ESCUELA SUPERIOR DE INGENIERÍA

INGENIERÍA TÉCNICA EN INFORMÁTICA DE SISTEMAS

SISTEMA MÓVIL INTERACTIVO PARA EL DESARROLLO DE LAS HABILIDADES CULINARIAS

JOSÉ MARÍA RODRÍGUEZ – IZQUIERDO FLORES

18 DE ABRIL DE 2016

Escuela Superior de Ingeniería

ESCUELA SUPERIOR DE INGENIERÍA

INGENIERÍA TÉCNICA EN INFORMÁTICA DE SISTEMAS

SISTEMA MÓVIL INTERACTIVO PARA EL DESARROLLO DE LAS HABILIDADES CULINARIAS

Departamento: Ingeniería Informática

Director del proyecto: Manuel Palomo Duarte

Autor del proyecto: José María Rodríguez – Izquierdo Flores

Puerto Real, 18 de abril de 2016

Fdo: José María Rodríguez – Izquierdo Flores

Agradecimientos

Quiero dar las gracias a mi familia y a toda la gente que me ha apoyado y que ha creído en mí. En especial, a mis verdaderos amigos, que han estado ahí en todo momento cuando he necesitado un consejo, una opinión o fuerzas para seguir adelante, y a mi hermano Agustín por el espectacular trabajo del diseño gráfico de la aplicación que ha desempeñado, sabiendo plasmar exactamente todo lo que yo tenía en mente.

Además, agradecer la dedicación y el empeño de los profesores que he tenido durante todos estos años por su afán de transmitirnos de la mejor forma posible todos los conocimientos necesarios para conseguir el título universitario y las aptitudes para continuar en la vida con la profesión de ingeniero técnico informático. Sobre todo, a mi tutor, Manuel Palomo, por interesarse desde un principio por mi idea, guiarme y animarme durante todo el tiempo que he estado realizando el proyecto.

Resumen

Las tecnologías móviles han evolucionado mucho en los últimos años, y con ellas, la necesidad de la sociedad de tener un mundo de posibilidades al alcance de su bolsillo en cualquier momento y lugar. Es tal este desarrollo, que en varias pulsaciones de pantalla tenemos millones de aplicaciones de todo tipo. Aprovechar esta coyuntura se convierte en un punto muy interesante y fuerte para la realización de un PFC.

Es un proyecto de aplicación móvil para dispositivos *Android*, con la intención de guiar y enseñar en la ejecución de recetas de cocina a usuarios con cualquier nivel de aptitud en el ámbito culinario. De manera gráfica e interactiva, al usuario se le mostrarán los pasos que conlleva la elaboración de un cierto plato, para aplicarlos de forma y en tiempo real en una cocina física, como haría un profesor humano.

La aplicación tendrá una modalidad del tipo juego-aventura, donde se comenzará desde lo más básico, como puede ser hacer una tortilla, y, consecutivamente, con la realización las fases (recetas), se irá avanzando en dificultad, añadiendo así una funcionalidad didáctica al proyecto.

El sistema dispondrá de un recetario clasificado mediante niveles de dificultad. Además de esta clasificación, existirán otras más, como por ejemplo tipos de recetas por alimentos (carne, pescado, verdura…), contenido de alérgenos, cálculo de calorías, tiempo de realización, etc.

Existen funcionalidades, además de animaciones gráficas de la interfaz, para hacer más atractiva la aplicación. Las dos principales son un cronómetro con alarma para poder llevar el tiempo de cocción de los alimentos de forma interactiva por el usuario. Y, la más importante, un sistema de TIPS (o consejos-trucos), que se irán desbloqueando a medida que se van realizando recetas al aplicarlos, y que quedan almacenados en una categoría especial para que se puedan consultar en cualquier momento, dando la posibilidad al usuario de usarlos a la hora de cocinar por su propia cuenta. Estos consejos, por poner ejemplos, son del tipo de “cómo saber que el aceite está listo para freír un alimento”, “formas de cortar un determinado vegetal”, “truco para hacer un buen empanado”, y demás sugerencias culinarias para agilizar y hacer más cómoda la experiencia y el aprendizaje.

Por último, hay un modo “batidora”, donde el usuario registrará una serie de ingredientes, y el programa le dará como resultado un conjunto de recetas aproximadas que podrían elaborarse con ellos.

El sistema se liberará bajo licencia libre.

Licencia

Notación y formato

Para la concreción de palabras técnicas, como pueden ser el nombre de una aplicación, funciones, rutas, bibliotecas, etcétera, la fuente irá en formato cursiva:

Android.

En el formato para citar líneas de código, se utilizará la fuente *Consolas* a tamaño 10 con la configuración de colores que viene predefinida en *Eclipse Luna*. A continuación expongo el código que viene de ejemplo en *Preferences -> Java -> Editor -> Syntax Coloring*:

```
/**
 * This is about <code>ClassName</code>.
 * {@link com.yourCompany.aPackage.Interface}
 * @author author
 * @deprecated use <code>OtherClass</code>
 */
public class ~~ClassName~~<E> extends AnyClass implements InterfaceName<String> {
	enum Color { RED, GREEN, BLUE };
	/* This comment may span multiple lines. */
	static Object staticField;
	// This comment may span only this line
	private E field;
	private AbstractClassName field2;
	// TASK: refactor
	@SuppressWarnings(value="all")
	public int foo(Integer parameter) {
		abstractMethod(inheritedField);
		int local= 42*hashCode();
		staticMethod();
		return bar(local) + parameter;
	}
}
```

Los enlaces a las fuentes o sitios web de interés irán en azul y subrayado:

http://albuche.blogspot.com.es/

Tabla de contenidos

Índice de figuras

Capítulo 1

Introducción

1.1 Motivación

Desde hace ya muchos años, la cocina es una disciplina que me ha llamado mucho la atención. Tras una temporada de autoaprendizaje y experimentación, decidí dar un paso más y cocinar en momentos puntuales para mis familiares y amigos, obteniendo valoraciones bastante positivas por su parte. Tal fue el interés de varios de mis allegados que me convencieron para crear un espacio donde difundir mis recetas mediante un blog llamado *¡Al Buche!* [1], usando la plataforma de *Blogger* de *Google*.

Con el apoyo de las redes sociales y el seguimiento de mis incondicionales, el blog cobró cierta relevancia en mi vida, llegando a tener una cantidad aceptable de seguidores en *Facebook*, aunque por falta de tiempo y por poca afinidad al formato acabé abandonándolo.

Cuando se me presentó la oportunidad de comenzar el PFC, mi principal motivación era pensar en algo que me hiciera sentir realizado por conocimientos por alcanzar, resultados palpables y por sintonía personal con la temática. Lo que pasó por mi cabeza fue hacer una aplicación móvil orientada a la cocina, visto el auge del sector de las tabletas y móviles inteligentes, la facilidad que ello te da para llegar a las personas y la importancia de adquirir nuevos conocimientos en tecnologías al alza. También, el vivir en un país puntero en el ámbito gastronómico, con la gran cantidad de programas de televisión sobre esta temática, hizo que la situación coyuntural, por el factor mediático y el tecnológico, reforzasen mi decisión de elegir este camino.

No me considero un maestro de la cocina, pero tengo las suficientes habilidades como para explicar de forma breve, fácil y clara cómo hay que proceder para realizar una cierta receta para que salga bien. Al fin y al cabo, la cocina y la programación no son disciplinas tan lejanas, puesto que la base de ambas son los algoritmos, como bien se puede apreciar en la definición de la RAE:

> "algoritmo: 1. m. Conjunto ordenado y finito de operaciones que permite hallar la solución de un problema."

A todo esto se le une que tanto para mí como para la gente interesada en mis recetas, sobre todo amigos que estudian en otras ciudades o que están independizados, con un nivel no muy destacable en habilidades culinarias, era la mejor vía para canalizar mis conocimientos y poder

compartirlos de forma sencilla con ellos, además de aumentar el alcance a cualquier otra persona interesada mediante *Google Play*. De este planteamiento concreté que la aplicación no sólo tuviera un objetivo divulgativo, sino que trasversalmente también pudiera ayudar a la gente a ir adquiriendo nuevas habilidades mediante consejos y trucos, además de una progresión de dificultad de menor a mayor, dotando al proyecto de un cariz educativo.

Tras concretar estas primeras ideas de lo que quería hacer, contacté con la Oficina de Software Libre de la universidad por si algún profesor estaba interesado en ser mi tutor, recibiendo varias respuestas orientativas sobre mi planteamiento. Finalmente obtuve el ofrecimiento de Manuel Palomo para ser mi guía durante el trascurso de mi proyecto, comentándome que la idea era de su agrado.

1.2 Objetivos

Explicado de forma básica, la finalidad principal es crear una aplicación móvil para *Android* que sea capaz de representar recetas tomadas de una base de datos. En profundidad, mi proyecto consta de dos propósitos más concretos: lo que quiero conseguir para mí mismo y lo que quiero aportar al resto del mundo.

Para mi persona, me gustaría obtener la suficiente cualificación y conocimientos como para que en un futuro pudiera dedicarme al desarrollo de aplicaciones móviles, ya fuera por cuenta propia o ajena. Me parece que esta vertiente de la informática tiene una cantidad inmensa de posibilidades a largo plazo.

En mi segundo propósito, lo que me interesa fundamentalmente es transmitir mis conocimientos culinarios de manera simple y clara a los usuarios de la futura aplicación mediante la descarga e instalación de la misma en sus dispositivos móviles *Android*. De manera trasversal al objetivo puramente divulgativo, surge también la posibilidad de ir mejorando la habilidad en la cocina mediante la diferenciación de niveles de dificultad y tiempo en cada receta y a base de pequeños consejos se irán introduciendo al usar la aplicación.

1.3 Estructura del documento

La estructura de este documento es la siguiente:

1. **Introducción:** motivaciones por las que comenzó este proyecto, objetivos que se quieren alcanzar y un resumen de la estructura del mismo.
2. **Descripción general:** descripción más concreta de los aspectos del proyecto y de las características básicas.
3. **Planificación:** explicación sobre cómo se ha planificado el proyecto y los diferentes cometidos desempeñados durante todo su desarrollo.
4. **Análisis:** periodo de análisis del sistema, empleando la metodología seleccionada, donde se reflejarán los requisitos funcionales del sistema, diagramas de caso de uso, diagramas de secuencia y los contratos de las operaciones.

5. **Diseño:** realización del diseño del sistema, diagramas de secuencia y descripción de las clases aplicadas al diseño.
6. **Implementación:** explicación de los aspectos más relevantes durante la fase de implementación del proyecto y problemas que han ido surgiendo durante este tramo.
7. **Pruebas:** diferentes comprobaciones llevadas a cabo en la aplicación, con la intención de verificar su correcto funcionamiento y el cumplimiento de las expectativas.
8. **Conclusiones:** interpretaciones obtenidas tras la consecución del proyecto.
9. **Apéndices:** explicación de herramientas usadas durante el desarrollo y manual de usuario para la instalación y uso de la aplicación final.
10. **Manual de usuario:** apartado donde se hallará la información para poder obtener y manejar la aplicación resultante.
11. **Bibliografía:** referencias a manuales, libros y documentos consultados durante todo el desarrollo del proyecto.
12. **Licencia *GNU GFDL*:** copia del texto completo de la licencia *GNU GFDL* en inglés.

1.4 Definiciones, acrónimos y abreviaturas

- ***Android*:** es un sistema operativo diseñado principalmente para dispositivos móviles como teléfonos inteligentes y tabletas, normalmente táctiles, aunque hoy en día también es funcional en relojes inteligentes, televisores y otros dispositivos electrónicos, y está basado en el núcleo *Linux*. Fue creado por la compañía *Android Inc.*, posteriormente adquirida por *Google*.
- **Dispositivo móvil:** aparato electrónico portátil capaz de procesar datos, conectarse a Internet, con memoria y con un cometido específico, pero con capacidad de para realizar otros trabajos más generales.
- **Smartphone:** en español, teléfono inteligente, es un dispositivo móvil de telefonía que, además de tener las características comunes de cualquier dispositivo telefónico portable, es capaz de cumplir tareas típicas de un ordenador personal mediante una buena capacidad de procesado, un tamaño de memoria considerable y una conectividad casi continua a Internet.
- **Tablet:** comúnmente llamado tableta en español, son dispositivos con características muy similares a los *Smartphones*, normalmente de mayor tamaño, con cualidades menos orientadas en el tema de telefonía y más encaminadas al uso como ordenador personal.
- **Google Play (Store):** es la plataforma de distribución en línea de aplicaciones de *Google* para dispositivos *Android*.
- **Software libre:** es la denominación de software que, por decisión del autor, puede ser copiado, estudiado, manipulado y redistribuido con o sin cambios. Su declaración va enlazada a la creación del *Proyecto GNU y de* la *Free Software Foundation*, encabezados por Richard Stallman. Se caracteriza por estar bajo una licencia del tipo *Copyleft*.

- **App:** es la abreviatura para referirse a aplicación móvil, en este caso basada en *Android*.
- **API:** corresponde a las siglas en inglés de *Application Programming Interface*, interfaz de programación de aplicaciones en español. Agrupado en una biblioteca, es el conjunto de funciones y procedimientos (métodos, en el ámbito de la programación orientada a objetos), que se ofrece para ser utilizado por otro software como una capa de abstracción.
- **SDK:** siglas en inglés de *Software Development Kit*, kit de desarrollo de aplicaciones, es un conjunto de herramientas destinadas al desarrollo software que aportan al programador el soporte necesario para crear programas para un sistema concreto, como son paquetes de software, *frameworks*, plataformas de hardware, computadoras, sistemas operativos, etcétera.
- **IDE:** *Integrated Development Environment*, en español entorno de desarrollo integrado, es un conjunto de herramientas de programación englobado en una aplicación informática, cuyo cometido es facilitar la tarea del programador. Una *IDE* puede usarse para programar en uno o varios lenguajes de programación, como, por ejemplo *Eclipse*, que es la elegida para desarrollar este proyecto.
- **Eclipse:** es un entorno de desarrollo integrado de código abierto con soporte para varios lenguajes de programación. Fue desarrollado en un principio por IBM, pero ahora el proyecto está a cargo de la ONG *Eclipse Foundation*.
- **ADT:** *Android Development Tools* (herramientas de desarrollo de *Android*) es el plugin necesario para que Eclipse adquiera la capacidad de desarrollar software *Android*.
- **AVD:** *Android Virtual Device*, dispositivo virtual *Android* en castellano, es la máquina virtual que proporciona la *SDK* para poder hacer comprobaciones sin necesidad de tener o conectar un dispositivo físico al ordenador del desarrollador.
- **Java:** es un lenguaje de programación orientado a objetos, el principal para crear aplicaciones para *Android*.
- **xml:** *eXtensible Markup Language* (lenguaje de marcas extensible) es un lenguaje de marcas usado para codificar información. En *Android* se usa para definir los *layouts* o vistas de diseño.
- **Activity o Actividad:** en resumidas cuentas, es lo que el usuario puede hacer en un momento exacto, y suele coincidir con la pantalla enfocada en ese momento en el dispositivo. Desde el punto de vista de la programación se puede comentar que son clases que comprenden la lógica de la aplicación heredadas de la clase base *Android.app.Activity*.
- **Layout o vista:** en *Android*, son contenedores de vistas que se diseñan mediante xml, donde se define el formato de cada elemento que tendrá una pantalla o *Activity*. La principal característica es que son anidables, para así poder crear composiciones.

Capítulo 2

Descripción general del proyecto

2.1 Explicación básica sobre la aplicación

Esta aplicación móvil de cocina representa recetas de forma básica, sencilla y clara. Con una interfaz amigable, simple e interactiva, a modo de juego, el usuario podrá ir leyendo recetas de forma guiada mediante pasos bien definidos, pudiendo interactuar para adquirir más información sobre algunos pasos. Ha sido creada para dispositivos móviles que funcionen bajo sistema operativo *Android* y se cumplimentará usando el entorno de desarrollo integrado *Eclipse*.

2.2 Características particulares

Cada receta comenzará con una pantalla donde se puede obtener información varia sobre los alérgenos o las kilocalorías. A continuación, conoceremos lista de ingredientes con las cantidades de cada uno. Lo siguiente es la visualización de cada paso ordenado, uno tras otro, que hay recorrer para conseguir como resultado el plato que nos indica la receta. Finalmente, se incluye la opción de tomar una foto del plato una vez confeccionado.

Mediante el chequeo individual de cada paso de una receta se puede conseguir de un modo muy visual completar su realización.

Las diferentes formas de realizar las recetas son:

- Desde un modo “aventura”, focalizado en la mejora de las habilidades culinarias mediante la elección de cinco recetas al azar repartidas en un mapa de menor a mayor dificultad, que convenientemente no hayan realizadas o acabadas anteriormente, y sean todas de diferentes tipos.
- Mediante un modo “a cocinar”, que hace las veces de recetario, listando las recetas por tipos.
- Usando el modo “batidora”, que permite al usuario introducir varios alérgenos que no quiere que tenga la receta o múltiples ingredientes con las cantidades que él mismo elija, dando como resultado una o varias recetas ordenadas por porcentaje de acierto, de mayor a menor aproximación a lo que el usuario ha elegido.

Como apoyo a la hora de cocinar, la aplicación trae consigo un reloj con cuenta regresiva y alarma para ayudar a los usuarios a poder cronometrar con exactitud los tiempos de cocción.

Para finalizar, se pueden ver en un listado los consejos desbloqueados, que no son más que frases de ayuda que se van obteniendo al pulsar sobre los pictogramas de técnica de cada paso de receta para conseguir desbloquearlos. Estos se pueden aprovechar posteriormente para aplicarlos a otras recetas en las que, por criterio propio del usuario, pueda pensar que es conveniente usarlos.

2.2.1 Definición de los elementos principales

A continuación se citan varios componentes claves para entender el funcionamiento que tiene la aplicación.

- **Receta:** conjunto finito de pasos ordenados que combinan ingredientes y técnicas de cocina para terminar obteniendo como resultado un plato listo para comer.
- **Tipos de Receta:** conjuntos que contienen recetas dependiendo del ingrediente principal o característico.
- **Paso:** es la acción de aplicarle técnicas a ingredientes y son expresados mediante cadenas de texto. Una aglomeración finita de pasos en un determinado orden componen una receta.
- **Ingrediente:** es uno de los alimentos que componen un paso.
- **Tipos de ingrediente:** conjuntos que agrupan cada ingrediente en una categoría dependiendo de las características alimenticias del mismo, como pueden ser vegetales, carnes, cereales, pescados, etcétera.
- **Alérgeno:** ingrediente o parte de un ingrediente que puede provocar algún tipo de reacción alérgica a personas sensibles.
- **Kilocaloría:** unidad que mide la energía que contienen los alimentos.
- **Técnica:** es toda aquella práctica o proceso que se le aplica a los alimentos.
- **Tipos de técnica:** conjuntos que agrupan cada técnica en una categoría dependiendo del tipo de operación que sea, como por ejemplo cortar, añadir, batir, hornear, etcétera.
- **Modo de juego:** posibilidades que tiene la aplicación para que el usuario acceda a las recetas u ofrecerle algún recurso, como son el modo aventura, a cocinar, batidora, el cronómetro y los consejos.
- **Usuario:** persona que se identifica con un nombre único para acceder a la aplicación y comenzar o continuar con su progreso, que es individual para cada uno de los usuarios que estén dados de alta.
- **Pictograma:** son las diferentes imágenes que representarán una técnica o ingrediente.

2.3 Concepto gráfico

Es conveniente reseñar que, cuando la idea del proyecto empezó a forjarse, me puse de acuerdo con Agustín Rodríguez – Izquierdo Flores para acordar el diseño gráfico de la aplicación. Establecimos unos primeros bocetos sobre la representación pictórica de la identidad corporativa y de varios de los elementos principales.

Como primer paso, establecimos el nombre de la aplicación de cara al púbico tras barajar diferentes posibilidades. Acordamos que la marca para su difusión sería "*aCubierto*" y que el logotipo sería el siguiente:

Figura 2.1: Logotipo aCubierto

Coincidimos en que el formato visual debe ser sencillo, al igual que lo es el manejo de la aplicación, para intentar la máxima información a partir un número mínimos de elementos.

