UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

Grado en Ingeniería Informática en Ingeniería del Software

Trabajo Fin de Grado

Asistente para identificación de gustos culinarios en grupos de personas

Alfonso Montero Barquilla

Noviembre, 2017

RESUMEN

Cada vez más, en nuestra sociedad, se presentan momentos de incertidumbre a la hora de elegir la carta de los restaurantes. ¿Me gustará esté plato? ¿Seré alérgico a estos ingredientes? o incluso preguntarse esto mismo hacia tus compañeros de comida. Con frecuencia sucede, que tenemos que preguntar al camarero que ingredientes posee una tapa o un plato.

Ante esta situación, tenemos un problema ya que todo este tipo de incertidumbre o desconocimiento puede dar lugar a intoxicaciones alimentarias o a discusiones y malas interpretaciones entre los compañeros de comida, como pensar que se han elegido los platos que menos nos agradan, por ser los que más agradan a otro, o pensar que no se ha contado con nuestra opinión para elegir la comida. Otro de los problemas que conlleva elegir comida para compartir es que no sabemos que pedir o no, sin saber los gustos de tus compañeros.

Existen numerosas soluciones para evitar este tipo de problemas, entre ellas, destacar las siguientes:

- Crear una carta en la que aparezcan exactamente todos los ingredientes de todos los platos/tapas para evitar intoxicaciones o tener que preguntar al camarero constantemente. Para este caso en concreto, la carta sería muy difícil de leer, poco atractiva visualmente y no solventaría los problemas asociados a las discusiones entre grupos de personas.

- Intentar ponernos de acuerdo sin discusiones a la hora de elegir los platos/tapas de los restaurantes entre los compañeros de comida. Este proceso es realmente difícil, ya que cada persona es diferente y tiene diferentes gustos y criterios, por lo que alguien quedaría insatisfecha por la elección.

- Ser independientes y que cada persona pida individualmente sus tapas/platos. Para esta solución, estamos ocasionando problemas al camarero en el momento de abonar la comida, ya que cada persona tendría que abonar su pedido individualmente, aun estando en grupo.

- Hacer uso de la tecnología para evitar todos los problemas detallados. Dicha solución, sería un engorro sobre todo para aquellas personas que quisieran distraerse del mundo tecnológico y no quieran hacer uso de la misma cuando visitan restaurantes.

En este último punto se centrará el proyecto ya que vamos a hacer uso de una tecnología que la mayoría disponemos y que solamente una pequeña minoría de

personas considera un engorro. Este es el caso, de los **dispositivos móviles** que presentan una penetración muy alta en nuestra sociedad, ya que casi todo el mundo dispone de uno.

Justamente, el objetivo de este proyecto es hacer uso de un dispositivo móvil para evitar los problemas anteriormente detallados. La manera de evitarlos sería, mediante la creación de un App móvil.

La aplicación móvil facilitará a las personas o grupos de personas, su elección de platos en cualquier tipo de local de comida y evitará problemas alérgicos. Este proceso se llevará a cabo mediante algoritmos que descifren los gustos culinarios de cada persona y mediante una tecnología para compartir preferencias culinarias entre comensales. Inicialmente la aplicación tomará datos culinarios muy genéricos para una primera valoración, que se irán refinando a medida que el usuario interactúe con la misma.

En este documento se detallarán los pasos que se han seguido para desarrollar esta aplicación. Inicialmente, se estuvo haciendo un estudio de las tecnologías y del desarrollo en general, pensando en que la solución final sea la más adecuada. Seguidamente, se investigaron, los posibles competidores y se desarrolló un resumen sobre las decisiones tomadas. Continuamos, con unas explicaciones sobre el manual para el usuario y el programador. Además, este proyecto ofrece la posibilidad de poderse ampliar y mejorar en un futuro. Finalmente se hizo un análisis de resultados con valoraciones y conclusiones.

___CONTENIDO___

___ILUSTRACIONES___

___TABLAS___

___1. INTRODUCCIÓN___

1.1 DESCRIPCIÓN DEL PROYECTO

Cuando visitamos un **restaurante** o algún local de comida son numerosas las veces que no sabemos exactamente que tapa o plato elegir. Además, también sucede que tenemos que preguntar constantemente al camarero cuales son los **ingredientes** que posee una tapa o un plato para comprobar si somos **alérgicos** a él o no. Todo esto, también se produce cuando acudimos a algún restaurante de forma **grupal**. Consecuencias de ello son discusiones entre las personas participantes en la comida ya que alguien podría quedar desconforme con la elección, debido a que no sabemos los gustos de todas las personas. Otra consecuencia de mayor importancia, podría ser una intoxicación alimentaria debido a que ingiramos algún alimento al que seamos alérgicos.

Para evitar estas situaciones, se podría hacer uso de la **tecnología** para intentar evitar los problemas que puedan surgir. En este contexto podría realizarse la implementación de una página web en la que podamos resolver todos estos problemas que se nos presentan. Las diferentes formas de acceder a esta web podrían ser entre otras desde un ordenador, móvil o tablet. Resultaría engorroso cargar constantemente con el ordenador o la tablet a cualquier local de comida que visitemos, por lo que una solución sería hacer uso del móvil ya que una gran mayoría dispone de uno y lo lleva siempre consigo. Acceder a cualquier web desde el móvil, podría presentar muchos problemas de accesibilidad ya que fueron creadas para otro tipo de tamaño. Por lo tanto, lo más oportuno sería crear una aplicación móvil con mayor grado de accesibilidad, manejo y entendimiento en la que poder resolver todos estos problemas.

Actualmente existen varias aplicaciones móviles de restaurantes que poseen algunas de las siguientes funcionalidades:

- Posibilidad de poder elegir que platos o tapas vas a comer.
- Pedir para llevar la comida que deseemos.
- **Reservar** una mesa en uno de tus restaurantes favoritos.
- Realizar y ver **comentarios** sobre los platos que queramos.
- Ver listas de restaurantes cercanos mediante **geolocalización**.

Este es el caso de aplicaciones móviles como "*el tenedor*" [17] o "*atrápalo restaurantes*" [18] que presentan algunas de estas funcionalidades. Ambas, presentan los problemas que se detallan anteriormente ya que ninguna es capaz de resolver la incertidumbre, discusiones o problemas alérgicos que se puedan ocasionar.

Este proyecto quiere dar un paso más allá en el desarrollo de aplicaciones móviles sobre comida, ya que no va abordar las funcionalidades básicas que poseen este tipo de aplicaciones sobre restaurantes sino que va a ser capaz de **pensar** por

nosotros a la hora de elegir algún plato o tapa y nos va a ofrecer unas **posibilidades** sobre prevención de intoxicaciones alimentarias, que ninguna o pocas aplicaciones nos ofrece en la actualidad.

Debido entre otras cosas, a los problemas existentes en la actualidad que se han detallado anteriormente, para este proyecto se ha abordado la elaboración de una aplicación móvil **inteligente** capaz de facilitar la elección de platos en locales de comida. No obstante, también evita problemas de alergias ya que cumple también con esta función (avisarnos en el caso de que seamos alérgicos a un ingrediente).

En la primera toma de contacto con la aplicación, esta nos tomará algunos parámetros en los que basarse para sacar sus primeros resultados.

En principio, la idea es que el usuario haga interacción de la misma aplicación para todos los locales a los que acuda y que acumule información acerca de sus **gustos** y **alergias**. A medida que el usuario utilice la aplicación, esta tendrá más probabilidad de que sus resultados sean **óptimos**. El proceso de uso de la aplicación, va a ser elegir un local de comida y elegir los platos que se desea comer.

La aplicación también está pensada para **grupos de personas** ya que en pocas ocasiones comemos solos. Para llevar a cabo este proceso, cada usuario intercambiará sus preferencias con sus compañeros, de tal modo que cuando acabe el intercambio, cada usuario tendrá un nuevo informe que sea conforme a todas las personas que participen en la comida. En este contexto de intercambio de preferencias, tuvimos que hacer uso de una tecnología novedosa y en pleno desarrollo conocida como **Wifi Direct.**

Los datos de cada usuario no van a ser guardados en la nube o en algún tipo de base de datos que se pueda acceder mediante internet. En este caso la aplicación va a ser **offline**, es decir, no vamos a necesitar conexión a internet para interactuar con ella. Por lo tanto los datos serán guardados en la memoria interna del móvil mediante **SQLite**. Este beneficio podrá ser de gran utilidad cuando viajemos a un país extranjero o cuando nos hayamos quedado sin datos de red en el dispositivo.

Como posibles mejoras para este proyecto destacar entre otras, la implementación de notificaciones push o realizar un informe al restaurante con información acerca de sus platos. Todas las posibles mejoras se comentarán más adelante en un apartado.

1.2 INTERNET DE LAS COSAS

Cada vez, con más frecuencia, somos testigos de cómo la tecnología influye en nuestras tareas rutinarias. Tal es el caso de que casi cinco mil millones de cosas están conectadas a internet. En este contexto, surge el término **Internet de las Cosas** (*Internet Of Things*), cuyo objetivo principal es intentar conectar todos los objetos que nos rodean, siendo capaces de recoger información sobre ellos, procesarla y compartirla. Este **proyecto** se hace partícipe de este término ya que somos capaces de **recoger** información culinaria de las personas, **procesando** la información de tal modo, que ayudamos al usuario en su elección e incluso tenemos la posibilidad de **compartir** estos datos con nuestros compañeros.

1.3 MOTIVACIÓN DEL PROYECTO

Personalmente el proyecto resultó ser un gran RETO para mí, debido entre otros motivos a que me he encontrado trabajando mientras desarrollaba parte del mismo y sobre todo a la posibilidad de emprender un futuro profesional con él.

Este proyecto surgió como iniciativa **propia** basándose anteriormente en unos criterios impuestos. En realidad, no existe ninguna aplicación con exactamente las mismas funciones, de ahí, que fuese tan apetitoso iniciar un proyecto con estas características.

Destacar no solo la novedad, sino también la utilidad que tendríamos cuando visitásemos locales de comida en grupos de persona. Seríamos capaces de **ponernos de acuerdo** entre tantas personas gracias al uso de esta aplicación.

Otro incentivo hacia el desarrollo de esta aplicación es intentar resolver **problemas alérgicos**. Aquellas personas alérgicas estarán en todo momento avisadas de aquellos platos a los que son alérgicos o no pueden tolerar.

La conectividad **Wifi Direct** debido a su uso para intercambiar información culinaria ha sido uno de los retos de implementación de este proyecto ya que cualquier otro tipo de conectividad no hubiese sido tan eficaz.

En último lugar, destacar la motivación que nos ofreció poder gestionar la privacidad de los datos. Con esto, estoy refiriéndome a que toda nuestra información personal almacenada en el dispositivo no se va a enviar a ningún servidor externo, sino que va a ser propia. Esto es muy importante debido a que, cada vez, más empresas como FaceBook o Google poseen mucha información personal, de tal modo, que estamos evitando darle más información a estas empresas.

2. OBJETIVOS

Generalmente, los objetivos, son implementar una aplicación móvil que resuelva algunos problemas que se pueden dar a la hora de elegir platos en un local de comida o a la hora de pedir algún plato al que seamos alérgicos. Para más información se detallarán una serie de sub-objetivos:

- **Conseguir que el coste de desarrollo sea lo más bajo posible**. En la mayoría de proyectos hay que añadir algún coste extra para poder llevar a cabo su desarrollo. En este, nos basta con disponer de algún dispositivo móvil y un portátil.

- **Hacer posible una funcionalidad sin conexión a internet (modo offline)**. Habitualmente la mayoría de las personas poseemos conexión a internet, pero también habitualmente queremos evitar gastar datos de red continuamente, y, si disponemos de una aplicación móvil que evite gastar datos será mejor para nuestros beneficios. Este objetivo tiene gran importancia cuando viajemos a un país extranjero o nos quedemos sin datos de red en el dispositivo.

- **Evitar problemas alérgicos**. El principal objetivo referente a temas sanitarios es evitar que una persona pueda ignorantemente intoxicarse por la ingestión de algún alimento al que es alérgico.

- **Conseguir intercambiar nuestras preferencias culinarias con las de nuestros compañeros y viceversa**. Generalmente, nuestras preferencias culinarias las sabemos, pero las del resto de acompañantes no, por lo que este objetivo se considera un importante factor social para la aplicación.

- **Evitar situaciones de incertidumbre**. Sería muy beneficioso sobre todo a nivel de tiempo y discusión el que nos pongamos de acuerdo entre varias personas, o incluso entre nosotros mismos, sobre que platos o tapas vamos a elegir.

- **En general, mejorar la vida de las personas**. La mayoría de aplicaciones móviles intenta mejorar la vida de las personas, por eso de su desarrollo. Esta aplicación cumple la función de mejorar la vida de las personas tanto en temas sanitarios como en temas culinarios y sociales.

- **Investigar y entender la conectividad Wifi Direct.** Esta conectividad es muy novedosa y eficaz en cuanto a intercambio de datos nos referimos. Por ello, es muy importante, en nuestra actualidad tecnológica, investigar y poder conocer mejor esta conectividad para transferencia de datos.

3. POSIBLES COMPETIDORES

En el siguiente apartado, haremos un estudio de las posibles competencias a este proyecto. Después de indagar y hacer constantes búsquedas por la **web**, se han encontrado algunos competidores referidos a temas culinarios que tienen otros objetivos y que mostraré a continuación.

3.1 YELP

Yelp, es una red social donde los usuarios recomiendan sus sitios favoritos escribiendo **reseñas** sobre restaurantes, tiendas, locales de copas… Gracias a la **geolocalización** permite descubrir lugares cercanos a nosotros: bares, restaurantes, tiendas de comida, pastelerías… por orden de cercanía, por precio o tipo de comida.

Por todo esto, con Yelp, nunca podrás llevarte un chasco cuando visites cualquier tipo de negocio. Restaurantes, bares de tapas y picoteo, salones de belleza, clínicas veterinarias, centros de wellness... desde el primer momento, puedes saber a ciencia cierta qué es lo que te vas a encontrar.

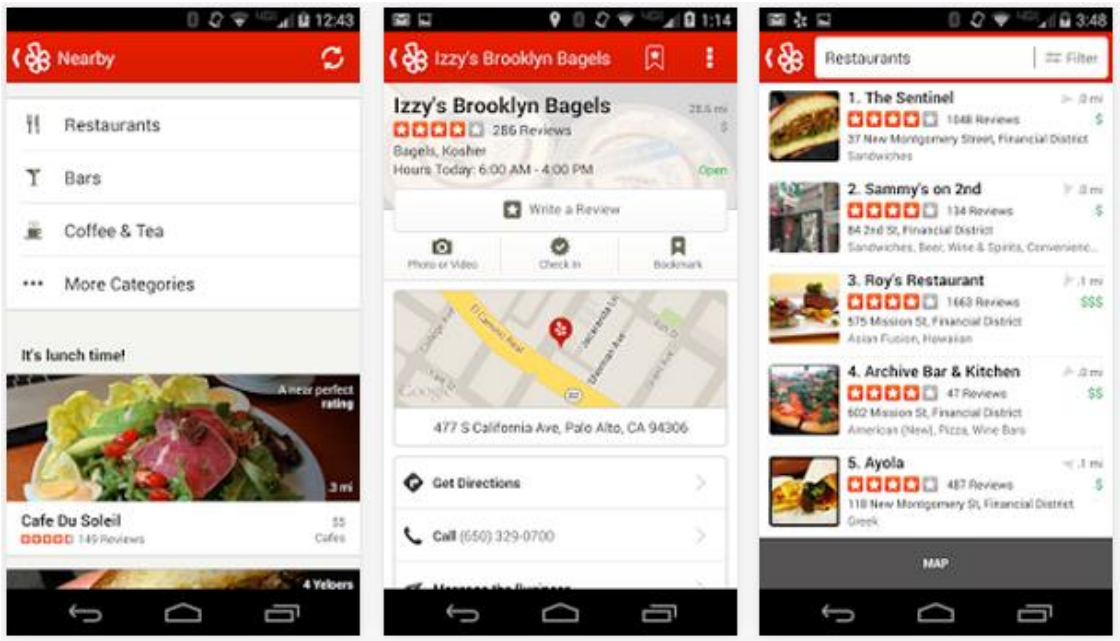

Ilustración 1- Yelp [32]

En el apartado “Entorno” puedes echar un vistazo a los negocios **cerca de ti**, y en “Buscar” tienes la opción de realizar búsquedas por proximidad, o por palabras clave los negocios de las categorías que te interesen.

Además, tienes la posibilidad de hacer **check-in** (registro) en los establecimientos que visites, así como añadirlos a favoritos, y cuentas con funciones rápidas para publicar un comentario o compartir fotos o vídeos.

Todas estas funcionalidades que posee la aplicación y que he detallado anteriormente son muy positivas, aunque respecto a mi proyecto no veo ninguna funcionalidad muy similar. Adicionalmente, a pesar de ser una aplicación social debido a poder compartir opiniones entre usuarios, en ninguno momento **comparte** otro tipo de información o **genera datos** a partir de los datos culinarios de nuestros compañeros.

3.2 EL TENEDOR

Ilustracion 2. El Tenedor [33]

La archiconocida web de reservas de restaurantes lanzó su aplicación inicialmente con una base de datos **de 6.000** restaurantes de toda España.

Actualmente, El Tenedor es la app para reservar restaurantes **LÍDER** en España. Con presencia en 12 países, te permite elegir entre más de 40.000 restaurantes,

disfrutar de promociones exclusivas de hasta el 50% y siempre con el apoyo de cerca de 7 millones de opiniones.

En esta app podemos descubrir restaurantes **cercanos** y ver información bastante detallada sobre ellos: fotos, opiniones de usuarios, menús, ofertas… Permite **buscar** restaurantes por criterios como el tipo de comida o el precio, y por supuesto desde la propia app se puede reservar mesa en un restaurante en pocos segundos, disfrutando a veces de descuentos en la carta. También podemos dejar que nos envíe alertas avisándonos de las **ofertas** de los restaurantes que están a nuestro alrededor y ver su disponibilidad a tiempo real con las mejores promociones.

Se trata de una aplicación muy potente, con numerosas funcionalidades y muy llamativas, como recibir mensajes de alertas con las ofertas de los restaurantes que están a nuestro alrededor o tener buenos filtros de búsqueda. Sin embargo, a diferencia del presente proyecto, no posee ninguna funcionalidad similar a pensar por el usuario y aconsejarle sobre que plato elegir. Tampoco ofrece la posibilidad de poder compartir preferencias culinarias entre varias personas.

3.3 GUÍA MICHELIN ESPAÑA

Ilustración 3- Guía Michelín [34]

La app oficial de la **Guía Michelín** de Restaurantes ofrece un buscador multicriterio para que encuentres el restaurante que más te guste entre los condecorados con alguna Estrella Michelín en el presente año.

Busca por ciudad, por tipo de comida, por clasificación de estrellas, por precio, etc. También incorpora **opiniones** de los propios usuarios y **fotografías** de los restaurantes y los platos.

En cada uno de los restaurantes, la opinión de los inspectores de la guía, los comentarios de los internautas, las fotos, los precios y la información práctica te ayudarán en tu **elección**. También, puedes consultar la ruta a seguir para llegar hasta allí, dejándote guiar desde tu posición.

Esta aplicación a diferencia de las anteriores y del proyecto que se está llevando a cabo, acota el registro de restaurantes ya que solo sirve para aquellos restaurantes que posean una estrella Michelín. Aunque resulta novedoso, ver todos los restaurantes con estrella Michelin, no tiene ninguna funcionalidad similar a generar una **carta** con **puntuaciones** en los platos en base a nuestros propios **gustos**. De tal modo que el usuario tendría más **claro** que pedirse cuando visite este tipo de restaurantes con estrella Michelin. Adicionalmente, la aplicación tampoco ofrece la virtud social que presenta el proyecto, debido a que no abordará que varias personas coman juntas. En cambio, este proyecto permite a varias personas juntar sus preferencias culinarias para elaborar una carta según estas preferencias.

3.4 ATRÁPALO RESTAURANTES

Ilustración 4- Atrápalo restaurantes [35]

Otra aplicación popular para buscar restaurante es Atrápalo Restaurantes, que permite **descubrir** y **reservar** restaurantes. Tiene una importante componente **social**, ya que es posible acceder a las **listas de restaurantes favoritos** de tus amigos.

Atrápalo Restaurantes es muy parecida a las aplicaciones detalladas anteriormente pero con alguna funcionalidad extra, como el acceso a la lista de restaurantes favoritos de tus amigos. Esto es, un punto a favor de la funcionalidad **social** de la aplicación. Sin embargo, en ningún momento se puede comparar esta funcionalidad con la posibilidad de **compartir** nuestros **gustos** culinarios y ver una carta ordenada en base a tus preferencias y las de tus amigos.

3.5 BURGUER KING

Burguer King, también conocida como BK, es una cadena de establecimientos de comida rápida estadounidenses. Como es habitual en nuestra sociedad, una cadena de establecimientos tan importante como esta, deberá poseer una aplicación móvil.

Ilustración 5- Burguer King [36]

En esta aplicación puedes encontrar **ofertas** exclusivas sólo disponibles en la App, cupones descuento para disfrutar de tus menús y productos favoritos al mejor precio y todas las promociones de Burguer King para que no se te escape nada. Además, podrás encontrar el restaurante más **cercano** y te ayudará a llegar para que no te pierdas la experiencia de visitarles, incluso se podrá disfrutar del **servicio a domicilio**.

En general, posee funcionalidades muy llamativas para el cliente como son, el poder disfrutar del servicio a domicilio o tener promociones que solo existan en la aplicación móvil. Esto da lugar a que las personas se descarguen la aplicación e interactúen con ella, ya que podrán ver un ahorro monetario. A pesar de ello, y de estas funcionalidades, en ningún momento la aplicación nos va a decir si somos alérgicos a alguna hamburguesa, por lo que vamos a estar continuamente pendiente de los ingredientes de todas las hamburguesas o productos que podamos pedir.

3.6 CONCLUSIONES

Como **conclusión** a todo este proceso de estudio de posibles empresas competidoras, destacar que aunque hemos observado que las posibles empresas tienen funciones **similares** sobre temas culinarios, ninguna de las funciones se parece a las funcionalidades que presenta el proyecto. Claro ejemplo de estas funciones son estudiar los **gustos culinarios** de las personas o evitarle **intoxicaciones** alimentarias.

	Reservas	Promociones	Geolocalización	Opiniones	Control alergias	Carta inteligente
YELP	NO	NO	SI	SI	NO	NO
EL TENEDOR	SI	SI	SI	SI	NO	NO
GUÍA MICHELÍN ESPAÑA	NO	NO	NO	SI	NO	NO
ATRÁPALO RESTAURANTES	SI	SI	SI	SI	NO	NO
RÉSTALO	SI	NO	SI	SI	NO	NO
BURGUER KING	SI	SI	SI	NO	NO	NO

Tabla 1- Conclusiones aplicaciones competidoras.

