: donde se encontraría todos los procesos de negocio y la representación de datos, es decir, los servicios de aplicación y los transfers.
- Integración: donde se encontraría todo lo relacionado con la comunicación a la base de datos y la persistencia de estos, es decir, los data Access object (DAO).

El objetivo era crear un modelo altamente cohesivo y débilmente acoplado, por lo que conté con interfaces y singletons para evitar acoplamientos y finalmente tener una especificación completa de lo que iba a ser el sistema final.

Una vez realizado el diseño, procedí a la implementación del sistema siguiendo la arquitectura establecida, por lo que cada capa representaba un paquete y dentro de ellas debía existir un paquete para cada módulo. Durante este proceso dependí mucho de mis compañeros ya que necesitaba la parte fundamental de la aplicación para las imágenes; los algoritmos, las APIs y los scripts de conexión al servidor aparte de la seguridad.

La parte más complicada durante la fase de implementación fue la creación de algunas vistas o comunicación entre ellas, ya que mi objetivo era que la interfaz gráfica fuera intuitiva, tuviera buena estética y que proporcionara bastante información. Por lo que tuve que coordinarme con Jose para la creación de la estructura de la base de datos, ya que existen partes en las que se muestra información muy específica de los alimentos o del usuario y no quería provocar redundancias en las tablas. Tras conseguir esto, también me centré mucho en que el usuario tuviera muchas facilidades:

- Todas las vistas que incluyen el ingreso de datos, si existen datos de ese usuario en la BD, se autocompletarán para hacer el proceso más dinámico.
- Siempre se le validaran los campos, esto ayuda a mantener la persistencia en la BD y que el usuario calcule dosis erróneas.
- El usuario puede acceder a todas las imágenes de las que disponga su terminal y su nube de Google aparte de la opción de poder realizar una fotografía desde la cámara.
- La opción de recortar un alimento, que a la vez fue la vista más complicada, debido a que mi objetivo era ser muy preciso con el procesamiento del alimento a añadir para que la base de conocimientos del algoritmo de mi compañera Laura fuera consistente y con valores semejantes. Por lo que tomé la decisión de utilizar el mismo proceso de lo que se conoce hoy en día como *sticker* para poder recortar una imagen a tu gusto con fondo transparente y obtener las medias y pixeles con el menor margen de error posible.

Además de esto, tuve que aprender sobre el funcionamiento de los hilos en Android y las clases de las que disponía para poder realizar procesos asíncronos que realicen tareas paralelas al flujo principal de la aplicación, como era, por ejemplo, la obtención de los alimentos una vez procesada la imagen mientras el usuario espera la lista de predicciones.

Finalmente, hay que añadir que dediqué mucho tiempo a la realización de pruebas, en esta fase me dedicaba a probar los métodos que iba implementando, tras ello probaba las clases relacionadas, para comprobar que todas las funcionalidades nuevas se ejecutaban correctamente. Las pruebas de sistema realizadas las hice junto a mis compañeros que también se prestaron a solucionar errores del sistema.

JOSE ANTONIO

En la fase inicial del proyecto, una de las dudas más importantes eran las tecnologías a utilizar. Una de las más sencillas de decidir fue el sistema operativo de Android, ya que todos los miembros del proyecto utilizábamos dispositivos Android. En consecuencia, y debido a nuestra experiencia en esta interfaz de desarrollo, nuestra herramienta principal para desarrollar fue Android Studio. Pero el resto de las tecnologías a utilizar no estaban tan claras como esta, por lo que se dedicó un tiempo a la investigación de tecnologías que más nos convenían.