2.4 Visión general del sistema

La *app* interactúa con el usuario de manera que de él comience a surgir un afán incremental por descubrir trucos y realizar recetas mediante detalles visuales, e implicarlo de forma activa en la mejora de sus dotes culinarias creando una atmósfera de satisfacción por medio de la autosuperación. Un valor añadido para motivar es la posibilidad de capturar tus platos al finalizarlos para poder compartirlos en las redes sociales. Todo esto apoyado por una interfaz agradable, que hace que para el usuario cocinar se convierta en un juego. Para clarificarlo, se puede observar el siguiente esquema:

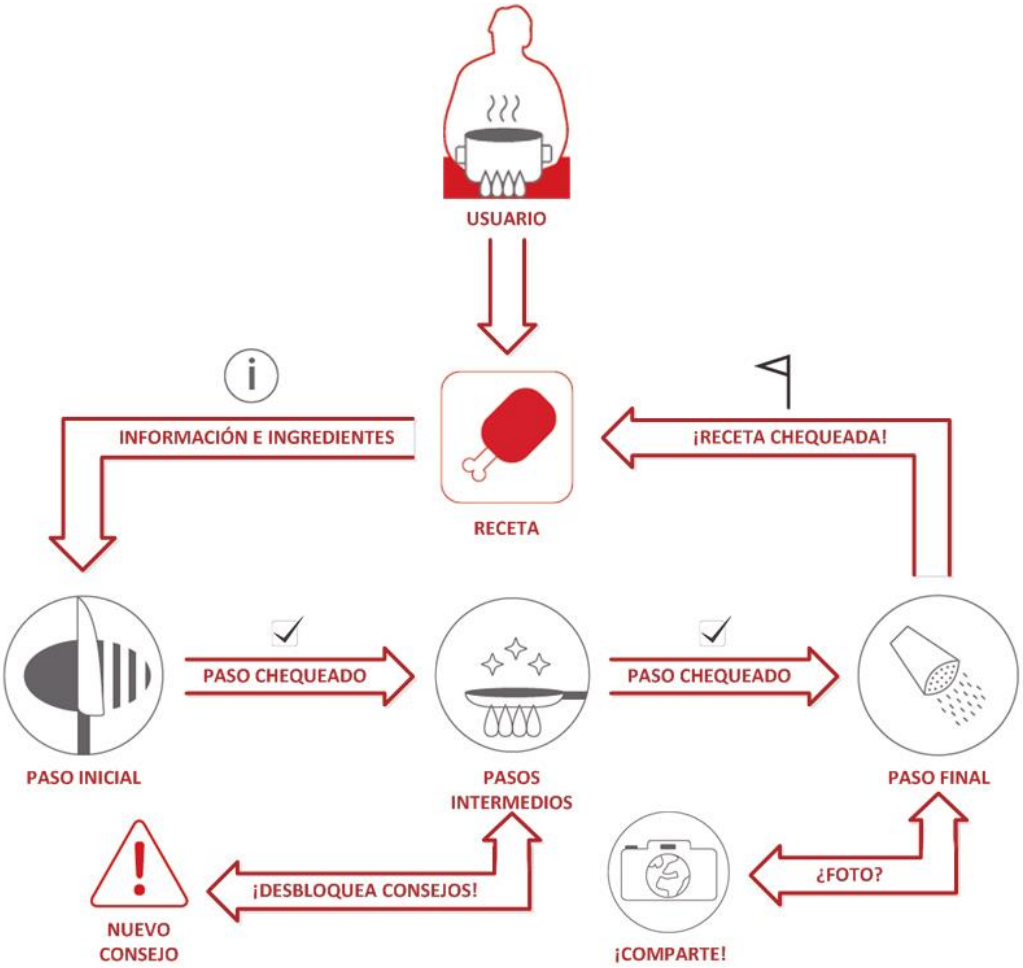


Figura 2.2: Diagrama visión general del sistema

2.5 Alcance

La intención inicial es introducir la aplicación resultante del proyecto en *Google Play* para que cualquier usuario de *Android* con una versión superior a la mínima requerida pueda disfrutar de ella. El público al que va dirigida la app resultante es, sobre todo, personas con interés por mejorar sus habilidades culinarias cuya base es inicialmente baja o nula. Aunque cualquiera, obviando el nivel de conocimientos, interesado por aprender nuevas recetas, conocer nuevos trucos, o, simplemente, tener un recetario a mano, también es público objetivo.

2.6 Colaboraciones

La idea original de la aplicacióny el apartado puramente informático, como son el análisis y el diseño del sistema han sido desarrollados por el autor de estas memorias. La parte de diseño gráfico, como anteriormente se ha mencionado, ha sido trabajo conjunto con Agustín Rodríguez – Izquierdo Flores. Esa parte del trabajo abarca todo lo relativo a la imagen visual y pictogramas de la aplicación, junto con su disposición en las diferentes pantallas. Menos relevante desde el punto de vista de la informática, pero aun así de gran importancia, es el manual de identidad corporativa [2] que contiene elementos como toda la explicación de los

componentes básicos del diseño gráfico, posibles campañas publicitarias, productos de *merchandising* y los formatos de todas las pantallas, así como el análisis de los elementos que diferencian esta aplicación de otras existentes en *Google Play Store*.

2.7 Metodología de desarrollo

Para finalizar la descripción y dar paso a la planificación, se debe plantear a priori qué modelo de ciclo de vida del software se va a usar. Esta es una decisión crítica que nos va a acompañar durante el resto del recorrido del proyecto y de la existencia de la aplicación. Tras haber plasmado y revisado los conceptos originarios, el más adecuado puede ser el modelo en cascada, por ser el más riguroso y estructurado. La decisión se apoya en que es la primera vez que hago frente a un proyecto de tal envergadura, y para no derivar en errores nacidos de la ignorancia sobre otras filosofía de trabajo más profundas esta es la mejor opción.

Las fases de este modelo son muy claras, secuenciales y rigurosas, características que pueden ayudar a no desviarse del objetivo principal, que no es más que crear una aplicación móvil sobre cocina para *Android* aplicando los métodos correctos para el desarrollo. Las etapas que se establecen en este enfoque metodológico son la planificación, el análisis de requisitos, el diseño del sistema, la implementación, las pruebas y el posterior mantenimiento.

Capítulo 3

Planificación

La planificación llevada a cabo para el desarrollo del proyecto está dividida en las siguientes partes:

3.1 Fase inicial

El proyecto empezó a formarse a partir de la situación comentada anteriormente en el apartado de motivaciones. Como consecuencia de las mismas empecé a plantear hasta dónde quería llegar, pensando por encima en las características básicas que quería integrar, apuntando en una libreta ideas sueltas a las que después se les fueron dando una forma más concreta en las fases posteriores de desarrollo.

La siguiente decisión en el inicio fue qué herramientas y medios se iban a usar para llevar a cabo la idea. Siempre dando prioridad al software libre sobre el privativo, la decisión fue usar el *SDK* oficial de *Android* junto con el *IDE Eclipse*, por la versatilidad que te da un entorno de desarrollo multilenguaje con gran cantidad de *plugins*.

3.2 Fase de análisis y diseño

En este segundo nivel, el planteamiento estuvo centrado en qué debía hacer exactamente la aplicación y en los recursos que iban a ser necesarios durante toda la consecución, máximamente, el tiempo empleado. Se definieron las características de la aplicación mediante la especificación de requisitos, vislumbrando las funcionalidades, y se hicieron bocetos para estimar cómo éstas estarían repartidas en el espacio visual de las pantallas.

El primero de ellos era definir exactamente qué estructura conceptual de almacenamiento iban a tener la piedra angular de la aplicación: las recetas. A partir de esto, el resto de la aplicación se construiría aportando las diferentes funcionalidades en torno al modelo de datos de las recetas.

Habiendo establecido la estructura de las recetas, lo siguiente fue definir con más exactitud los requisitos funcionales de la aplicación mediante la pregunta “qué debe hacer la aplicación en cada momento” y “cómo va a abordar lo que debe hacer”. Para poder abordar esta parte sin que se convirtiera en un auténtico caos por la cantidad de posibilidades que se habían planteado inicialmente, se decidió actuar con la premisa “divide y vencerás”. Tomando las

ideas más generales como son los modos de juego, login o la representación de recetas, y haciendo subdivisiones sucesivas con una concepción recursiva, partiendo cada gran bloque en cometidos más concretos y particularizando mediante tareas el funcionamiento de los mismos, conseguimos, tras colapsar el árbol que se crearía con este sistema, ir dando la funcionalidad completa y correcta a cada apartado de la aplicación. Como ejemplo, el desarrollo del modo de trabajo para el caso de la representación de las recetas quedaría de la siguiente forma:

- Representación de receta
    - Presentación previa
        - Chequeo de receta
            - ¿Todos los pasos están chequeados?
        - Kilocalorías que tiene
            - Suma de las kilocalorías de cada ingrediente dependiendo de la cantidad
        - Alérgenos que tiene
            - Comprobar si cada ingrediente contiene algún alérgeno
    - Lista de ingredientes
        - Cantidad e ingrediente
            - Listar ingredientes de una receta con su cantidad
    - Presentación de pasos
        - Número de paso
            - Paso número n / total de pasos de la receta
        - Chequeo de paso
            - Comprobar si estaba chequeado anteriormente o no
            - Capacidad de chequear y deschequear
        - Técnica
            - Obtener tipo de técnica
            - Obtener descripción de la técnica
        - Ingredientes
            - Obtener ingredientes implicados en el paso

Una vez aclarado lo anteriormente comentado, se procedió a aplicar las herramientas propias para la continuación de esta fase para completar el análisis y el diseño.

Paralelamente a la cumplimentación de este apartado, se fueron perfeccionando los bocetos de la parte gráfica mediante el contacto ininterrumpido con la persona encargada del tema.

3.3 Fase de aprendizaje

Se planteaba como uno de los momentos más arduos del proyecto, y así lo fue. Era la primera vez que tenía que enfrentarme al reto de adquirir el suficiente nivel en un lenguaje de programación, totalmente desconocido y por cuenta propia, para desarrollar la aplicación.

Para afrontar el reto, comencé por ver qué material didáctico debía tener a mano. Para empezar, necesitaba un libro a modo de manual para poder hacer las consultas adecuadas sobre las bases de *Java*, para así ir asimilando conceptos comparando mis conocimientos en

C++ con la parte más elemental de *Java*. Tras ojear varios, decidí adquirir el libro *Java 7 Bases del lenguaje y de la programación orientada a objetos* [3], que contiene las explicaciones de los conceptos básicos del lenguaje.

Lo siguiente sería empezar a entender las directrices para programar en *Android*. Teniendo ya una base de Java, lo primordial era conocer los conceptos más relevantes de la *API*, como son las *Activities* y los *layouts*. Me decanté por adquirir, al menos, un documento escrito, el *Android, Curso de desarrollo de aplicaciones* [4], para leerlo a la vez que hacía pruebas con programas de tipo "Hola Mundo". Otra serie de recursos en formato digital, como el curso online (MOOC) de la Universidad Politécnica de Valencia [5] sobre *Introducción en desarrollo de aplicaciones para Android*, las guías y tutoriales de aprendizaje oficiales [6] y las dudas de otros desarrolladores en el foro *stack overflow* [7] fueron fuentes de grandísima ayuda para todo el proceso de aprendizaje, y también de desarrollo.

Después de un tiempo escudriñando las diferentes facetas de *Android* y conociendo por encima las características básicas que la aplicación tendría, llegué a la conclusión de que, para representar ciertos comportamientos, iba a necesitar usar alguna tecnología para almacenar información de las diferentes opciones que se ofrecen en la *API*. Habiéndolas revisado, vi necesario repasar lo que había aprendido durante mis años de estudio y ampliar mis conocimientos sobre *SQL* (*lenguaje de consulta estructurada*), en concreto, las particularidades de *SQLite*, que es para el que la *API* tiene soporte integrado.

Centrándonos en la materia principal que trata la aplicación, la cocina, debía precisar sobre ciertos conceptos con carácter particular. Por cuenta propia, leyendo artículos y blogs en la red y analizando varios libros sobre recetas que tenía de antes, como *Platos únicos* [8] o *Carnes, la cocina de casa con el toque de los grandes chefs* [9], empecé a concretizar ideas tales como cuál es la mejor forma de englobar diferentes tipos de técnicas, categorías de alimentos, qué es un alérgeno (14 principales, información obtenida de la web del Consejo Europeo de Información sobre la Alimentación [10]), etcétera. Incluso tuve varios encuentros con un conocido que es estudiante de la Escuela de Hostelería de Sevilla para escuchar una opinión más técnica.

3.4 Fase de desarrollo

Tras adquirir las suficientes nociones básicas en los temas comentados anteriormente, se comenzó la primera fase de desarrollo con varios planteamientos claros.

Teniendo desarrollados todos los árboles de las funcionalidades generales del apartado de análisis, se tuvo que revisar las dependencias entre ellos, puesto que es evidente que entre actividades de *Android*, para su correcto funcionamiento, se debían compartir ciertos datos como son la id del usuario, la id de una receta, etcétera.

El mayor de los esfuerzos estuvo centrado en crear un buen sistema de clases con el que fuera posible abarcar todas las funcionalidades y también tuviera una buena capacidad para facilitar mejoras de la aplicación en un futuro, dando así una mejor proyección, adaptabilidad y

versatilidad de cara a una posterior ampliación, siempre respetando el resultado del análisis y el diseño.

3.5 Fase de pruebas y correcciones

Usando la estructura de árbol de la fase de desarrollo, se creaba la posibilidad de ir implementando la aplicación por partes, con lo que paulatinamente podían ir haciéndose comprobaciones de verificación del funcionamiento.

Estas pruebas se llevaban a cabo en diferentes dispositivos móviles con diferentes versiones para prever los posibles fallos en caso de diferentes versiones. Se ha usado principalmente un *Smartphone* de gama media, una *Tablet* de gama media y la *AVD* que viene con la *SDK* de *Android* en la versión mínima contemplada por la aplicación. Las pruebas, más que en la correcta visualización de los objetos en pantalla, estaban orientadas a que las actividades y sus componentes funcionaran debidamente.

Se tuvieron muy en cuenta casos de cambios entre actividades, como entrada-salida-entrada de las mismas, para comprobar si el tránsito de datos entre ellas era el correcto. La mayoría de los fallos en la fase de implementación fueron de ese tipo. Otro tipo de fallo común fue con respecto al manejo de ciertas clases de *Java*, en las que ciertos métodos no llegaban a funcionar como se pensaba que lo iban a hacer por falta de práctica con el lenguaje.

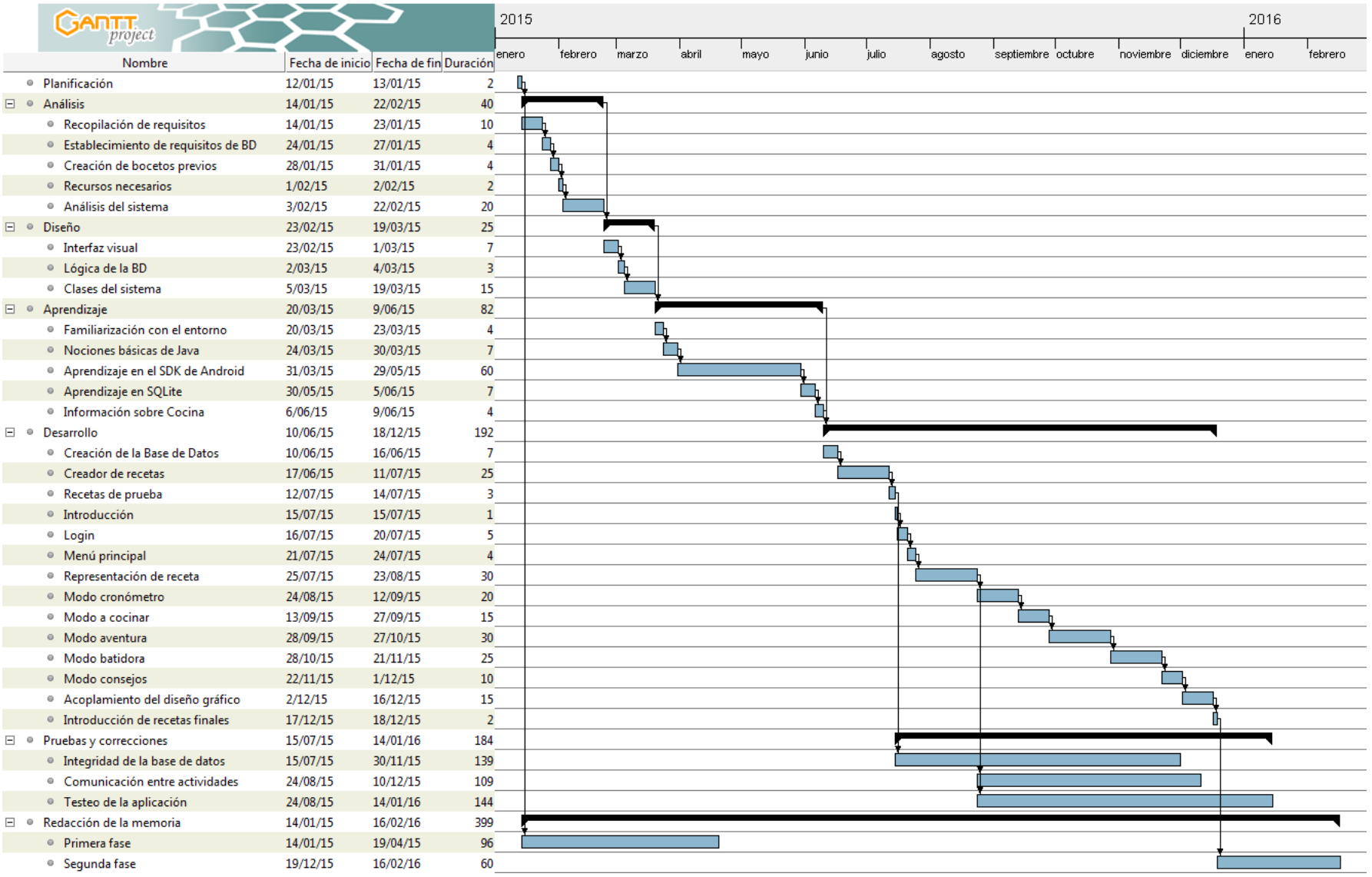
3.6 Fase de redacción de la memoria

La memoria se ha ido redactando a la vez que se han ido cumplimentando el resto de fases del proyecto. Llegado cierto punto, en el momento de la fase de implementación, la redacción perdió un poco de paralelismo con el desarrollo, puesto que todos los esfuerzos estuvieron centrados en crear un código sólido que cumpliera todas las funcionalidades de la manera más eficiente posible. Para que se evitara perder información importante para la redacción en ese momento, se utilizó papel y bolígrafo para ir anotando cada dato relevante para después tenerlo en cuenta en la continuación de la memoria.

Tras acabar la fase de desarrollo, se procedió a continuar con la redacción, no sin antes hacer una revisión de lo que ya se había escrito, para que el resultado final fuera lo más fiel al trascurso del proyecto.

3.7 Diagrama de Gantt

Como resumen de toda la planificación del proyecto, mostramos a continuación el diagrama de Gantt con el desglose en el tiempo de todas las fases que han tenido que ser abordadas:



Nombre	Fecha de inicio	Fecha de fin	Duración
Planificación	12/01/15	13/01/15	2
Análisis	14/01/15	22/02/15	40
Recopilación de requisitos	14/01/15	23/01/15	10
Establecimiento de requisitos de BD	24/01/15	27/01/15	4
Creación de bocetos previos	28/01/15	31/01/15	4
Recursos necesarios	1/02/15	2/02/15	2
Análisis del sistema	3/02/15	22/02/15	20
Diseño	23/02/15	19/03/15	25
Interfaz visual	23/02/15	1/03/15	7
Lógica de la BD	2/03/15	4/03/15	3
Clases del sistema	5/03/15	19/03/15	15
Aprendizaje	20/03/15	9/06/15	82
Familiarización con el entorno	20/03/15	23/03/15	4
Nociones básicas de Java	24/03/15	30/03/15	7
Aprendizaje en el SDK de Android	31/03/15	29/05/15	60
Aprendizaje en SQLite	30/05/15	5/06/15	7
Información sobre Cocina	6/06/15	9/06/15	4
Desarrollo	10/06/15	18/12/15	192
Creación de la Base de Datos	10/06/15	16/06/15	7
Creador de recetas	17/06/15	11/07/15	25
Recetas de prueba	12/07/15	14/07/15	3
Introducción	15/07/15	15/07/15	1
Login	16/07/15	20/07/15	5
Menú principal	21/07/15	24/07/15	4
Representación de receta	25/07/15	23/08/15	30
Modo cronómetro	24/08/15	12/09/15	20
Modo a cocinar	13/09/15	27/09/15	15
Modo aventura	28/09/15	27/10/15	30
Modo batidora	28/10/15	21/11/15	25
Modo consejos	22/11/15	1/12/15	10
Acoplamiento del diseño gráfico	2/12/15	16/12/15	15
Introducción de recetas finales	17/12/15	18/12/15	2
Pruebas y correcciones	15/07/15	14/01/16	184
Integridad de la base de datos	15/07/15	30/11/15	139
Comunicación entre actividades	24/08/15	10/12/15	109
Testeo de la aplicación	24/08/15	14/01/16	144
Redacción de la memoria	14/01/15	16/02/16	399
Primera fase	14/01/15	19/04/15	96
Segunda fase	19/12/15	16/02/16	60

Figura 3.1: Diagrama de Gantt

Capítulo 4

Análisis de requisitos

4.1 Requisitos funcionales del sistema

Vamos a centrarnos en qué debe hacer la aplicación en cada modo de la misma, empezando con una breve explicación textual, y apoyándonos en el procedimiento especificado en el apartado de análisis descrito en el capítulo de planificación, un desglose en tareas cada vez más básicas que comprenden una estructura de árbol.

Cabe destacar que este apartado evoluciona con el tiempo, puesto que durante las siguientes fases del proyecto pueden aparecer detalles que no estaban contemplados previamente y / o modificaciones para una mejor funcionalidad y cumplimiento de las expectativas.

Comenzaremos de forma ordenada, desde el inicio de la aplicación hasta cualquiera de los modos de juego.

4.1.1 Inicio de la aplicación

Se lanza la actividad para dar paso a la bienvenida.

- Inicio de la *app*
    - Bienvenida

4.1.2 Login de usuarios

Gestión de los usuarios para comenzar el juego. Permite crearlos y eliminarlos, borrando todos sus datos de juego. Se deberá elegir uno para poder comenzar.

- Login
    - Nuevo jugador
    - Eliminar jugador
    - Seleccionar jugador
    - Comenzar juego

4.1.3 Selección de modos de juego

Da acceso a cualquiera de los modos que ofrece la aplicación, como son el modo aventura, el "a cocinar", batidora, cronómetro y consejos.