4. RECURSOS DISPONIBLES

En este cuarto apartado del documento se van a describir los **recursos** hardware que se han ido usando a lo largo del desarrollo del proyecto, detallando las **características** más importantes.

4.1 LAPTOP

Mi **fiel** aliado, tanto en el proyecto como en el largo desarrollo del Grado Universitario ha sido un ordenador portátil **Toshiba Satellite P850-135**. En él, se ha llevado a cabo todo el desarrollo del proyecto, tanto la implementación, como la presente memoria. Entre sus características más técnicas podemos destacar:

- **Procesador**: Intel(R) Core(TM) i7-3610QM CPU @ 2.30GHz.
- **Memoria RAM**: 8.00 Gb.
- **Gráficos**: NVIDIA GeForce GT 630M.
- **Sistemas operativos**: Windows 10 PRO y Ubuntu 14.04.

4.2 SMARTPHONES

Otra de las piezas claves para la implementación de este proyecto han sido los conocidos **smartphones**, que han servido para realizar pruebas software de la aplicación, con el fin de verificar el correcto funcionamiento de la misma.

Los dispositivos móviles utilizados para este proyecto se describirán a continuación:

4.2.1 SAMSUNG GALAXY S5

Principal dispositivo móvil utilizado a la hora de **desarrollar** la aplicación. Ha participado durante todo el proceso de implementación de la aplicación. Sus características físicas son las siguientes:

- **CPU**: Qualcomm MSM8974AC Snapdragon 801.
- **Memoria RAM**: 2GB.
- **Almacenamiento**: 32GB.
- **Tamaño**: 5.1 pulgadas.
- **Resolución de pantalla**: 1920 x 1080.
- **Versión de Android**: 6.0.1.

4.2.2 SAMSUNG GALAXY A5

Este dispositivo ha sido usado principalmente en las primeras pruebas que se estuvieron haciendo para **compartir preferencias** con la tecnología **Wifi Direct**. Además también ha servido para generar una **demo** final de la aplicación. Sus características físicas son las siguientes:

- **CPU**: Qualcomm MSM8916 Snapdragon 410 quad-core 1.2GHz.
- **Memoria RAM**: 2GB.
- **Almacenamiento**: 16GB.
- **Tamaño**: 5.0 pulgadas.

- **Resolución de pantalla**: 1280x720.
- **Versión de Android**: 4.4.4.

4.2.3 SAMSUNG GALAXY J3

Finalmente veremos este dispositivo, que ha sido utilizado únicamente para comprobar que la aplicación funciona correctamente con **más de 2 dispositivos** y para realizar una **demo de la aplicación**. Sus características físicas son las siguientes:

- **CPU**: Spreadtrum SC9830.
- **Memoria RAM**: 1.5GB.
- **Almacenamiento**: 16GB.
- **Tamaño**: 5.0 pulgadas.
- **Resolución de pantalla:** 1280x720.
- **Versión de Android**: 5.1.1 Lollipop.

5. DECISIONES TOMADAS

A continuación se va a realizar un **resumen** por apartados de las **decisiones** más importantes que se han ido tomando a lo largo del desarrollo del proyecto. Se ha elegido explicar las decisiones en estos puntos y no en otros, debido a que son los más relevantes y más influyentes a la hora de desarrollar el proyecto, ya que cualquier decisión tomada en estos, supondría un gran cambio en el mismo. En concreto veremos por un lado, las decisiones que hemos tomado respecto a la **tecnología** usada para el compartimento de preferencias. Por otro lado, se hará una explicación de las decisiones tomadas en el **almacenamiento de la información** y en la **gestión de preferencias**. Por último, comentaremos las decisiones tomadas en el **compartimento** de preferencias entre comensales. Destacar que habitualmente, las decisiones han sido tomadas mediante la ayuda del profesor tutor, aunque algunas de ellas han sido por criterio propio.

5.1 TECNOLOGÍA PARA INTERCAMBIO DE DATOS

Al principio del proyecto se estuvo haciendo un **estudio** sobre cómo hacer el *compartimento* de preferencias entre los comensales. Uno de las principales causas para decantarnos por alguna tecnología fue la existencia del modo ***offline***, es decir, no es necesario tener una conexión a internet para poder intercambiar información entre dispositivos. Después de estar varios días investigando sobre cómo ***implementarlo***, hemos acotado nuestra elección a usar una de las siguientes alternativas:

5.1.1 BLUETOOTH LOW ENERGY

Ilustración 6- Bluetooth low energy [37]

Es el nuevo sistema **bluetooth** 4.0 que viene incorporado en los **smartphone** a partir del 2013. Emite una señal de 2.4Ghz con un alcance de hasta **100** metros, con la gran ventaja de que **consume** muy poca energía y permite estar emitiendo la señal durante meses e incluso años sin tener que recargar la batería.

Como cualquier sistema bluetooth, es un sistema **inalámbrico** que sirve para conectar varios aparatos electrónicos e intercambiar información entre ellos vía bluetooth, pero en esta ocasión de bajo consumo. Permite emitir durante **meses** sin cambiar la **batería** del emisor o del receptor. La mayoría usa pilas normales, ni siquiera baterías.

Este tipo de sistema cambiará la forma de interactuar con nuestro teléfono. Las tiendas y grandes superficies ya lo están utilizando para la llamada *publicidad por proximidad*. Es muy sencillo. Mediante una baliza informativa, se va a emitir una señal Bluetooth Low Energy, de tal modo, que cuando un smartphone pase cerca de la baliza, esta podrá enviar mensajes al smartphone.

Respecto a la implementación de BLE en Android Studio, podemos encontrar una **API** [10] para el **envío** y **recepción** de paquetes mediante este protocolo.

Finalmente está tecnología fue **fuertemente** rechazada debido a su principal problema (los **paquetes de datos a enviar** entre dispositivos son relativamente **pequeños**, por lo que no nos valdría para transmitir toda la información necesaria sobre los datos culinarios de las personas). Estamos ante una **decisión** muy **importante** para el desarrollo del proyecto ya que compartir datos entre comensales es una de sus principales **funcionalidades**. De tal modo, que no podemos permitirnos enviar solo parte de la información que almacenamos. En tal caso, estaríamos poniendo en **peligro** a aquellas personas que sean alérgicas a un producto, ya que podría ocurrir que la información tipo **alérgica** no se envíe y se le permitiría escoger un plato al que es alérgico. Imaginemos que la aplicación posee gran cantidad de información culinaria del usuario, entre ellas, su alergia al marisco, ¿Qué sucedería si está alergia no se envía? El resultado sería muy negativo ya que el usuario podría pedir inconscientemente aquellos platos que posean marisco entre sus ingredientes.

5.1.2 LTE DIRECT

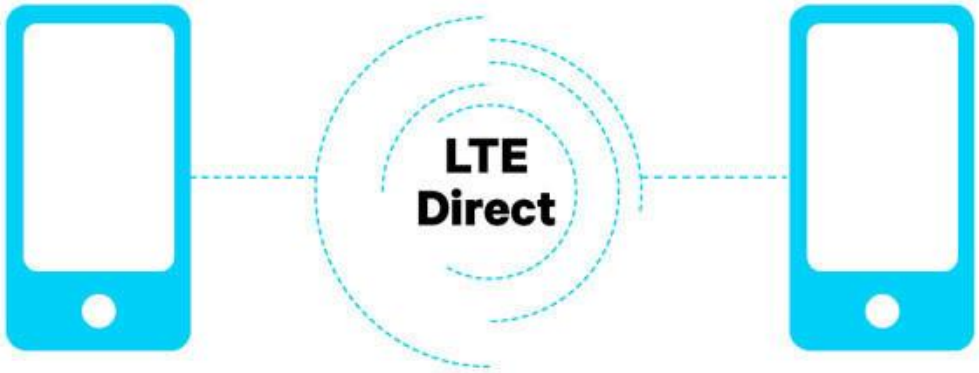

Ilustración 7- LTE Direct [39]

LTE Direct es un desarrollo impulsado por Qualcomm, donde compañías como **Facebook** y **Yahoo** están ayudando en las primeras pruebas, las cuales consisten en conectar y trasmitir información entre dos dispositivos a un máximo de **500** metros de distancia.

LTE Direct se le puede considerar un **gran avance** en cuanto a los servicios de proximidad, ya que nos proporcionará mayores funciones con sólo poseer un smartphone, donde una de sus ventajas es que sólo se trata de una actualización del actual protocolo **LTE**, por lo que no necesitaremos mayores implementaciones, ya que vendrá habilitado en los futuros procesadores.

Finalmente **rechazamos** está propuesta debido a que es una tecnología que está comenzando a desarrollarse y existe muy poca documentación sobre ella. En cualquier caso, podemos visitar su plataforma **oficial [29].**

5.1.3 WIFI AWARE

Ilustración 8- Wifi Aware [40]

Wi-Fi Aware se trata de una tecnología similar a las anteriores, que permitirá a dispositivos Wi-Fi cercanos comunicarse entre sí, sin intervención humana, y sin necesidad de conectarse a un punto de acceso ni, requiere conexión a Internet.

Si activas el modo **Wi-Fi Aware**, el dispositivo emite una especie de "latido" de forma continua, y al mismo tiempo capta todos los latidos que hay a su alrededor. Entonces se establece una conexión entre ambos dispositivos y se intercambian ciertos datos predefinidos. Tienes un control total sobre con qué te conectas.

El término "latido" tiene sentido porque en realidad esta señal WiFi se apaga y se enciende varias veces por segundo para ahorrar energía. Wi-Fi Alliance asegura que es de baja intensidad y consume muy poco. Además como no se conecta a Internet, mejora la seguridad.

Mediante este sistema, dispositivos Wi-Fi Aware podrán enviar datos a otros cercanos, por ejemplo ofertas y cupones cuando pasas delante de una tienda, o una orden de que se enciendan las luces cuando entres en casa.

La ventaja sobre Bluetooth LE, usado por los beacons y otros dispositivos del Internet de las Cosas, es que al parecer Wi-Fi Aware es más rápido y tiene mayor alcance.

Finalmente esta tecnología fue rechazada debido a que requiere **Android 8.0** para su funcionamiento y a que posee muy poca documentación para su desarrollo, debido a su reciente lanzamiento.

5.1.4 WIFI DIRECT

Ilustración 9- Wifi Direct Logo [38]

Wifi Direct se trata de una de las **alternativas** más útiles para la transferencia de archivos. Hablamos de tasas de transferencia de archivos más **elevadas** (hasta **250** Mbps frente a los 25Mbps de Bluetooth 4.0). Aunque Bluetooth 4.0 haga uso de protocolos **802.11**, no alcanza las velocidades de WiFi Direct.

WiFi Direct ofrece mayor **tasa de transferencia** y un **cifrado** más seguro que Bluetooth.

También se tiene un mayor grado de **seguridad**, ya que con bluetooth se comparte una serie de información privada. Y ésta, en manos de alguien que intercepte dichos paquetes puede ser bastante peligrosa. En palabras técnicas, Bluetooth 4.0 cuenta con cifrado AES de 128 bit. Mientras que WiFi Direct cuenta con el cifrado WPA2 que es AES de 256 bits.

Como conclusión, a diferencia de Bluetooth, Wifi Direct permite transferir los datos por wifi de manera más **rápido** que con bluetooth. Además Wifi Direct es más **seguro** debido entre otras cosas a su cifrado, y la **distancia** mínima de funcionamiento, puede ser mayor que con bluetooth.

Por lo tanto, esta tecnología que será mejor detallada en el siguiente apartado, fue la **ganadora**, debido a sus diferencias con Bluetooth y a la facilidad que tenemos para implementar a través de la web oficial de desarrollo de ***Android Studio***.

La API de Android Studio para el desarrollo de la tecnología Wifi Direct se llama ***WifiP2P*** [9].

5.2 ALMACENAMIENTO DE DATOS

Ilustración 10- Almacenamiento de datos [41]

Para la realización de este proyecto, en varias ocasiones, surge la situación de guardar una serie de datos en un **almacén de datos**. Para la implementación, hubo que hacer un **estudio** de las diferentes formas de almacenamiento existentes para la plataforma Android.

La primera intención fue usar un tipo de base de datos que almacenase la información en la **nube** para poder acceder desde cualquier **dispositivo**. Un ejemplo para este tipo de base de datos es hacer uso de ***Firebase***. Está idea fue rechazada debido a los siguientes motivos:

- Suponía un **incremento** importante en el desarrollo del proyecto.
- Implica una **conexión a internet**.
- Consideramos un mal menor el no poder acceder desde un nuevo dispositivo a nuestras preferencias almacenadas en el dispositivo anterior. Este mal podría solventarse transfiriendo las preferencias culinarias del dispositivo antiguo al nuevo.

En este punto, decidimos usar una base de datos **local** en el que la información se pudiese almacenar en cada dispositivo. Sin embargo, surgió la **duda** de que hacer para guardar el **listado** de restaurantes con sus platos.

Respecto al listado de restaurantes que incluye nombres de platos e ingredientes decidimos guardarlo en el propio proyecto, es decir, en un archivo "txt" incluido en el proyecto. Esta decisión fue tomada así, debido a que la información iba a ser siempre la misma y no iba a cambiar de un usuario a otro.

Caso contrario fue la decisión sobre donde guardar las preferencias de cada usuario, tanto **datos personales** (usuario, contraseña, email,…) como **datos culinarios** o **datos alérgicos**. En este aspecto decidimos usar **SQLite,** debido entre otros motivos a su sencillez y sobre todo a su innecesaria conexión a internet para su correcto funcionamiento.

SQLite es un motor de bases de datos muy popular en la actualidad por ofrecer características tan interesantes como su pequeño tamaño, no **necesitar** servidor, precisar poca configuración, ser transaccional y por supuesto ser de **código libre**.

Android incorpora de serie todas las herramientas necesarias para la creación y gestión de bases de datos **SQLite**, y entre ellas una completa **API** para llevar a cabo de manera sencilla todas las tareas necesarias.

En el siguiente apartado sobre las Tecnologías Utilizadas podremos ver en detalle esta base de datos.

5.3 GESTIÓN DE PREFERENCIAS

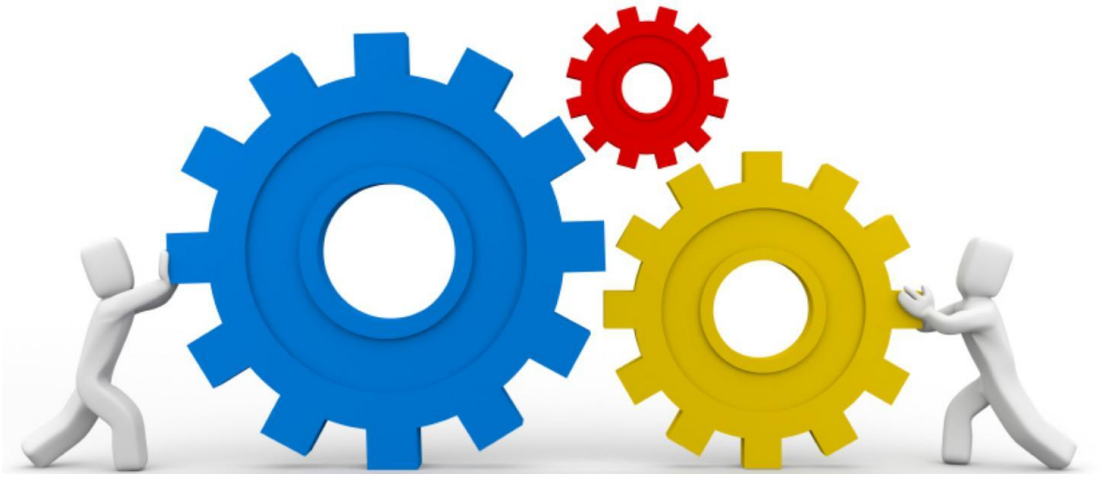

Ilustración 11- Gestión de preferencias [42]

Desde el primer momento, nos hicimos las siguientes preguntas: ¿**De dónde sacamos los datos culinarios o alérgicos de cada usuario**? ¿Cómo implementamos el algoritmo para **ordenar** los platos según preferencias? ¿Qué método logístico usamos para **incrementar** la puntuación de cada plato? En definitiva, surgieron un conjunto de dudas con incógnitas que en cualquier caso, había que resolver.

Inicialmente, la intención era llevar a cabo una primera obtención de información sobre gustos **culinarios** y **alergias** a nivel **genérico** sobre el usuario. De tal modo, que estos datos se irían **refinando** a medida que el usuario fuese escogiendo platos o tapas en los diversos **restaurantes**. Este primer paso se resolvió implementando en el **registro** un pequeño **formulario** muy genérico sobre tipos de comida y alergias. Existen miles de **ingredientes**, diferentes **alergias** y obviamente no podemos poner en un formulario **todos** los ingredientes que existen y **todas** las alergias que puedan existir. Por lo que la resolución fue hacer formularios genéricos y cortos para tener una idea inicial acerca de las preferencias de cada usuario.

En segundo lugar, resolvimos la duda acerca de cómo **ordenar** los platos según preferencias. Esta duda fue resuelta mediante una **lista** de **objetos** que contienen atributos como el **nombre** de la tapa o la **puntuación**. Este nuevo objeto implementa un **comparable** por puntuación, por lo que cuando recorramos la lista, va a salir **ordenada** en base a la puntuación de cada tapa.

A medida que el usuario utiliza la aplicación, habrá que mejorar la información sobre ese usuario y observar cómo van evolucionando sus gustos. Este proceso se decidió hacer mediante un **incremento,** en las puntuaciones que contienen los ingredientes de los platos que el usuario escoja. El incremento, se resolvió de la

siguiente manera: **dividimos** 1 entre el **número de ingredientes** que contiene esa tapa o plato. El resultado de esta división es el incremento que aplicamos a cada **ingrediente**.

Para entender mejor esta decisión veamos el siguiente ejemplo: supongamos que elegimos **solomillo al roquefort** que posee **4** ingredientes (*carne de cerdo* → **5** puntos, *queso* → **2** puntos, *pimiento* → Sin evaluar y *patatas* →Sin evaluar). El incremento que va a obtener el usuario para cada ingrediente va a ser de ¼. De tal modo que se obtendrán las siguientes puntuaciones: carne de cerdo → **5.25**, queso → **2.25**, pimiento → **0.25** y patatas → **0.25**.

¿Qué método aplicamos cuando **tengamos** preferencias de varias personas? .La resolución de este caso fue *simple* y *eficaz*. Para esta hipótesis, la lista final de tapas generada deberá contener las mismas puntuaciones para todos los dispositivos que hayan compartido sus preferencias. Por lo tanto, se va a generar una **nueva lista** que va a contener la información de todos los comensales que han compartido sus preferencias. Con esta nueva **lista** vamos a generar un listado **final** de tapas que contendrá las preferencias de cada comensal. Con el siguiente ejemplo entenderemos mejor su funcionamiento:

Imaginemos que existen 3 usuarios con estas preferencias: **usuario 1** (*carne de cerdo* → **5** puntos, *queso* → **2** puntos, *pimiento* → 3 puntos y *patatas* →4 puntos), **usuario 2** (*carne de pollo* → **4** puntos y *queso* → **1** puntos) y **usuario 3** (*carne de pollo* → **5** puntos, *queso* → **5** puntos y *pimiento* → 2), el resultado final de puntuaciones será el siguiente:

- Carne de cerdo → 5 puntos → Usuario 1.
- Queso → 8 puntos → Usuarios 1,2 y 3.
- Pimiento → 5 puntos → Usuarios 1 y 3.
- Patatas → 4 puntos → Usuario 1.
- Carne de pollo → 9 puntos → Usuarios 2 y 3.

Como conclusión, destacar que la gestión de preferencias ha sido uno de los apartados que más **dudas** ha generado, ya que hay que controlar **multitud** de aspectos e intentar hacerlo todo de la mejor manera posible para averiguar las preferencias culinarias de cada usuario.

5.4 COMPARTO DE PREFERENCIAS

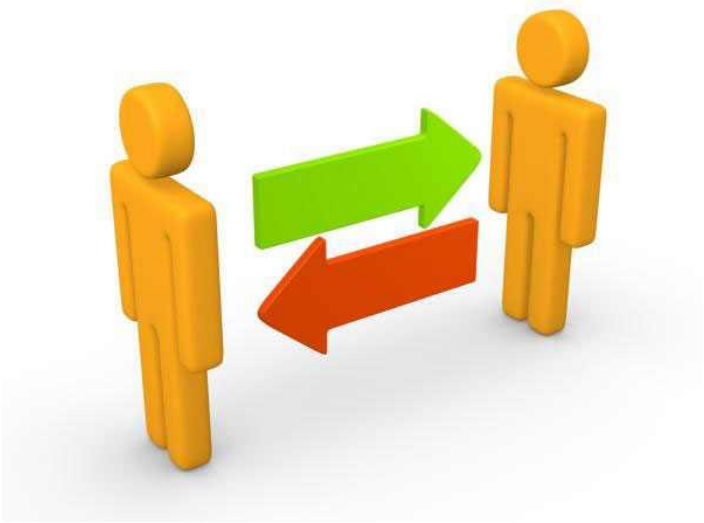

Ilustración 12- Comparto de preferencias [43]

Cuando un grupo de personas visitan algún tipo de local de comida, en numerosas ocasiones se preguntan ¿Qué podemos pedir para que **nos guste a todos**? . Gracias a este proyecto intentamos **mejorar** este tipo de incertidumbre. En este contexto, surge la duda de cómo implementar el comparto de preferencias entre comensales. Ya sabemos que la tecnología a usar va a ser Wifi Direct, sin embargo, no sabemos cómo hacerlo. Una de los aspectos que teníamos claro para esta implementación, era intentar, que el usuario **interactúe** lo más mínimo con la aplicación.

En un principio la idea era sencilla, hacer una conexión **many** to **many** mediante Wifi Direct y enviarse las preferencias entre dispositivos. Esta idea fue **rechazada** debido a que Wifi Direct no contempla este tipo de conexiones. La única opción poco parecida a esta era hacer que un dispositivo crease un **grupo** y crear una conexión **one** to **many**. El dispositivo creador de grupo se va a mantener a la **espera** de peticiones por parte de los clientes, para posteriormente enviar los datos. A la inversa (enviar datos del dispositivo que actúa como servidor a los clientes) implicaría una **interacción** muy **dificultosa** e incoherente para el usuario y se acabaría errando el **comparto** de preferencias, ya que tendría que conectarse con cada dispositivo que haya recibido peticiones de forma manual. Un punto en contra de las conexiones one to many es que no todos los móviles con Wifi Direct van a soportar este tipo de conexiones, por lo que serán solo aptos para conexiones one to one. Por todo lo detallado, esta decisión fue rechazada.

Viendo rechazada estas 2 opciones acudimos al recurso de hacer **conexiones individuales bidireccionales**. Después de hacer varias pruebas y de investigar mucho sobre cómo implementar esta opción llegamos a la conclusión de que esta opción era inviable. Esto es, debido a que para cada conexión entre dispositivos uno actúa como

cliente y otro como **servidor** (punto de acceso). Cualquiera de los dos podría enviar información al otro especificándolo con un parámetro al crear la conexión. El problema aparece cuando al crear la conexión asignamos a cada dispositivo un rol, no podemos intercambiar los roles de cada dispositivo para la misma conexión.