En lo personal, me centré más en investigar y sopesar pros y contras del cliente de base de datos a utilizar, ya que había diferentes opciones que podrían ser viables para nuestra aplicación, como SQLite, Cloud SQL o MySQL. Otra de las investigaciones en las que me centré fue en profundizar en la configuración de Apache para hacerlo funcionar correctamente para nuestros casos de uso. En cuanto al cálculo matemático que realizamos en ciertas fases del proyecto, la tecnología utilizada fue MatLab. Esta tecnología fue recomendada por nuestros tutores, pero al tener poca experiencia en ella, yo y mis compañeros dedicamos un tiempo a investigar cómo funcionaba y como se estructuraba y comportaba su lenguaje para poder sacar el máximo partido a la funcionalidad que ofrece.

Para conseguir los datos sobre los alimentos para nuestra aplicación, necesitábamos una base de datos para consultar los nutrientes de dichos alimentos. Yo me encargué de la investigación de base de datos que contuvieran esa información y encontré la base de datos de USDA. Esta base de datos, aparte de ofrecer los datos que necesitábamos, ofrece una API para realizar consultas mediante peticiones, por lo que conseguí una clave para usar esa API e implementamos las consultas en nuestra aplicación.

Debido a que esta base de datos estaba en inglés, necesitábamos un sistema para traducir las búsquedas que el usuario pudiese necesitar. Para ello, investigué sobre ello y encontré la API de traducción que Google ofrece, Google Translate. Sin embargo, esta API no ofrecía servicio directamente a Android. Aun así, disponía de un servicio REST que permite mandar peticiones HTTP para traducir palabras disponiendo de una clave API. Investigué cómo conseguir esa clave API y qué consultas se podían realizar al servicio REST y, con ayuda de mis compañeros, implementamos las consultas en nuestra aplicación para así poder traducir desde ella.

Para acceder a la base de datos del servidor desde el cliente es necesario que el servidor ofrezca unas rutas para realizar peticiones. Apache se encarga de ofrecer este servicio, por lo que decidimos implementarlo en el servidor. Al realizar una petición a una ruta específica del servidor, necesitábamos que el servidor ejecutase scripts que accediesen a base de datos y la actualizasen con los parámetros que el cliente hubiese mandado en la petición. Para ello, investigué cómo realizar scripts en PHP y acceder a base de datos desde ellos, y los situé para que Apache lo utilizase para responder a las peticiones que recibiese desde el cliente.

Teniendo esto completado en el proyecto, necesitábamos saber la estructura de nuestra base de datos. Coordinándome con mis compañeros, ideé una estructura inicial de nuestra base de datos que contenía usuarios, alimentos, etc. Esta estructura fue puliéndose a medida que avanzaba el proyecto por necesidades nuevas o no contempladas en el inicio de éste.

En etapas más avanzadas del proyecto, concluimos que sería positivo aumentar la seguridad que ofrecía nuestra aplicación para protegerla de posibles ataques informáticos. Llevé a cabo la investigación de posibles mejoras de seguridad para nuestra aplicación

como la encriptación de datos sensibles usando AES ECB, la activación del HTTPS para conectar cliente y servidor, la restricción de permisos de la aplicación, etc.

Así como la seguridad, tuvimos en cuenta cómo llevar a cabo el mantenimiento tanto de la aplicación como de los datos contenidos en el servidor. Para ello, añadí una entrada al *crontab* del servidor para realizar un *backup* diario de nuestra base de datos y me encargué de documentar propiamente el código de la aplicación para facilitar su mantenibilidad en el futuro. Además, junto con mis compañeros, realizamos arreglos de bugs que íbamos encontrando en las pruebas que ejecutábamos sobre nuestra aplicación.

Conclusiones

El método de segmentación proporcionado por el algoritmo *KMeans* se ha mostrado exitoso en la segmentación de imágenes. No solo se ha conseguido una separación idónea por colores, sino que además se ha obtenido en un tiempo competente. Por su parte, los resultados obtenidos gracias al clasificador *Knn* mediante el cálculo de distancias euclídeas nos han proporcionado los datos convenientes para llevar a cabo la correcta clasificación de los distintos alimentos.