- Menú principal
    - Ir a modo aventura
    - Ir a modo a cocinar
    - Ir a modo batidora
    - Ir a modo cronómetro
    - Ir a modo consejos

4.1.4 Modo aventura

Muestra una serie de 5 recetas de diferentes tipos, ordenadas por dificultad de menor a mayor y preferiblemente chequeadas, sobre un mapa, mediante el icono del tipo de cada una. El usuario podrá ver su progreso global pulsando un botón de inicio y comenzar una nueva aventura cuando todas las recetas de la actual estén chequeadas haciendo clic en un botón de meta. Se podrá seleccionar cualquier receta del mapa, sin importar el orden, mostrando el nombre de la misma, el tipo, el nivel dentro del modo, el número de pasos, el tiempo y dando la opción de comenzar con ella.

- Modo aventura
    - Asignación previa de 5 recetas de diferente tipo (1ª vez)
        - Selección de recetas de diferente tipo y preferiblemente no chequeadas
    - Visualización previa de receta
        - Nombre, tipo, nº de pasos, tiempo y nivel
        - Comenzar receta
    - Consultar progreso
        - Veces que se ha concluido una aventura y recetas que faltan para la actual
    - Iniciar nueva aventura (si todas las recetas están chequeadas)
        - Asignación de 5 recetas con las condiciones anteriores

4.1.5 Modo a cocinar

Contiene todas las recetas organizadas por apartados, en concreto, por tipos. Permite seleccionar cualquiera de ellos y ver el listado de recetas, pudiéndolas ordenar por tiempo de realización o por dificultad. Se podrá distinguir cuál de ellas ha sido realizada y cuál no, y acceder a un resumen de la misma, donde se podrá dar paso a su realización.

- Modo a cocinar
    - Ir al listado de recetas de cada tipo
        - Visualización del listado, realizadas o no realizadas
        - Ordenar por dificultad o por tiempo
        - Visualización previa de receta

- Nombre, tipo, nº de pasos, tiempo y nivel
- Comenzar receta

4.1.6 Modo batidora

Permite configurar un listado de ingredientes y alérgenos para dar con recetas que se aproximen a las necesidades plasmadas. Los alérgenos se podrán elegir de una lista de los oficialmente reconocidos para que las recetas no los contengan, y los ingredientes podrán elegirse por tipo o por orden alfabético, pudiéndoles añadir una cantidad asociada. Tras esto, la aplicación da una serie de resultados con un índice de aproximación a la realización de la receta, y ofrece la oportunidad de acceder a su resumen y de comenzar con ellas.

- Modo batidora
  - Listado de alérgenos
    - Posibilidad de selección de cualesquiera
  - Listado de ingredientes
    - Listado por tipos
    - Listado completo por orden alfabético
  - Mostrar resultados
    - Lista de recetas ordenadas de mayor a menor porcentaje de acierto
    - Visualización previa de receta
      - Nombre, tipo, nº de pasos, tiempo y nivel
      - Comenzar receta

4.1.7 Modo cronómetro

Este modo es un simple cronómetro de cuenta atrás para que el usuario pueda servirse de él cuando vaya a cocinar algo de forma libre. Se deberá añadir un tiempo dividido en horas, minutos y segundos, por separado. La cuenta atrás comenzará pulsando el botón indicado, y podrá pausarse pulsando el mismo botón. Dejando pulsado el botón reset, todo volverá a su estado inicial. Cuando concluya la cuenta atrás, la aplicación activará un sonido como aviso.

- Modo cronómetro
  - Introducción de horas, minutos y segundos
    - Comenzar y pausar cuando esté en marcha
    - Posibilidad de resetear todos los valores
    - Activa aviso cuando llega a cero

4.1.8 Modo consejos

Contiene el listado con un resumen de los consejos que se hayan ido desbloqueando durante la realización de recetas. Se puede pulsar en cada uno de ellos para ver el consejo completo, junto con la receta al que pertenece y el tipo de técnica al que se le aplica. Puede verse qué consejos no han sido desbloqueados de cada receta mediante un botón.

- Modo consejos
    - Visualizar listado de consejos desbloqueados
        - Acceder a cada uno de los consejos
    - Posibilidad de visualizar un resumen de los consejos no desbloqueados

4.1.9 Realizar receta

Todos los puntos anteriormente comentados giran en torno o desembocan en este. Cuando se comienza una receta se podrá conocer mediante los respectivos botones qué alérgenos contiene y cuántas kilocalorías, junto con una bandera que indicará si ha sido totalmente chequeada con anterioridad o no. Se mostrará una lista con qué ingredientes son necesarios y en qué cantidad. Cada paso de la receta estará compuesto de una técnica con un pictograma característico de tipo, el cual podrá desbloquear un consejo que esté ligado a dicha técnica pulsando en él, y que se le aplica a uno o varios ingredientes, dando como resultado un formato legible y lógico. Cada paso irá numerado y podrá ser chequeado por el usuario con un clic en el elemento de la pantalla destinado para ello. Además, si lleva enlazado un tiempo de preparación, se podrá activar el modo cronómetro mediante un botón especial para poder llevar un cálculo exacto. Llegado al último paso, el pictograma de técnica indicará que se puede tomar una fotografía de cómo quedó la receta, ofreciendo el compartirla en las redes sociales. La navegación consecutiva entre introducción de la receta, listado de ingredientes y pasos podrá llevarse a cabo pulsando en los botones anterior y siguiente, y deslizando el dedo sobre la pantalla de forma natural, de izquierda a derecha o de derecha a izquierda correlativamente.

- Representación de receta
    - Navegación entre fases de receta hacia delante y hacia atrás
        - Botones de navegación
        - Deslizamiento natural del dedo sobre la pantalla
    - Presentación previa
        - Receta completamente chequeada o no
        - Mostrar kilocalorías
        - Mostrar alérgenos
    - Lista de ingredientes
        - Cantidad e ingrediente
    - Presentación de pasos
        - Número de paso
        - Posibilidad de chequeo o des-chequeo de paso por parte del usuario
        - Posibilidad de activación de cronómetro si existe tiempo asociado al paso
        - Descripción del paso
            - Técnica
            - Ingrediente(s)
        - Posibilidad de consejo clicando en el pictograma de técnica
            - Consejo desbloqueado por primera vez
            - Consejo leído anteriormente

        - Posibilidad de tomar fotografía en paso final
            - Posibilidad de compartir en redes sociales

4.1.10 Requisitos de la interfaz gráfica

Para poder representar todas las funciones que se acaban de explicar, se van a necesitar una serie de pictogramas, iconos y vistas de pantallas que compondrán la identidad visual del producto. Como se ha expuesto en capítulos anteriores, la intención es que el aspecto visual de la aplicación sea sencillo, intuitivo y claro. Para poder ir adelantando el trabajo de esta área, mediante la revisión exhaustiva de los requisitos anteriores se han definido una serie de elementos gráficos, que en principio serán los indispensables.

- Pantallas de presentación de las funcionalidades
    - Inicio de la aplicación
    - Login
        - Nuevo jugador
        - Eliminar jugador
    - Menú principal
    - Modo aventura
        - Progreso global
        - Nueva aventura
        - Mapa de aventura
    - Modo a cocinar
        - Listado de tipos de receta
        - Listado de recetas de cada tipo
    - Modo batidora
        - Selector de alérgenos, ingredientes y lista de seleccionados
        - Listado con resultados y porcentajes de acierto
    - Modo cronómetro
    - Modo consejos
        - Listado de consejos desbloqueados
        - Listado de consejos bloqueados
    - Receta
        - Resumen
            - Kilocalorías
            - Alérgenos
        - Introducción
        - Lista de ingredientes
        - Pasos
- Conjunto de pictogramas
    - Pictogramas para tipos de receta
    - Pictogramas para tipos de técnicas
- Iconos comunes
    - Logo de la aplicación

    - Home
    - Salida y meta
    - Kilocalorías y alérgenos
    - Receta y paso chequeado
    - Paso anterior y paso siguiente
    - Activación de cronómetro
    - Tiempo y dificultad
- Estilo de botones
- Tipografías

4.2 Requisitos no funcionales del sistema

A continuación, vamos a pasar a definir los requisitos que van a rodear el sistema que comprende la aplicación.

Interfaz hardware y software externa

Como se ha comentado anteriormente, la aplicación está diseñada para dispositivos móviles con sistema operativo *Android*. El nivel mínimo de la API que se ha decidido usar es el 16, correspondiente a la primera versión de la *Jelly Bean (Android 4.1.2)*, que tiene la característica de que unifica las dos versiones anteriores, la de *tablet* y la de *smartphone*. A fecha de febrero de 2015, más del 85% de los dispositivos *Android* funcionan con esta versión.

Requisitos de eficiencia

Toda aplicación o programa debe tender tener un buen funcionamiento. Por ello, varios puntos a tener en cuenta, habiendo sido usuario de dispositivos móviles *Android* desde hace más de cinco años, son la memoria que ocupará en el dispositivo, la fluidez de funcionamiento y la utilización de la red de datos.

Se intentará evitar el derroche de estos recursos a la hora de generar la aplicación, creándola con parámetros aceptables para el usuario dentro de las posibilidades del desarrollador.

4.3 Modelo de casos de uso

Para proseguir con la fase de análisis, el siguiente paso es describir de una forma concreta los comportamientos del sistema referidos en los requisitos funcionales mediante *UML* (lenguaje de modelado unificado), que da una visión exacta mediante diagramas de qué es lo que exactamente hace la aplicación.

4.3.1 Diagramas de casos de uso

Para llevar una estructura lógica, vamos a seguir el orden que se ha definido en el desarrollo de los requisitos funcionales.

Los diagramas de casos de uso examinan los diferentes escenarios que se dan en el sistema, y son, hoy en día, fundamentales para describir modelos de sistemas orientados a objetos. Analizan la interacción de los diferentes actores con el sistema.

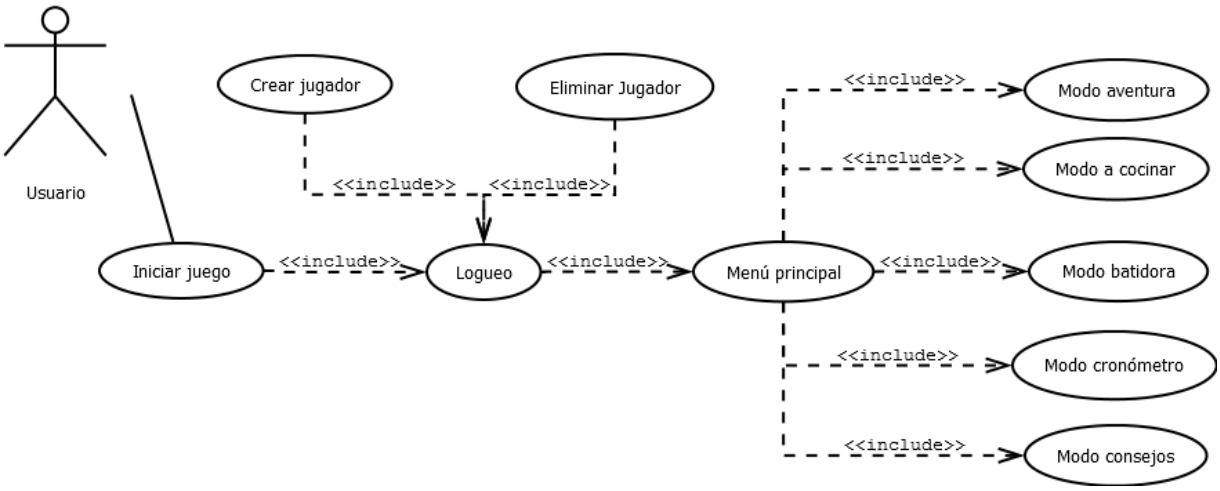


Figura 4.1: Análisis: Diagrama de casos de uso

- **Caso de uso:** Iniciar juego.

Descripción: Se muestra la pantalla de bienvenida de la aplicación.

Actores: Usuario.

Precondiciones: Ninguna.

Postcondiciones: El usuario accede a la siguiente pantalla.

Escenario principal:

1. El usuario ejecuta la aplicación.
2. El sistema muestra la pantalla de inicio con las indicaciones para proseguir.
3. El usuario pulsa sobre la pantalla y accede al login.

Extensiones – flujo alternativo:

3a. El usuario pulsa la tecla “atrás” del dispositivo.

1. El sistema cierra la aplicación.

- **Caso de uso:** Logueo.

Descripción: El usuario se identifica con un nombre en el sistema.

Actores: Usuario.

Precondiciones: El sistema se encuentra en espera en la pantalla de login.

Postcondiciones: El usuario accede al menú principal.

Escenario principal:

1. El usuario selecciona un nombre de la lista de jugadores.
2. El sistema marca el nombre del jugador seleccionado en la cabecera.
3. El usuario pulsa comenzar y accede a la pantalla de menú principal.

Extensiones – flujo alternativo:

*a. El usuario pulsa la tecla "atrás" del dispositivo.

1. El sistema cierra la ventana de login y retorna a la pantalla de inicio.

*b. El usuario selecciona crear jugador.

1. El sistema muestra la ventana para introducir nuevo jugador.

1. El usuario introduce un nombre de jugador.

2. El sistema valida el nombre, lo incluye en la lista de usuarios seleccionables, cierra la ventana y vuelve al paso 1 del caso de uso.

1a. La lista de jugadores está vacía.

1. El usuario deberá crear un jugador (*b) para continuar.

3a. El usuario selecciona eliminar jugador.

1. El sistema pregunta si se desea eliminar dicho jugador.

2. El usuario pulsa sí.

3. El sistema elimina los datos del jugador, cierra la ventana y vuelve al paso 1 del caso de uso.

3b. El usuario selecciona eliminar jugador.

1. El sistema pregunta si se desea eliminar dicho jugador.

2. El usuario pulsa no.

3. El sistema cierra la ventana y vuelve al paso 1 del caso de uso.

- **Caso de uso:** Menú principal.

Descripción: Visualización del acceso a los cinco diferentes modos de juego, donde se podrá elegir a cuál entrar mediante los respectivos botones.

Actores: Usuario.

Precondiciones: El usuario está correctamente logueado.

Postcondiciones: Ninguna.

Escenario principal:

1. El sistema muestra el menú principal de modos de juego.
2. El usuario selecciona el botón de acceso al modo aventura.
3. El sistema carga el modo aventura.

Extensiones – flujo alternativo:

2a. El usuario pulsa la tecla “atrás” del dispositivo.

1. El sistema cierra el menú principal y vuelve a la pantalla de inicio.

2b. El usuario selecciona el modo a cocinar.

1. El sistema carga el modo a cocinar.

2c. El usuario selecciona el modo batidora.

1. El sistema carga el modo batidora.

2d. El usuario selecciona el modo cronómetro.

1. El sistema carga el modo cronómetro.

2e. El usuario selecciona el modo consejos.

1. El sistema carga el modo consejos.

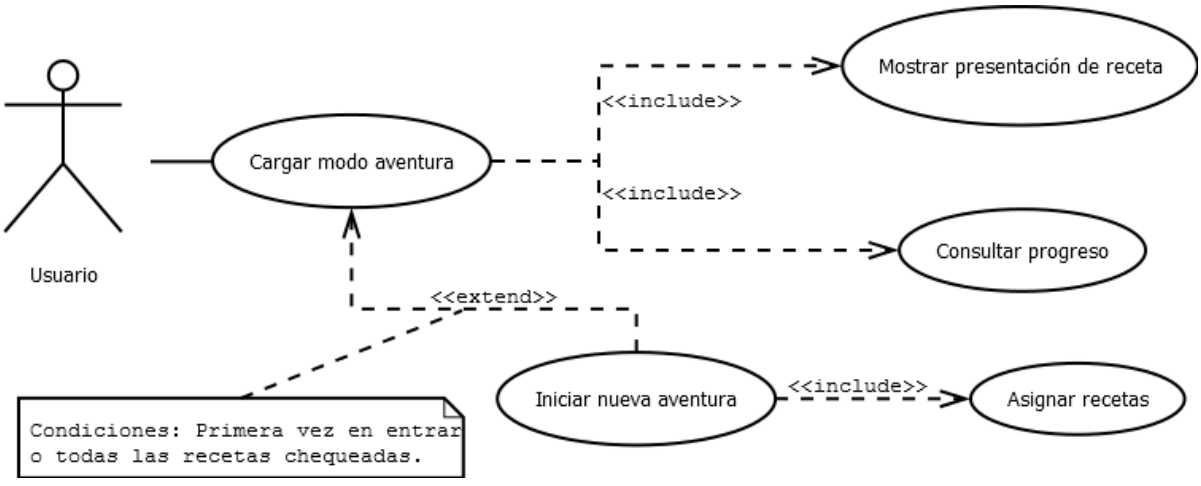


Figura 4.2: Diagrama de caso de uso cargar modo aventura

- **Caso de uso:** Cargar modo aventura.

Descripción: El usuario accede y juega al modo aventura, procediendo a realizar una receta de las varias que se asignan aleatoriamente.

Actores: Usuario.

Precondiciones: El usuario ha accedido al modo aventura desde el menú principal.

Postcondiciones: Ninguna.

Escenario principal:

1. El sistema muestra la pantalla del modo aventura en progreso que esté guardado.
2. El usuario selecciona una receta cualquiera.
3. El sistema muestra el resumen de la receta.
4. El usuario pulsa el botón comenzar la receta.
5. El sistema carga los datos de la receta y da paso a ella.

Extensiones – flujo alternativo:

1a. El sistema no detecta ningún modo aventura en progreso guardado.

1. El sistema selecciona las nuevas recetas, crea un nuevo modo aventura y vuelve al paso 1 del caso de uso.

2a. El usuario pulsa la tecla “atrás” del dispositivo.

1. El sistema cierra el modo aventura y vuelve a la pantalla de menú principal.

2b. El usuario selecciona el botón de comienzo para ver su progreso.

1. El sistema carga el progreso total del usuario en el modo aventura y lo muestra.
2. El usuario pulsa la tecla “atrás” del dispositivo para volver al modo aventura.
3. El sistema cierra la ventana y vuelve a comenzar el caso de uso.

2c. El usuario selecciona el botón de meta para iniciar nueva aventura.

1. El sistema comprueba y verifica afirmativamente que las recetas estén chequeadas.
2. El sistema selecciona las nuevas recetas y crea un nuevo modo aventura.
3. El sistema cierra la ventana y vuelve a comenzar el caso de uso.

2d. El usuario selecciona el botón de meta para iniciar nueva aventura.

1. El sistema comprueba y verifica negativamente que las recetas estén chequeadas.

3. El sistema cierra la ventana y vuelve a comenzar el caso de uso.

4a. El usuario pulsa la tecla “atrás” del dispositivo.

1. El sistema cierra el resumen de la receta y vuelve a comenzar el caso de uso.

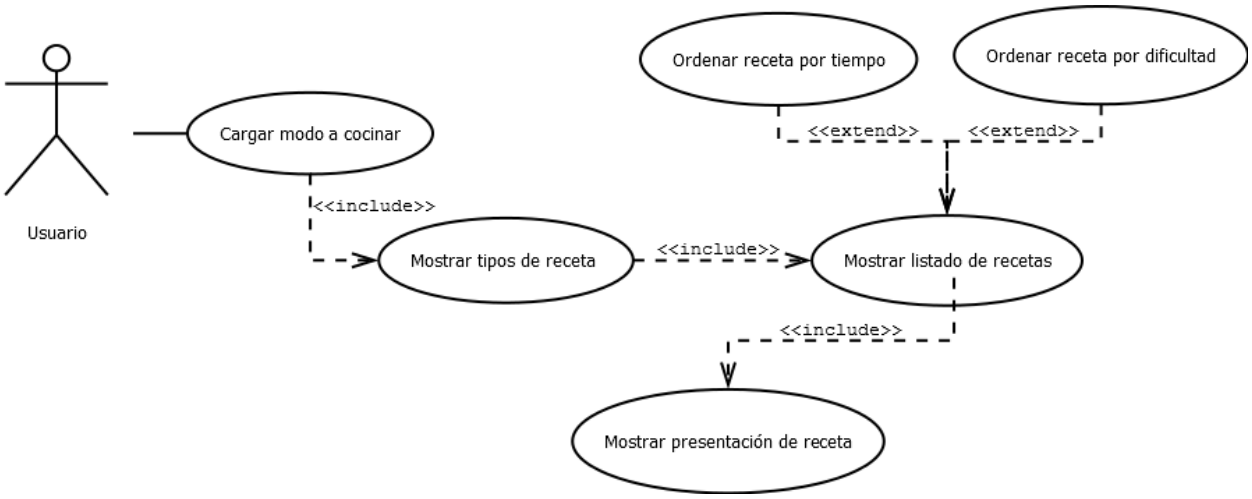


Figura 4.3: Diagrama de caso de uso cargar modo a cocinar

- **Caso de uso:** Cargar modo a cocinar.

Descripción: El usuario accede al modo a cocinar, mostrándose un listado de tipos de recetas. Se puede acceder a cada tipo para ver la lista de recetas y realizar una de ellas.

Actores: Usuario.

Precondiciones: El usuario ha accedido al modo a cocinar desde el menú principal.

Postcondiciones: Ninguna.

Escenario principal:

1. El sistema muestra la pantalla del modo a cocinar.
2. El usuario selecciona un tipo receta cualquiera.
3. El sistema genera un listado con las recetas del tipo seleccionado y lo muestra.
4. El usuario selecciona una receta cualquiera del listado mostrado.
5. El sistema muestra el resumen de la receta.
6. El usuario pulsa en comenzar la receta.

7. El sistema carga los datos de la receta y da paso a ella.

Extensiones – flujo alternativo:

2a. El usuario pulsa la tecla “atrás” del dispositivo.

1. El sistema cierra el modo a cocinar y vuelve a la pantalla de menú principal.

4a. El usuario selecciona el botón para ordenar las recetas por tiempo.

1. El sistema ordena las recetas mediante la relación establecida por la duración de cada una de ellas y retorna al paso 3 manteniendo la relación.