En este aspecto se decidió hacer **2 conexiones** simulando 1 conexión individual bidireccional. En este caso, controlamos el momento en el que se reciben los datos al dispositivo para cerrar la conexión y crear una nueva intercambiando los **roles** y enviando los datos al otro dispositivo.

En este momento nos hicimos la siguiente pregunta ¿**Cómo sabemos que dispositivo nos ha enviado los datos para después crear la conexión con él**? .Al crear la conexión pasamos el número **MAC** con el dispositivo al que queremos conectar, por lo que al enviar las **preferencias** culinarias, también enviamos la dirección **MAC** del dispositivo emisor, con el fin de poder crear la conexión con ese dispositivo.

Una vez controlado el tema de como intercambiar las preferencias, nos surgió la duda de como acoplar estas **preferencias** a las **preferencias** propias del usuario. Esta incógnita fue resuelta **dividiendo** las puntuaciones de cada ingrediente entre el número de personas que intercambian la información. En el caso de que existan ingredientes ***comunes*** para varias personas, simplemente ***sumamos*** las puntuaciones y seguidamente las **dividimos** entre el número de personas que intercambian la información.

Después de hacer varias pruebas y ver como las puntuaciones de los platos iban cambiando a medida que íbamos pidiendo tapas, nos preguntamos cómo **normalizar** las puntuaciones si tenemos tapas con altas puntuaciones (superiores a 10). Esto sería un problema ya que no sabes la **cota superior máxima** que puede obtener una tapa. En este contexto, se decidió en primer lugar, comprobar si tenemos alguna puntuación **mayor que 10**. En caso negativo **dejamos** las puntuaciones tal cual, pero en caso afirmativo le damos el valor 10 a la tapa con mayor puntuación. Para las otras tapas, simplemente hacemos una *regla de 3* usando como valor máximo la puntuación mayor.

6. TECNOLOGÍAS UTILIZADAS

En el siguiente apartado se hace un estudio de las **tecnologías** más importantes que han sido utilizadas para desarrollar el proyecto.

6.1 ANDROID

Ilustración 13- Android [44]

En primer lugar comenzaremos con la explicación de la tecnología **Android** y por qué nos hemos decantado a usar Android entre otras. Además veremos en profundidad la **arquitectura** que posee esta plataforma.

6.1.1 ¿QUÉ ES ANDROID?

Android es un sistema operativo inicialmente pensado para teléfonos móviles, al igual que *iOS, Symbian y Blackberry OS*. Lo que lo hace diferente es que está basado en **Linux**, un núcleo de sistema operativo **libre**, **gratuito** y **multiplataforma**.

El sistema operativo proporciona todas las interfaces necesarias para desarrollar aplicaciones que accedan a las funciones del teléfono (como el GPS, las llamadas, la agenda, etc.) de una forma muy sencilla en un lenguaje de programación muy conocido como es **Java**.

6.1.2 ¿POR QUE ANDROID?

La diferencia de Android con respecto a todos los sistemas operativos, es que es un software de **código abierto**, no cerrado. El usuario no tiene que conocer esta característica pero es importante de cara al mercado.

Por tanto, no es de extrañar que en torno al **80** por ciento de los dispositivos móviles en Europa y el mundo entero funcionan con Android, tal y como ha señalado la comisaria europea de Competencia.

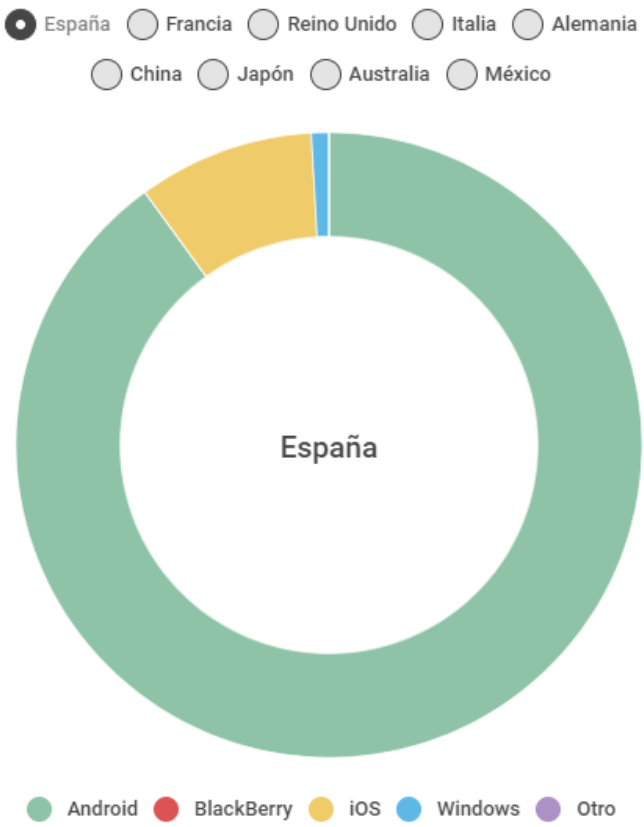

Ilustración 14- Gráfica uso Android [45]

Como se puede observar en la gráfica anterior, la tecnología Android **prevalece** en España ante otros sistemas operativos. En menor medida podemos ver como se encuentran los otros sistemas operativos como *iOS, Windows o BlackBerry*. Destacar que ni sumando las puntuaciones del resto de sistemas operativos obtendríamos una puntuación mayor que para los dispositivos Android, por lo que tenemos una **mayoría** absoluta.

Otros de los aspectos importantes para decantarnos por esta tecnología y no por iOS es que no está tan **limitado** como esta. Por ejemplo, un dispositivo iOS no podría actuar como **punto de acceso** en una conexión **Wifi Direct**. Sin embargo un dispositivo Android si podría hacerlo.

6.1.3 ARQUITECTURA DE LA PLATAFORMA

Android es una pila de software de código abierto basado en **Linux** creada para una variedad amplia de dispositivos y factores de forma. En el siguiente diagrama se muestran los componentes principales de la plataforma Android.

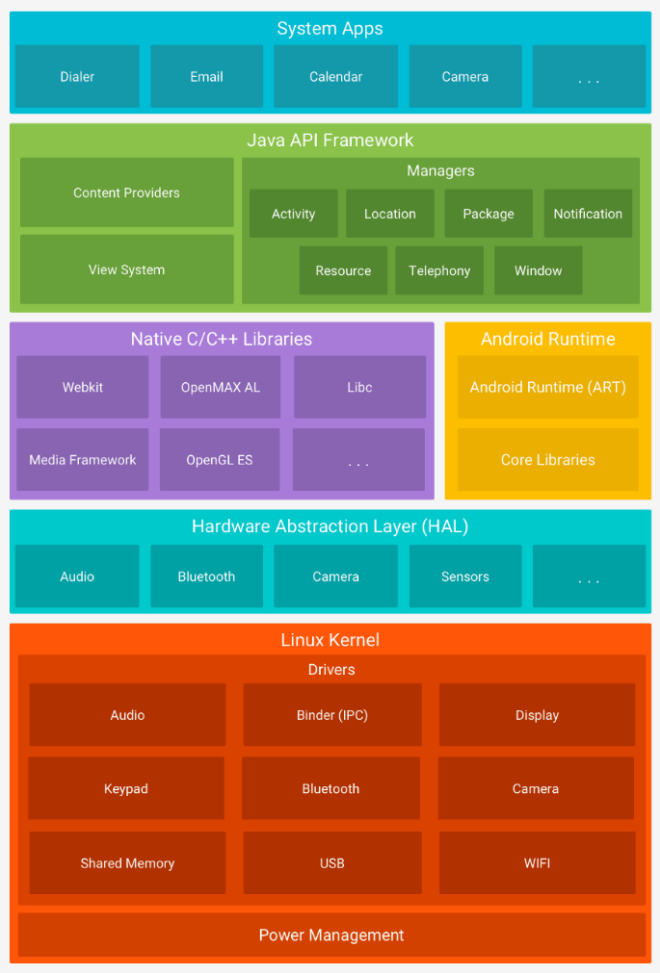

Ilustración 15- Arquitectura de la plataforma [46]

- **Kernel de Linux**

En el nivel más **bajo** nos encontramos con el Kernel de Linux, aquí, se encuentran la base de **servicios** del sistema como gestión de memoria y procesos, seguridad, modelo de controladores y pila de red. Permite **abstraer** al resto de capas, del acceso al hardware.

- **Capa de abstracción de hardware (HAL)**

La capa de abstracción de hardware (HAL) brinda interfaces estándares que **exponen** las capacidades de hardware del dispositivo, al framework de la API Java de nivel más alto.

- **Tiempo de ejecución de Android**

Al mismo nivel que las librerías se encuentra el **tiempo de ejecución de Android**, que proporciona un conjunto de **librerías** base, que proporcionan la mayor parte de las funciones disponibles en las bibliotecas base del lenguaje Java

- **Bibliotecas C/C++ nativas**

En el tercer nivel tenemos las **librerías de Android**, estas librerías escritas en C/C++, son utilizadas por varios componentes del sistema y se exponen a los desarrolladores para su uso a través de la capa **Application Framework**. Por ejemplo, puedes acceder a **OpenGL** a través de la *API Java OpenGL* del framework de Android, para agregar a tu app compatibilidad con los dibujos y la manipulación de gráficos 2D y 3D.

- **Framework de la Java API**

Esta capa proporciona a los desarrolladores un conjunto de librerías para que las aplicaciones **interactúen** con el sistema operativo. Además es la capa de software que proporciona el sistema operativo para, sobre éste, construir nuestras aplicaciones. Haciendo uso de esta capa, nuestras aplicaciones podrán por ejemplo acceder al **calendario** del dispositivo, hacer una **foto**, acceder a la **ubicación** del dispositivo.

- **Apps del sistema**

En Android se incluye un conjunto de apps centrales para correo electrónico, mensajería SMS, calendarios, navegación en Internet y contactos, entre otros elementos. Las apps incluidas en la plataforma no tienen un estado especial entre las apps que el usuario elije instalar; por ello, una app externa se puede convertir en el navegador web, el sistema de mensajería SMS o, incluso, el teclado predeterminado del usuario (existen algunas excepciones, como la app Settings del sistema).

6.1.4 ANDROID STUDIO

Ilustración 16- Android Studio [47]

6.1.4.1 ¿QUÉ ES ANDROID STUDIO?

Android Studio es el **entorno de desarrollo** integrado (IDE) oficial para el desarrollo de aplicaciones para **Android** y se basa en IntelliJ IDEA . Además del potente editor de códigos y las herramientas para desarrolladores de IntelliJ, Android Studio ofrece aún más **funciones** que aumentan tu productividad durante la compilación de apps para Android, como son las siguientes:

- Un sistema de **compilación** basado en Gradle flexible.
- Un **emulador** rápido con varias funciones.
- Un entorno **unificado** en el que puedes realizar desarrollos para todos los dispositivos Android.
- Instant Run para aplicar **cambios** mientras tu app se **ejecuta** sin la necesidad de compilar un nuevo APK.
- Integración de plantillas de código y **GitHub** para ayudarte a compilar funciones comunes de las apps e importar ejemplos de código.
- Gran cantidad de herramientas y **frameworks** de prueba.
- Herramientas **Lint** para detectar problemas de rendimiento, usabilidad, compatibilidad de versión, etc.

- Compatibilidad con **C++** y NDK.
- Soporte incorporado para **Google Cloud Platform**, lo que facilita la integración de Google Cloud Messaging y App Engine.

6.1.4.2 ESTRUCTURA DEL PROYECTO

Cada proyecto en Android Studio contiene uno o más **módulos** con archivos de código fuente y archivos de recursos. Entre los tipos de módulos se incluyen los siguientes:

- Módulos de **apps** para Android.
- Módulos de **bibliotecas**.
- Módulos de **Google App Engine**.

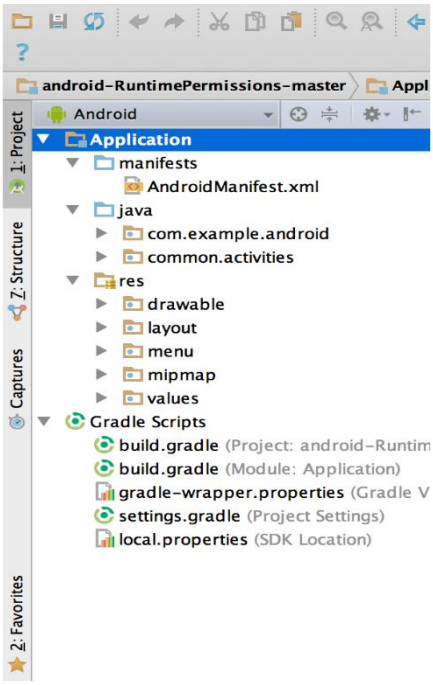

Ilustración 17- Estructura de proyecto [48]

De manera predeterminada, Android Studio muestra los archivos de tu proyecto en la vista de proyectos de Android, como se muestra en la ilustración 17.

Esta vista se organiza en **módulos** para proporcionar un rápido acceso a los archivos de origen, clave de tu proyecto.

Todos los archivos de compilación son visibles en el nivel superior de secuencias de comando de **Gradle,** y cada módulo de la aplicación contiene las siguientes carpetas:

- **Manifests**: contiene el archivo **AndroidManifest.xml**.
- **Java**: contiene los archivos de código fuente de **Java**, incluido el código de prueba **JUnit**.
- **Res**: Contiene todos los **recursos**, como diseños XML, cadenas de IU e imágenes de mapa de bits.

La estructura del proyecto para Android en el disco, difiere de esta representación plana. Para ver la estructura de archivos real del proyecto, selecciona Project en la lista desplegable **Project** (en la ilustración 17 se muestra como Android).

También puedes **personalizar** la vista de los archivos del proyecto para concentrarte en aspectos específicos del desarrollo de tu app. Por ejemplo, al seleccionar la vista Problems de tu proyecto, aparecerán **enlaces** a los archivos de origen, que contengan **errores** conocidos de codificación y sintaxis, como una etiqueta de cierre faltante para un elemento XML en un archivo de diseño.

6.2 WIFI DIRECT

Ilustración 18- Wifi Direct [38]

A continuación veremos con más detalle una de las tecnologías más **innovadora** que hemos usado en el proyecto. Esta tecnología posee muy poca documentación para su desarrollo en dispositivos Android. Dicha tecnología es conocida como **Wifi Direct**.

6.2.1 ¿QUÉ ES WIFI DIRECT?

Wifi Direct es una tecnología nacida de la WiFi Alliance. Una **mejora** de los protocolos WiFi 802.11 que permite convertir un cliente en un punto de acceso. De este modo, se pueden conectar a él otro tipo de dispositivos y establecer una conexión estable.

WiFi Direct convierte uno de los dispositivos en un ***Punto de Acceso.***

En primer lugar tenemos que identificar las diferentes partes que hay dentro de una comunicación mediante WiFi Direct. En una conexión tenemos dos **extremos**: **Punto de Acceso Inalámbrico y el cliente**. Pues WiFi Direct, mediante software (Soft AP), es capaz de convertir un dispositivo con **wifi** en un **punto de acceso**.

Wifi Direct permite el **intercambio** de datos entre dos dispositivos directamente mediante wifi, la diferencia con el uso habitual es que con Wifi Direct no es necesario conectar los **smartphones** a un **router** para que estos se puedan

comunicar, sino que ellos se ven **directamente** entre sí, funciona exactamente igual que con *bluetooth* donde para intercambiar fotos asociamos los dos teléfonos directamente.

6.2.2 CARACTERÍSTICAS DE WIFI DIRECT

Esta tecnología se encuentra presente en muchos **dispositivos**. Teléfonos, cámaras, impresoras, ordenadores, consolas de videojuegos y **televisores** se pueden conectar con este protocolo, que a su vez es muy seguro, ya que está protegido con el último cifrado para redes.

Un dispositivo con WiFi Direct activado emite una señal hacia otros artefactos en la sala, haciendo saber que está disponible para una conexión. Los usuarios pueden enviar una **invitación** o recibir una, para efectuar dicha conexión. Cuando ambos dispositivos están conectados (puede que pida una clave, disponible en el mismo menú) se puede empezar a compartir archivos.

Google incorpora WiFi Direct en todos los teléfonos con Android 4.0 ó superior. **Samsung** lo hizo antes, y lo permitió a partir del Galaxy S2, es decir, con Android 2.3, aunque sólo para conectarse con otros gadgets Samsung. Otros fabricantes como LG, HTC o Sony también lo han habilitado.

En los móviles Samsung la función es muy sencilla de usar. Una vez activada y pareados los dispositivos, los usuarios pueden enviar fotos o archivos desde el típico menú de compartir. Solo hace falta tocar la opción de enviar por Wifi Direct.

6.2.2 USO DE WIFI DIRECT

Respecto a la utilidad de Wifi Direct podemos destacar, por ejemplo, imprimir un archivo desde el portátil ó el smartphone con una impresora inalámbrica, enviar una imagen al móvil de una persona, o por ejemplo, reproducir en la pantalla del televisor un vídeo que tenemos guardado en el móvil. Todo esto sin cables y sin internet.

Esta tecnología es de gran utilidad cuando quieres recibir o enviar alguna información a alguien pero no quieres dar datos personales. Por ejemplo, estás en un concierto, se hace un pequeño grupo entre desconocidos y empiezan los selfies. Quieres esas fotografías para recordar el momento, pero no deseas ofrecer tu número de móvil para que las envíen por WhatsApp o tampoco dar la dirección de email ó Facebook. Mediante WiFi Direct vas a poder compartir esos archivos mucho más rápido que por Bluetooth, sin gastar datos de internet y sin exponer datos privados.

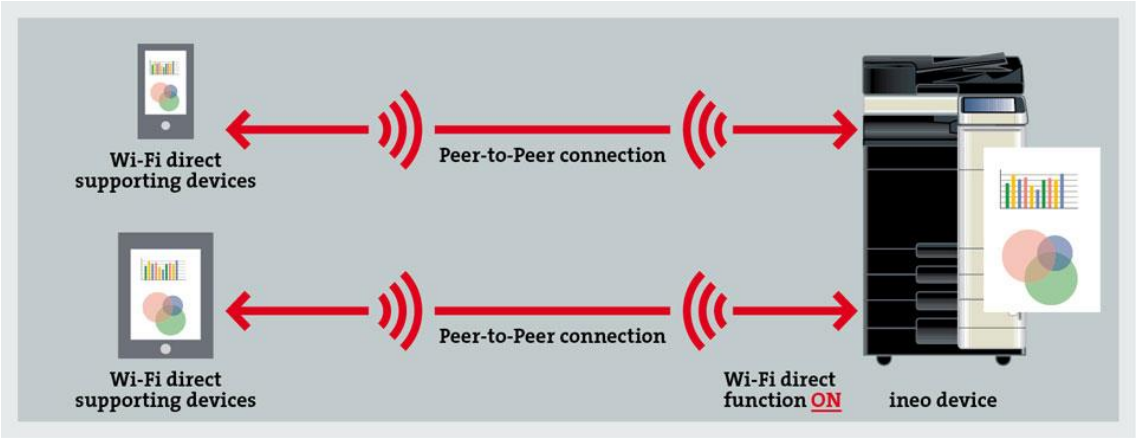

Ilustración 19- Wifi Direct esquema [49]

En la imagen anterior podemos observar una conexión Wifi Direct entre una impresora y un dispositivo. Como ya se ha detallado, esto es de gran utilidad para imprimir archivos a distancia, sin necesidad de tener conectado ningún cable de por medio.

Una de sus funcionalidades más populares, es la posibilidad de hacer que nuestro dispositivo móvil actúe como "router" dotando de internet a otros dispositivos. Para evitar que todos los dispositivos se conecten a nuestro router para consumir datos, Wifi Direct permite editar la contraseña de acceso para su conexión, de tal modo, que solo tendrán acceso aquellos dispositivos que posean la contraseña.

Otra de las funcionalidades importantes que posee Wifi Direct es darnos la posibilidad de ver el **contenido** de la **pantalla** de nuestro dispositivo en la Televisión. Sin embargo, habitualmente para realizar este tipo de funcionalidades se usa google **ChromeCast** [1], por lo que esta funcionalidad está un poco en desuso.

1- Dispositivo de reproducción multimedia en streaming que se conecta al puerto HDMI de la televisión.

6.2.3 ¿CÓMO CONECTAR 2 DISPOSITIVOS CON WIFI DIRECT?

Para conectar dos dispositivos, basta que uno de los dos sea **compatible** con Wifi Direct, esto es: el primer dispositivo **activa** el Wifi Direct de manera que el segundo dispositivo se **conecta** a su wifi. Este segundo dispositivo se conecta a la wifi del primero de la misma forma que lo haría en un **router** wifi, por lo que no necesita ser compatible con Wifi Direct, mientras que el primero sí tiene que tener Wifi Direct integrado en su sistema.

Ilustración 20- Como usar Wifi Direct [51]

Como ya se ha detallado los teléfonos Android a partir de la versión **4.0** tienen incluido Wifi Direct ya de **serie**. Los dispositivos de Apple con iOS tienen un Wifi Direct propio, llamado **Airdrop**, a partir de la versión 7.0, que sólo es compatible entre ellos. **Airdrop** utiliza por un lado **wifi,** para transmitir datos pero por otro utiliza **bluetooth,** para sincronizar los dos dispositivos, así que no cumple con el estándar de Wifi Direct en Android.

Para resumir, si tienes **iOS** y arrancas **Airdrop** los dispositivos Android no se podrán comunicar con él, sin embargo al revés si funciona: si tienes **Android** y activas **Wifi Direct** los dispositivos **iOS** se podrán conectar **perfectamente** con tu Android, debido a lo detallado anteriormente.

6.2.4 APORTE AL PROYECTO

Para este proyecto se ha hecho uso de la tecnología Wifi Direct para **compartir** datos culinarios entre comensales. Esta tecnología se usa a través de **WifiP2P** que es una **API** que tiene Android para hacer uso de esta tecnología.

WifiP2P permite a dispositivos Android 4.0 (nivel de API 14) o dispositivos posteriores con el hardware apropiado, poder **conectarse directamente** entre ellos a través de wifi sin un punto de acceso intermedio. Con estas **API**, puede **descubrir** y **conectarse** a otros **dispositivos** mientras el dispositivo sea compatible con WifiP2P.

La clase ***WifiP2pManager*** es la encargada de permitirte *descubrir*, *solicitar* y *conectarte* con tus compañeros.

WifiP2P posee *listener* (escuchadores) que nos permiten recibir **notificaciones** sobre si ha tenido *éxito* o *fracaso* las llamadas a la clase WifiP2pManager.

WifiP2P tiene **intent´s** (objetos de acción) que pueden notificarnos sobre **eventos detectados** como son una conexión fallida o un dispositivo recién descubierto.

En general, la clase WifiP2pManager proporciona métodos que nos permiten **interactuar** con el hardware de wifi en nuestro dispositivo, para hacer cosas como descubrir y conectarse con compañeros.