Además, los métodos utilizados para el cálculo de las cantidades y proporciones de carbohidratos y, por lo tanto, del cálculo de nutrientes y la dosis de insulina apropiada para cada usuario han reflejado datos fiables y seguros.

Asimismo, para dotar a la aplicación de una funcionalidad total se ha añadido la posibilidad de agregar alimentos manualmente en el caso de fallo del algoritmo de clasificación. Consiguiendo de esta forma, no solo proporcionar siempre información nutricional útil, sino también lograr que la aplicación continúe siempre aprendiendo y mejorando a través del propio usuario.

Por último, toda la información generada por la aplicación es considerada de gran utilidad y, por lo tanto, es guardada de forma dinámica en la BBDD.

De esta forma podríamos afirmar que se ha conseguido desarrollar una aplicación Android funcional con una estructura cliente servidor capaz de segmentar y clasificar imágenes mediante el estudio de color.

A partir de este trabajo y los resultados obtenidos queda una amplia base de conocimiento para el trabajo dentro del campo de procesamiento de imágenes a través del color. En un futuro podría ampliarse a, no solo el estudio por color, sino también por textura, lo que proporcionaría una mayor cantidad de información al clasificador haciéndolo más fiable y puliría la segmentación de la imagen. No sería el único aspecto a poder mejorar, pudiendo utilizar algoritmos más complejos como el de Sobel para identificar bordes y nos permitiría recortar el plato automáticamente y así realizar las mismas tareas mejorando los resultados actuales.

El campo de la aproximación, no solo de carbohidratos, sino también de dietas añadiría mayor funcionalidad a la aplicación haciéndola más completa, pudiendo proponer al usuario recetas a partir de los alimentos que ha proporcionado o dando más información al usuario mediante email o incluso proporcionándole un gestor nutricional en función de su altura y peso.

También sería interesante el desarrollo de esta aplicación para distintas plataformas, como iOS, ampliando así la cobertura a una mayor cantidad de usuarios.

En cuanto a seguridad, es necesario cambiar las peticiones a HTTPS para evitar intrusiones o robos de sesión en redes que no sean de confianza, por otro lado, el servidor debería de permanecer en una extranet para poder garantizar la seguridad de los datos almacenados en la base de datos y restringir el acceso al servidor como administrador.

Por último, para el mantenimiento de la aplicación sería mejorar el sistema de monitorización y de alertas, para ello, se utilizaría un servicio ELK, el cual mediante *elasticsearch* accedería a los logs o archivos de interés con el fin de obtener métricas para finalmente obtener una gráfica, tabla, etc. que permita plasmar esa información en datos relevantes y métricas útiles para mantener el sistema, mientras que, por otro lado, añadir

también una herramienta de alertas que pueda acceder a esta información con el fin de notificar a las personas responsables de la aplicación. Además, crear un sistema de *disaster recovery* por si se cayera el servidor, que siempre hubiera otro servidor con los datos sincronizados y preparado para seguir dando el servicio.

Conclusions

The segmentation method supplied by the *KMeans* algorithm has been proven very successful in the image segmentation area. It has not only achieved a suitable separation by color, it has also made it in an acceptable time. In the other hand, the results acquired by the *Knn* classifier through the calculation of Euclidean distances, had provided convenient data to take on the classification of the different aliments.

Furthermore, the methods used for the calculus of the quantities and proportions of carbohydrates and, therefore, the calculus of nutrients and the appropriate doses of insulin for each user have been reflected reliable and secure.

Likewise, with the aim of achieving total functionality it has been attach the possibility to add manually aliments in case of failure of the classification algorithm. That way, we have not only accomplished providing useful nutritional information, but also that, the application continues learning and improving through the user itself.

Lastly, all the information generated by the application is considered highly useful and, therefore, it will be saved dynamically in the DDBB.

We could affirm that we have developed a totally functional Android application with a client server structure that is able to segment and classify images through color study.