4b. El usuario selecciona el botón para ordenar las recetas por dificultad.

1. El sistema ordena las recetas mediante la relación establecida por la dificultad de cada una de ellas y retorna al paso 3 manteniendo la relación.

4c. El usuario pulsa la tecla “atrás” del dispositivo.

1. El sistema cierra el listado de recetas y vuelve a comenzar el caso de uso.

6a. El usuario pulsa la tecla “atrás” del dispositivo.

1. El sistema cierra el resumen de la receta y vuelve al paso 3 del caso de uso.

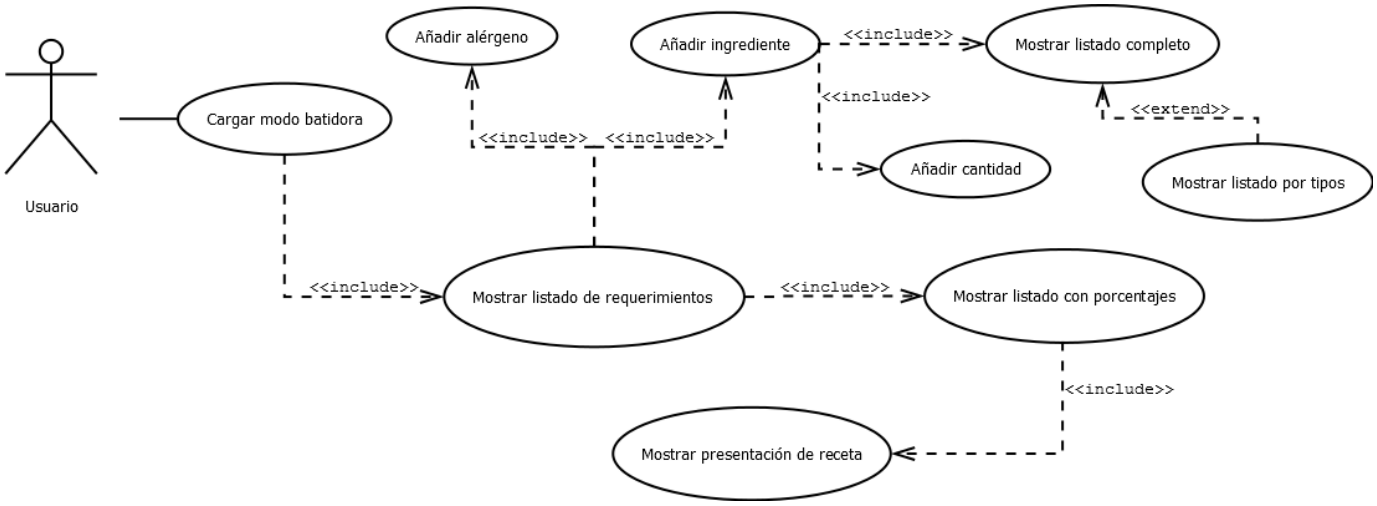


Figura 4.4: Diagrama de caso de uso cargar modo batidora

- **Caso de uso:** Cargar modo batidora.

Descripción: El usuario accede al modo batidora, pudiendo introducir ingredientes y su cantidad para buscar recetas que lo contengan y alérgenos para no mostrar recetas con ellos. Se expondrá un listado con los porcentajes de acierto con una serie de recetas y se podrán realizar.

Actores: Usuario.

Precondiciones: El usuario ha accedido al modo batidora desde el menú principal.

Postcondiciones: Ninguna.

Escenario principal:

1. El sistema muestra la pantalla del modo batidora.
2. El usuario selecciona añadir un ingrediente.
3. El sistema da a elegir entre listado completo o por tipos.
4. El usuario elige listado completo.
5. El sistema carga en una lista todos los ingredientes por orden alfabético.
6. El usuario selecciona un ingrediente cualquiera.
7. El sistema muestra una ventana para asociar una cantidad.
8. El usuario escribe una cantidad.
9. El sistema agrega el ingrediente y la cantidad a la lista de requerimientos.
10. El usuario pulsa el botón para obtener resultados.
11. El sistema compara los ingredientes con cantidades y alérgenos de todas las recetas con los de la lista de requerimientos.
12. El sistema crea y muestra una lista con recetas y sus aproximaciones ordenada de mayor a menor porcentaje de acierto.
13. El usuario selecciona una receta cualquiera del listado mostrado.
14. El sistema muestra el resumen de la receta.
15. El usuario pulsa en comenzar la receta.
16. El sistema carga los datos de la receta y da paso a ella.

Extensiones – flujo alternativo:

2a. El usuario pulsa la tecla “atrás” del dispositivo.

    1. El sistema cierra el modo batidora y vuelve a la pantalla de menú principal.

2b. El usuario selecciona añadir un alérgeno.

1. El sistema muestra el listado de alérgenos.
2. El usuario selecciona un alérgeno de la lista.
3. El sistema añade el alérgeno a la lista de requerimientos.

2c. El usuario selecciona añadir un alérgeno.

1. El sistema muestra el listado de alérgenos.
2. El usuario pulsa la tecla “atrás” del dispositivo.
3. El sistema cierra el listado de alérgenos y vuelve a comenzar el caso de uso.

4a. El usuario pulsa la tecla “atrás” del dispositivo en cualquier momento de la introducción de ingredientes.

1. El sistema cierra el listado de ingredientes y vuelve a comenzar el caso de uso.

10a. El usuario hace clic en introducir otro ingrediente.

1. El caso de uso vuelve al paso 3 del escenario principal.

10b. El usuario selecciona añadir un alérgeno.

1. El caso de uso toma el flujo del paso 2b de los escenarios alternativos.

13a. El usuario pulsa la tecla “atrás” del dispositivo.

1. El sistema cierra el listado de recetas y vuelve a comenzar el caso de uso.

15a. El usuario pulsa la tecla “atrás” del dispositivo.

1. El sistema cierra el resumen de la receta y vuelve al paso 12 del caso de uso.

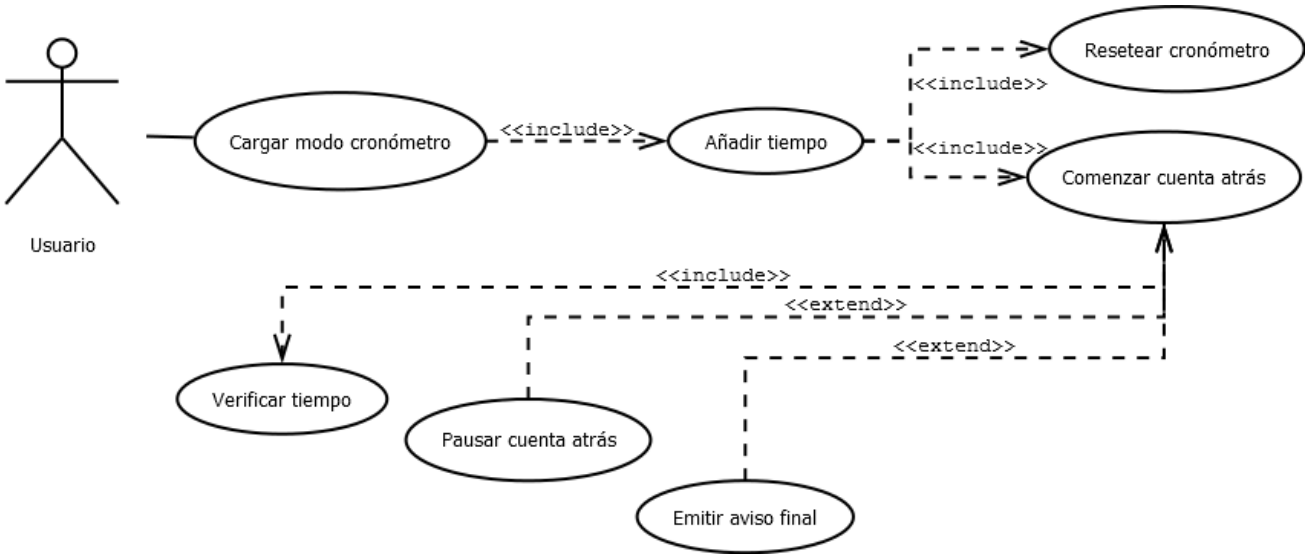


Figura 4.5: Diagrama de caso de uso cargar modo cronómetro

- **Caso de uso:** Cargar modo cronómetro.

Descripción: El usuario accede al modo cronómetro, pudiendo introducir un tiempo en horas, minutos y segundos para comenzar un reloj con cuenta regresiva, que avisará cuando llegue a cero.

Actores: Usuario.

Precondiciones: El usuario ha accedido al modo cronómetro desde el menú principal.

Postcondiciones: Ninguna.

Escenario principal:

1. El sistema muestra la pantalla del modo cronómetro.
2. El usuario introduce un tiempo en horas, minutos y segundos.
3. El usuario pulsa en el botón comenzar.
4. El sistema configura el cronómetro con el tiempo indicado por el usuario verificándolo.
5. El sistema comienza la cuenta atrás.
6. El sistema lanza el aviso cuando la cuenta atrás llegue a cero.

Extensiones – flujo alternativo:

2a. El usuario pulsa la tecla “atrás” del dispositivo.

   1. El sistema cierra el modo cronómetro y vuelve a la pantalla de menú principal.

6a. El usuario mantiene pulsado el botón reset.

1. El sistema se reestablece poniendo todo a cero volviendo al paso 1 del caso de uso.

6b. El usuario hace clic en el botón pausa.

1. El sistema detiene la cuenta atrás y conserva el tiempo.

2. El usuario selecciona continuar.

3. El sistema continúa con la cuenta atrás por donde había quedado parado el cronómetro.

4. El sistema lanza el aviso cuando la cuenta atrás llegue a cero.

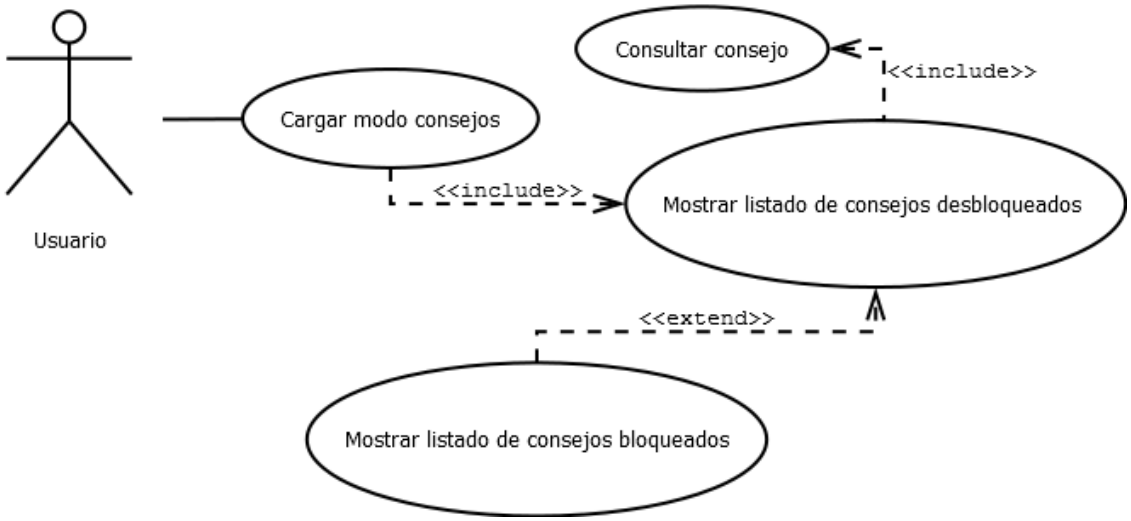


Figura 4.6: Diagrama de caso de uso cargar modo consejos

- **Caso de uso:** Cargar modo consejos.

Descripción: El usuario accede al modo consejos, pudiendo conocer los consejos desbloqueados y obteniendo la descripción de los mismos.

Actores: Usuario.

Precondiciones: El usuario ha accedido al modo consejos desde el menú principal.

Postcondiciones: Ninguna.

Escenario principal:

1. El sistema muestra la pantalla del modo consejos con el listado de consejos desbloqueados.

2. El usuario accede a un consejo.

3. El sistema muestra la descripción del consejo.

Extensiones – flujo alternativo:

2a. El usuario pulsa la tecla “atrás” del dispositivo.

1. El sistema cierra el modo consejos y vuelve a la pantalla de menú principal.

2b. El usuario elige la opción de consejos bloqueados.

1. El sistema muestra un resumen de los consejos sin desbloquear.

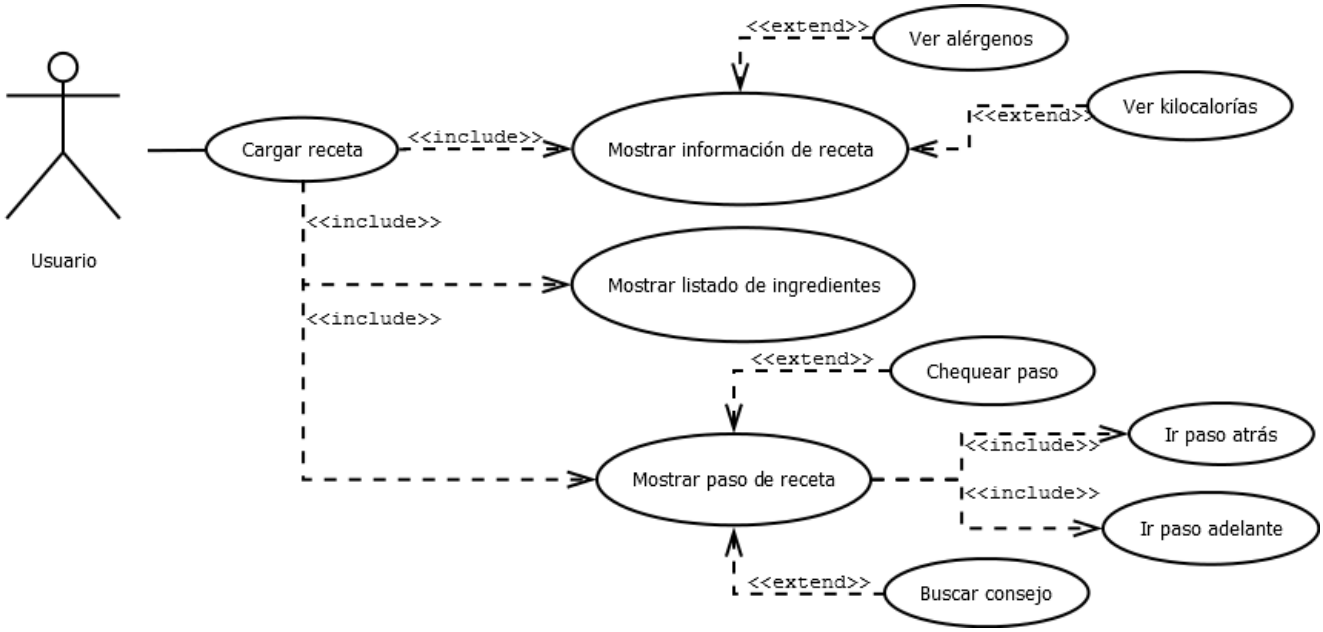


Figura 4.7: Diagrama de caso de uso cargar receta

- **Caso de uso:** Cargar receta.

Descripción: El usuario comienza una receta cualquiera de las que contiene la aplicación y recorre todas sus partes hasta que se finalizan los pasos.

Actores: Usuario.

Precondiciones: El usuario ha accedido a una receta cualquiera desde los modos de juego aventura, a cocinar o batidora.

Postcondiciones: Ninguna.

Escenario principal:

1. El sistema muestra la pantalla de presentación de una receta cualquiera.
2. El usuario pulsa en el botón comenzar.
3. El sistema muestra el listado de ingredientes con sus cantidades.
4. El usuario pulsa continuar para acceder a los pasos de la receta.
5. El sistema muestra el primer paso de la receta junto con todos los datos del mismo.
6. El usuario hace clic en siguiente para continuar con los pasos.
7. El sistema carga el paso pertinente.
8. El usuario pulsa siguiente y llega al último paso.
9. El sistema avisa de ello y ofrece tomar una foto del resultado de la receta.
10. El usuario pulsa el icono de "home".
11. El sistema regresa a la pantalla del modo desde el que se haya accedido a la receta.

Extensiones – flujo alternativo:

2a. El usuario pulsa la tecla "atrás" del dispositivo.

1. El sistema regresa a la pantalla del modo desde el que se haya accedido a la receta.

2b. El usuario selecciona el botón de kilocalorías.

1. El sistema muestra la cantidad de kilocalorías por cada 100 gramos de la receta en una ventana emergente.

2c. El usuario selecciona el botón de alérgenos.

1. El sistema muestra los alérgenos que contiene la receta en una ventana emergente.

4a. El usuario pulsa la tecla "atrás" del dispositivo o de la pantalla del listado.

1. El sistema regresa a la pantalla de la presentación de la receta.

6a. El usuario pulsa la tecla "atrás" del dispositivo o de la pantalla del paso.

1. El sistema regresa a la pantalla de la lista de ingredientes.

6b. El usuario pulsa sobre el pictograma de representación de la técnica.

1. Si el paso tiene consejo asociado, el sistema lo muestra en una ventana emergente.

6c. El usuario pulsa sobre la casilla de chequeo del paso, estando el paso no chequeado.

1. El sistema señala el paso como chequeado.

6d. El usuario pulsa sobre la casilla de chequeo del paso, estando el paso chequeado.

1. El sistema señala el paso como no chequeado.

8a. El usuario pulsa la tecla “atrás” del dispositivo o de la pantalla del paso.

1. El sistema regresa al paso anterior de la receta.

8b. El usuario pulsa sobre el pictograma de representación de la técnica.

1. Si el paso tiene consejo asociado, el sistema lo muestra en una ventana emergente.

8c. El usuario pulsa sobre la casilla de chequeo del paso, estando el paso no chequeado.

1. El sistema señala el paso como chequeado.

8d. El usuario pulsa sobre la casilla de chequeo del paso, estando el paso chequeado.

1. El sistema señala el paso como no chequeado.

10a. El usuario pulsa la tecla “atrás” del dispositivo o de la pantalla del paso.

1. El sistema regresa al penúltimo paso de la receta.

10a. El usuario pulsa la tecla “atrás” del dispositivo o de la pantalla del paso.

1. El sistema regresa al penúltimo paso de la receta.

10b. El usuario pulsa sobre el pictograma de realizar fotografía.

1. El sistema lanza la cámara del dispositivo.

2. El usuario toma la fotografía.

3. El sistema almacena la fotografía en el dispositivo y pregunta si se quiere compartir en las aplicaciones externas que el dispositivo permita.

4. El usuario elige una de ellas y la fotografía será exportada a la aplicación elegida para que haga uso de ella.

5. El sistema recupera el control tras salir de la aplicación externa mostrando la pantalla del paso final de la receta.

10c. El usuario pulsa sobre el pictograma de realizar fotografía.

1. El sistema lanza la cámara del dispositivo.

2. El usuario pulsa “atrás” en cualquier momento de la operación que comprende el flujo alternativo 10b.

3. El sistema cancela la acción y muestra la pantalla del paso final de la receta.

4.4 Modelo conceptual de datos

Continuamos con el estudio de las diferentes dependencias que existen entre los requisitos del sistema. Para ello, vamos a apoyarnos en la realización de un diagrama de clases de tipo entidad-relación, donde se verá expuesto claramente cómo se comunican las diferentes entidades relevantes que componen los casos de uso anteriormente comentados.

El siguiente diagrama entidad-relación representa de forma general cómo interactúan los diferentes casos de uso que hemos venido analizando. Cabe destacar que cuando haya que aplicarlo al diseño surgirán más clases derivadas de las expuestas para mejorar la consistencia y evitar problemas lógicos.