6.3 SQLITE

Ilustración 21- SQLite [52]

SQLite es una herramienta de software libre, que permite almacenar información en dispositivos empotrados de una forma sencilla, eficaz, potente, rápida y en equipos con pocas capacidades de hardware, como puede ser una PDA o un dispositivo móvil. SQLite implementa el estándar SQL92 y también agrega extensiones que facilitan su uso en cualquier ambiente de desarrollo. Esto permite que SQLite soporte desde las consultas más básicas hasta las más complejas del lenguaje SQL, y lo más importante es que se puede usar tanto en dispositivos móviles como en sistemas de escritorio, sin necesidad de realizar procesos complejos de importación y exportación de datos, ya que existe compatibilidad al 100% entre las diversas plataformas disponibles, haciendo que la portabilidad entre dispositivos y plataformas sea transparente.

Sus desarrolladores destacan, que su principal característica, es su completo soporte para tablas e índices en un único archivo por base de datos, soporte transaccional, rapidez (unas 2 veces más veloz que MySQL y PostgreSQL), escaso tamaño (unas 25 mil líneas de código C) y su completa portabilidad.

De la mezcla de **Android** y **SQLite** surge la clase **SQLiteOpenHelper**. SQLite es una pequeña biblioteca, que implementa un motor de bases de datos. Básicamente permite **gestionar bases de datos relacionales,** que ocupan una pequeña cantidad de memoria. El limitado uso de la memoria es de gran utilidad en los dispositivos móviles y, de hecho, es el motivo de la estrecha relación entre Android y SQLite.

En Android, la forma típica para crear, actualizar, y conectar con una base de datos SQLite será a través de una clase auxiliar llamada SQLiteOpenHelper, o para ser más exactos, de una clase propia que derive de ella y que debemos personalizar para adaptarnos a las necesidades concretas de nuestra aplicación.

La clase SQLiteOpenHelper tiene tan sólo un constructor, que normalmente no necesitaremos sobrescribir, que deberemos personalizar con el código necesario para crear nuestra base de datos y para actualizar su estructura respectivamente. Para más información sobre esta clase podemos visitar su página [21]

SQLiteOpenHelper nos permitirá **crear** y **eliminar** tablas, registros, hacer **actualizaciones**, etc.

Los métodos heredados de la clase que extiende de SQLiteOpenHelper son **onCreate**, llamado al **crear** la base de datos y **onUpgrade**, llamado para hacer una **modificación** de la base de datos.

6.3.1 APORTE AL PROYECTO

SQLite ha sido la base de datos que hemos usado para **guardar** la información del usuario relativa a los ***datos personales*** como el nombre de usuario, la contraseña o el email. Además, también se han guardado otro tipo de datos del usuario, como son los **datos culinarios** o los **datos alérgicos**.

7. MANUAL DEL PROGRAMADOR

En este apartado se explica con un poco más de detalle cómo se ha llevado a cabo el **desarrollo** del proyecto. El manual del **programador**, hará un resumen que contendrá información acerca de los **requisitos del sistema** (casos de uso, arquitectura de la aplicación,…), y una **explicación detallada** de cómo se ha ido diseñando e implementado el proyecto. En estos puntos se tratarán aspectos como la estructura del proyecto, cuáles son sus componentes principales, como se ha llevado a cabo el diseño visual de la aplicación y la explicación detallada de varios métodos que son de vital importancia en la implementación del proyecto.

7.1 METODOLOGÍA

Para llevar a cabo una solución a los problemas contemplados en este proyecto, hemos decidido seguir un marco de desarrollo software denominado **Proceso Unificado**. Este proceso se caracteriza por ser:

- **Iterativo e incremental**

El Proceso Unificado es un marco de desarrollo iterativo e incremental compuesto de cuatro fases denominadas Inicio, Elaboración, Construcción y Transición. Cada una de estas fases es a su vez dividida en una serie de iteraciones. Estas iteraciones ofrecen como resultado un incremento del producto desarrollado que añade o mejora las funcionalidades del sistema en desarrollo.

- **Dirigido por Casos de Uso**

En el Proceso Unificado los **casos de uso** se utilizan para capturar los **requisitos funcionales** y para definir los **contenidos de las iteraciones**. La idea es que cada iteración tome un conjunto de casos de uso o escenarios y desarrolle todo el camino a través de las distintas disciplinas: diseño, implementación, prueba, etc.

7.2 REQUISITOS DEL SISTEMA

Respecto a los **requisitos** del sistema podemos destacar los siguientes:

REQUISITOS FUNCIONALES

- El proyecto debe permitir el **registro** de un usuario para cada aplicación incluyendo el registro de **datos personales**, **datos culinarios** y **datos alérgicos**.
- La aplicación podrá controlar que los datos introducidos en los campos de texto sean **correctos** y no estén vacíos.
- Será posible **iniciar sesión** en la aplicación una vez nos hayamos registrado.
- El usuario tendrá la posibilidad de ver un **listado de restaurantes** y poder seleccionar alguno de ellos.
- El usuario **elegirá** sobre su deseo de intercambiar preferencias con sus compañeros de comensal.
- El cliente va a tener voto para elegir los compañeros con los que **compartir** sus preferencias.
- El cliente tendrá la posibilidad de ver un **listado de platos o tapas** ordenadas según sus puntuaciones.
- Será posible permitir al usuario elegir uno o un **conjunto de platos** o tapas.
- La aplicación podrá actualizar la información culinaria de un usuario logueado.
- El usuario tendrá derecho para elegir un plato o tapa al que sea **alérgico** avisando consecuentemente de sus consecuencias.
- Cada persona que inicie sesión, deberá saber cuáles son las tapas o platos que contienen ingredientes a los que es alérgico, visualizando en **tonos** rojizos los datos de la tapa.
- Cada usuario tendrá la posibilidad de salir de la aplicación, una vez hayamos iniciado sesión.

- La aplicación debe permitir volver atrás en las pantallas que así se decida.

REQUISITOS NO FUNCIONALES

- A cada usuario se le permitirá tener **guardada** toda su información en la memoria interna del teléfono.

- NO se podrá denegar al usuario interactuar con la aplicación sin conexión a internet.

7.2.1 CASOS DE USO

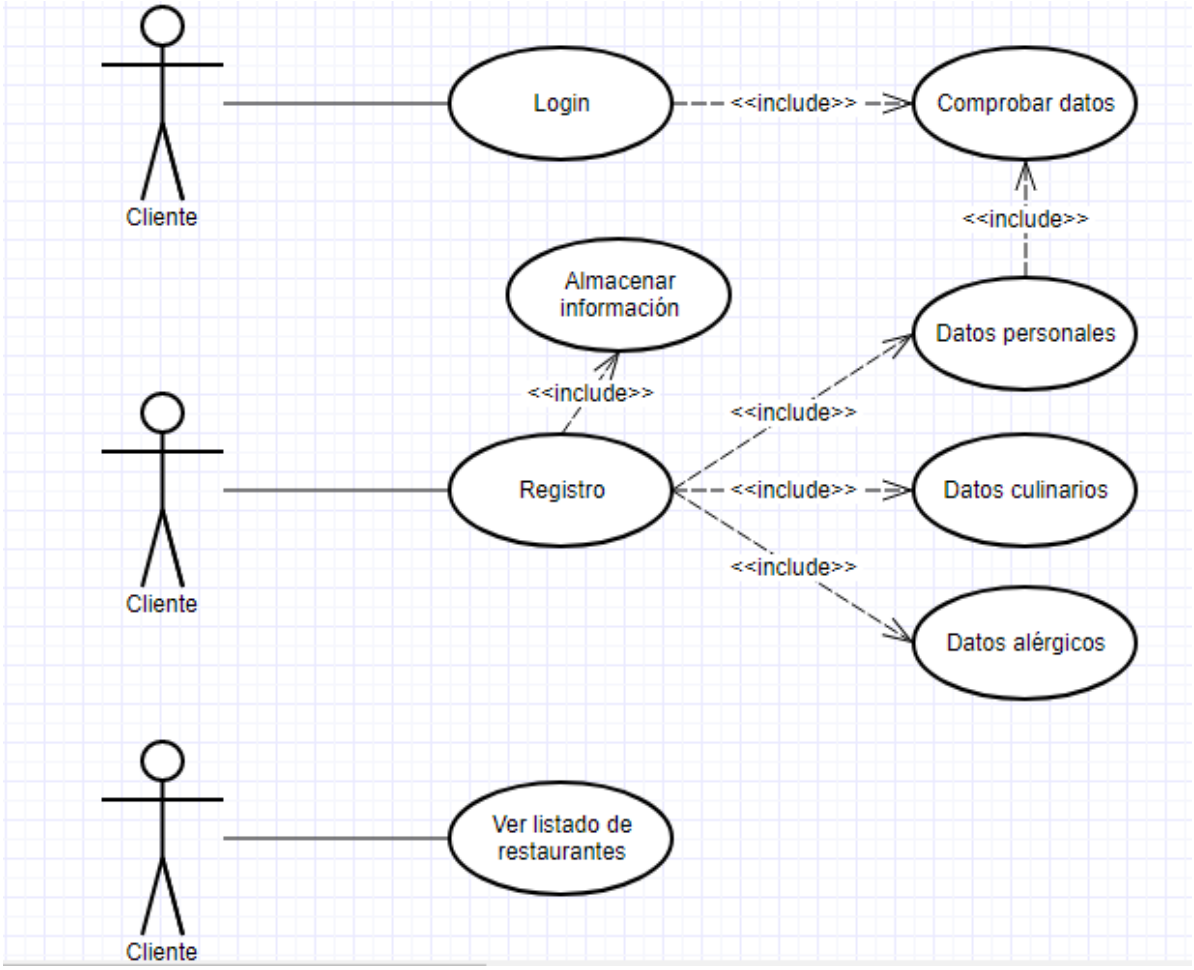

Ilustración 22- Casos de uso (1)

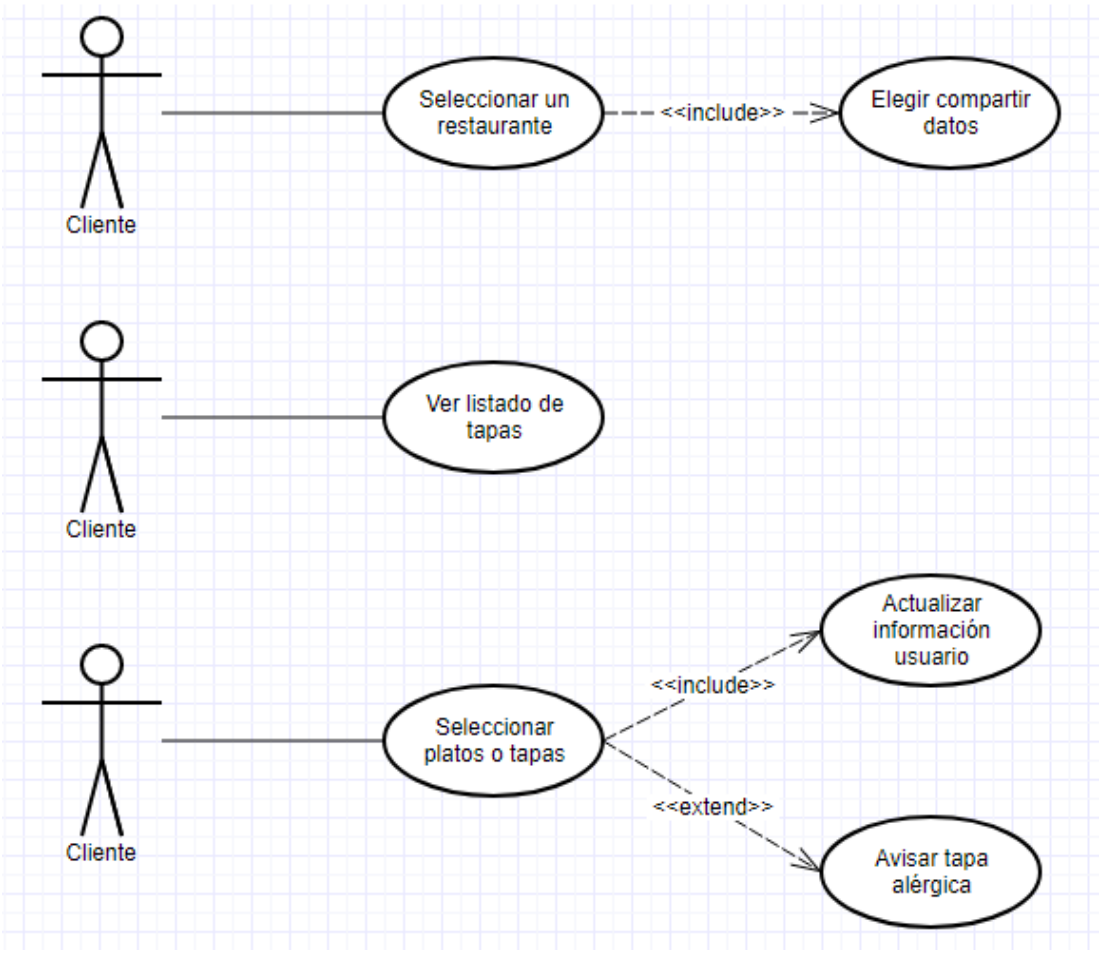

Ilustración 23- Casos de uso (2)

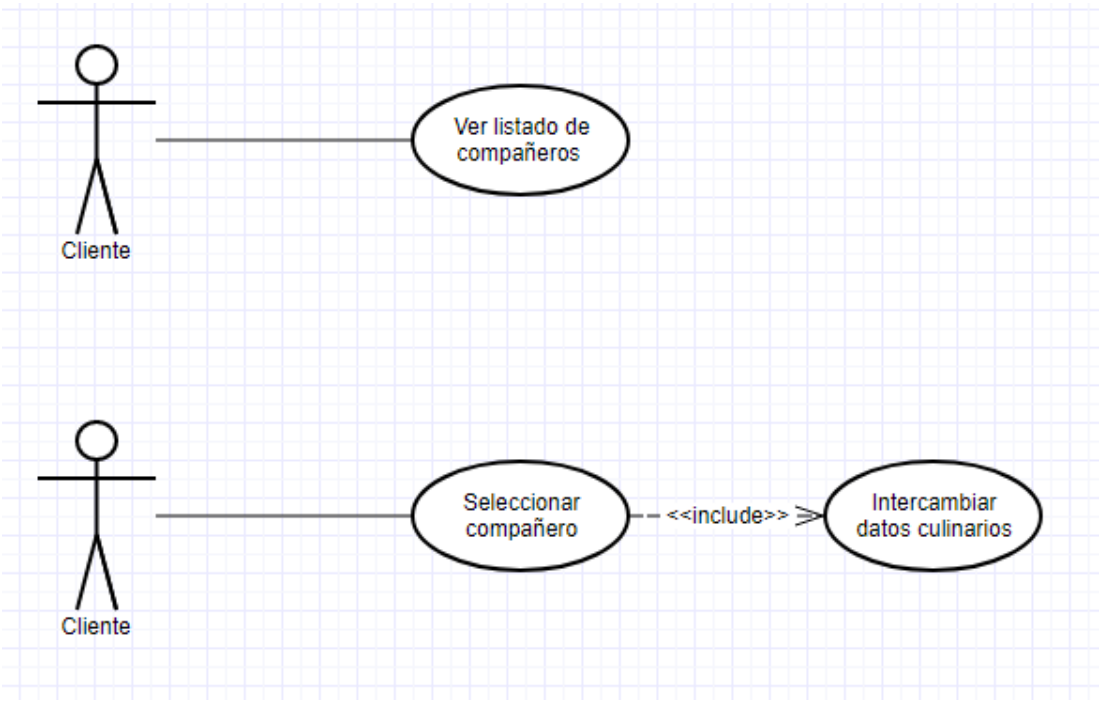

Ilustración 24- Casos de uso (4)

7.3 ARQUITECTURA

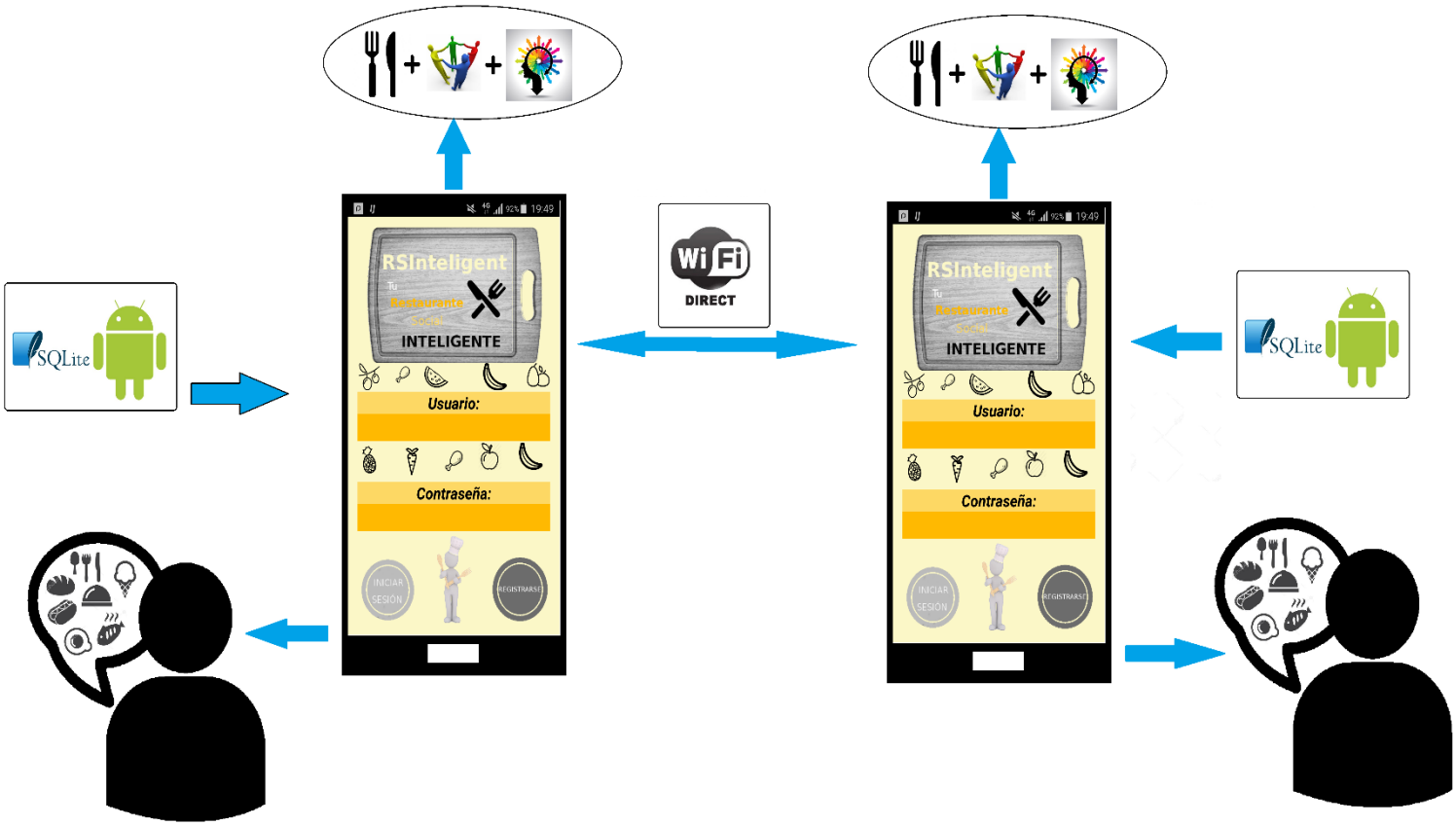

Ilustración 25- Arquitectura

En el centro de la imagen podemos observar 2 dispositivos con la aplicación móvil, cada uno, va a contener el perfil de los gustos culinarios de la persona. Mediante SQLite la aplicación va a ser capaz de almacenar la información del usuario. Respecto al intercambio de datos culinarios entre los 2 dispositivos, usaremos Wifi Direct. El resultado de este proceso es mejorar la relación existente entre restaurantes, sociedad e inteligencia.

7.3.1 DIAGRAMA DE TECNOLOGÍAS

A continuación, en la siguiente imagen se puede ver de forma general cuáles son las **principales** tecnologías que integran la aplicación. Este **diagrama** servirá para tener una visión más general de cómo está **estructurada** la solución.

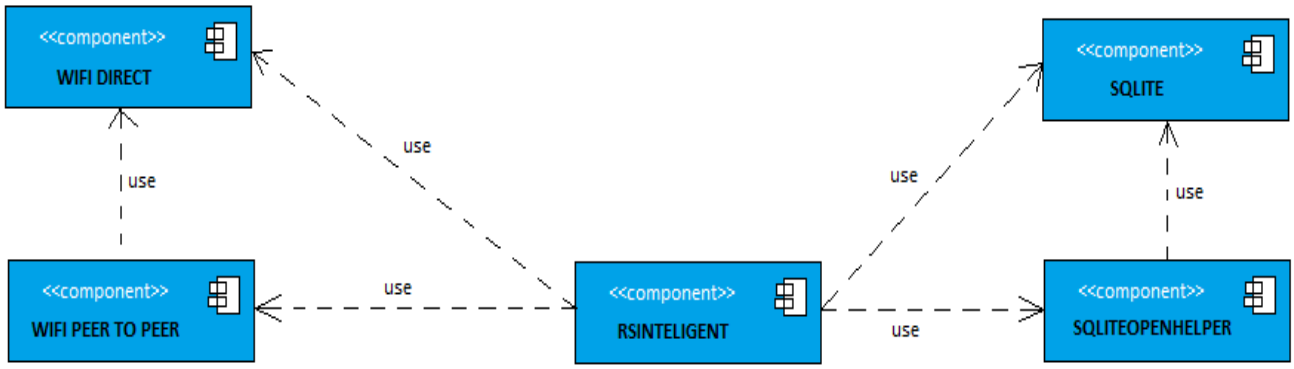

Ilustración 26- Diagrama de tecnologías integradas

En el centro del diagrama nos encontramos con la aplicación. Ésta tiene a su derecha **SQLite** que es la base de datos que hemos usado para almacenar la información. **SQLiteOpenHelper** es la API que usa Android Studio para interactuar con esta base de datos. Por la izquierda podemos observar a **Wifi Direct** que es la tecnología mediante la cual compartimos nuestras preferencias culinarias. **WifiP2P** es la API que posee Android para interactuar con la tecnología.

7.3.2 DIAGRAMA DE CLASES

En el siguiente apartado se va a ir explicando por partes el diagrama de clases del proyecto. En concreto, veremos el diagrama para los casos de uso más importantes.