From this project and the results obtained remains a broad knowledge base of work within the field of image processing through color. In the future it could be extend to, not only to the study through color but through texture, which would provide a larger quantity of information for the classifier making it more reliable, and also, would refine the image segmentation. It wouldn't be the only aspect to upgrade; we could use more complex algorithms such as the Sobel algorithm to identify edges and allow us to cut the plate automatically to improve the actual results.

Besides, it could be interesting adding not only the estimation of carbohydrates but also the approach on diets; making the application more complex, including recipes containing the aliments supplied by the user or giving more information by email or even providing a nutritional manager in function of its height or weight.

Finally, it would be compelling the development for different platforms, such as iOS, increasing the coverage of users.

In terms of security, it's necessary to chague the type of requests to HTTPS to avoid instrusions or theft of user sessions on networks that are not trusted, on the other hand, the server should remain on an extranet in order to guarantee the security of the information in the database and finally, restrict access to the server as administratos.

Finally, the maintenance of the application would be to improve the monitoring and alerts system, for this, an ELK service could be the best option, because it uses *elasticsearch* service tha access to logs or files of interest in order to obtain metrics to finally show every important data in a graphic, table, etc. With ELK, we could translate this information into relevant data and useful metrics to maintain the system, however, also add an alert tool that can access to this information with the purpose of notifying the people who are responsible for the application. In addition, create a disaster recovery

system to use in case the server goes down, in this way there is always another server with the data synchronized and ready to continue giving the service.

Lista de referencias

[1] J.I.Hidalgo, et al., *Modeling glycemia in humans by means of Grammatical Evolution*, Appl. Soft Comput J.(2013),https://doi.org/10.1016/j.asoc.2013.11.006

[2] Esmeralda Colino. (2015*). Fundación para la Diabetes: Tipos de diabetes.* Recuperado de https://www.fundaciondiabetes.org/infantil/177/tipos-de-diabetes-ninos

[3] Fundación para la Diabetes (2016) *Fundación para la Diabetes: La diabetes en España*. Recuperado de https://www.fundaciondiabetes.org/prensa/297/la-diabetes-en-espana

[4] Shapiro, L. & Stockman, G. (2000) *Computer Vision.* Seattle, Washington. EEUU: Prentice Hall Editorial.

[5] Maravall Gómez-Allende, D. (1993) *Reconocimiento de formas y visión artificial.* Madrid, España: RA-MA Editorial.

[6] *Deep Learning, Inteligencia Artificial y Machine Learning*. Recuperado de https://www.blog.andaluciaesdigital.es/deep-learning-inteligencia-artificial-y-machine-learning/

[7] Sucar, L.Enrique & Gómez, Giovani (2011) *Visión Computacional.* Instituto Nacional de Astrofísica, Óptica y Electrónica, México. https://www.researchgate.net/profile/Luis_Sucar/publication/267295870_Vision_Computacional/links/54d8cae30cf2970e4e7940c1/Vision-Computacional.pdf

[8] Statcounter: Mobile Vendor Market Share Worldwide http://gs.statcounter.com/vendor-market-share/mobile

[9] *Android vs iPhone: la guerra de los smartphones en cifras.* Recuperado de https://computerhoy.com/reportajes/industria/android-vs-iphone-guerra-smartphones-cifras-271447

[10] Cloud SQL. Recuperado de https://cloud.google.com/sql/?hl=es

[11] Android Documentation: Device Storage. Recuperado de https://developer.android.com/training/data-storage/files

[12] About SQLite. Recuperado de https://www.sqlite.org/about.html

[13] MySQL Documentation. Recuperado de https://dev.mysql.com/doc/

[14] About MariaDB. Recuperado de https://mariadb.org/about/

[15] Pajares, G & De la Cruz, JM (2008) *Visión por computador: imágenes digitales y aplicaciones.* Ra-Ma Editorial.