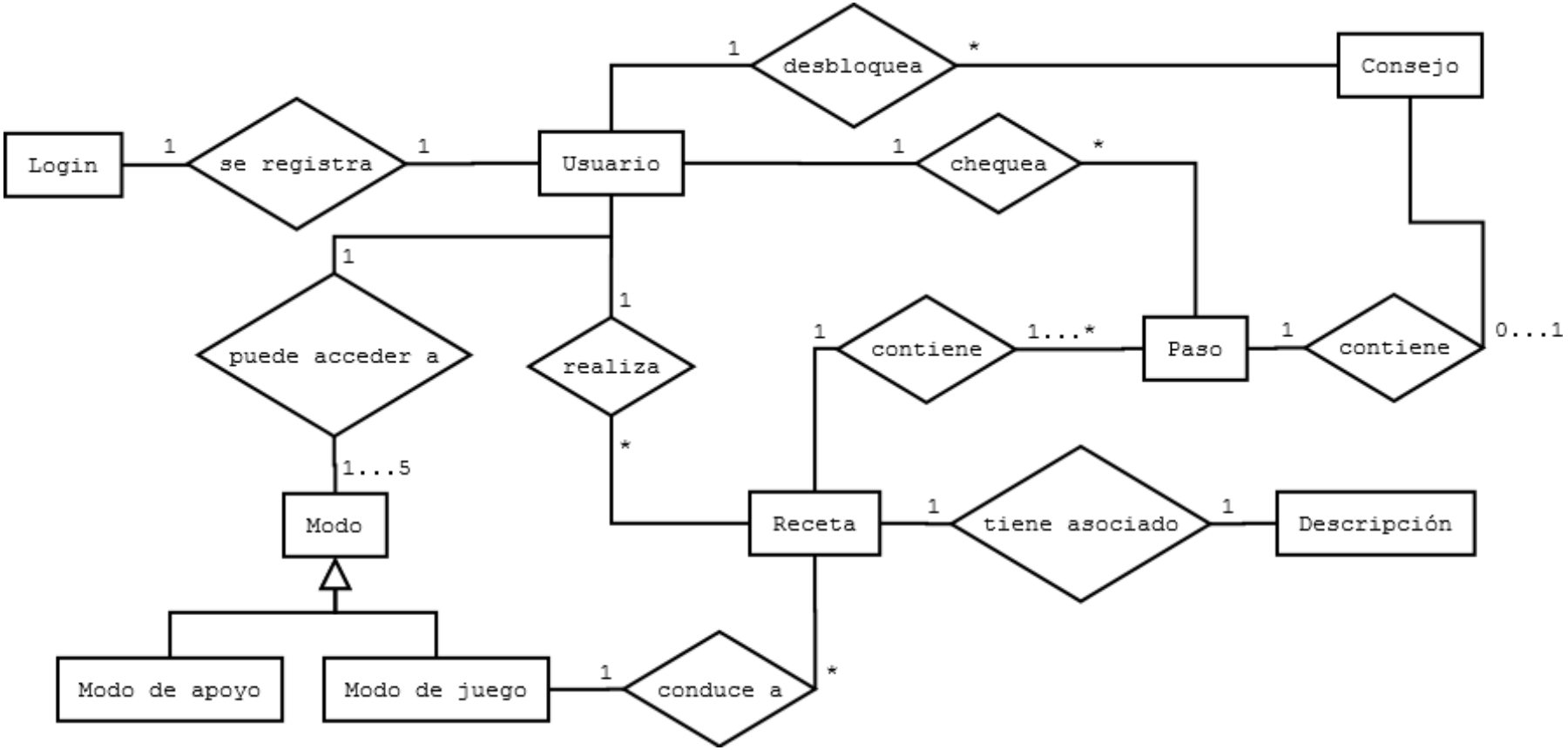


Figura 4.8: Análisis: Diagrama entidad-relación

4.5 Modelo de comportamiento del sistema

Para finalizar el análisis vamos a proceder a realizar la definición del comportamiento del sistema que tendrá dos apartados diferenciados: los diagramas de secuencia del sistema, que muestra los eventos que se suceden entre el actor (usuario) y el sistema, y los contratos de las operaciones, donde se define qué función específica tiene cada procedimiento.

Para no engrosar de sobremanera el contenido de la memoria de sobremanera, en este apartado nos centraremos en los escenarios más relevantes de cada caso de uso, intentando aportar el máximo de información y evitando la duplicidad de elementos de menor importancia.

Caso de uso: Iniciar juego

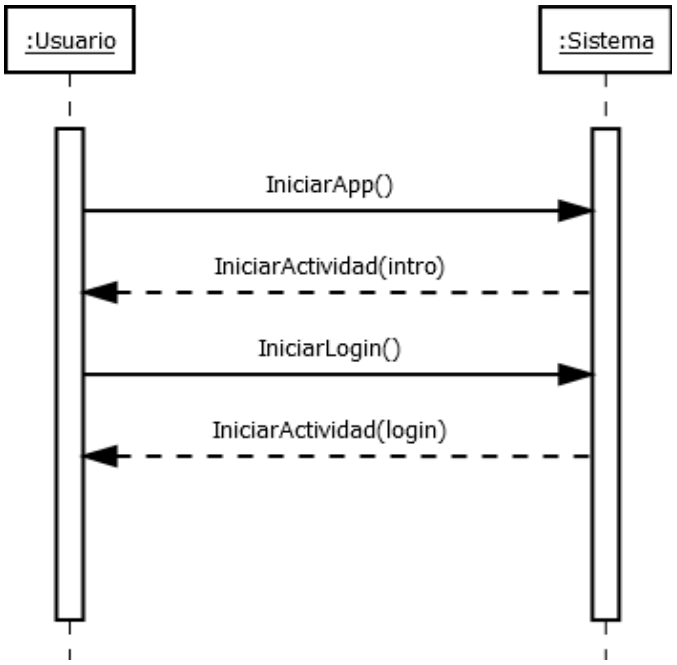


Figura 4.9: Análisis: Diagrama de secuencia Iniciar juego (escenario principal)

Operación: IniciarApp()

Responsabilidades: Inicia la aplicación y lanza la actividad que muestra la pantalla de introducción.

Precondiciones: Ninguna.

Postcondiciones: El sistema carga la aplicación y se muestra la pantalla de introducción.

Operación: IniciarLogin()

Responsabilidades: Lanza la actividad que muestra la pantalla de logueo de jugadores.

Precondiciones: La pantalla de introducción está lanzada y el usuario ha pulsado en ella.

Postcondiciones: Se muestra la pantalla de login.

Caso de uso: Logueo

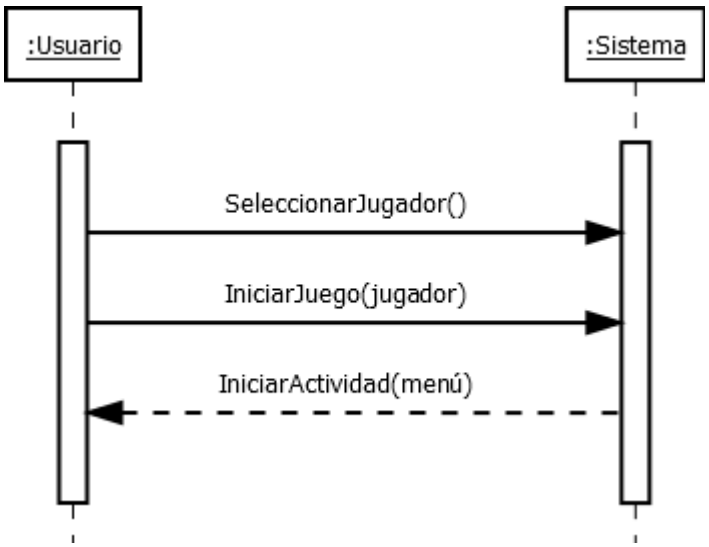


Figura 4.10: Análisis: Diagrama de secuencia Logueo (escenario principal)

Operación: SeleccionarJugador()

Responsabilidades: Se selecciona un jugador de la lista de jugadores disponibles.

Precondiciones: La pantalla de login está lanzada y el usuario accede a la lista de jugadores.

Postcondiciones: Un jugador queda seleccionado de forma visible.

Operación: IniciarJuego(jugador)

Responsabilidades: Lanza la actividad que muestra la pantalla de menú de juego cargando los datos del jugador seleccionado.

Precondiciones: La pantalla de login está lanzada y hay un jugador seleccionado.

Postcondiciones: Se muestra la pantalla de menú de juego.

Caso de uso: Logueo (escenario *b)

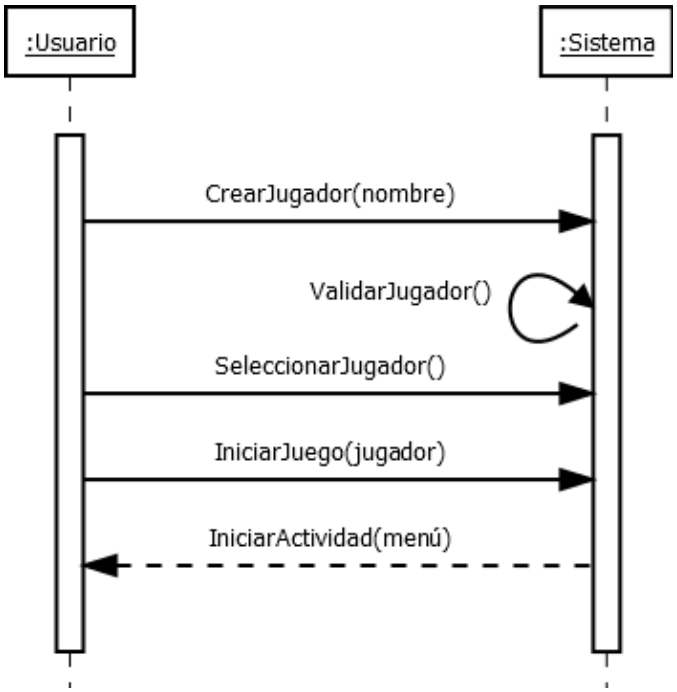


Figura 4.11: Análisis: Diagrama de secuencia Logueo (escenario *b)

Operación: CrearJugador(nombre)

Responsabilidades: Se crea un jugador con un nombre dado por el usuario, aplicando ValidarJugador().

Precondiciones: La pantalla de login está lanzada y el usuario accede a nuevo jugador.

Postcondiciones: Tras ValidarJugador(), se incorpora el nuevo nombre de jugador a la lista.

Operación: ValidarJugador()

Responsabilidades: Se revisa que el nombre de jugador aportado por el usuario sea válido y que no exista. No permitirá que el jugador se cree hasta que el nombre sea válido.

Precondiciones: La pantalla de crear jugador está lanzada y el usuario ha introducido un nuevo nombre para añadirlo a la lista de jugadores.

Postcondiciones: Tras validar el nuevo nombre de jugador, se prosigue con la creación del jugador.

Operación: EliminarJugador(nombre)

Responsabilidades: Se elimina un jugador seleccionado por el usuario.

Precondiciones: La pantalla de login está lanzada, el usuario ha seleccionado un jugador mediante SeleccionarJugador() y el usuario accede a eliminar jugador.

Postcondiciones: Se elimina el jugador seleccionado de la lista de jugadores y se borran todos sus datos.

Caso de uso: Menú principal

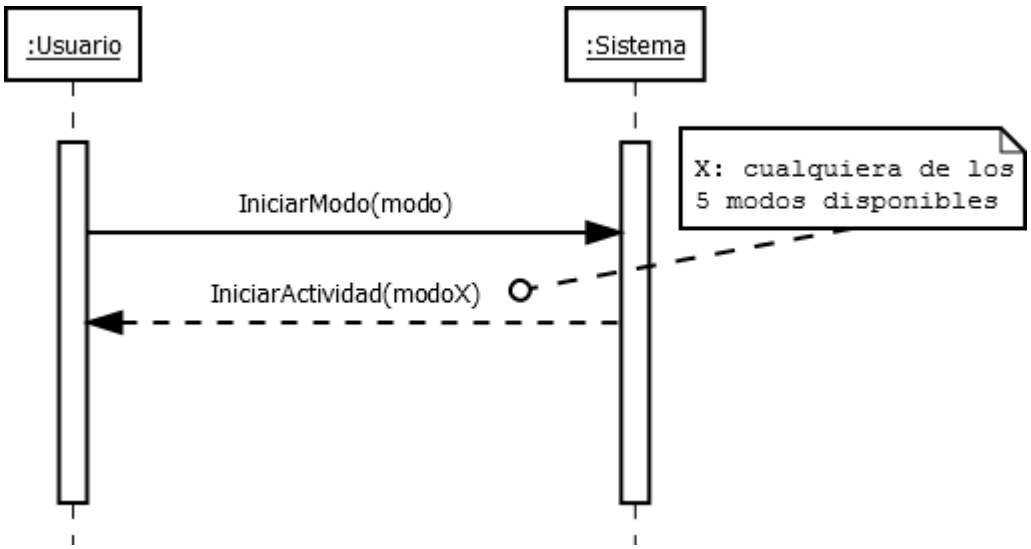


Figura 4.12: Análisis: Diagrama de secuencia Menú principal (escenario principal)

Operación: IniciarModo(modo)

Responsabilidades: Lanza la actividad que muestra cualquiera de las pantallas de las cinco modalidades de juego, dependiendo cuál elija el usuario.

Precondiciones: La pantalla de menú de juego está lanzada.

Postcondiciones: Se muestra la pantalla de uno de los cinco modos de juego.

Caso de uso: Modo aventura

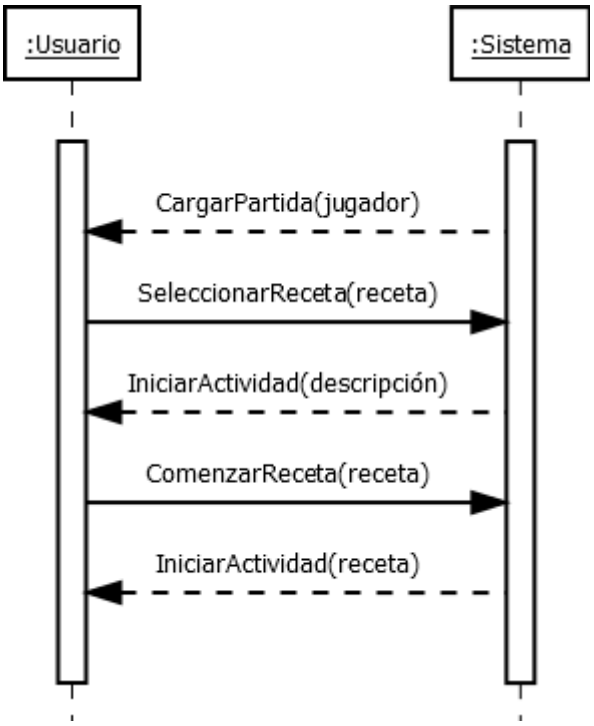


Figura 4.13: Análisis: Diagrama de secuencia Modo aventura (escenario principal)

Operación: IniciarModo(aventura)

Responsabilidades: Lanza la actividad que muestra la pantalla con las cinco recetas de la aventura si se ha iniciado anteriormente o carga cinco recetas aleatorias, de diferentes tipos y que preferiblemente no se hayan chequeadas antes por el jugador.

Precondiciones: El usuario ha seleccionado el modo aventura en el menú de juego.

Postcondiciones: Se muestra la pantalla del modo aventura con cinco recetas seleccionables.

Operación: SeleccionarReceta(receta)

Responsabilidades: Lanza la actividad que muestra la pantalla con el resumen de la receta que haya sido seleccionada por el usuario.

Precondiciones: El usuario ha seleccionado una receta disponible del modo en cuestión.

Postcondiciones: Se muestra la pantalla de resumen de receta con los datos de la receta seleccionada.

Operación: ComenzarReceta(receta)

Responsabilidades: Lanza la actividad que muestra la pantalla de introducción de la receta seleccionada por el usuario, cargando todos los datos necesarios para proceder al visualizado de la información relacionada con la misma y su consecución.

Precondiciones: El usuario ha accedido a comenzar una receta cualquiera a través de su pantalla de resumen, habiendo accedido desde alguno de los modos habilitados para ello, como lo son el modo aventura, el modo a cocinar y el modo batidora.

Postcondiciones: Se muestra la pantalla de inicio para la receta seleccionada y los botones de acceso a la información de la misma, como son las kilocalorías o los alérgenos que contiene. Además, se visualizará si la receta ha sido chequeada mediante el indicador para este cometido.

Caso de uso: Modo aventura (escenario 2b)

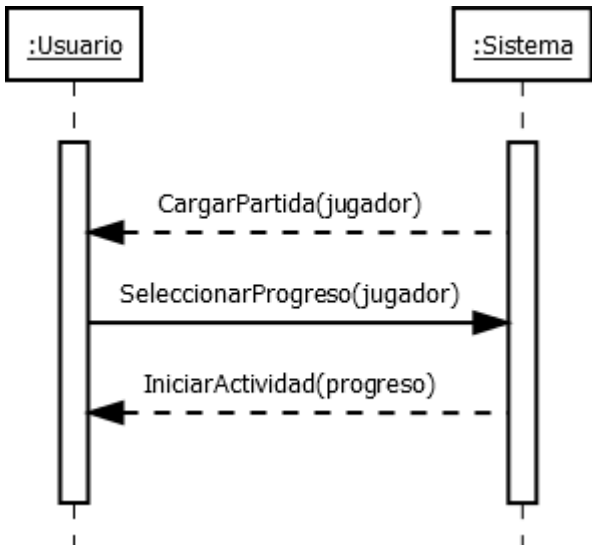


Figura 4.14: Análisis: Diagrama de secuencia Modo aventura (escenario 2b)

Operación: SeleccionarProgreso(jugador)

Responsabilidades: Lanza la actividad que muestra la pantalla de progreso donde se visualiza cuántas veces ha finalizado el usuario “jugador” una aventura.

Precondiciones: El usuario ha seleccionado el botón de progreso en la pantalla del modo aventura.

Postcondiciones: Se muestra la pantalla de progreso del modo aventura con las estadísticas del jugador dentro de este modo.

Caso de uso: Modo aventura (escenario 2c)

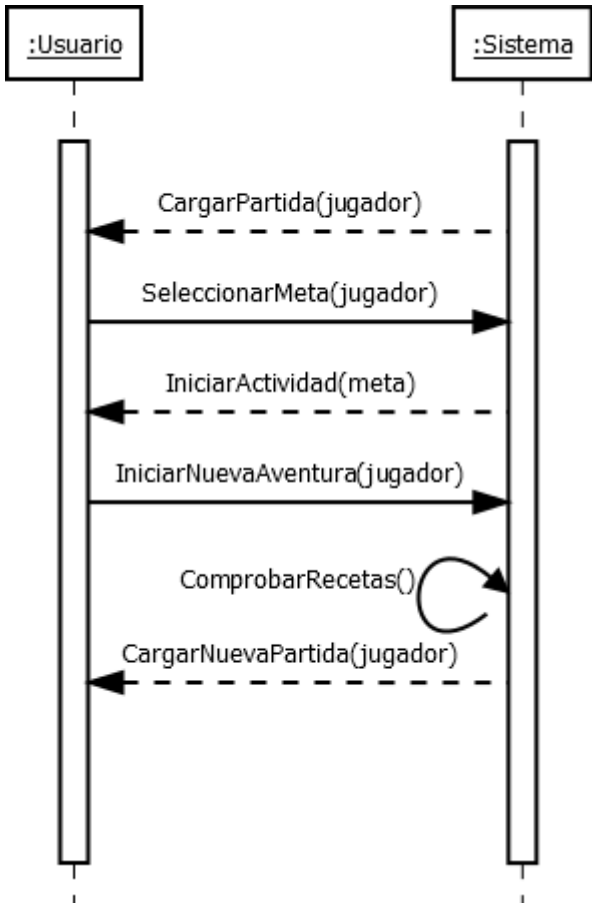


Figura 4.15: Análisis: Diagrama de secuencia Modo aventura (escenario 2c)

Operación: SeleccionarMeta(jugador)

Responsabilidades: Lanza la actividad que muestra la pantalla de meta donde se visualiza las recetas de la aventura actual que le quedan al usuario "jugador" para completarla.

Precondiciones: El usuario ha seleccionado la opción de meta en la pantalla del modo aventura.

Postcondiciones: Se muestra la pantalla de meta del modo aventura con la información del jugador sobre la aventura actual.

Operación: IniciarNuevaAventura(jugador)

Responsabilidades: Se cargan cinco nuevas recetas aplicando ComprobarRecetas().

Precondiciones: La pantalla de meta se encuentra lanzada y el usuario ha seleccionado nueva aventura.

Postcondiciones: Tras ComprobarRecetas (), se eligen cinco nuevas recetas de diferentes tipos que preferiblemente no hayan sido chequeadas por el jugador anteriormente y se vuelve a mostrar la pantalla del modo aventura con cinco nuevas recetas seleccionables.

Operación: ComprobarRecetas()

Responsabilidades: Se revisa que el jugador que haya sido seleccionado por el usuario tenga chequeadas todas las recetas del modo aventura actual. En caso contrario, avisa de que faltan recetas por chequear.

Precondiciones: El usuario ha seleccionado nueva aventura en la pantalla de meta y ha confirmado la decisión.

Postcondiciones: Tras revisar si todas las recetas están chequeadas, prosigue con el inicio de la nueva aventura para dicho jugador.

Caso de uso: Modo a cocinar

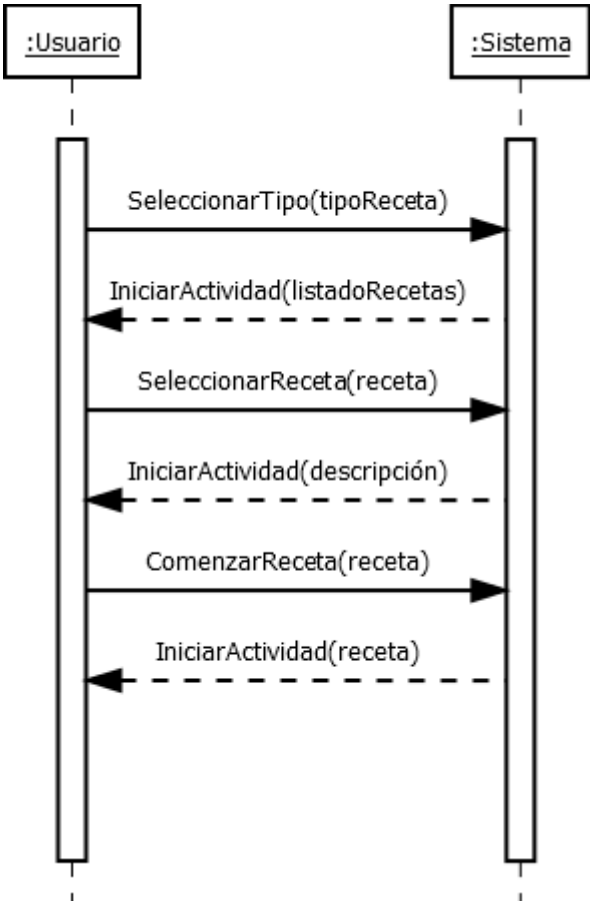


Figura 4.16: Análisis: Diagrama de secuencia Modo a cocinar (escenario principal)

Operación: SeleccionarModo(aCocinar)

Responsabilidades: Lanza la actividad que muestra la pantalla de selección de tipos de receta donde se visualizan los diferentes iconos seleccionables asociados a cada tipo.

Precondiciones: El usuario ha seleccionado el modo a cocinar en el menú de juego.