REGISTRO

En primer lugar, comenzaremos viendo el diagrama para el **registro** en la aplicación. Este diagrama posee la actividad (activity) de **login,** que es la primera página que vamos a ver una vez la aplicación esté cargada. En ella, nos encontramos con métodos para *inicializar los componentes* o para ***consultar*** si los campos introducidos en los campos de texto se corresponden con alguna persona registrada. En caso de no estar registrado acudiremos al *registro1*, actividad donde se podrán introducir los ***datos personales*** del usuario. El *registro2* estructurado de la misma manera que la anterior y que el siguiente, es el encargado de controlar las **valoraciones** de las comidas. Para ello, usamos un método del ciclo de vida llamado **onResume**() que actúa instantáneamente en caso de que pulsemos en cualquier *checkbox* de puntaciones. Referente al último registro, tiene la estructura similar de los anteriores y es el encargado de gestionar las **alergias**. Finalmente destacar, las 3 clases que extienden de **SQLiteOpenHelper** (API detallada anteriormente) para guardar la información a cerca de cada tipo de dato.

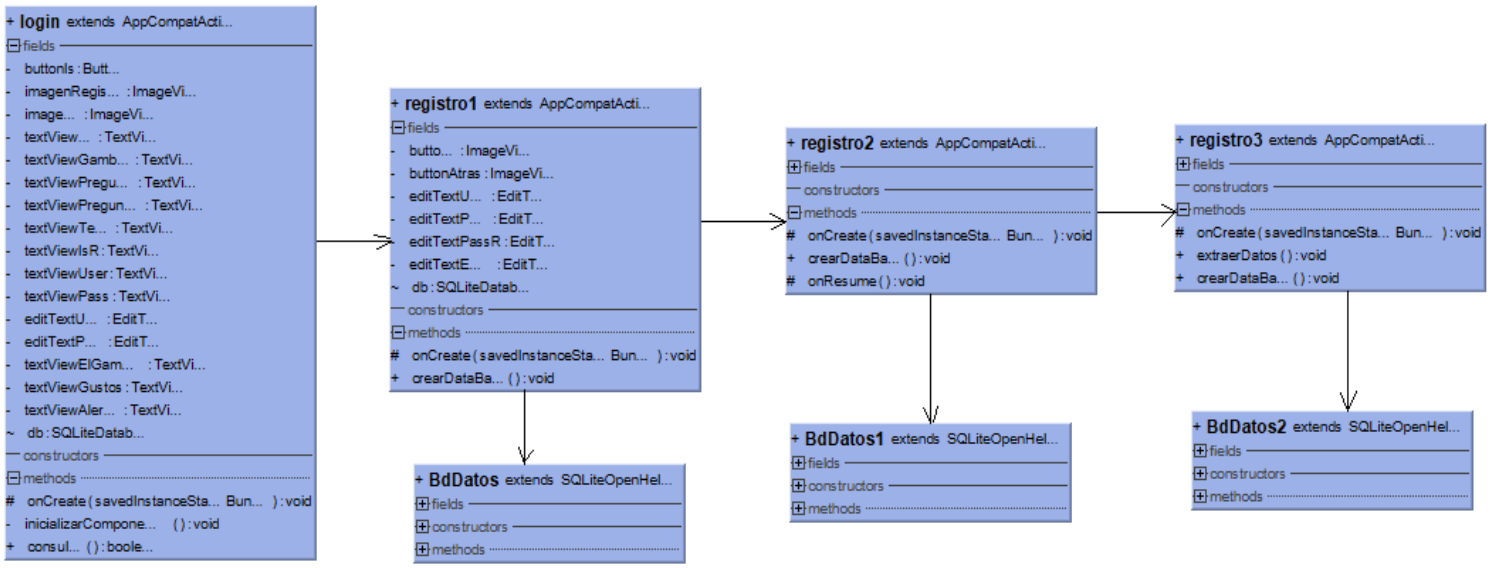

Ilustración 27- Diagrama de clases (1).

LOGIN

Una vez detallado el diagrama de clases para el caso de uso de poder registrarse en la aplicación, continuaremos con el diagrama de clases para **iniciar sesión** en la aplicación (**login**).

Este diagrama simplemente contiene una actividad llamada **login**(detallada anteriormente) que va a ser la encargada de comprobar si los datos introducidos en los campos de usuario y contraseña se **corresponden** con la persona almacenada en la memoria interna del dispositivo. Para ello, se hace uso de **SQLiteOpenHelper** mediante una consulta **sql**.

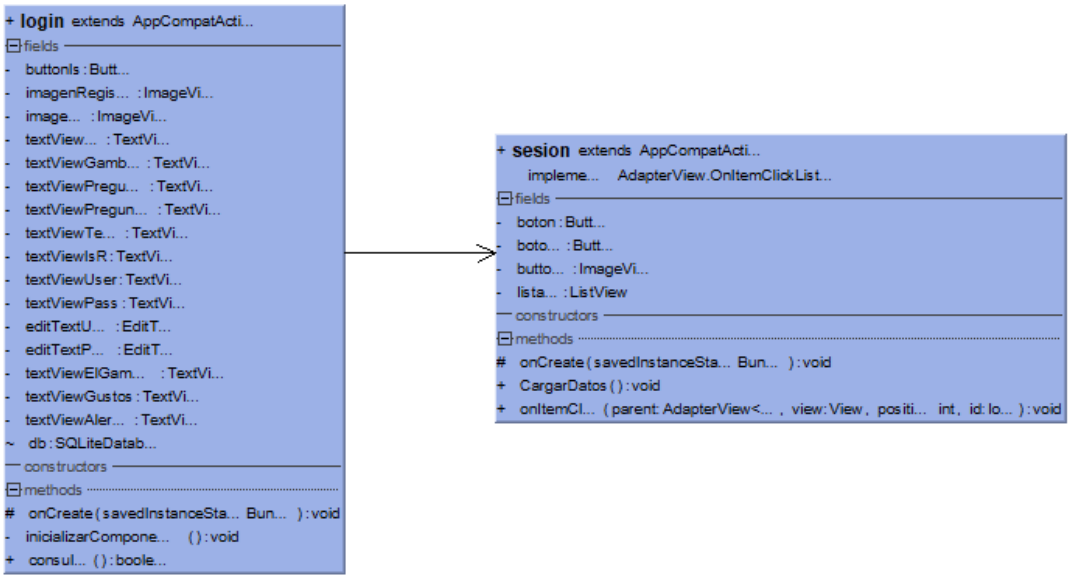

Ilustración 28- Diagrama de clases (2).

COMPARTO DE PREFERENCIAS

Seguidamente vamos a continuar viendo en detalle el diagrama de clases para el caso de uso de **intercambiar** preferencias culinarias entre los diferentes comensales.

En primer lugar, para acceder a la ubicación del diagrama que se contempla a continuación, debemos de haber **iniciado sesión** en la aplicación y haber **elegido** un restaurante. Respecto a la elección de restaurantes, la realizamos desde la clase "sesion", que implementa un **AdapterView.OnItemClickListener,** que se trata de una definición de interfaz que será invocada cuando el usuario haga clic en un elemento de ese **AdapterView**.

Cuando la aplicación detecte un dispositivo, van a aparecer en la pantalla mediante el adaptador **MyAdapter,** que extiende de una *RecyclerView.Adapter* que recogerá todos los dispositivos detectados incluyendo su nombre y su dirección mac.

Cuando la conexión vía Wifi Direct se inicia entre 2 dispositivos, se va a ejecutar una tarea **asíncrona** a la actividad principal llamada **DataServerAsyncTask** que va a ser la encargada de **esperar** información por parte del cliente, y además en el dispositivo emisor de información culinaria, se ejecutará un nuevo hilo llamado **DataTransferService**.

Adicionalmente, poseemos un receptor de **broadcast** (Broadcast receiver) llamado ***WifiDirectBroadcastReceiver*** que se trata de un componente Android que permite el registro de eventos del sistema. Todos los receptores registrados serán notificados por Android una vez que estos ocurran. Por ejemplo, en esta clase existen eventos registrados para cuando iniciamos la búsqueda de dispositivos mediante Wifi Direct en nuestro alcance, de tal modo que somos notificados cuando esta búsqueda se inicie o se pare.

Las **preferencias** de los compañeros serán almacenadas en una lista que posee la clase "MainActivity". Mediante la tarea asíncrona "DataServerAsyncTask", vamos a insertar los datos culinarios de nuestros compañeros en la lista, una vez, hayamos recibido información de nuestros compañeros. Además, DataServerAsyncTask será el encargado de comprobar si ya existen los ingredientes en la lista. En caso de que existan, sumaremos sus puntuaciones, en caso contrario, insertaremos ese nuevo ingrediente a la lista. Cuando acabemos el intercambio de preferencias, esta lista será la encargada de interactuar con la información culinaria del usuario principal para generar un nuevo listado de platos.

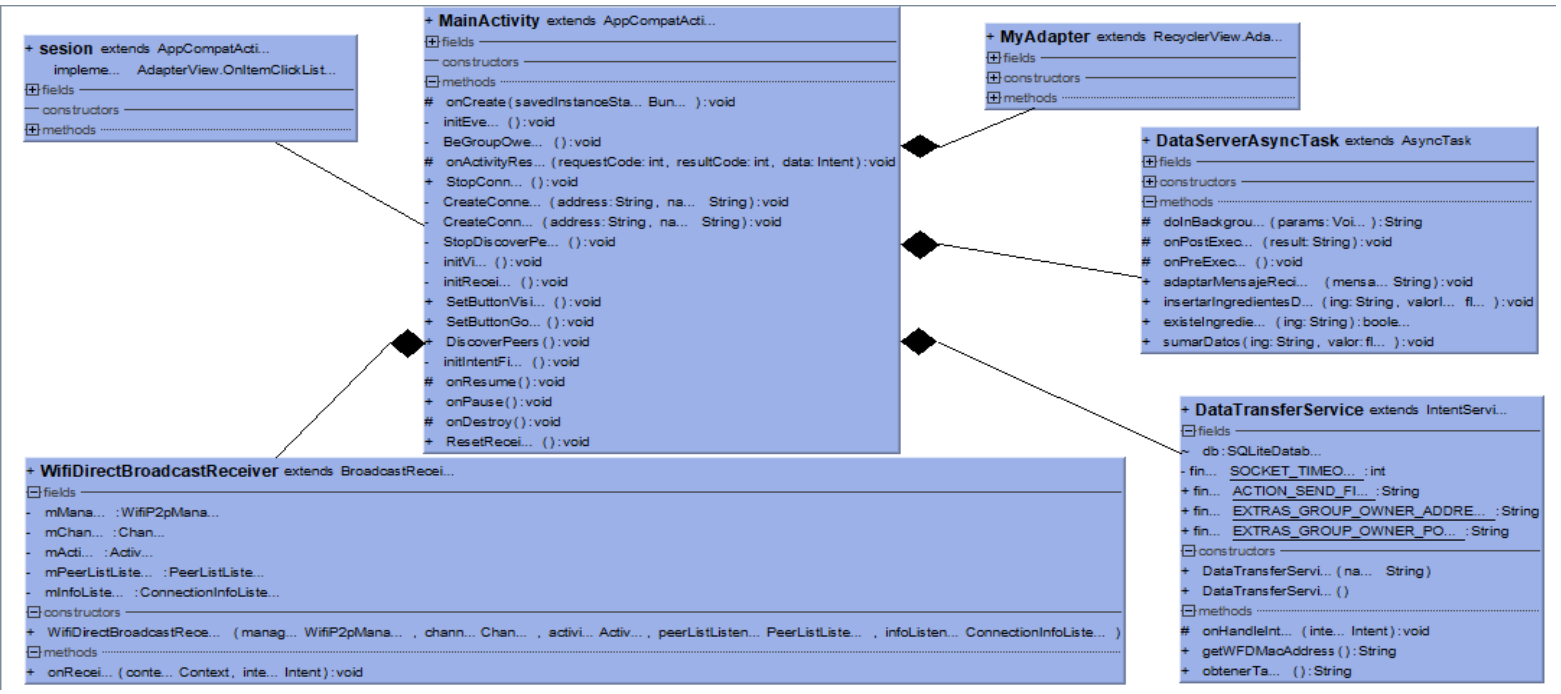

Ilustración 29- Diagrama de clases (3).

GESTIÓN DE TAPAS O PLATOS

Finalmente, veremos el diagrama de clases sobre **la elección de un plato** o tapa en un restaurante y su correspondiente actualización culinaria. La actividad que gestiona este primer proceso se llama "*resultadoI*" que como podéis observar se trata de la clase más **extensa** del proyecto debido a todas las gestiones sobre **ingredientes**, **tapas** o **puntaciones** que se llevan a cabo. En ella, podemos observar un *ArrayAdapter* llamado *CustomArrayAdapter* que será el encargado de crear las vistas de la lista (ListView) a partir de los datos almacenados en un array.

Para evitar consultas constantes a la base de datos decidimos crear ***2 objetos*** que podemos observar en el diagrama. El primero a detallar será "**Item**", que implementa un *comparable* por **puntuación** con el fin de tener una **lista ordenada** en base a las **puntuaciones** de las tapas. En cambio, "***ItemIngrediente***", es el objeto mediante el cual gestionamos los ingredientes que poseemos y sus puntuaciones.

Cuando el usuario selecciona una o varias tapas, se va **actualizar** la información culinaria de este. Para ello, haremos uso de métodos de la clase "*resultadoI*" para insertar o actualizar ingredientes y puntuaciones en "*BdDatos1*", dependiendo de si el **ingrediente** existía o no en la base de datos. En caso de que exista, **sumaremos** las puntuaciones, en caso contrario, **añadiremos** ese nuevo ingrediente a la base de datos.

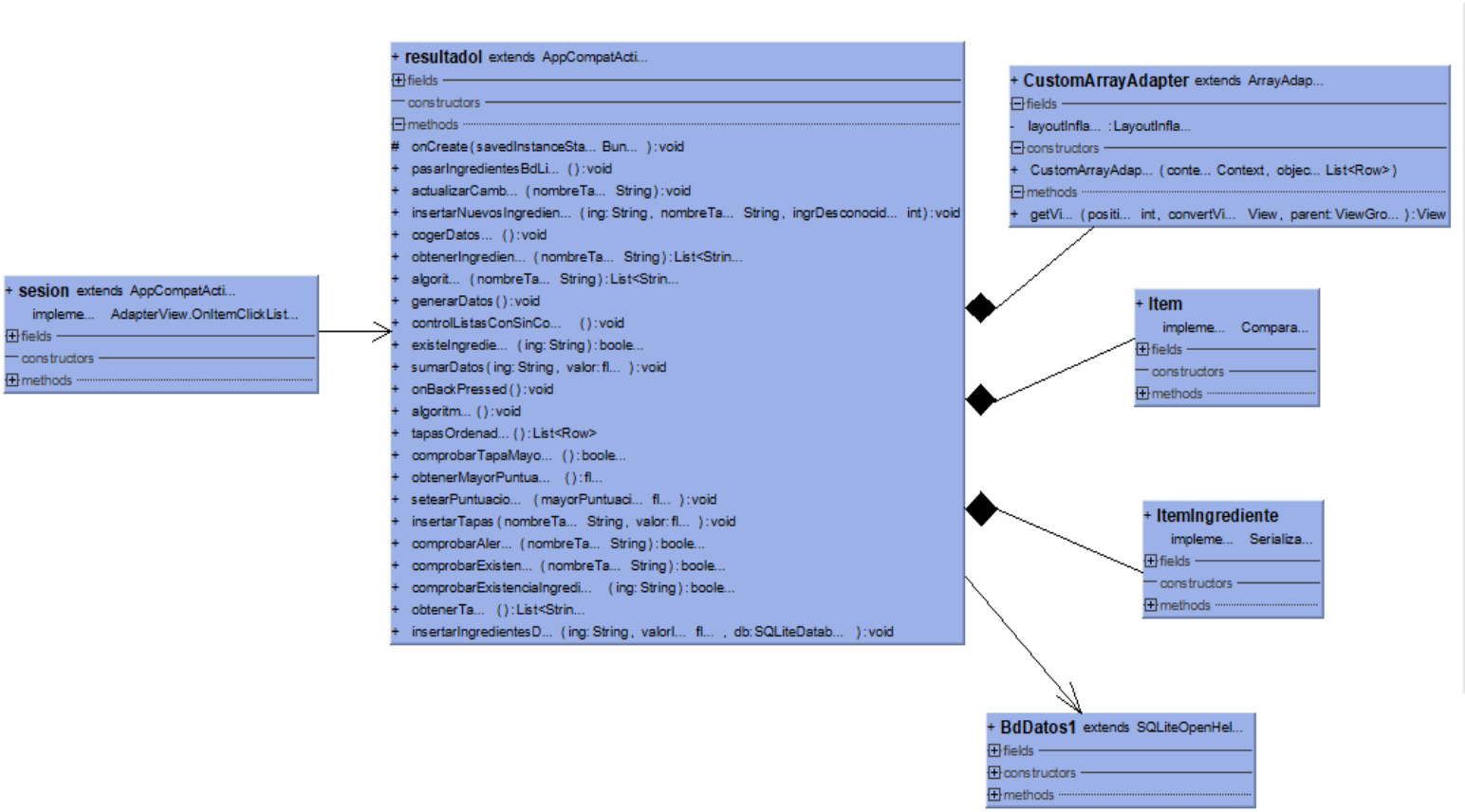

Ilustración 30- Diagrama de clases (4)

7.3.3 DIAGRAMA DE BASES DE DATOS

A continuación, para acabar el apartado de **Arquitectura**, veremos finalmente el diagrama de bases de datos. En concreto, podremos observar las **tablas y atributos** que hemos utilizado para almacenar la información culinaria y personal del usuario. Como ya se ha detallado anteriormente, la base de datos ha sido SQLite. Para el almacenamiento de información nos hemos creado 3 tablas; una llamada **Usuario** que sirve para almacenar la información personal del usuario, otra denominada **DatosAlergias** para almacenar la información sobre aquellos alimentos a los que somos alérgicos y finalmente otra tabla llamada **DatosCulinarios** que sirve para almacenar los datos sobre ingredientes y sus correspondientes puntuaciones.

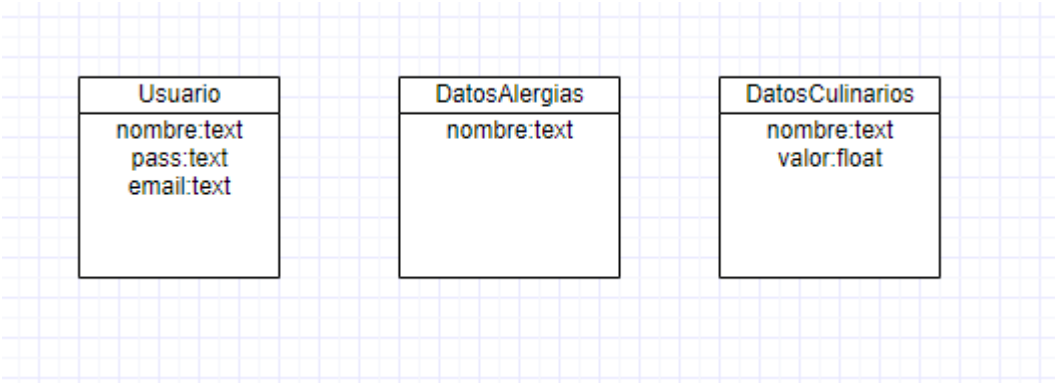

Ilustración 31- Diagrama de Base de Datos

7.4 DISEÑO E IMPLEMENTACIÓN

En este apartado vamos a explicar cómo se ha ido diseñando e implementado el proyecto. Todo se ha ido desarrollando mediante la herramienta Android Studio, que posee una **web** para desarrolladores muy importante.

A lo largo de todo el documento se ha estado trabajando, en base a los **casos de uso**, por ello mostraremos este apartado según los casos de uso impuestos anteriormente.

7.4.1 ASPECTO VISUAL

Antes de comenzar con el diseño e implementación del proyecto siguiendo los casos de uso, comenzaremos explicando cómo se ha llevado a cabo la parte **visual** de la aplicación ya que considero un aspecto importante debido al **tiempo** que ha llevado. ***Android Studio*** posee una herramienta muy potente para el diseño del aspecto visual de la aplicación. Internamente, esta herramienta lo que está haciendo es crear un archivo "**xml**" que nosotros podemos *visualizar* y *editar*. Los diseños de las diferentes páginas se encuentran en la carpeta "*layout*". En ella, se encuentran los diseños del proyecto que son objetos que representan el espacio contenedor de todas las vistas (Views) dentro de una actividad. Una actividad es la parte lógica de cada una de las pantallas o vistas que forman una aplicación**.**

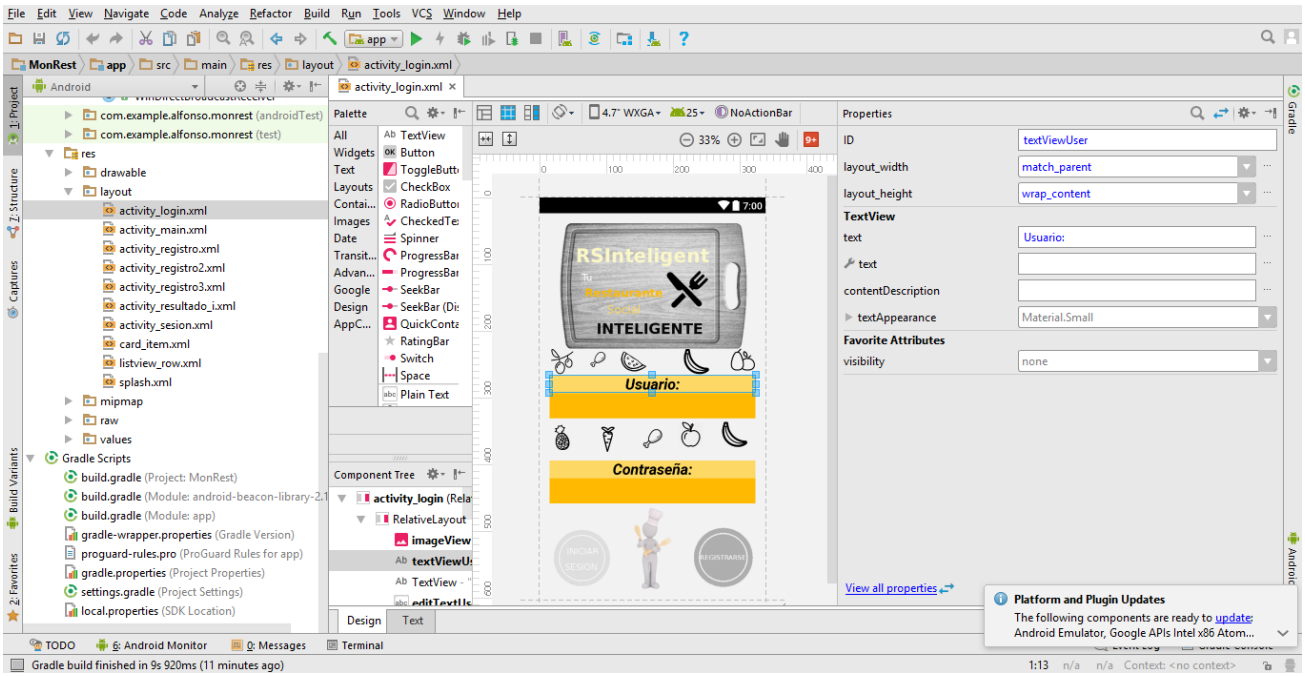

Ilustración 32- Diseño en Android Studio.