[16] MacQueen, J. (1867) *Some methods for classification and analysis of multivariate observations.* University of Cailfornia, Los Angeles. EEUU.
[17] Hong Yao & Qingling Duan & Daoliang Li & Jianping Wang (2013) *Mathematical and Computer Modelling.* Pekín, China. https://www.journals.elsevier.com/mathematical-and-computer-modelling
[18] Maravall Gómez-Allende, D. (1993) *Reconocimiento de formas y visión artificial.* Madrid, España: RA-MA Editorial.
[19] Mitchell, Tom M. (1997) *Machine learning.* Pittsburgh, Pensilvania. EEUU. WCB McGraw-Hill Publisher.
[20] Arquitectura cliente-servidor. Recuperado de http://somebooks.es/arquitectura-clienteservidor/
[21] Modelo Vista Controlador (MVC). Recuperado de https://si.ua.es/es/documentacion/asp-net-mvc-3/1-dia/modelo-vista-controlador-mvc.html
[22] Android Manifest. Recuperado de https://developer.android.com/guide/topics/manifest/manifest-intro?hl=es-419
[23] ¿Qué es el Gradle? Recuperado de https://androidstudiofaqs.com/conceptos/que-es-gradle-en-android-studio
[24] *Activity*. Recuperado de https://developer.android.com/guide/components/activities.html?hl=es-419
[25] *Intent.* Recuperado de https://developer.android.com/guide/components/intents-filters?hl=es-419
[26] *Fragments.* Recuperado de https://developer.android.com/guide/components/fragments?hl=es-419
[27] KMeans en Matlab. Recuperado de https://es.mathworks.com/help/stats/kmeans.html
[28] NDB API. Recuperado de https://ndb.nal.usda.gov/ndb/doc/index#
[29] *Cloud Translation.* Recuperado de https://cloud.google.com/translate/
[30] AsyncTask. Recuperado de https://corochann.com/asynctask-usage-summary-341.html
[31] Cálculo de la dosis de insulina. Recuperado de https://dtc.ucsf.edu/es/tipos-de-diabetes/diabetes-tipo-2/tratamiento-de-la-diabetes-tipo-2/medicamentos-y-terapias-2/prescripcion-de-insulina-para-diabetes-tipo-2/calculo-de-la-dosis-de-insulina/

[32] About Apache. Recuperado de https://httpd.apache.org/ABOUT_APACHE.html https://developer.android.com/training/articles/security-tips?hl=es-419
[33] Patrick Favre-Bulle (2018) *Symmetric Ecnryption with AES in Java and Android.* Recuperado de https://proandroiddev.com/security-best-practices-symmetric-encryption-with-aes-in-java-7616beaaade9
[34] What is a block cipher? Recuperado de https://www.wolfssl.com/what-is-a-block-cipher/
[35] ECB – AES electronic codebook mode encryption. Recuperado de https://infocenter.nordicsemi.com/index.jsp?topic=%2Fcom.nordic.infocenter.nrf52832.ps.v1.1%2Fecb.html
[36] *Cipher*. Recuperado de https://developer.android.com/reference/javax/crypto/Cipher
[37] *SecretKeySpec*. Recuperado de https://developer.android.com/reference/javax/crypto/spec/SecretKeySpec
[38] Sugerencias de seguridad Android. Recuperado de https://developer.android.com/training/articles/security-tips?hl=es-419
[39] IPC mechanisms. Recuperado de https://stackoverflow.com/questions/5740324/what-are-the-ipc-mechanisms-available-in-the-android-os
[40] Apps de diabetes. Recuperado de https://republikadiabetes.com/apps-contar-raciones-calculo-dosis-registro-datos/
[41] Certificación SQAS. Recuperado de https://www.bureauveritas.es/home/about-us/our-business/our-business-certification/su-sector/transport-and-distribution/transporte-sqas
[42] USDA Food Composition Database. Recuperado de https://ndb.nal.usda.gov/ndb/
[43] ¿Qué es REST? Recuperado de https://www.arquitecturajava.com/que-es-rest/