Postcondiciones: Se muestra la pantalla principal del modo a cocinar con los correspondientes iconos de tipos de receta seleccionables.

Operación: SeleccionarTipo(tipoReceta)

Responsabilidades: Lanza la actividad que muestra un listado en la pantalla con las recetas asociadas a dicho tipo “tipoReceta”.

Precondiciones: El usuario ha seleccionado un tipo de receta dentro de la pantalla que contiene los iconos con los tipos de receta.

Postcondiciones: Se muestra la pantalla con el listado de recetas contempladas en el tipo de receta seleccionado.

Caso de uso: Modo a cocinar (escenario 4a / 4b)

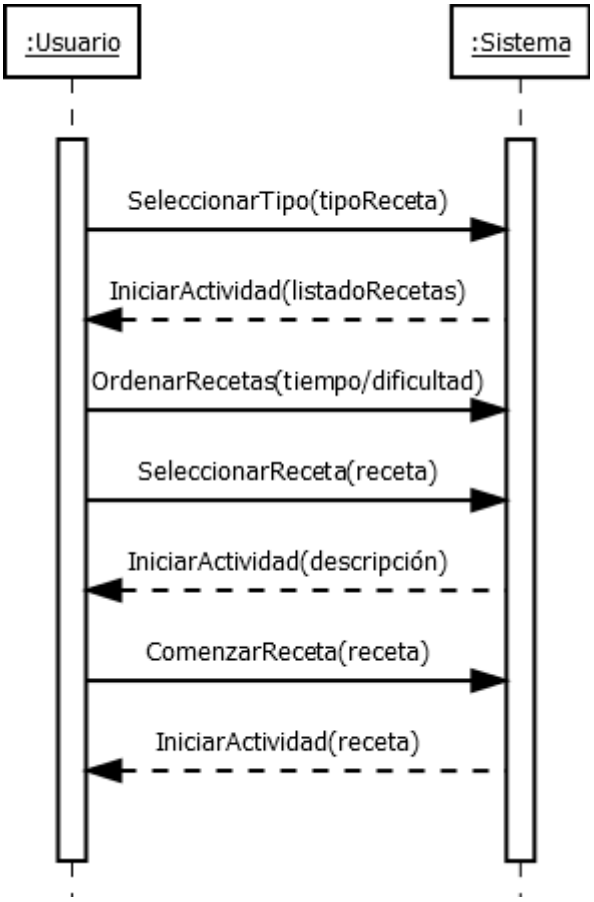


Figura 4.17: Análisis: Diagrama de secuencia Modo a cocinar (escenario 4a / 4b)

Operación: OrdenarRecetas(tiempo / dificultad)

Responsabilidades: Se ordenan las recetas del listado de recetas disponibles, de un tipo en cuestión, por tiempo de realización o por dificultad, de mayor a menor o al contrario, dependiendo del icono que esté seleccionado.

Precondiciones: El listado de recetas de un tipo en cuestión se encuentra visualizado en pantalla, tras haber accedido a él mediante la selección de dicho tipo.

Postcondiciones: Las recetas quedan ordenadas de menor a mayor tiempo si se selecciona el icono de tiempo, de mayor a menor tiempo si se deselecciona, de menor a mayor dificultad si se selecciona el icono de dificultad, y de mayor a menor dificultad si se deselecciona.

Caso de uso: Modo batidora (escenario principal / 10a)

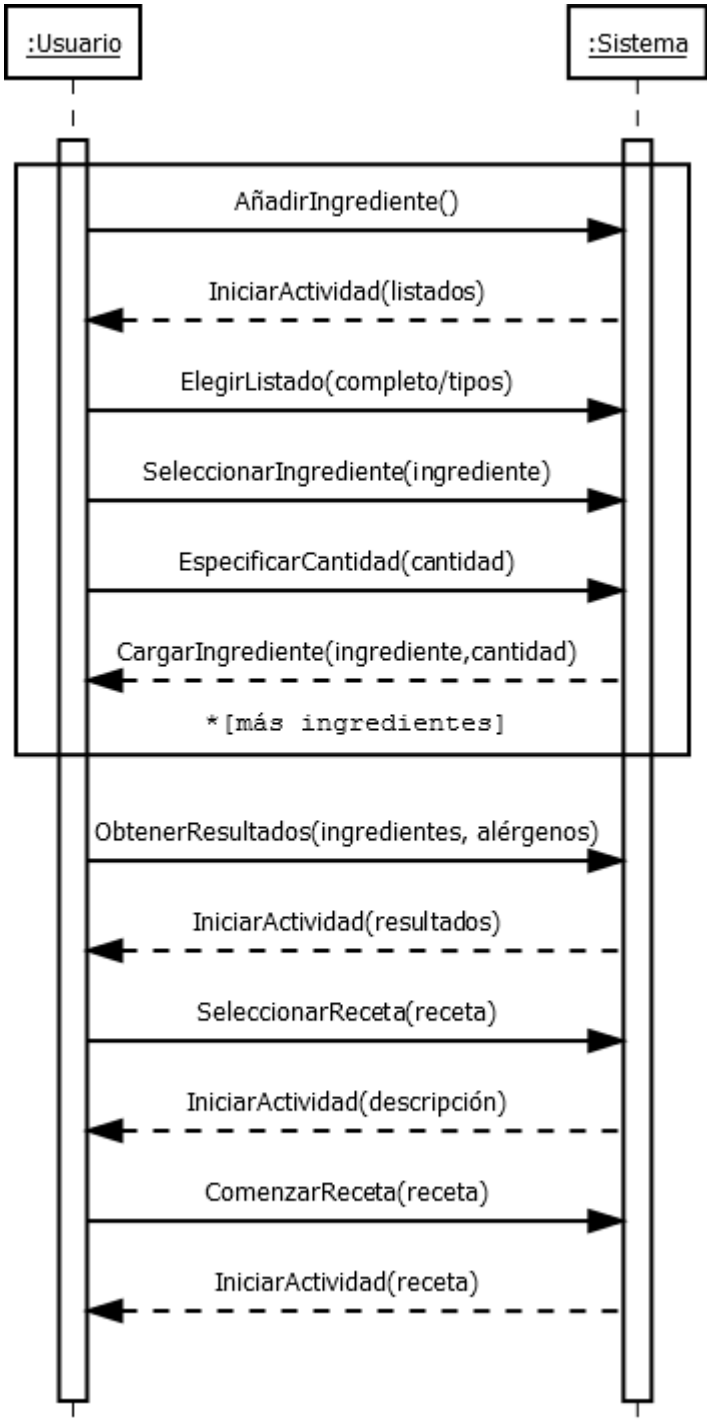


Figura 4.18: Análisis: Diagrama de secuencia Modo batidora (escenario principal / 10a)

Operación: SeleccionarModo(batidora)

Responsabilidades: Lanza la actividad que muestra la pantalla del modo batidora, donde se ofrece la opción de añadir alérgenos que no contengan las recetas e ingredientes que formen parte de las mismas, y proceder a la obtención de resultados procesando el listado inicialmente vacío.

Precondiciones: El usuario ha seleccionado el modo batidora en el menú de juego.

Postcondiciones: Se muestra la pantalla principal del modo batidora con los botones para añadir alérgenos e ingredientes, un listado vacío para añadir éstos y otro botón para obtener resultados.

Operación: AñadirIngredientes()

Responsabilidades: Lanza la actividad que muestra las dos opciones que existen para visualizar los ingredientes disponibles: por tipos de ingredientes o por orden alfabético.

Precondiciones: El usuario ha seleccionado la opción para añadir ingredientes que se encuentra en la pantalla del modo batidora.

Postcondiciones: Se visualiza un par de botones que reflejan las diferentes opciones que se contemplan para visualizar el listado de ingredientes, que son por tipos de ingredientes o todos ellos en orden alfabético.

Operación: ElegirListado(completo / tipos)

Responsabilidades: Carga y muestra el modo de listado de ingredientes que haya sido seleccionado por el usuario, descartando los que ya estén añadidos a la lista del modo batidora, y acotando por tipos si se elige esta opción.

Precondiciones: El usuario ha seleccionado cualquiera de las dos opciones disponibles para que se muestre el listado, completo o dividido por tipos de ingrediente.

Postcondiciones: Se muestra el listado según haya elegido el usuario una opción u otra, con la posibilidad de seleccionar un ingrediente del mismo para continuar con su inclusión en la lista de la batidora.

Operación: SeleccionarIngrediente(ingrediente)

Responsabilidades: Guarda la selección de un ingrediente realizada por el usuario de la lista de ingredientes disponible mostrada tras la elección de un tipo de listado.

Precondiciones: Se ha accedido al listado de ingredientes mediante la selección de una de las dos posibilidades, mostrando los ingredientes disponibles en ese mismo instante, eligiéndose uno de ellos.

Postcondiciones: Se guarda el ingrediente elegido por el usuario para continuar con su inclusión en la lista de la batidora.

Operación: EspecificarCantidad(ingrediente, cantidad)

Responsabilidades: Adjunta una cantidad concreta al ingrediente seleccionado por el usuario, que dependerá de la unidad de medida del mismo, y la añade a la lista del modo batidora. Si no se indica, la cantidad podrá ser la máxima necesitada en cualquier receta.

Precondiciones: Se ha seleccionado con anterioridad por parte del usuario un ingrediente disponible en el listado, que tiene ligado un tipo de unidad de medida.

Postcondiciones: Se enlaza la cantidad especificada por el usuario al ingrediente y estos datos pasan a visualizarse en el listado del modo batidora, cerrándose la pantalla de añadir ingrediente.

Operación: ObtenerResultados(ingrediente, alérgenos)

Responsabilidades: Invoca a la actividad que realiza una búsqueda entre todas las recetas comparando ingredientes y sus cantidades y descartando los que contengan algún tipo de alérgeno de los que se hallan en la lista de la batidora, dando como resultado un listado de recetas junto con un porcentaje de aproximación de su posible realización.

Precondiciones: Se ha añadido al menos un ingrediente y cero o varios alérgenos a la lista de la batidora mediante las operaciones anteriores y se ha seleccionado batir en la pantalla principal del modo batidora.

Postcondiciones: Se muestra en una nueva pantalla las posibles recetas realizables dependiendo de los alimentos y los alérgenos seleccionados por el usuario, mostrándose un listado de recetas seleccionables.

Caso de uso: Modo batidora (escenario 2b / 10b)

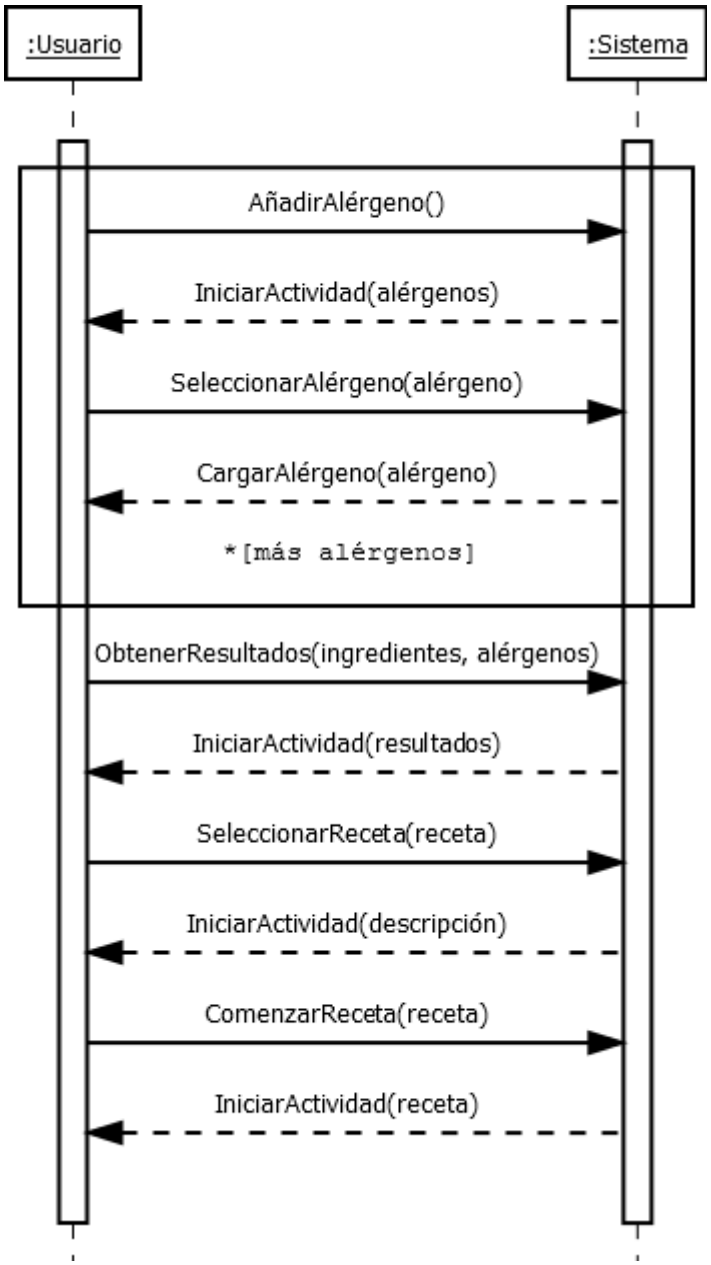


Figura 4.19: Análisis: Diagrama de secuencia Modo batidora (escenario 2b / 10b)

Operación: AñadirAlérgeno()

Responsabilidades: Lanza la actividad que muestra las lista de todos los alérgenos oficialmente reconocidos para poder seleccionarlos y excluirlos de los resultados de la batidora.

Precondiciones: El usuario ha seleccionado la opción para añadir alérgeno que se encuentra en la pantalla del modo batidora.

Postcondiciones: Se visualiza la lista de los alérgenos no seleccionados por orden alfabético para poder seleccionarlos y añadirlos a la lista de la batidora.

Operación: SeleccionarAlérgeno(alérgeno)

Responsabilidades: Guarda la selección de un alérgeno realizada por el usuario de la lista de alérgenos disponibles y la añade al listado de la batidora para que excluirlo de las recetas.

Precondiciones: Se ha accedido al listado de alérgenos mediante la pantalla principal del modo batidora, mostrando los alérgenos disponibles en ese mismo instante, eligiéndose uno de ellos.

Postcondiciones: Se guarda el alérgeno elegido por el usuario y se incluye en el listado de elementos de la pantalla principal del modo batidora para excluirlo del resultado de las recetas.

Caso de uso: Modo cronómetro (escenario principal / 6b)

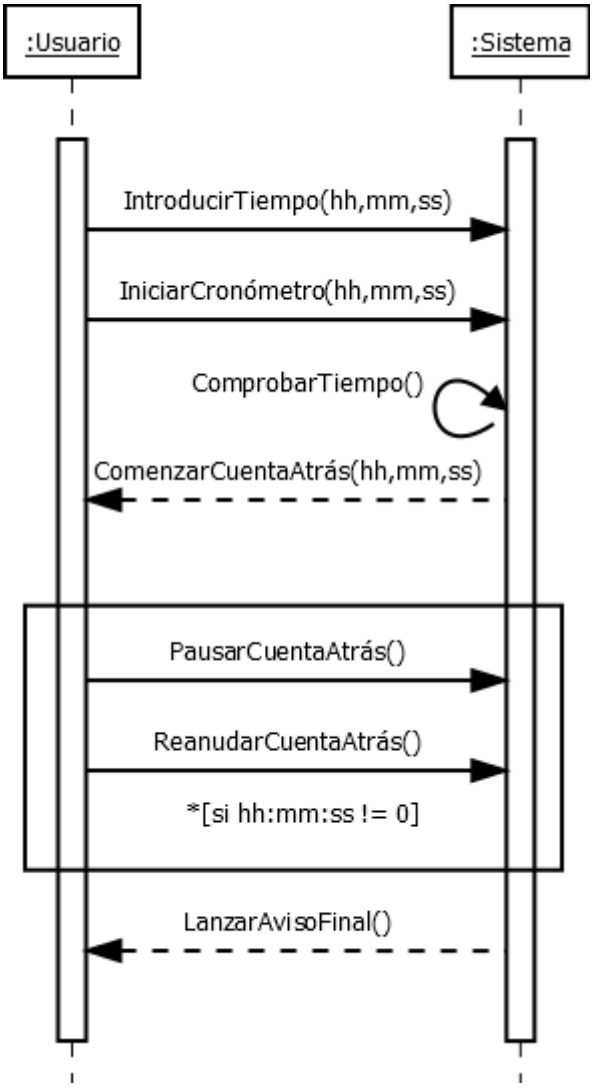


Figura 4.20: Análisis: Diagrama de secuencia Modo cronómetro (escenario principal / 6b)

Operación: SeleccionarModo(cronómetro)

Responsabilidades: Lanza la actividad que muestra la pantalla del modo cronómetro, donde se puede establecer una nueva cuenta atrás, visualizar el estado, pausarla y resetearla.

Precondiciones: El usuario ha seleccionado el modo cronómetro en el menú de juego.

Postcondiciones: Se muestra la pantalla principal del modo cronómetro con los botones para iniciar una nueva cuenta atrás mediante la introducción de horas, minutos y segundos e interactuar con ella.

Operación: IntroducirTiempo(hh, mm, ss)

Responsabilidades: Permite al usuario introducir valores numéricos en tres casillas diferentes, correspondientes a horas, minutos y segundos.

Precondiciones: El usuario ha seleccionado el modo cronómetro en el menú de juego, y la pantalla está preparada para introducir valores numéricos para el tiempo.

Postcondiciones: Se guardan los valores numéricos de horas, minutos y segundos que el usuario haya introducido en cada casilla designada para ello, quedando a la espera de comenzar la cuenta atrás.

Operación: IniciarCronómetro(hh, mm, ss)

Responsabilidades: Lanza la cuenta atrás del tiempo introducido por el usuario en los recuadros destinados para dicho cometido, visualizándose de forma gráfica en la pantalla principal de la actividad.

Precondiciones: El usuario ha introducido algún valor numérico en alguna de las tres casillas destinadas para ello.

Postcondiciones: Se cargan los valores numéricos de tiempo introducidos por el usuario en los recuadros de horas, minutos y segundos en el cronómetro principal, comenzando la cuenta atrás, no sin antes haber validado el dato de tiempo aportado por el usuario mediante ComprobarTiempo(). Lanza un aviso gráfico y sonoro cuando la cuenta atrás llega a cero.

Operación: ComprobarTiempo()

Responsabilidades: Se revisa que el usuario haya introducido valore de tiempo en horas, minutos y segundos correctos, de tal forma que no se pueda comenzar la cuenta atrás hasta que los valores no sean correctos.

Precondiciones: El usuario ha introducido valores numéricos de horas, minutos y segundos y ha pulsado el botón que inicia la cuenta atrás.

Postcondiciones: Da paso a la continuación de la cuenta atrás si los valores de segundos están entre 0 y 59, los de minutos entre 0 y 59, y los de horas entre cero y un valor máximo razonable, como es 5. Si todo es cero, no hace nada.

Operación: PausarCuentaAtrás()

Responsabilidades: Para la cuenta atrás en el tiempo indicado en ese momento por el reloj cronómetro principal de la pantalla.

Precondiciones: La cuenta atrás está en curso y el usuario decide seleccionar pausa para detenerla.

Postcondiciones: Se detiene la cuenta atrás conservando el valor de tiempo visualizado en el cronómetro principal del modo, mostrando gráficamente el estado en el momento de la pausa.

Operación: ReanudarCuentaAtrás()

Responsabilidades: Reanuda la cuenta atrás en el tiempo indicado en ese momento por el reloj cronómetro principal de la pantalla.

Precondiciones: La cuenta atrás se ha comenzado y pausado por el usuario en algún momento anterior, y decide continuarla.

Postcondiciones: Se reanuda la cuenta atrás a partir del valor de tiempo visualizado en el cronómetro principal del modo.

Caso de uso: Modo cronómetro (escenario 6c)

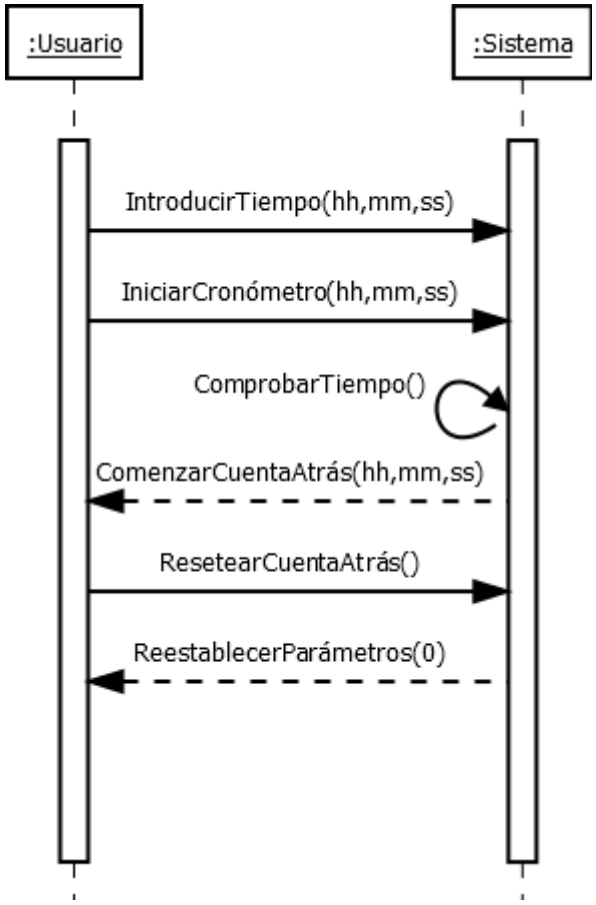


Figura 4.21: Análisis: Diagrama de secuencia Modo cronómetro (escenario 6c)

Operación: ResetearCuentaAtrás()

Responsabilidades: Devuelve todos los parámetros del cronómetro a su estado inicial, poniendo todos los datos a cero, tanto las casillas para configurar el tiempo como el cronómetro principal.