Como se puede observar en la imagen anterior, estamos ante la pantalla **principal** de la aplicación. La pantalla está formada por *textView*(textos), *button*(botones), *imageView*(imagenes), *editText*(texto editable). Todos estos componentes se pueden crear y editar tanto desde la **pantalla de diseño** como desde el propio **xml**.

Varias de las imágenes que poseen el aspecto visual de la aplicación, han sido creadas por mí a través de la herramienta **GIMP** (programa de manipulación de imágenes). Esta herramienta resulta muy útil para hacer *fondos transparentes* o para retocar la **imagen** en general. Lo mejor de esta herramienta es su sencillez y que es **gratuita**. Podéis descargar la aplicación desde su web oficial [54].

Ilustración 33- Logo gimp [54]

7.4.2 CASOS DE USO: REGISTRO Y CONTROL DE ERRORES

El siguiente paso en el diseño e implementación de la aplicación fue preparar el caso de uso de **registrarse** un usuario. El registro se compone de varias **páginas** (datos personales, datos culinarios y datos alérgicos). Cada una de ellas posee un formulario con el que añadiremos información propia a la aplicación. A continuación explicaré la implementación de cada uno de ellos:

- **DATOS PERSONALES:**

Se trata de un típico **formulario** de *registro* en el que debemos introducir datos personales *como nombre de usuario, contraseña, repetición de contraseña o email*. Estos campos están totalmente **validados** mediante **expresiones regulares**. Por ejemplo, respecto a la validación del **email** he usado la siguiente expresión regular:

^[_a-z0-9-]+(.[_a-z0-9-]+)*@[a-z0-9-]+(.[a-z0-9-]+)*(.[a-z]{2,4})$

En caso de introducir algún **campo erróneo**, la aplicación no nos va a dejar continuar el registro, avisándonos del problema con un mensaje.

DATOS CULINARIOS:

Esta página del caso de uso de registro ha sido la más compleja de implementar ya que ha implicado un **esfuerzo** bastante mayor. Estos problemas se han debido a los **aspectos visuales** que veréis a continuación.

Ilustración 32- Registro datos culinarios

Como podemos observar, para obtener los datos culinarios de cada usuario utilizamos puntuación en base a **estrellas**. Cada estrella es un **checkbox** diferente, de ahí uno de sus principales motivos de complejidad. La idea es controlar desde su actividad cuantas estrellas hemos pulsado por cada dato. En el caso de pulsar primero 5 o cuatro y después un número menor, las estrellas tienen que marcarse y desmarcarse **automáticamente**. Este proceso se ha realizado de la siguiente manera:

- **XML**

```
<LinearLayout
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="42dp"
    android:id="@+id/lave">
```

```
    <TextView
        android:text="Ave"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/ave"
        android:layout_weight="1"
        android:textColor="@android:color/darker_gray"
        android:textSize="20sp"
        android:textAlignment="center" />

    <CheckBox
        android:layout_width="30dp"
        android:layout_height="wrap_content"
        android:id="@+id/taga1"
        android:tag="1"
        style="?android:attr/starStyle"
        android:background="@android:color/background_dark"
        android:theme="@style/yourStyle"/>
    <CheckBox
        android:layout_width="30dp"
        android:layout_height="wrap_content"
        android:id="@+id/taga2"
        android:tag="2"
        style="?android:attr/starStyle"
        android:background="@android:color/background_dark"
        android:theme="@style/yourStyle"/>
    <CheckBox
        android:layout_width="30dp"
        android:layout_height="wrap_content"
        android:id="@+id/taga3"
        android:tag="3"
        style="?android:attr/starStyle"
        android:background="@android:color/background_dark"
        android:theme="@style/yourStyle"/>
    <CheckBox
        android:layout_width="30dp"
        android:layout_height="wrap_content"
        android:id="@+id/taga4"
        android:tag="4"
        style="?android:attr/starStyle"
        android:background="@android:color/background_dark"
        android:theme="@style/yourStyle"/>
    <CheckBox
        android:layout_width="30dp"
        android:layout_height="wrap_content"
        android:id="@+id/taga5"
        android:tag="5"
        style="?android:attr/starStyle"
        android:background="@android:color/background_dark"
        android:theme="@style/yourStyle"/>


</LinearLayout>
```

Hay que prestar especial atención al atributo **android:tag=””** ya que a través de este **atributo** y del **identificador** podemos capturar cuando pulsamos sobre las estrellas desde la actividad.

- **LÓGICA (ACTIVITY)**

```
lave = (LinearLayout) findViewById(R.id.lave);
for (int i = 1; i <= 5; i++) {
    ave = (CheckBox) lave.findViewWithTag(String.valueOf(i));
    ave.setOnClickListener(avesListener);

}


private View.OnClickListener avesListener = new
View.OnClickListener() {

    public void onClick(View v) {
        int tag = Integer.valueOf((String) v.getTag());
        tagAve = tag;
        for (int i = 1; i <= tag; i++) {
            ave = (CheckBox)
lave.findViewWithTag(String.valueOf(i));
            ave.setChecked(true);
        }
        for (int i = tag + 1; i <= 5; i++) {
            ave = (CheckBox)
lave.findViewWithTag(String.valueOf(i));
            ave.setChecked(false);
        }
    }

};
```

Este trozo de código se encarga de **chequear** y **quitar** el chequeo en las estrellas. De tal modo, que si pulsamos sobre la **quinta** estrella se van a pintar todas las estrellas. De la misma forma, si tenemos todas las estrellas pintadas, en caso de pulsar en una menor, las restantes dejan de estar chequeadas.

Destacar, que debido a que la información a representar no cabía en la pantalla se decidió poner una vista con ***scroll*** para la representación de los datos culinarios.

La última decisión resuelta para esta página fue como cambiar el **color** que se pone automáticamente al pulsar en cualquier *checkbox*. Android Studio pone un color por defecto y habitualmente se puede usar en el XML la propiedad “**buttonTint**” para cambiar el color. Sin embargo, esta propiedad no funciona ya que ponía siempre el **checkbox** del mismo color aun pulsando sobre él. Por lo que después de mucho *investigar* logramos hacerlo funcionar de la siguiente manera: en primer lugar, debemos de irnos al archivo donde tenemos los estilos, en mi caso llamado *style.xml* y crear un nuevo estilo.

```
<style name="stilo" parent="Base.Theme.AppCompat">
```

```
    <item name="colorAccent">#000</item> <!-- for uncheck state -->
    <item name="android:textColorSecondary">#000</item> <!-- for
check state -->
</style>
```

Como podemos comprobar usamos **2 colores** uno para cuando el checkbox esté en estado chequeado y otro al contrario. Una vez tenemos ya creado el estilo, simplemente añadimos la siguiente línea al checkbox:

```
android:theme="@style/stilo"
```

De esta manera podremos observar que el checkbox utiliza los colores anteriormente indicados.

DATOS ALÉRGICOS:

En la última página de registro atacaremos **los datos alérgicos** del usuario. Para ello, simplemente usamos checkbox para seleccionar aquellos productos a los que somos alérgicos.

Ilustración 34- Datos alérgicos

Una vez hemos finalizado el registro, vamos a ver el caso de uso de **guardar** toda la información del usuario en la memoria interna del teléfono a través de **SQLite.**

7.4.3 CASO DE USO: ALMACENAMIENTO DE INFORMACIÓN CULINARIA

Como ya se ha comentado anteriormente, para guardar y actualizar los datos de cada usuario, usamos **SQLite**. A continuación, veremos cómo se ha desarrollado todo este proceso para guardar datos, actualizarlos y hacer consultas.

En primer lugar comenzaremos creando una clase que extienda de **SQLiteOpenHelper** como la siguiente:

```
public class BdDatos1 extends SQLiteOpenHelper {
    String sqlDatosC = "CREATE TABLE DatosTapas (nombre TEXT, valor
float)";
    public BdDatos1(Context context, String name,
SQLiteDatabase.CursorFactory factory, int version) {
        super(context, name, factory, version);
    }
    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(sqlDatosC);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int
newVersion) {
    }
}
```

Como se puede observar en color verde, tenemos el código **SQL** que va a crear la tabla con esas columnas.

Seguidamente, continuaremos con la **inserción** de datos en esta base de datos. Para ello, simplemente debemos de observar este ejemplo para que se entienda perfectamente.

```
BdDatos2 DatosA = new BdDatos2(this,"DatosAlergias",null,1);
int codigo = 1;
db = DatosA.getWritableDatabase();
 extraerDatos();

if(db != null){
    db.execSQL("INSERT INTO DatosAlergias (nombre , valor) "+"
VALUES ("'Leche', "+ Leche + ")");
    db.execSQL("INSERT INTO DatosAlergias (nombre , valor) "+"
VALUES ("'Trigo', "+ Trigo + ")");
    db.close();
}
```

Para las actualizaciones, en lugar de usar CREATE o INSERT usaremos **UPDATE**. A continuación se muestra un ejemplo de actualización:

```
BdDatos1 DatosTapas = new BdDatos1(this,"DatosTapas",null,1);
db = DatosTapas.getWritableDatabase();
boolean b = false
      float desconocidos = (float)ingrDesconocidos;
      float valor = (float) (1.0/desconocidos);
      if(db != null){
                db.execSQL("UPDATE DatosTapas SET
      valor=valor+"+valor+" WHERE nombre='"+ing+"' ");
        db.close();
      }
```

Por último haremos una consulta de **selección** en el que vamos a poder recuperar los datos que queramos según la **consulta** que usemos. Hay que tener en cuenta que cuando vayamos a hacer modificaciones en la base de datos usaremos el método ***getWritableDatabase***(). Para casos contrario, como en las consultas de selección, usaremos el método ***getReadableDatabase***(). En el siguiente ejemplo veremos cómo se hacen consultas de selección con SQLiteOpenHelper:

```
BdDatos2 datosA = new BdDatos2(this, "DatosAlergias", null, 1);
db2 = datosA.getReadableDatabase();
if (db2 != null) {

    Cursor c = db2.rawQuery(" SELECT * FROM DatosAlergias ", null);
    if (c.moveToFirst()) {
        do {
            String nombre = c.getString(1);
            Integer var = c.getInt(2);
            System.out.println("Comprobando alergia "+ nombre);
        } while (c.moveToNext());
    }
    db2.close();


}
```

7.4.4 CASO DE USO: INTERCAMBIO DE PREFERENCIAS

A continuación se va a hacer un resumen de como se ha ido implementando la funcionalidad de **intercambiar preferencias** mediante Wifi Direct. Destacar que este proceso ha sido el que más tiempo nos ha llevado debido a la **complejidad** de implementación que tiene. Como comenté en el apartado de decisiones, la primera opción fue hacer conexiones **many** to **many** entre dispositivos. Debido a que las opciones de conexión many to many o conexiones individuales bidireccionales no eran posible de realizar, se decidió realizar **conexiones individuales automáticas** de tal modo que simulamos conexiones individuales bidireccionales.

Para llevar a cabo el compartimento de datos mediante Wifi Direct hay que tener **claro** cierta información. Cada dispositivo que inicia una conexión a otro tiene la posibilidad de elegir si va a actuar como **cliente** o como **servidor** (punto de acceso). Por lo tanto, vamos a poder elegir si queremos **enviar** datos o **recibirlos**. En caso de recibir datos, no vamos a poder enviarlo ya que son roles diferentes. Este fue uno de los motivos por los que no funcionan las conexiones individuales bidireccionales.

Para llevar a cabo una conexión mediante Wifi Direct ejecutaremos la función que podremos ver a continuación. La siguiente función toma como parámetro la dirección **MAC** del dispositivo al que queremos conectarnos. Por lo tanto, para iniciar una comunicación mediante Wifi Direct con otro dispositivo, deberemos saber su dirección **MAC** y encontrar este dispositivo en la **lista de dispositivos** encontrados tras hacer una búsqueda. En primer lugar, crearemos un objeto de tipo *WifiP2pConfig* para configurar la dirección del dispositivo al que queremos conectarnos (**deviceAddress**) y asignar un valor a la propiedad ***groupOwnerIntent*** (entero que toma valores entre **0 y 15** para indicar el grado de inclinación a ser propietario del grupo). El uso de esta propiedad es muy importante para saber quién va a enviar los datos en la comunicación. Si el valor es igual a **0** sabemos que quien envía los datos va a ser quien inicia la comunicación. Por el contrario, si hubiésemos puesto el valor **15** quien inicia la comunicación es el que recibe la petición de conexión. En este proyecto se han usado los valores que poseen la cota superior o inferior de la propiedad ***groupOwnerIntent***. En caso de haber usado los valores 1 y 14 respectivamente, hubiese funcionado igualmente, aunque sin embargo, si usamos valores como 7 u 8 podría dar lugar a equivocaciones ya que no podríamos identificar correctamente los roles en la comunicación. Finalmente, la conexión se **crea** mediante el método **connect**() para el canal especificado y con la configuración indicada.

```
 private void CreateConnect(String address) {


        WifiP2pConfig config = new WifiP2pConfig();
        Log.i("xyz", address);
        config.deviceAddress = address;

        config.groupOwnerIntent = 0;

        Log.i("address", "lingyige youxianji" +
String.valueOf(config.groupOwnerIntent));

        mManager.connect(mChannel, config, new
WifiP2pManager.ActionListener() {

            @Override
            public void onSuccess() {
        System.out.println("Conexion bien realizada");


            }


            @Override
            public void onFailure(int reason) {
         System.out.println("Conexion mal realizada");

            }
        });
    }
```

Respecto a la funcionalidad de **reenviar** datos es imprescindible haberse creado un método para obtener la dirección **MAC** del dispositivo que crea la conexión. A continuación, se puede observar el siguiente método que obtiene la dirección MAC del dispositivo mediante la clase **NetworkInterfaze**.

```
public String getWFDMacAddress(){
    try {
        List<NetworkInterface> interfaces =
Collections.list(NetworkInterface.getNetworkInterfaces());
        for (NetworkInterface ntwInterface : interfaces) {

            if (ntwInterface.getName().equalsIgnoreCase("p2p0")) {
                byte[] byteMac = ntwInterface.getHardwareAddress();
                if (byteMac==null){
                    return null;
                }
                StringBuilder strBuilder = new StringBuilder();
                for (int i=0; i<byteMac.length; i++) {
                    strBuilder.append(String.format("%02X:",
byteMac[i]));
                }

                if (strBuilder.length()>0){
                    strBuilder.deleteCharAt(strBuilder.length()-1);
                }
```

```
                return strBuilder.toString();
            }

        }
    } catch (Exception e) {

    }
    return null;
}
```

Una vez la conexión se ha creado y aceptado vamos a poder controlar que fragmento de código va a ejecutar cada uno de los dispositivos. Para ello, haremos uso del método **onConnectionInfoAvailable** que será ejecutado de forma automática cuando cambie el *estado* de la conexión. Este método ejecutará código diferente en caso de que el dispositivo actúe como punto de acceso o no, con el objetivo de que dependiendo del rol que tenga va a enviar información o recibirla. En la hipótesis de que el dispositivo actúe como punto de acceso vamos a crearnos una actividad ***asíncrona*** que permanezca a la espera de información por parte del cliente. Por el contrario, si no somos propietarios del grupo vamos a ser quien envié la información al servidor o punto de acceso.

```
        final WifiP2pManager.ConnectionInfoListener mInfoListener =
new WifiP2pManager.ConnectionInfoListener() {

            @Override
            public void onConnectionInfoAvailable(final WifiP2pInfo
minfo) {
                System.out.println(minfo.toString());
                Log.i("xyz", "InfoAvailable is on");
                info = minfo;
                if (info.groupFormed && info.isGroupOwner) {
                    Log.i("xyz", "owmer start");
                 //   SetButtonVisible();
                    mDataTask = new
DataServerAsyncTask(MainActivity.this, MainActivity.this);

mDataTask.executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR);



                } else if (info.groupFormed) {
                   // SetButtonVisible();

                    Intent serviceIntent = new
Intent(MainActivity.this,
                            DataTransferService.class);

serviceIntent.setAction(DataTransferService.ACTION_SEND_FILE);
```

```
serviceIntent.putExtra(DataTransferService.EXTRAS_GROUP_OWNER_ADDRES
S,

info.groupOwnerAddress.getHostAddress());

                    Log.i("address", "owenerip is " +
info.groupOwnerAddress.getHostAddress());

serviceIntent.putExtra(DataTransferService.EXTRAS_GROUP_OWNER_PORT,
                            8988);
                    MainActivity.this.startService(serviceIntent);

                }
            }
        };
        mReceiver = new WifiDirectBroadcastReceiver(mManager,
mChannel, this, mPeerListListerner, mInfoListener);

    }
```

Cada vez que el punto de acceso o servidor reciba datos por parte del cliente, vamos a ir añadiéndolos en una lista para poder enviarla mediante **intent** con la propiedad "**putExtra**" a la actividad que contiene el listado de tapas con restaurantes.

Una duda que se planteó, fue qué realizar si recibimos más de un paquete de datos con algunos **ingredientes** *iguales* y otros *diferentes*. En el caso de recibir ingredientes iguales a los existentes en la lista, simplemente **sumamos** sus puntuaciones. Si no existe un ingrediente lo **añadimos** a la lista. Imaginemos que tenemos en la lista: ajo → **5**, queso → **6** y atún → **9**. En este momento recibimos un paquete de datos con la siguiente información: ajo →**2**, atún →**1** y cebolla →**7**. La lista final generada contendrá: **ajo →7, queso →6, atún →10, cebolla →7**.

Otro aspecto a destacar es saber el **número de veces** que recibimos datos, es decir, el número de veces que nuestros compañeros nos envían sus preferencias. Desde la actividad asíncrona de recepción de datos, aumentamos el valor a una variable de la actividad padre cada vez que recibamos datos con el objetivo de poder enviar esta información a la actividad que contiene el listado de tapas. Cuando un dispositivo recibe datos de otro dispositivo a través de Wifi Direct, va a ejecutar el método **onPostExecute** propio de un ***Asynctask***. En él, hacemos las operaciones anteriormente citadas y que podréis ver a continuación:

```
@Override
protected void onPostExecute(String result) {

    Log.i("xyz", "data onpost");
    Toast.makeText(activity, "result" + result,
Toast.LENGTH_SHORT).show();
    String macAdress = result.split("-")[0];
    String preferencias = result.split("-")[1];

    activity.Adress = macAdress;
```

```
    if (result != null) {
        activity.StopConnect();
        activity.DiscoverPeers();
        activity.SetButtonVisible();
        adaptarMensajeRecibido(preferencias);
        activity.numeroCompaneros = activity.numeroCompaneros + 1;

    }
}



public void adaptarMensajeRecibido(String mensaje) {

    for (int j = 0; j < mensaje.split(";").length; j++) {
        String tapasConValor = mensaje.split(";")[j];
        String nombreIngrediente = tapasConValor.split(":")[0];
        String valorIngrediente = tapasConValor.split(":")[1];
        float valor = Float.parseFloat(valorIngrediente);
        insertarIngredientesDTS(nombreIngrediente, valor);

    }
}


public void insertarIngredientesDTS(String ing, float valorIng) {
    if (activity.numeroCompaneros == 1) {
        ItemIngrediente item = new ItemIngrediente();
        item.setPuntuacion(valorIng);
        item.setNombreIngrediente(ing);
        item.setPropio(1);
        activity.listaIngredientes.add(item);
    } else {

        if (existeIngrediente(ing) == true) {
            sumarDatos(ing,valorIng);
        } else {
            ItemIngrediente item = new ItemIngrediente();
            item.setPuntuacion(valorIng);
            item.setNombreIngrediente(ing);
            item.setPropio(1);
            activity.listaIngredientes.add(item);
        }

    }
}
```

7.4.5 GESTIÓN DE PREFERENCIAS

En este apartado podremos ver los casos de uso relacionados con la gestión de preferencias culinarias.

CASOS DE USO: LISTADO DE TAPAS Y DE RESTAURANTES.

Antes de nada, comentar como se ha gestionado el **listado de tapas** y de **restaurantes**. Para ello, simplemente guardamos en un fichero de texto de la aplicación todos los datos acerca de cuáles son los restaurantes existentes, cuáles son sus tapas o platos y para cada una de ellas, cuáles son sus ingredientes. Una vez que tenemos esta información y la información de gustos culinarios del usuario, realizamos una **conexión** entre ambos, generando una lista ordenada de platos o tapas en base a los propios gustos del usuario. De tal modo, que en la parte superior posarán aquellos platos que **más** nos gusten y en la parte inferior posaran aquellos platos que menos nos gusten o seamos alérgicos. Esto es simplemente un "**arrayList**" de un objeto que contiene datos como el nombre de la tapa o su puntuación. Este objeto se ha creado en una clase externa.

En el caso de que un ingrediente o su familia existan tanto en la información de restaurantes como en la información personal del usuario **aumentamos** la puntuación para esa tapa. En el caso contrario dejamos el valor a **0**.

Para que se entienda mejor todo este procedimiento de gestión de preferencias lo mostraré explicando los siguientes **ejemplos**:

Supongamos que tenemos una tapa, por ejemplo: **solomillo al roquefort (cerdo, queso, patatas, pimiento y huevo)**. Por otro lado, tenemos al usuario que ha valorado la carne de cerdo con **3** estrellas, al queso con **2** y las demás no las ha valorado ya que no vamos a contemplar todos los alimentos existentes. El resultado será una tapa de solomillo al roquefort con una puntuación de **5**. En el caso de que la tapa contenga algún ingrediente al que seamos alérgicos se va a añadir a la lista con una puntuación de **0**.

CASO DE USO: SELECCIONAR UNA TAPA

Por otro lado, supongamos que el usuario decide **pedir** esta tapa de solomillo al roquefort. En este caso, vamos a usar el método que se mostrará a continuación. Este método hace una **comprobación** de si los *ingredientes* que posee la tapa existen en los datos *culinarios* del usuario. En caso afirmativo, actualiza su puntuación. Sin embargo, al contrario, inserta un nuevo **ingrediente** en sus datos culinarios. Referente al código, creamos una instancia de la base de datos, abrimos esta instancia en modo escritura y hacemos la comprobación de existencia de ingredientes. En caso verdadero, ejecutamos la sentencia SQL **update** y en caso contrario realizamos un **insert**.

```
    public void insertarNuevosIngredientes(String ing, String
nombreTapa, int ingrDesconocidos){
        String nombreRestaurante = restaurante;
        BdDatos1 DatosTapas = new
BdDatos1(this,"DatosTapas",null,1);
        int codigo = 1;
        db = DatosTapas.getWritableDatabase();
        float desconocidos = (float)ingrDesconocidos;
        float valor = (float) (1.0/desconocidos);
        if(db != null){
```

```
            if(comprobarExistenciaIngrediente(ing)==false) {
                db.execSQL("INSERT INTO DatosTapas (cogUser, nombre
, valor) "+" VALUES (" + codigo + ",'"+ing+"', "+ valor + ")");
//Actualizamos la puntuacion de la tapa

            }else{
                db.execSQL("UPDATE DatosTapas SET
valor=valor+"+valor+" WHERE nombre='"+ing+"' ");
            }
            db.close();
        }

    }
```

Solomillo al roquefort posee **5** ingredientes (**cerdo, queso, patatas, pimiento y huevo**). Al entrar en este método anterior los ingredientes como patatas, pimiento y huevo los va a insertar por primera vez en sus datos culinarios con una puntuación de **0.2** (**1** entre el número de ingredientes de esa tapa). Para la hipótesis de que un ingrediente exista en nuestros datos culinarios sumamos a su valor actual **0.2**. De tal modo, ingredientes como cerdo y queso que ya existían en nuestros datos culinarios, obtendrán una puntuación de **3.2 y 2.2**. La próxima vez que accedamos a este restaurante, el solomillo marcará una puntuación de **6**.