Precondiciones: Ninguna.

Postcondiciones: Se reinician todos los valores del cronómetro estableciendo todos los parámetros a cero, tanto los elementos gráficos como los parámetros internos, volviendo a la configuración inicial del modo cronómetro.

Caso de uso: Modo consejos

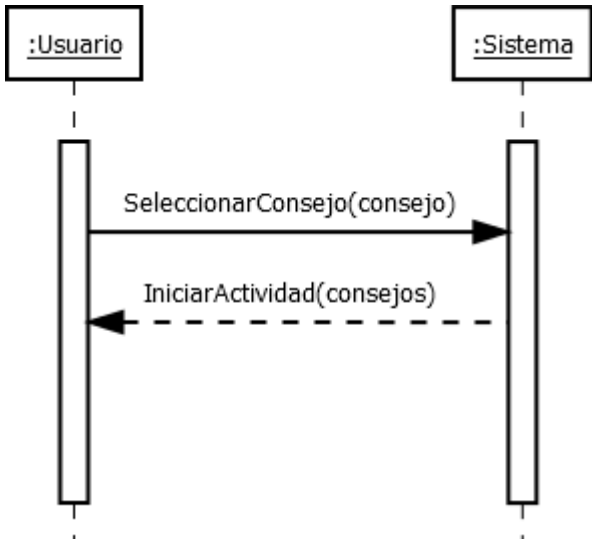


Figura 4.22: Análisis: Diagrama de secuencia Modo consejos (escenario principal)

Operación: SeleccionarModo(consejos)

Responsabilidades: Lanza la actividad que muestra la pantalla del modo consejos, donde se visualiza el listado del resumen ampliable de los consejos desbloqueados, el apartado de consejos aún sin desbloquear y el número de consejos desbloqueados sobre los que aún no lo están.

Precondiciones: El usuario ha seleccionado el modo consejos en el menú de juego.

Postcondiciones: Se muestra la pantalla principal del modo consejos donde aparece el número de consejos desbloqueados sobre los totales, un listado de consejos que se han ido desbloqueando en otros modos, con una breve descripción de cada uno, el tipo de técnica con el que tienen relación y las recetas donde se desbloquean. Si no se han desbloqueado consejos, no aparecerá ningún elemento en la lista.

Operación: SeleccionarConsejo()

Responsabilidades: Permite visualizar en pantalla la totalidad del consejo que haya sido seleccionado por el usuario.

Precondiciones: El usuario ha seleccionado el modo consejos en el menú de juego y hay al menos un consejo desbloqueado desde otros modos de juego para que se visualice en el listado y se pueda seleccionar.

Postcondiciones: Se abre una ventana emergente con la explicación completa del consejo acompañada de la técnica con el que está relacionado y de la receta del que proviene.

Caso de uso: Modo consejos (escenario 2b)

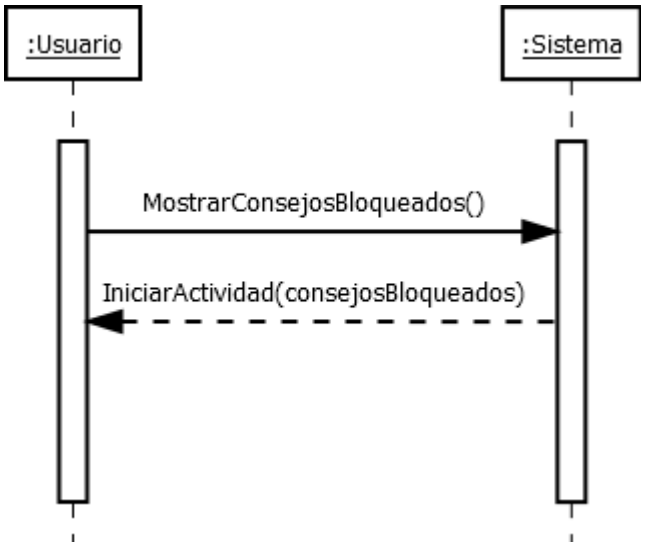


Figura 4.23: Análisis: Diagrama de secuencia Modo consejos (escenario 2b)

Operación: MostrarConsejosBloqueados()

Responsabilidades: Se lanza la actividad que muestra un listado con el resumen, tipo de técnica culinaria y receta de procedencia de los consejos que aún no han sido desbloqueados por el jugador.

Precondiciones: El usuario ha seleccionado el modo consejos en el menú de juego, ha accedido a los consejos no desbloqueados mediante el botón que existe para ello y hay al menos un consejo sin desbloquear entre todas las recetas.

Postcondiciones: Se muestra una pantalla donde aparece el listado de consejos que aún no han sido desbloqueados en el caso de que haya al menos un consejo que no haya sido desbloqueado de entre todas las recetas, donde se podrá apreciar un resumen del consejo, la técnica que tiene ligada y la receta donde desbloquearlo. Si todos los consejos han sido desbloqueados, mostrará un mensaje avisando de que todos los consejos están desbloqueados.

Caso de uso: Cargar recetas

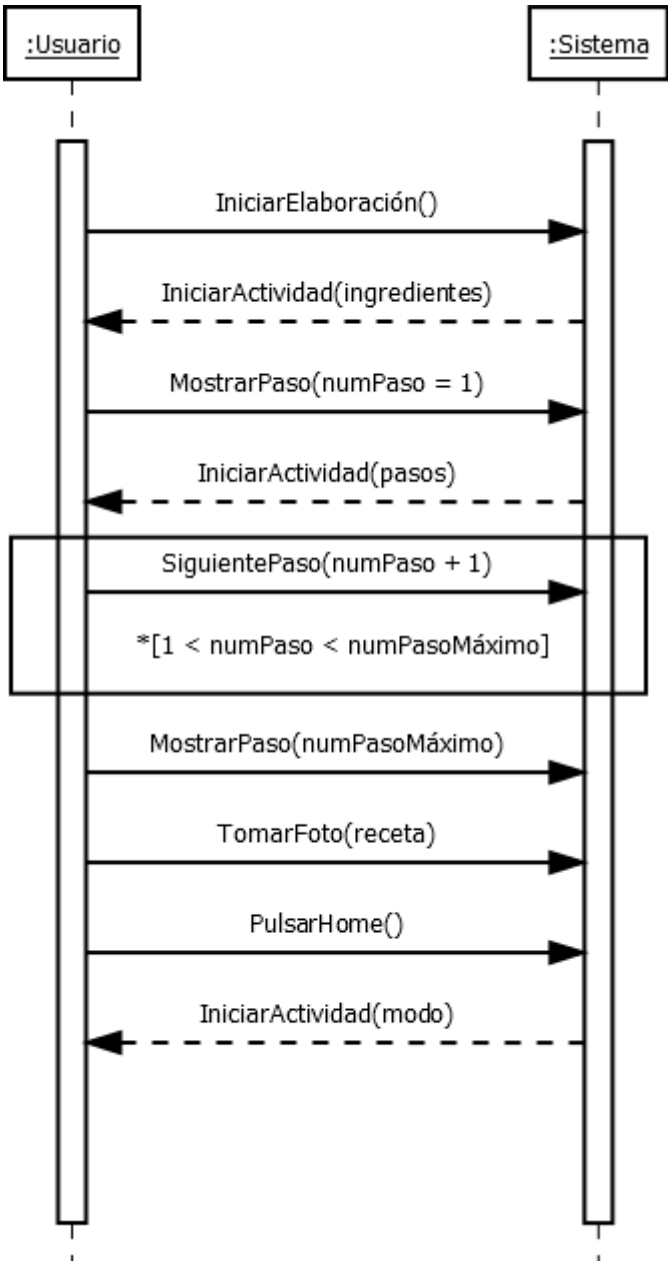


Figura 4.24: Análisis: Diagrama de secuencia Cargar receta (escenario principal)

Operación: IniciarElaboración(receta)

Responsabilidades: Lanza la actividad que muestra la pantalla del listado de ingredientes necesarios para llevar la receta seleccionada a cabo.

Precondiciones: El usuario ha pulsado la opción continuar con la elaboración de la receta que haya elegido desde cualquiera de los modos de juego con la opción de completar recetas disponible.

Postcondiciones: Se muestra la pantalla con la lista de ingredientes necesarios para poder realizar la receta acompañados de la cantidad necesaria de cada uno, además de los botones de navegación para poder acceder a los diferentes pasos.

Operación: MostrarPaso(numPaso)

Responsabilidades: Lanza la actividad que coordina el visualizado de cada uno de los pasos de una receta, configurando la información necesaria para que el usuario pueda seguirlos, dependiendo del número del paso en cuestión y permitiendo avanzar o retroceder en la secuencia.

Precondiciones: El usuario accede al visualizado de los pasos tras haberse mostrado los ingredientes necesarios para la receta.

Postcondiciones: Se carga en pantalla la descripción del paso, mostrando el orden que tiene dentro de la secuencia, junto con el icono que del tipo de técnica que está ligada a dicho paso y el estado de chequeo. Si se accede desde la lista de ingredientes, se mostrará el primer paso, y si el paso es el último de la receta, permitirá tomar una foto y compartirla mediante las diferentes aplicaciones del dispositivo que lo permitan. En los pasos que tengan ligados un tiempo de realización se dará la opción de lanzar el modo cronómetro con las horas, minutos y segundos que sean necesarios de forma predefinida.

Operación: SiguientePaso(numPaso + 1)

Responsabilidades: Carga en pantalla el paso siguiente en la secuencia de la receta, junto con la descripción del mismo y el icono de tipo de técnica que tiene ligado. Cuando llegue al último paso, dará la opción para tomar foto y compartir receta.

Precondiciones: El usuario ha iniciado el visualizado de pasos de la receta y se encuentra en uno de ellos.

Postcondiciones: Se muestra la información del paso siguiente a numPaso, según la secuencia de realización de la receta. Si es el último paso, permite hacer una foto y compartirla mediante medios externos a la aplicación.

Operación: TomarFoto(receta)

Responsabilidades: Lanza el servicio de la cámara que permite tomar una foto y la guarda en una carpeta en la galería de imágenes destinada para la aplicación.

Precondiciones: La receta se encuentra en su último paso y el usuario decide pulsar en el icono de captura imagen de la receta.

Postcondiciones: Se ejecuta la cámara del dispositivo para permitir tomar una foto de la receta finalizada y se guarda la captura en la carpeta de la galería para la aplicación.

Operación: PulsarHome(numPaso)

Responsabilidades: La receta se cierra regresando al punto previo desde el que se inició, continuando con el modo de juego desde donde se hubiera accedido a ella.

Precondiciones: Ninguna.

Postcondiciones: Finaliza la receta y devuelve el flujo de la aplicación a la pantalla del modo de juego desde donde se accedió, recargando la receta por si se hubiera chequeado completamente.

Caso de uso: Cargar recetas (escenario 2b / 2c)

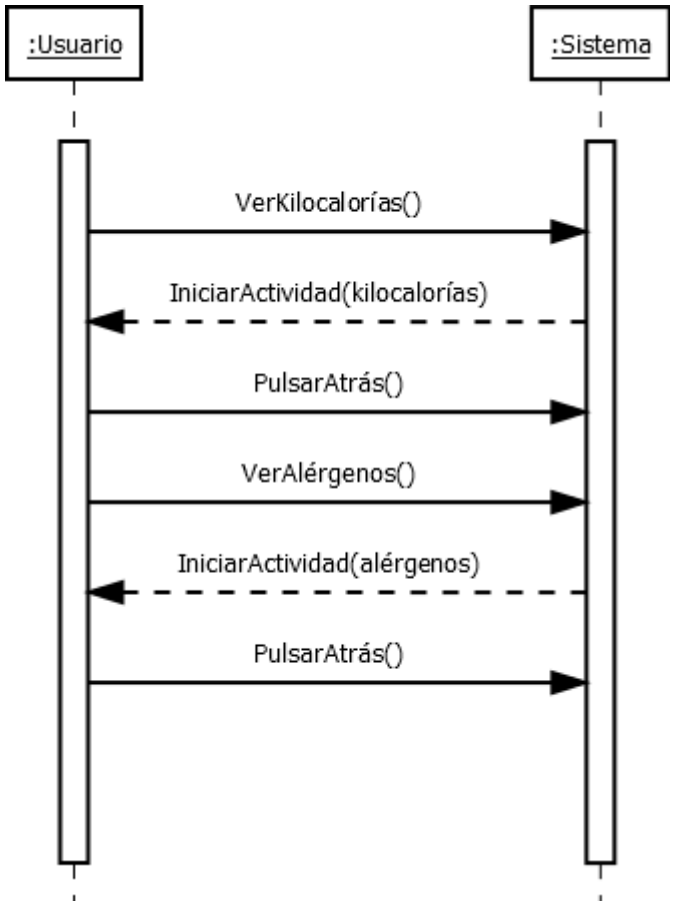


Figura 4.25: Análisis: Diagrama de secuencia Cargar receta (escenario 2b / 2c)

Operación: VerKilocalorías()

Responsabilidades: Calcula las kilocalorías que tiene la receta por cada 100 gramos a partir de las kilocalorías de cada ingrediente y su cantidad.

Precondiciones: Se ha accedido a comenzar la receta desde el resumen de la misma.

Postcondiciones: Lanza una ventana emergente que indica la cantidad de kilocalorías de la receta por cada 100 gramos.

Operación: VerAlérgenos()

Responsabilidades: Crea un listado con todos los alérgenos que tiene la receta a partir de la revisión de los alérgenos que contenga cada ingrediente.

Precondiciones: Se ha accedido a comenzar la receta desde el resumen de la misma.

Postcondiciones: Lanza una ventana emergente que muestra un listado con los ingredientes que contienen alérgenos en la receta, ligados a dichos alérgenos. Si no tuviera, en el listado se especificará que la receta no contiene alérgenos.

Operación: PulsarAtrás()

Responsabilidades: Vuelve al paso o pantalla anterior, sea cual sea la pantalla actual.

Precondiciones: Ninguna.

Postcondiciones: Retrocede a la pantalla previa a la que estemos.

Caso de uso: Cargar recetas (escenario 6c / 6d /8c /8d)

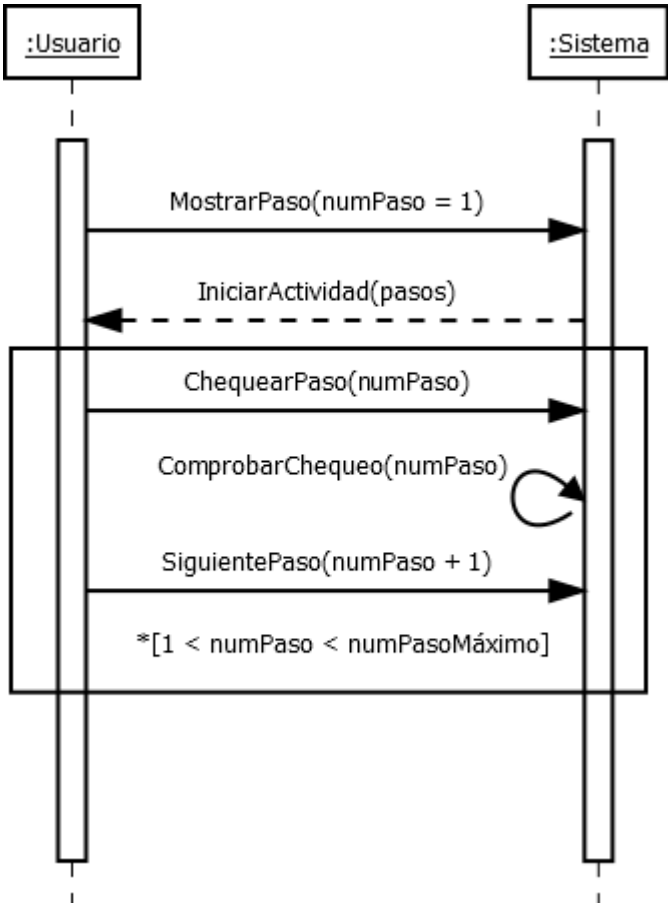


Figura 4.26: Análisis: Diagrama de secuencia Cargar receta (escenario 6c / 6d /8c /8d)

Operación: ChequearPaso()

Responsabilidades: Ofrece al usuario la posibilidad de marcar el paso en el que se encuentre como chequeado o deschequearlo, sin tener en cuenta el último paso y dependiendo del estado en el que se encuentre el paso (ya chequeado o no).

Precondiciones: La pantalla visible muestra cualquiera de los pasos de una receta, menos el paso final.

Postcondiciones: El paso en cuestión queda chequeado o deschequeado.

Caso de uso: Cargar recetas (escenario 6a / 8a / 10a)

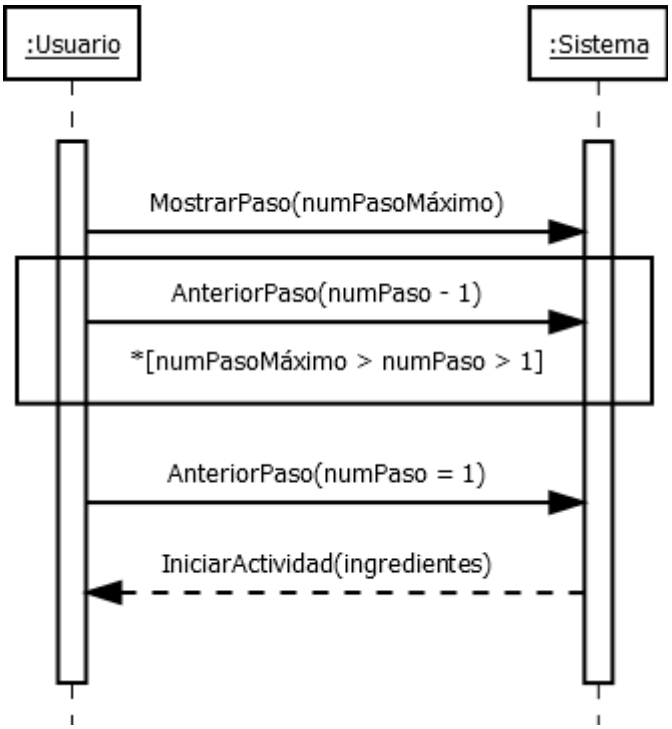


Figura 4.27: Análisis: Diagrama de secuencia Cargar receta (escenario 6a / 8a / 10a)

Operación: AnteriorPaso(numPaso - 1)

Responsabilidades: Carga en pantalla el paso anterior en la secuencia de la receta, junto con la descripción del mismo y el icono de tipo de técnica que tiene ligado. Cuando llegue al primer paso, volverá a mostrar la lista de ingredientes.

Precondiciones: El usuario ha iniciado el visualizado de pasos de la receta y se encuentra en uno de ellos.

Postcondiciones: Se muestra la información del paso anterior a numPaso, según la secuencia de realización de la receta. Si es el primer paso, se retornará al listado de ingredientes.

Caso de uso: Cargar recetas (escenario 8b)

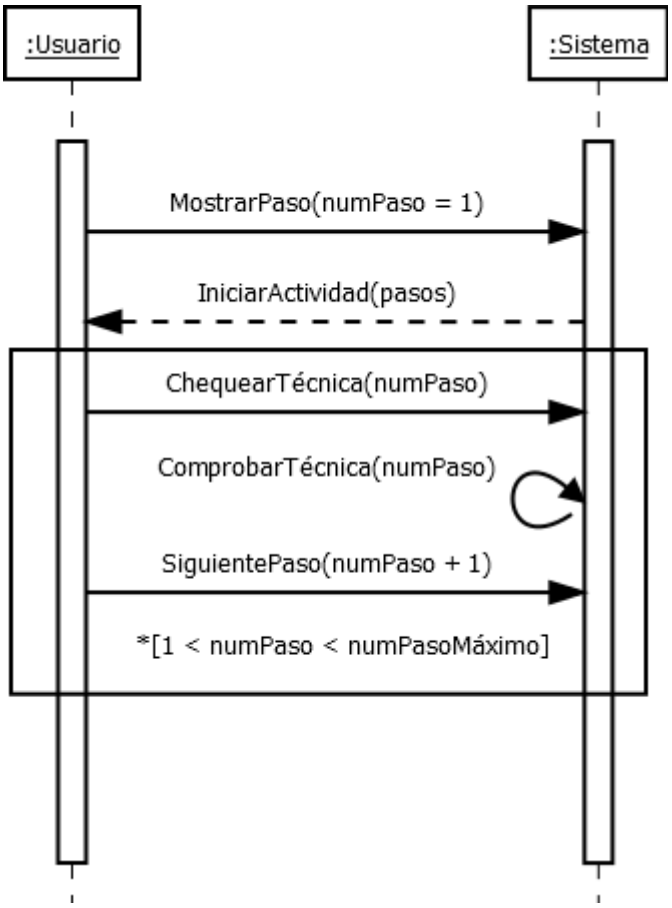


Figura 4.28: Análisis: Diagrama de secuencia Cargar receta (escenario 8b)

Operación: ChequearTécnica(numPaso)

Responsabilidades: Permite al usuario desbloquear consejos de un paso concreto si este tiene ligado alguno, o volver a visualizarlo si ya ha sido desbloqueado con anterioridad, mostrando la descripción del consejo.

Precondiciones: El usuario ha iniciado el visualizado de pasos de la receta y se encuentra en uno de ellos, que no sea el último.

Postcondiciones: Si existe consejo enlazado a ese paso, se muestra en una ventana emergente la descripción completa del consejo junto con el icono del tipo de técnica que tiene ligado.