Después de que el usuario haya **interactuado** en varias ocasiones con la aplicación y haya seleccionado varias tapas en varios restaurantes podemos encontrarnos con el problema de tener tapas con valores **mayores que 10**. En este punto, se decidió situar siempre las puntuaciones en una escala de **0 a 10**. De tal modo, que a medida que vayamos escogiendo algunas tapas se va a ir dando **mayor** peso a unos **ingredientes** y menos a otros. En la hipótesis de que tengamos puntuaciones mayores que 10 ejecutaremos el método que se explicará a continuación. Este método comprueba si disponemos de tapas con valores mayores que 10. En caso afirmativo averigua cual es la mayor puntuación de los platos para después cambiar las puntuaciones de las tapas. Este proceso se realiza simplemente con una regla de 3.

```
    public List<Row> tapasOrdenadas () {
        List<String> tapas = new ArrayList<String>();
        rows = new ArrayList<Row>();
        String nombreRest = restaurante;

        if (comprobarTapaMayor10()) {
            float mayor = obtenerMayorPuntuacion();
            setearPuntuaciones(mayor);
        }

        for (int i = 0; i < listaOrdenada.size(); i++) {


if(listaOrdenada.get(i).getNombreRestaurante().compareTo(restaurante
) == 0) {

                if (listaOrdenada.get(i).getPuntuacion() > 0) {
                    Row r = new
```

```
Row(listaOrdenada.get(i).getNombreTapa(), "Válido para su consumo.
Puntuación:" + listaOrdenada.get(i).getPuntuacion(), false);
                    rows.add(r);
                } else {
                    listaOrdenada.get(i).setPuntuacion(0);
                    Row r = new
Row(listaOrdenada.get(i).getNombreTapa(), "No apta para su consumo.
Puntuación:" + listaOrdenada.get(i).getPuntuacion(), false);

                    rows.add(r);
                }
            }
        }
return rows;
    }
```

En el siguiente ejemplo se podrá ver con detalle todo este proceso de regenerar las puntuaciones en una escala de 0 a 10.

LISTA DE TAPAS SIN CAMBIAR SUS PUNTUACIONES:

- **Solomillo al roquefort**. PUNTUACION → 24
- **Jamón Ibérico de bellota**. PUNTUACION → 12
- **Sepia a la plancha**. PUNTUACION → 8

Al aplicar a las siguientes tapas el método anterior, se genera la siguiente lista con sus puntuaciones alteradas:

- **Solomillo al roquefort**. PUNTUACIÓN → 10, ya que es la tapa con mayor puntuación de la lista.
- **Jamón ibérico de bellota**. PUNTUACIÓN → 12*10/ 24 → 5.
- **Sepia a la plancha**. PUNTUACION → 8*10/ 24 → 3.

7.4.6 CASO DE USO: TAPAS CON TONOS ROJIZOS

Respecto al pintar de un **color** u **otro** el texto de la lista de tapas (ListView), dependiendo de si la tapa contiene ingredientes alérgicos o no, ha resultado un proceso difícil de realizar. Esto es, debido a que se tiene que cambiar solo el texto de algunas filas de la tabla (aquellas cuya tapa sea **alérgico**). Finalmente y después de seguir investigando se decidió crear un **ArrayAdapter** en una clase externa que contendrá la lista de tapas. Esta clase que extendía de ArrayAdapter presenta un método propio llamado ***getView***. En este método fue donde aplicamos el proceso de cambiar el **color** de texto en aquellas tapas que seamos **alérgicos**. A continuación, podréis ver con más detalle la implementación de este método. En **negrita** podemos observar la parte del código que realiza este proceso. Si el texto asociado a la tapa comienza con la frase "**no apta**", sabemos que se trata de una tapa alérgica, por lo que obtenemos la vista de ese texto y cambiamos su color a tonos rojizos.

```
@Override
public View getView(int position, View convertView, ViewGroup
parent)
{
    // holder pattern
    Holder holder = null;
    if (convertView == null)
    {
        holder = new Holder();

        convertView = layoutInflater.inflate(R.layout.listview_row,
null);
        holder.setTextViewTitle((TextView)
convertView.findViewById(R.id.textViewTitle));
        holder.setTextViewSubtitle((TextView)
convertView.findViewById(R.id.textViewSubtitle));
        convertView.setTag(holder);
    }
    else
    {
        holder = (Holder) convertView.getTag();
    }

    Row row = getItem(position);
    holder.getTextViewTitle().setText(row.getNombreTapa());
    holder.getTextViewSubtitle().setText(row.getPuntuacion());
    if(row.getPuntuacion().startsWith("No apta")){
       // t.setText("hola",TextView.BufferType.EDITABLE);
        holder.getTextViewSubtitle().setTextColor(Color.RED);
    }else{
        holder.getTextViewSubtitle().setTextColor(Color.WHITE);

    }
   return convertView;
}
```

7.5 ARTEFACTOS GENERADOS

Para este capítulo, vamos a ver cuáles han sido los diferentes artefactos generados en las distintas releases(lanzamientos) a lo largo del desarrollo generado.

RELEASE 1

En la primera toma de contacto con la aplicación, se empezó con el registro. En concreto, se realizó, todo lo necesario para almacenar la información personal y culinaria del usuario. Con ello, ya tendríamos una primera aproximación genérica acerca de los gustos del usuario.

RELEASE 2

Seguidamente, se gestionó, que los datos personales introducidos anteriormente coincidiesen con los campos introducidos en el Login. Una vez estos campos fuesen validados, podríamos observar el listado de restaurantes.

RELEASE 3

El tercer lanzamiento sería añadir nuevas funcionalidades a lo anterior, de modo, que al seleccionar un restaurante viésemos sus platos o tapas con sus correspondientes puntuaciones.

RELEASE 4

En cuarto lugar, gestionamos la posibilidad de poder seleccionar una o varias tapas y que este proceso actualizase la información culinaria del usuario.

RELEASE 5

Sucesivamente, continuamos con el proceso de intercambiar preferencias culinarias entre los diferentes comensales.

RELEASE 6

En esta reléase, la aplicación además, sería capaz de mostrar un nuevo listado de tapas con los datos de todos los comensales que hubiesen intercambiado sus preferencias.

RELEASE 7

Para el último lanzamiento se estuvo perfeccionando generalmente la aplicación. En concreto, se modificaron aspectos visuales, se añadió una pantalla de splash(aquella pantalla que se muestra antes de iniciar la aplicación) y se cambió el color de las tapas alérgicas a tonos rojizos.

8. MANUAL DE USUARIO

En este apartado se va a explicar qué **posibilidades** ofrece esta aplicación y cómo ha de *utilizarse*. Lo primero a tener en cuenta es que esta aplicación puede ser utilizada sin *conexión a internet*. Además, se debe tener en cuenta que se puede usar entre **varios dispositivos**.

8.1 LOGIN Y REGISTRO

En la hipótesis de que el usuario acceda a la aplicación, verá la pantalla de **login**, donde se le solicitaran sus credenciales de acceso una vez, se haya cargado la pantalla de **splash** (pantalla inicial mientras se cargan los datos). En el caso de que el usuario acceda por primera vez a la aplicación, obviamente no tendrá credenciales de acceso por lo que deberá **registrarse**. Por contrario, si posee credenciales debido a que ya ha realizado el registro podrá acceder de forma habitual.

A continuación se va a mostrar una captura de pantalla sobre como el usuario vería la pantalla de **login**. Como podemos observar, sólo se le solicita un nombre de **usuario** y una **contraseña**. También podemos observar *2 botones*, uno para aquellos que posean credenciales y que ya se hayan registrado y otros para los que no las posean y deseen registrarse.

Ilustración 36- Login

En la siguiente imagen se muestra la pantalla de registro de un nuevo usuario (**datos personales**). En esta pantalla el usuario debe escribir cuales van a ser sus datos personales en la aplicación. Para **continuar** el proceso de registro, el usuario tendrá que pulsar sobre el icono en forma de plato que pone "**siguiente**". En la hipótesis que tengamos un error en el formulario, seguiremos en la misma página. Para cancelar la operación de registro simplemente pulsamos en "**volver**".

Ilustración 37- Registro datos personales

El proceso seguidor de registro va a ser introducir información genérica acerca de los datos culinarios. Para ello, el usuario deberá llevar a cabo un registro sobre sus preferencias culinarias. Simplemente deberá de ir puntuando mediante **estrellas** cada alimento. Para continuar el proceso de registro el usuario deberá pulsar en siguiente. Para volver a la anterior página de registro pulsaremos en "**volver**".

Ilustración 38- Registro datos culinarios

Para finalizar el registro, el usuario deberá seleccionar aquellos alimentos o grupos de alimentos a los que es **alérgico** y pulsar sobre el botón "**siguiente**". Para volver al proceso de registro anterior, el usuario deberá pulsar en "**volver**".

Ilustración 39- Registro datos alérgicos

8.2 LISTADO DE RESTAURANTES

Una vez finalizado el registro del usuario, podremos acceder a ver el **listado de restaurantes** disponibles. Está página va a ser la página de redirección una vez iniciemos sesión en la aplicación. En ella, el usuario podrá observar un listado de todos los restaurantes existentes en la aplicación. Para ver el listado de tapas de cada restaurante, el usuario deberá pulsar sobre el restaurante que desee. Si por el contrario, deseamos salir de la aplicación, deberemos pulsar sobre el botón de salir.

Ilustración 40- Listado de restaurantes

Cuando un usuario se decida a ver un **restaurante** se le mostrará un mensaje de **alerta** como el que se muestra a continuación, informándole sobre que desea realizar. En la hipótesis de que pulsemos en el botón "**cancelar**" el usuario verá el listado de tapas o platos de ese restaurante. Caso opuesto, el usuario accederá a una nueva pantalla en la que podrá compartir sus datos culinarios con el resto de compañeros.

Ilustración 41- Aviso preferencias

8.3 INTERCAMBIO DE DATOS

En el momento en que el usuario acceda a la pantalla de intercambio de datos culinarios, al principio, verá una lista vacía, ya que todavía no le ha dado tiempo a la aplicación a buscar dispositivos que tengan activado la conectividad **Wifi Direct**. En el momento de que ambos dispositivos se encuentren mutuamente cualquiera de los dos va a poder **iniciar** la conexión y **enviar** los datos simplemente con pulsar sobre él, en la lista. Por cada usuario que hayamos encontrado vamos a poder ver su alias y su dirección **MAC**. En las siguientes imágenes veremos un ejemplo de **2 dispositivos** que se han encontrado mutuamente y que están listos para que cualquiera de los dos inicie la conexión.

DISPOSITIVO 1

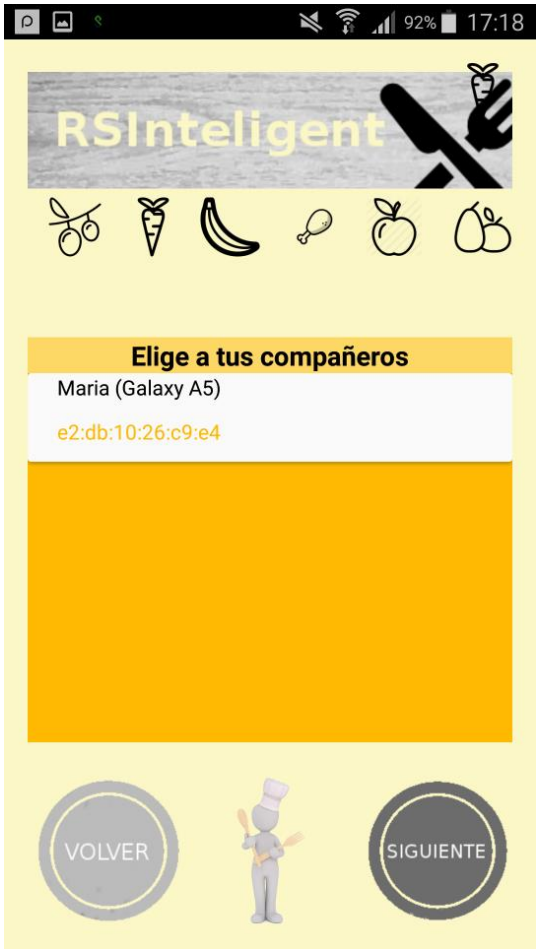

Ilustración 43- Intercambio de datos (1).

DISPOSITIVO 2

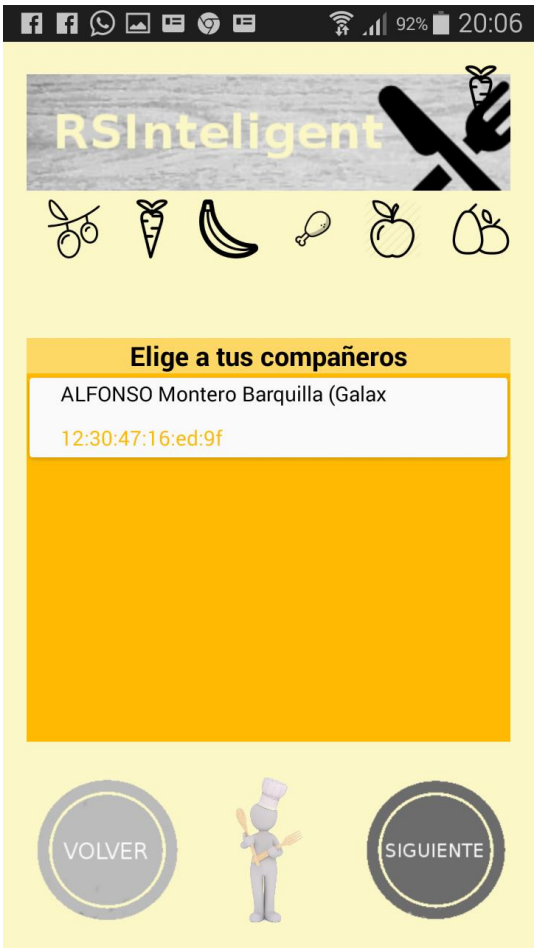

Ilustración 42- Intercambio de datos (2).

Cuando pulsemos sobre el dispositivo al que queremos conectarnos, le saltará un mensaje al dispositivo receptor para verificar si desea conectarse con el dispositivo emisor. En este caso, lanzaremos una conexión desde el dispositivo 2. En caso de aceptar, recibirá los datos culinarios del dispositivo emisor y se hará visible un botón para efectuar el envío de datos al dispositivo emisor.

DISPOSITIVO 1

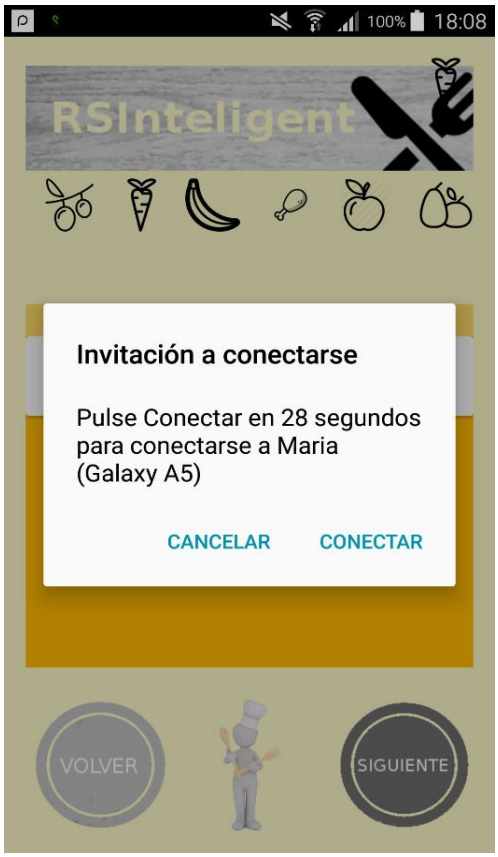

Ilustración 44- Intercambio de datos (3).

DISPOSITIVO 2

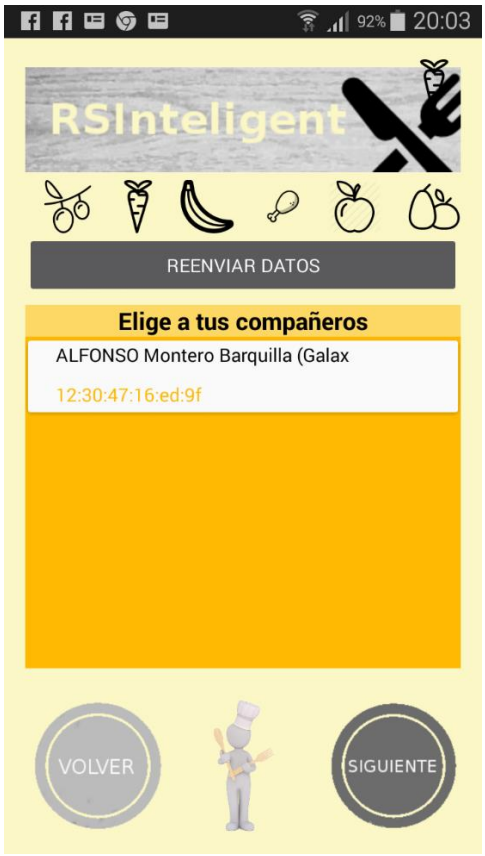

Ilustración 42- Intercambio de datos (4).

Finalmente, cuando el usuario decida que ya ha acabado el compartimento de datos, podrá pulsar sobre el botón siguiente para poder ver el **listado de tapas** con sus preferencias y la de sus compañeros. Obviamente, en caso de que todas las personas compartan datos entre sí, deberán tener las mismas puntuaciones en las tapas. Si deseamos volver a ver el listado de restaurantes, deberemos de hacer **clic** en el botón "**volver**".

8.4 GESTIÓN DE PREFERENCIAS

Para finalizar con el **manual de usuario** veremos la página final de la aplicación. Esta página posee el ***listado de tapas o platos*** del restaurante. Cada tapa va a poseer una **puntuación** y la **verificación** de si es apta o no para nuestro consumo. El usuario va a poder elegir cualquiera de las tapas existentes, incluidas las alérgicas, simplemente deberá pulsar en el **checkbox** posicionado a la izquierda de la tapa. Únicamente nos saldrá un mensaje de aviso en caso de elegir una tapa a la que somos alérgicos. Para confirmar el pedido nada más que tendremos que pulsar sobre el botón de ***confirmar***. Cada vez que seleccionemos tapas y la confirmemos van a ir **cambiando** las puntuaciones y más datos contendrá la aplicación sobre nuestros gustos culinarios.

Ilustración 43- Listado de tapas

9. CONCLUSIONES

En base a todo lo detallado anteriormente, se procede a comentar los resultados obtenidos durante las **etapas** realizadas a lo largo del desarrollo del **presente** proyecto. Se recalca el **éxito** total en lo referente al cumplimiento de los objetivos finales marcados a conseguir durante todo el desarrollo del proyecto. De los objetivos establecidos al inicio del proyecto, este sería el resultado después de terminar el proyecto:

- **Conseguir que el coste de desarrollo sea lo más bajo posible** → La solución final del proyecto es lo más económico posible, ya que no necesitamos ningún tipo de gasto extra. Únicamente necesitamos un dispositivo móvil con conectividad Wifi Direct
- **Hacer posible una funcionalidad sin conexión a internet (modo offline)** → Este objetivo ha sido conseguido debido al uso, entre otras cosas, de SQLite ya que no necesita conexión a internet para almacenar los datos internamente. Si hubiésemos utilizado alguna otra tecnología con almacenamiento en la nube como Firebase, este objetivo no se podría haber cumplido.
- **Conseguir intercambiar nuestras preferencias culinarias con las de nuestros compañeros y viceversa** → Este propósito es uno de los más importantes de la aplicación a nivel social, que fue logrado satisfactoriamente gracias a la conectividad Wifi Direct.
- **Evitar problemas alérgicos** → Es un objetivo muy importante y que cumplimos ya que no queremos que el usuario se intoxique alimentariamente pudiéndolo evitar. Este objetivo se controla por un lado con un formulario de alergias. Por otro, con avisos cuando escojamos un alimento al que somos alérgicos o visualizando las tapas alérgicas en color rojo
- **Evitar situaciones de incertidumbre**. → Otro de los objetivos más importantes ya que es uno de los pilares de la aplicación. Obviamente cumplimos este objetivo debido a la gestión que hacemos sobre las preferencias culinarias del usuario.
- **En general, mejorar la vida de las personas** → En cierto aspecto, somos responsables de mejorar la vida de las personas ya que evitamos al usuario situaciones incomodas de incertidumbre y posibles intoxicaciones alimentarias.
- **Investigar y entender la conectividad Wifi Direct** → Este último objetivo se cumplió con creces ya que esta tecnología se ha estado investigando durante todo el desarrollo del proyecto. Además, después de llevar a cabo esta investigación, se consiguió entender más esta conectividad.

En mi opinión los objetivos establecidos al inicio del proyecto se han cumplido, aunque durante la realización del proyecto fueron surgiendo nuevas ideas que se mostraran en el siguiente punto para futuras ampliaciones del proyecto.

Generalmente estoy satisfecho con el resultado del proyecto ya que he podido aprender distintas tecnologías conocidas pero no utilizadas y he aprendido numerosas utilidades que no conocía.

10. POSIBLES AMPLIACIONES

A continuación, se va a hacer una propuesta de una **serie de ampliaciones** que se podrían llevar a cabo en un **futuro**, en este proyecto. Los diferentes **apartados** o **mejoras** han ido surgiendo mientras se ha ido desarrollando el proyecto y principalmente no se han podido llevar a cabo por una cuestión de tiempo. Además, se proponen posibles mejoras sobre la implementación actual del proyecto.

10.1 IDENTIFICACIÓN DE COMPAÑEROS

Ilustración 44- Beacon [54]

Gracias a la tecnología Beacon, vamos a proponer un cambio futuro para el desarrollo de la **identificación de compañeros**. Esta tecnología ya ha sido investigada para el **intercambio** de datos culinarios entre comensales.

Desde play store podemos descargarnos una aplicación denominada **Locate Beacon** en la que podemos hacer pruebas de funcionalidad antes de ponernos a implementar su **API**. Hay que tener en cuenta que esta API, que posee la familia de código abierto no va a ser posible de implementar en todos los dispositivos móviles debido a problemas de versiones. Para tener más información acerca de esta API podréis acceder a su página oficial [30].

Más allá de haber usado o no esta **tecnología** en el proyecto, durante el desarrollo del mismo nos surgió la siguiente duda: imaginemos un **restaurante** con varias **mesas** todas repletas de comensales que utilizan la aplicación. En la página para intercambio de información culinaria seguramente nos saldrían todas las personas que están usando la aplicación. De alguna manera, habría que acotar las personas que nos aparecen y poner solamente aquellas que se encuentren comiendo con nosotros.

En este momento es cuando podríamos hacer uso de la API de Beacons ya que deberíamos comprobar a que distancia se encuentra cada dispositivo. En la hipótesis de encontrarse a menor distancia que una determinada, sabemos que está comiendo

con ***nosotros*** y por lo tanto, dejaríamos visualizarlo en el listado de compañeros. Por el contrario, si las distancias entre dispositivos son **mayores** a una distancia determinada sabemos que esta persona no está en nuestra mesa.

Respecto a la distancia entre comensales, también se pensó en la posición **GPS,** para saber que dispositivos están cerca unos del otro. Obviamente este proceso se rechazó debido a que la posición GPS tiene un gran margen de error en dispositivos cerrados y no estarían siendo cálculos positivos.

Con el uso de Beacons, también surgen nuevas ampliaciones como son las **notificaciones Push**. Estaría bastante bien, que cuando nos acerquemos a un restaurante nos saliese una notificación sobre ese restaurante en nuestro móvil. Este desarrollo habría que hacerlo mediante el dispositivo físico Beacons o mediante la geolocalización GPS.

10.2 LISTADO DE RESTAURANTES

Ilustración 45- Listado de restaurantes

Cuando iniciamos sesión en la aplicación nos aparece un **listado** de todos los restaurantes que posee la aplicación para elegir donde queremos **comer** y que queremos **pedir**. Esto es uno de los desarrollos que se pueden mejorar en diversos aspectos.

Imaginemos que poseemos miles de restaurantes en el listado y queremos encontrar uno que se encuentra en la mitad de la lista. Tardaríamos bastante tiempo en encontrarlo. Por lo que como posible mejora se haría una **búsqueda** de restaurantes por nombre.

Otra cuestión interesante seria hacer buen uso de la **geolocalización** GPS y mostrar restaurantes que se encuentren en un **radio** de un rango que hayamos introducido previamente. En este contexto podríamos usar también **Beacons** ya que se podría cumplir con la misma funcionalidad.

En general, para esta página se intenta realizar una mejora de sus funcionalidades. Simplemente carga todos los restaurantes directamente y los muestra. No estaría nada mal implementar **búsquedas** o hacer uso de la geolocalización GPS para filtrar los resultados.

10.3 ALMACENAMIENTO

Ilustración 46- Almacenamiento [56]

Uno de los problemas que posee la aplicación es **como** almacenar la información. En la aplicación se han usado **tecnologías** y se han llevado a cabo **decisiones sencillas** para centrarnos más en la gestión de preferencias y en el intercambio de información culinaria.

Los datos referentes a cada restaurante con sus tapas e ingredientes están almacenados en un archivo “txt” incrustado en la aplicación. Por otro lado, los datos personales, culinarios o alérgicos del usuario se encuentran almacenados en la memoria interna del teléfono.

Sería conveniente que todos los datos se encuentren en la nube o en una base de datos bien estructurada a la que podamos acceder en cualquier momento y desde

cualquier lugar. Este podría ser el caso de utilizar Firebase o alguna base de datos NoSQL que les tengo bastante estima en los tiempos que acontecen.

Una de las ventajas que posee el utilizar SQLite es, que no necesitamos tener conexión a internet. A su vez también implica perder los datos si cambiamos de teléfono.

10.4 FORMULARIO DE ALERGIAS

Ilustración 47- Datos alérgicos.

En la anterior imagen nos encontramos en la **última** página de registro de un usuario en la aplicación. Como podemos observar hemos elegido que somos alérgicos a la **leche**, a la **soja** y a los **frutos secos**. Ahora bien, que sucedería si somos alérgicos a algún alimento que ahí no se contempla. En este contexto es cuando vienen los problemas ya que el usuario va a poder consumir algún alimento siendo alérgico.

Una de las mejoras que podríamos llevar a cabo en este contexto es un espacio de alimentación más grande. En la imagen solo contemplamos 8 alimentos que son los más genéricos en intoxicaciones alimentarias, pero siempre existen excepciones.

Como solución a este problema se ha pensado en que el usuario pueda introducir el nombre de los alimentos a los que es alérgico. De tal modo, se generaría una lista de alimentos alérgicos que tendríamos que añadir a la información alérgica del usuario.

10.5 DISEÑO E IMPLEMENTACIÓN

Respecto al diseño de la aplicación, se podría mejorar algo, aunque como está realizada me resulta bastante llamativa.

Uno de los aspectos de diseño que si sería mejorable sería el llamado **responsive** (una técnica de diseño web que busca la correcta visualización de una misma página en distintos dispositivos. Desde ordenadores de escritorio a tablets y móviles). Aunque la aplicación se ve bien en los dispositivos mayor demandados habría que pensar en otro tipo de dispositivos como tablets o en aquellos que posean un tamaño considerablemente diferente.

El desarrollo del proyecto se ha realizado con la herramienta Android Studio. Esta herramienta solo desarrolla aplicaciones para dispositivos Android. En el caso de tener un dispositivo IOS no podríamos usar la aplicación. Por lo tanto, una de las mejoras que podríamos hacer es desarrollar el proyecto mediante el framework Ionic 2 basado en Angular 2 que crea una aplicación móvil para cualquier plataforma(window phone,ios, android,…). Para más información acerca de este framework podéis visitar su página oficial [31].

10.6 INFORMACIÓN EN LAS TAPAS

Para finalizar el apartado de posibles ampliaciones destacar una característica que posee la mayoría de aplicaciones sobre restaurantes o comida, como es la muestra de **fotos** u **opiniones** sobre los platos o tapas que posee un restaurante. Sería muy interesante que se pudiese llevar a cabo esta ampliación ya que **ayudaría** al consumidor en su decisión de platos.

Ilustración 48- Ejemplo de tapas con foto y comentarios [57]

___11. CONCLUSIÓN PERSONAL___

Personalmente, este proyecto ha significado un **gran reto** para mi debido entre otros motivos, a que la idea ha sido mía y a que estuve **trabajando** mientras desarrollaba parte del proyecto.

Adicionalmente, a pesar de haber cursado una asignatura como **ASEE** en la que hemos estudiado **Android Studio**, en este proyecto se tocan algunos aspectos que no se han cursado en la asignatura, como es trabajar con **Wifi Direct**.

Antes de saber definitivamente que iba a realizar este proyecto, mi objetivo era que el proyecto que realizase fuese lo más **innovador** posible y que tuviese la posibilidad de **emprender** una carrera empresarial con él. Uno de los objetivos se ha cumplido con creces ya que este proyecto no tiene competidores actuales con ideas parecidas. En cambio respecto a la posibilidad de emprender una carrera profesional con él, se podría llevar a cabo después de trabajar mucho sobre él y de mejorar algunos aspectos. Quizás en un futuro pueda dedicarle el tiempo que necesita para emprender con él un futuro profesional.

Otro de los aspectos con lo que me quedo después de realizar el proyecto y de acabar la carrera es enseñarme a buscar en internet. **Google** es muy sabio y como tal hay que saber buscar en él, para que el resultado devuelto sea lo más óptimo posible.

No puedo acabar mi conclusión sin agradecer su tiempo y ayuda a mi tutor en este proyecto, **Javier Berrocal**, que ha estado siempre atento durante el desarrollo y me ha servido de gran ayuda para llevar a cabo el proyecto.

10. BIBLIOGRAFÍA

1. Bluetooth Low Energy <http://wiki.makespacemadrid.org/index.php?title=Bluetooth_Low_Energy>[Disponible a 9 de Septiembre de 2017].

2. ¿Qué es Android?<https://www.xatakandroid.com/sistema-operativo/que-es-android>[Disponible a 9 de Septiembre de 2017].

3. ¿Por qué Android?<http://www.abc.es/tecnologia/informatica/software/abci-google-android-sistemas-operativos-201604202207_noticia.html>[Disponible a 10 de Septiembre de 2017].

4. Developer Android<https://developer.android.com/guide/platform/index.html>[Disponible a 10 de Septiembre de 2017].

5. Acerca de Android Studio<https://developer.android.com/studio/intro/index.html?hl=es-419>[Disponible a 10 de Septiembre de 2017].

6. Wifi Direct VS Bluetooth<https://elandroidelibre.elespanol.com/2016/10/wifi-direct-vs-bluetooth.html>[Disponible a 10 de Septiembre de 2017].

7. Características de Wifi Direct <http://www.abc.es/tecnologia/consultorio/20150212/abci-wifi-direct-como-usar-201502111739.html>[Disponible a 10 de Septiembre de 2017].

8. Librería Android para Beacon <https://altbeacon.github.io/android-beacon-library/> [Disponible a 30 de Septiembre de 2017].

9. API para desarrollo Wifi Direct <https://developer.android.com/guide/topics/connectivity/wifip2p.html> [Disponible a 30 de Septiembre de 2017].

10. BLE <http://www.areatecnologia.com/nuevas-tecnologias/bluetooth-le.html> [Disponible a 30 de Septiembre de 2017].

11. LTE Direct<https://www.xataka.com/otros/lte-direct-la-futura-conexion-movil-a-movil-que-no-pasa-por-las-antenas-de-telefonia> [Disponible a 30 de Septiembre de 2017].

12. Wifi Aware<http://computerhoy.com/noticias/internet/wi-fi-aware-nuevo-wi-fi-internet-31615> [Disponible a 30 de Septiembre de 2017].

13. SQLite<http://www.sgoliver.net/blog/bases-de-datos-en-android-i-primeros-pasos/> [Disponible a 30 de Septiembre de 2017].

14. Layout Android Studio <http://www.hermosaprogramacion.com/2014/09/android-layouts-views/> [Disponible a 30 de Septiembre de 2017].

15. Android y SQLite<http://blog.saducelabs.com/android/android-y-sqlite/> [Disponible a 30 de Septiembre de 2017].

16. Yelp<http://computerhoy.com/paso-a-paso/apps/que-es-yelp-asi-funciona-app-compartir-experiencias-22281> [Disponible a 3 de Octubre de 2017].

17. ElTenedor<https://play.google.com/store/apps/details?id=com.lafourchette.lafourchette&hl=es> [Disponible a 3 de Octubre de 2017].

18. Atrápalo_Restaurantes<https://play.google.com/store/apps/details?id=com.atrapalo.restaurantes>[Disponible a 3 de Octubre de 2017].

19. BurguerKing<https://play.google.com/store/apps/details?id=es.burgerking.android&hl=es> [Disponible a 3 de Octubre de 2017].

20. Wifi Direct <http://www.compartirwifi.com/blog/wifi-direct-para-que-sirve/> [Disponible a 3 de Octubre de 2017].

21. SQLiteOpenHelper <https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html> [Disponible a 3 de Octubre de 2017].

22. AdapterView.OnItemClickListener<https://developer.android.com/reference/android/widget/AdapterView.OnItemClickListener.html> [Disponible a 5 de Octubre de 2017].

23. AdapterView<https://code.tutsplus.com/es/tutorials/android-from-scratch-understanding-adapters-and-adapter-views--cms-26646> [Disponible a 5 de Octubre de 2017].

24. IntentService<https://developer.android.com/reference/android/app/IntentService.html> [Disponible a 3 de Octubre de 2017].

25. RecyclerView<https://developer.android.com/reference/android/support/v7/widget/RecyclerView.Adapter.html> [Disponible a 3 de Octubre de 2017].

26. Tareas asíncronas<http://www.sgoliver.net/blog/tareas-en-segundo-plano-en-android-ii-intentservice/> [Disponible a 3 de Octubre de 2017].

27. BroadcastReceiver<http://androcode.es/2012/11/android-services-24-broadcastreceiver/> [Disponible a 3 de Octubre de 2017].

28. ListView<http://www.androidcurso.com/index.php/tutoriales-android/34-unidad-3-actividades-e-intenciones/128-la-vista-listview> [Disponible a 3 de Octubre de 2017].

29. LTE Direct web oficial <https://www.qualcomm.com/invention/research/projects/lte-direct> [Disponible a 3 de Octubre de 2017].

30. Librería Beacon <https://altbeacon.github.io/android-beacon-library/> [Disponible a 5 de Octubre de 2017].

31. Framework Ionic 2 <https://ionicframework.com/docs/> [Disponible a 5 de Octubre de 2017].

32. Ilustración 1-Yelp <https://play.google.com/store/apps/details?id=com.yelp.android>[Disponible a 15 de Octubre de 2017].

33. Ilustración 2-El Tenedor <http://www.gastroeconomy.com/2012/09/marcos-alves-para-ser-emprendedor-hay-que-sonar/>[Disponible a 15 de Octubre de 2017].

34. Ilustración 3-Guía Michelín. <https://gastronomiaycia.republica.com/2016/11/27/app-de-la-guia-michelin-espana-y-portugal-2017/>[Disponible a 15 de Octubre de 2017].

35. Ilustración 4- Atrapalo Restaurantes<http://www.smartblog.es/2013/05/no-sabes-donde-comer-hoy-atrapalo-restaurantes-te-lo-pone-facil/>[Disponible a 15 de Octubre de 2017].

36. Ilustración 5- Burguer King <https://play.google.com/store/apps/details?id=es.burgerking.android>[Disponible a 15 de Octubre de 2017].

37. Ilustración 6- Bluetooth Low Energy <http://www.exuberantsolutions.com/bluetooth-low-energy-training.htm>[Disponible a 18 de Octubre de 2017].

38. Ilustración 7- Wifi Direct<http://www.lg.com/pe/microsites/Wi-FI-Direct>[Disponible a 18 de Octubre de 2017].

39. Ilustración 8- Lte Direct <https://www.xataka.com/otros/lte-direct-la-futura-conexion-movil-a-movil-que-no-pasa-por-las-antenas-de-telefonia>[Disponible a 18 de Octubre de 2017].

40. Ilustración 9- Wifi Aware <https://www.slashgear.com/wi-fi-aware-your-router-knows-youre-near-14392955/>[Disponible a 18 de Octubre de 2017].

41. Ilustración 10- Almacenamiento de datos <https://www.smechannels.com/study-70-percent-of-global-companies-wary-about-complexity-of-data-storage/>[Disponible a 18 de Octubre de 2017].

42. Ilustración 11- Gestión de preferencias <http://pedrocorvinosabogado.es/la-encomienda-de-gestion-como-tecnica-de-autoorganizacion-y-como-medio-de-contratacion-domestica/> [Disponible a 18 de Octubre de 2017].

43. Ilustración 12- Comparto de preferencias <https://www.google.es/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&ved=0ahUKEwi785Gv1fDWAhXLDxoKHaxQAGsQjxwIAw&url=http%3A%2F%2Fwww.bcgsearch.com%2Farticle%2F60335%2F10-ways-to-reciprocate-in-networking%2F&psig=AOvVaw0R8kBaQ8PCPdNevlhEo8YR&ust=1508089499460839>[Disponible a 18 de Octubre de 2017].

44. Ilustración 13- Android <https://www.taringa.net/posts/celulares/19354810/Las-5-mejores-aplicaciones-de-camara-para-Android.html>[Disponible a 20 de Octubre de 2017].

45. Ilustración 14- Grafica de uso Android <http://www.abc.es/tecnologia/informatica/software/abci-google-android-sistemas-operativos-201604202207_noticia.html>[Disponible a 20 de Octubre de 2017].

46. Ilustración 15- Arquitectura de la plataforma Android <https://developer.android.com/guide/platform/index.html>[Disponible a 20 de Octubre de 2017].

47. Ilustración 16- Android Studio < https://www.androidauthority.com/android-studio-tutorial-beginners-637572/>[Disponible a 20 de Octubre de 2017].

48. Ilustración 17- Estructura de un proyecto Android Studio <https://developer.android.com/studio/intro/index.html?hl=es-419>[Disponible a 20 de Octubre de 2017].

49. Ilustración 19- Esquema Wifi Direct <https://elandroidelibre.elespanol.com/2016/10/wifi-direct-vs-bluetooth.html>[Disponible a 20 de Octubre de 2017].

50. Ilustración 20- Wifi Direct vs BLE <https://elandroidelibre.elespanol.com/2016/10/wifi-direct-vs-bluetooth.html>[Disponible a 20 de Octubre de 2017].

51. Ilustración 21- Como usar Wifi Direct <http://www.compartirwifi.com/blog/wifi-direct-para-que-sirve/>[Disponible a 20 de Octubre de 2017].

52. Ilustración 22- SQLite < https://en.wikipedia.org/wiki/SQLite>[Disponible a 20 de Octubre de 2017].

53. Ilustración 31- Diseño en Android Studio <https://en.wikipedia.org/wiki/SQLite>[Disponible a 20 de Octubre de 2017].

54. Ilustración 32- Gimp < https://www.gimp.org/>[Disponible a 20 de Octubre de 2017].

55. Ilustración 45- Beacon < https://github.com/mmazzarolo/react-native-beacons-android>[Disponible a 20 de Octubre de 2017].

56. Ilustración 47- Almacenamiento < http://www.freepngimg.com/png/11641-database-free-download-png>[Disponible a 20 de Octubre de 2017].

57. Ilustración 49- Ejemplo de tapas con foto y comentarios <http://www.compudemano.com/noticias/discusion/aplicaciones-para-cenar-y-chuparse-los-dedos-hoy-atrapalo-restaurantes.69358/>[Disponible a 20 de Octubre de 2017].

58. Ilustración 50- Áreas lógicas en Android Studio <https://developer.android.com/studio/intro/index.html?hl=es419#interfaz_de_usuario> [Disponible a 20 de Octubre de 2017].

___ANEXOS___

ÁREAS LÓGICAS EN ANDROID STUDIO

A continuación, veremos la ventana principal de Android Studio que consta de varias áreas lógicas que se identifican en la siguiente figura. Seguidamente se explicarán cada una de estas áreas lógicas y sus funciones.

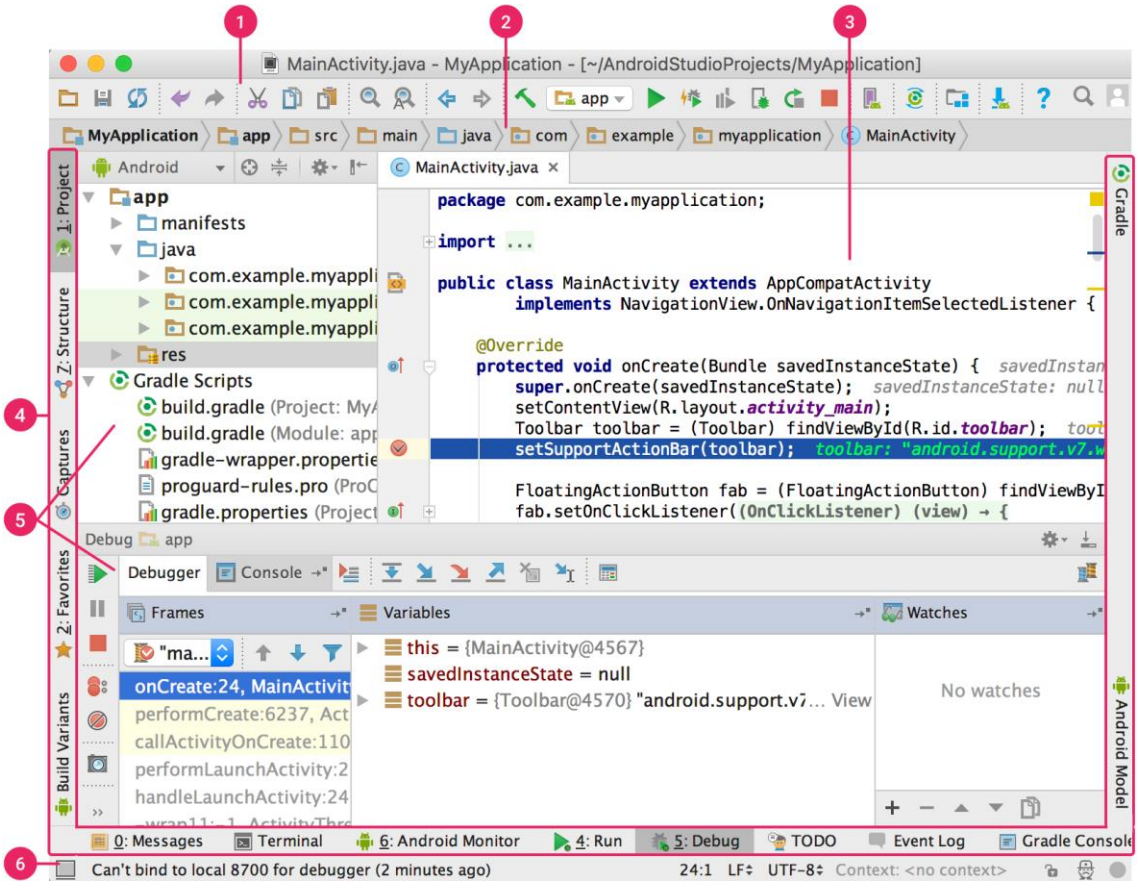

Ilustración 45- Interfaz Android Studio [58]

1. La barra de herramientas te permite realizar una gran variedad de acciones, como la ejecución de tu app y el inicio de herramientas de Android.

2. La barra de navegación te ayuda a explorar tu proyecto y abrir archivos para editar. Proporciona una vista más compacta de la estructura visible en la ventana Project.

3. La ventana del editor es el área donde puedes crear y modificar código. Según el tipo de archivo actual, el editor puede cambiar. Por ejemplo, cuando se visualiza un archivo de diseño, el editor muestra el editor de diseño.

4. La barra de la ventana de herramientas se extiende alrededor de la parte externa de la ventana del IDE (Entorno de Desarrollo Integrado) y contiene los botones que te permiten expandir o contraer ventanas de herramientas individuales.

5. Las ventanas de herramientas te permiten acceder a tareas específicas, como la administración de proyectos, las búsquedas, los controles de versión, etc. Puedes expandirlas y contraerlas.

6. En la barra de estado, se muestra el estado de tu proyecto y del IDE en sí, como también cualquier advertencia o mensaje.

Puedes organizar la ventana principal para tener más espacio en pantalla ocultando o desplazando barras y ventanas de herramientas. También puedes usar combinaciones de teclas para acceder a la mayoría de las funciones.

En cualquier momento, puedes realizar búsquedas en tu código fuente, bases de datos, acciones, elementos de la interfaz de usuario, etc., presionando dos veces la tecla Shift o haciendo clic en la lupa que se encuentra en la esquina superior derecha de la ventana de Android Studio. Esto puede ser muy útil, por ejemplo, si intentas localizar una acción específica del IDE que olvidaste cómo activar.

FIN