Caso de uso: Cargar recetas (escenario 10b)

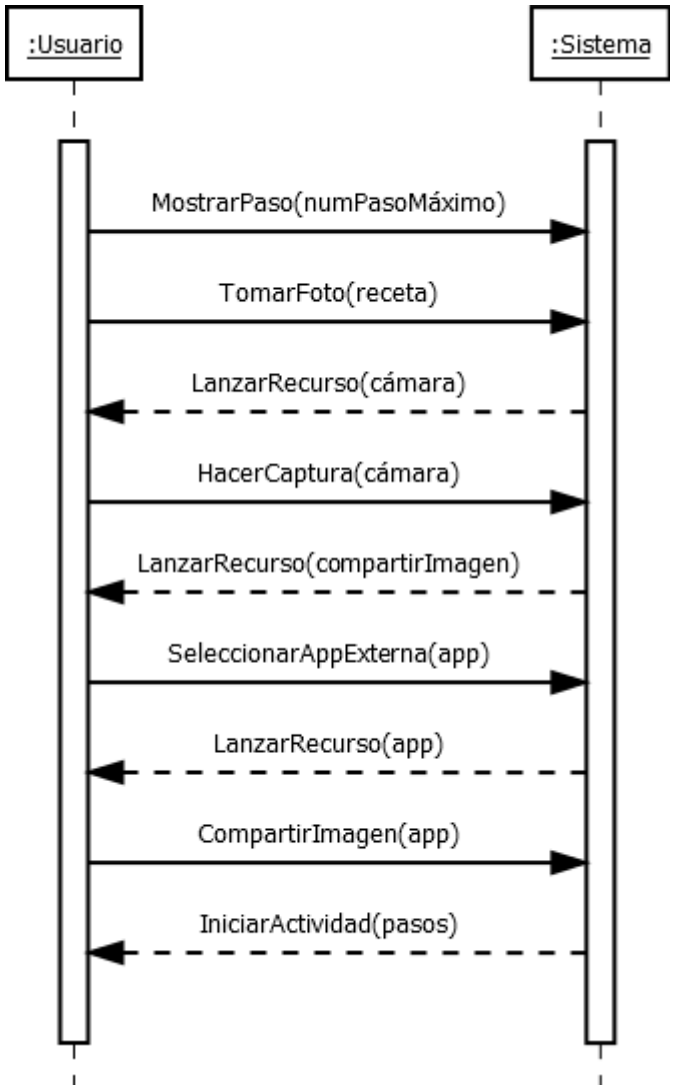


Figura 4.29: Análisis: Diagrama de secuencia Cargar receta (escenario 10b)

Operación: HacerCaptura(cámara)

Responsabilidades: Realiza una foto de la receta realizada por el usuario y la guarda en la carpeta de la galería de imágenes del dispositivo destinada a las capturas de la aplicación.

Precondiciones: Se ha accedido a realizar una foto de la receta desde el último paso de la misma, lanzando la cámara del dispositivo.

Postcondiciones: Toma la foto y la guarda en la carpeta de *aCubierto*, que se crea cuando se realiza la primera captura.

Operación: SeleccionarAppExterna(app)

Responsabilidades: Permite compartir la foto realizada con cualquier aplicación externa del dispositivo que permita compartir imágenes, añadiéndole una pequeña descripción.

Precondiciones: Se ha realizado satisfactoriamente la captura de la imagen de la receta con el dispositivo tras haber accedido a tomar la foto en el último paso.

Postcondiciones: El dispositivo lanza el servicio para compartir imágenes con otras aplicaciones y ejecuta la aplicación seleccionada por el usuario, permitiendo así compartir la imagen acompañada de texto descriptivo de la receta.

Operación: CompartirImagen()

Responsabilidades: Se comparte la fotografía de la receta junto con un texto descriptivo en una aplicación externa del dispositivo.

Precondiciones: Se ha seleccionado cualquiera de las aplicaciones disponibles en el menú de compartir y se lanza satisfactoriamente.

Postcondiciones: La imagen queda compartida en la aplicación seleccionada y se devuelve el flujo a la aplicación

Capítulo 5

Diseño del sistema

En este capítulo de la memoria vamos a desgranar cómo debe funcionar la aplicación para reproducir lo que queremos que haga. A partir de los requisitos funcionales y no funcionales establecidos, los casos de uso, el modelo conceptual y los de comportamiento del sistema y los contratos de las operaciones, nos será más fácil concretar los diferentes elementos del funcionamiento interno de nuestra aplicación, como son las clases que intervienen con sus atributos y la relación entre ellas.

Mediante el conveniente apoyo en la fase de análisis, conseguiremos distinguir las diferentes piezas que conforman el puzle y que darán vida al programa. Esta transición es de vital importancia para una posterior implementación del código y determinará en gran medida el nivel de calidad, robustez y posibles mejoras futuras de la aplicación en su conjunto.

Hay que prestar especial atención a la representación interna que van a tener los elementos relacionados con la cocina, puesto que, al ser un arte, no existen estándares, lo que puede llevar a un desarrollo sesgado o subjetivo de los elementos clave que conforman su funcionamiento y congruencia.

5.1 Diseño de la interfaz gráfica

Teniendo ya definido el funcionamiento que debe tener la aplicación, podemos y debemos plasmar los conceptos en un formato visual. Esto será de gran ayuda para terminar de ordenar las ideas que han sido desarrolladas de forma escrita.

Usaremos las que, casi con toda seguridad, serán las configuraciones finales de las pantallas de la aplicación, que han sido creadas por el diseñador gráfico a partir de los requisitos establecidos en la fase de análisis.

A continuación vamos a exponer cómo quedarían las diferentes pantallas y la forma en la que se navegaría entre ellas. Se dividirá en tres diagramas por la cantidad de contenido, en los que se podrá intuir con la ayuda de flechas el recorrido que podrá hacer el usuario a través de la aplicación.

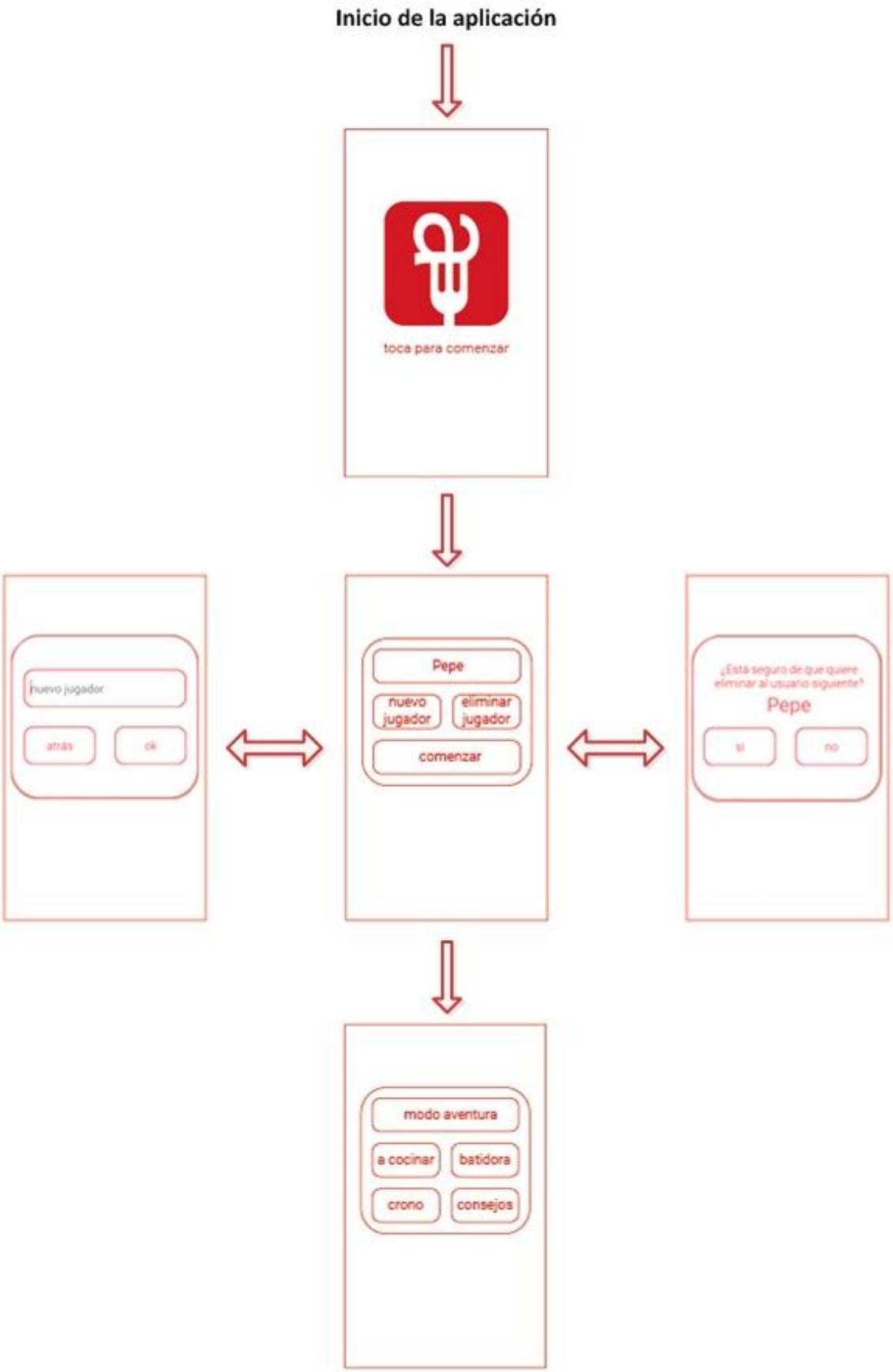


Figura 5.1: Diseño: Diagrama de inicio de la aplicación

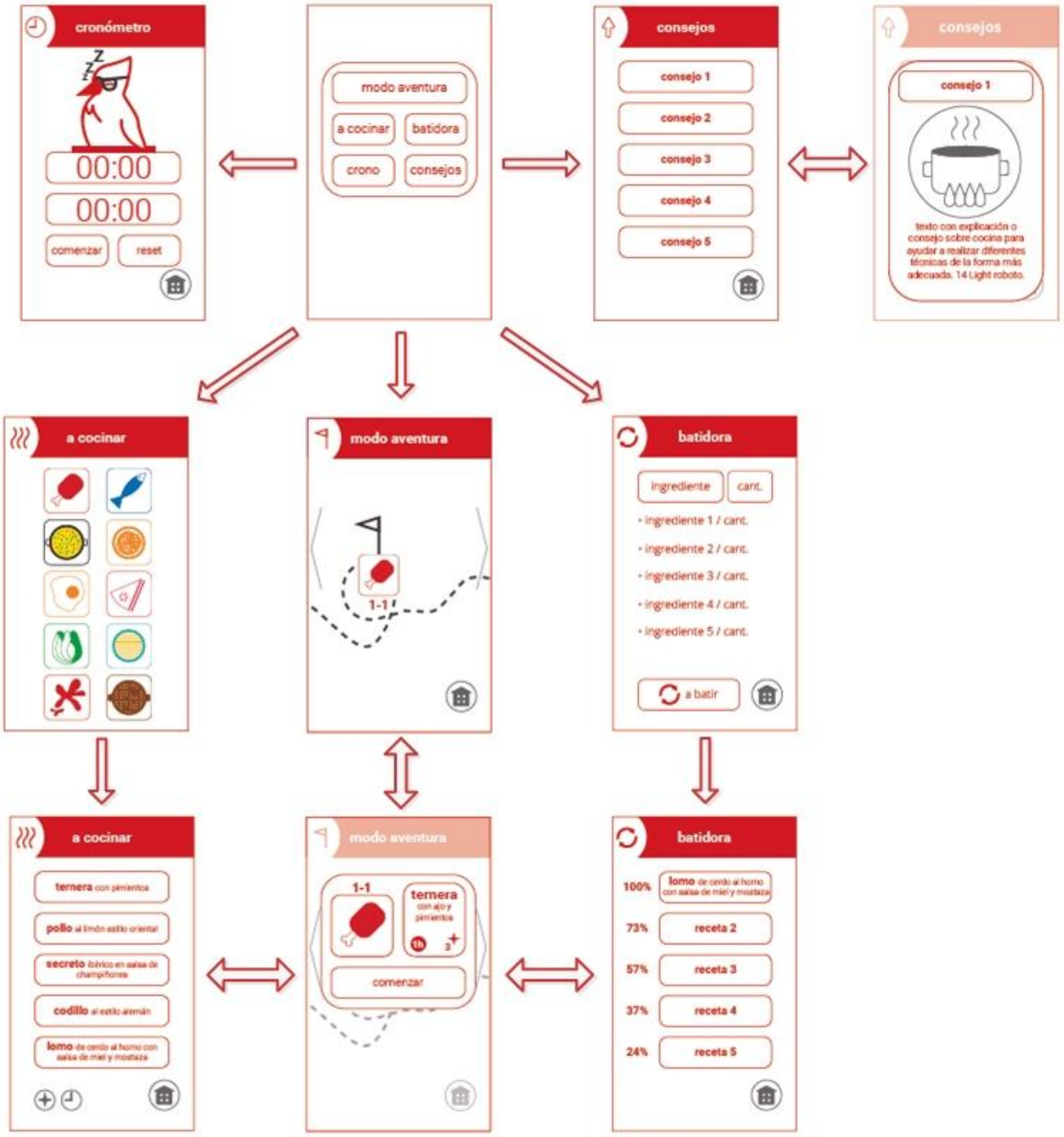


Figura 5.2: Diseño: Diagrama de navegación en modos

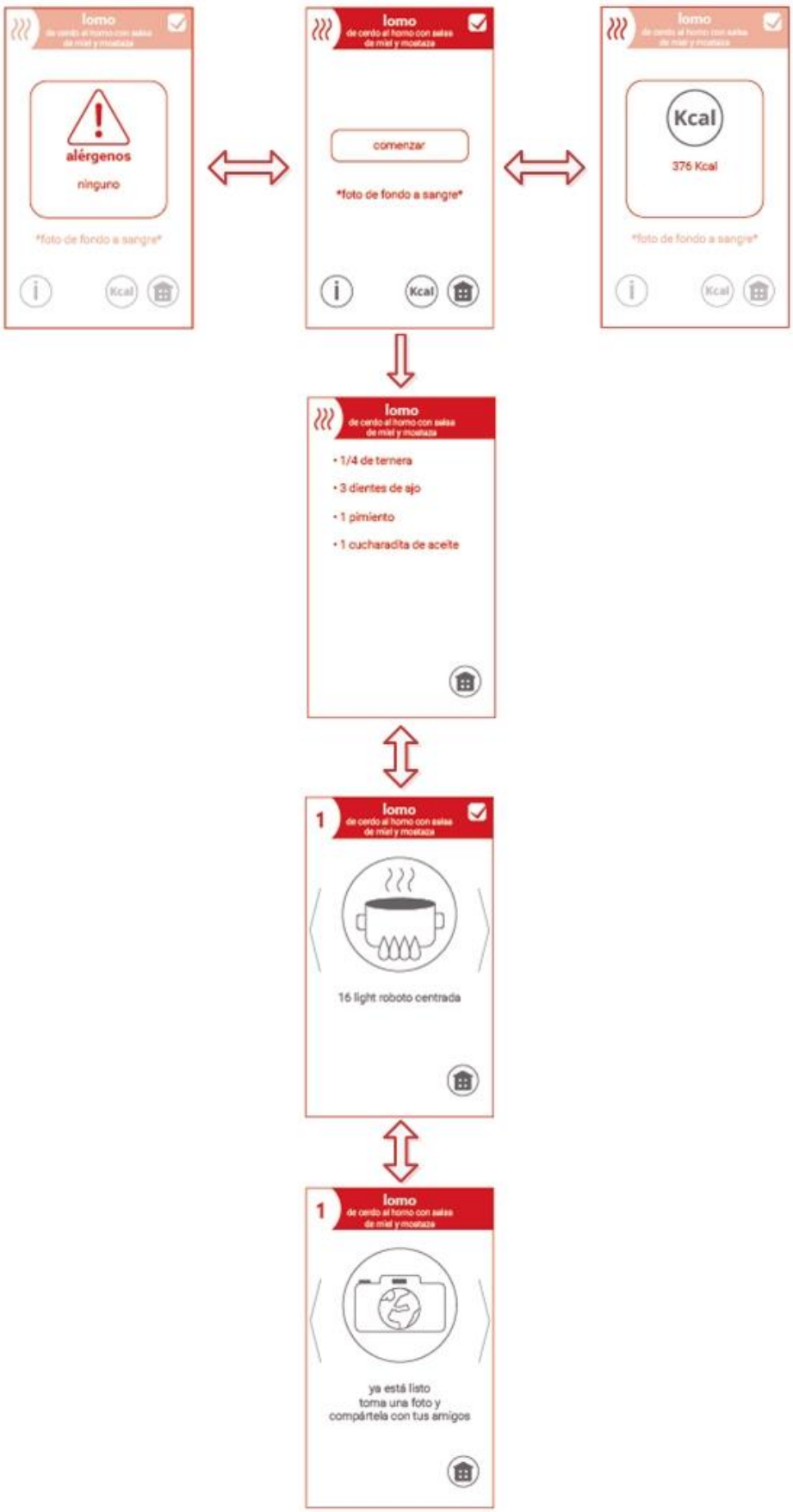


Figura 5.3: Diseño: Diagrama de navegación en recetas

5.2 Síntesis de los elementos necesarios para el diseño

Antes de proceder con los elementos característicos del apartado de diseño, vamos a exponer en forma de resumen qué datos necesitaríamos para poder plasmar las clases y sus atributos a partir de lo desarrollado en la parte de análisis.

Esto se convierte en una necesidad dentro de los requisitos para continuar por la complejidad que tiene representar un arte como es la cocina de la forma más objetiva y congruente posible. De aquí se desprenderá, revisando el capítulo anterior, qué información es la mínima necesaria para que nuestra aplicación tenga coherencia y robustez, y se pueda implementar cubriendo todos los requisitos que se han expresado. Lo mejor para atajar este planteamiento es localizar los elementos más generales y, aplicando el concepto “divide y vencerás”, separar cada bloque en componentes más pequeños, así hasta llegar a las piezas más básicas.

Para empezar, podemos discernir dos elementos principales que tiene la aplicación: las recetas y los usuarios. Son entidades que por su naturaleza quedan bien diferenciadas, pero que estarán relacionadas entre sí, porque los usuarios van a realizar las recetas.

Centrémonos en la parte de recetas, por ser el elemento principal alrededor del que gira toda la aplicación. Para identificar una receta necesitaremos que tenga un nombre. Además, las recetas se diferencian en tipos y tienen asignadas un tiempo aproximado de realización. Con esto podremos mostrar algunos de los elementos de los resúmenes desde las diferentes pantallas que permiten realizar recetas, clasificarlas y ordenarlas por tiempo en el modo a cocinar, y elegirlas de tipos diversos en el modo aventura.

Las recetas están compuestas por pasos, que son un conjunto de ingredientes al que se les aplica una técnica. Necesitaremos saber qué técnica y qué ingredientes se usan en cada uno de ellos. Gracias a esto, podremos contabilizar los pasos que tiene una receta, representar la descripción de un paso, asignarles un tipo de técnica a partir de la particularización, ordenarlas por dificultad en el modo a cocinar y mostrarla según la cantidad de pasos en los resúmenes.

La existencia de las técnicas va ligada a su uso en alguna de las recetas. Hace falta darles un nombre o más bien descripción, ligarlas a un tipo de técnica más general y saber si tienen un tiempo de ejecución asignado. Así cubriremos los requisitos como son que uno o varios ingredientes puedan describirse junto a una técnica para conformar un paso, donde a su vez se permita desbloquear un consejo para visualizarlo en el modo pertinente, y que se pueda activar el cronómetro a través de un paso.

Del párrafo anterior se desprende que la descripción de una técnica estará enlazada a un tipo más general, en los que se aglutinarán las técnicas que sean comunes, como diferentes formas de cortar. De esta manera, se podrán vincular tipos de técnica con los consejos y con los pasos de cada receta.

Los consejos se relacionan directamente con los pasos, con lo que a partir de un paso de una receta podremos o no desbloquear un consejo. A su vez, como los pasos dependen de una técnica y esta tiene un tipo específico, se podrá conocer la dependencia de forma transitiva.

En cuanto a los ingredientes, los datos que deberemos conocer para plasmar los requisitos serán nombre y tipo de cada uno, las calorías, los alérgenos, y, para representarlos en cadenas conservando el rigor

ortográfico, los determinantes artículos determinados. Toda esta información son propiedades intrínsecas a cada uno de los ingredientes.

Lo que sí depende de las circunstancias de cada ingrediente, por lo tanto, son características extrínsecas de los mismos, es la cantidad de cada uno que se necesitará en las diferentes recetas y la unidad de medida que tendrá enlazada. A partir de esto podremos saber qué alérgenos tiene cada receta, el listado de ingredientes con sus cantidades y, a partir de esto, hacer funcionar el modo batidora.

Habiendo desgranado la parte de las recetas, ahora veamos cómo cubrir los requisitos expresados en el análisis para la parte de los usuarios. Necesitaremos poder crear varios jugadores con un nombre dado por cualquier usuario de la aplicación, para así dar funcionalidad al login.

A partir de la asignación de un nombre, los jugadores podrán ir desbloqueando y chequeando elementos, como las diferentes fases del modo aventura, los pasos de cada receta con posibles consejos ligados, y una receta de forma completa, habiendo chequeado todos los pasos de la misma.

5.3 Diagrama de la lógica de la base de datos

Comencemos por establecer de forma rigurosa cómo quedaría la base de datos para almacenar toda la información imprescindible para el funcionamiento de la aplicación.

Gracias al apartado anterior, podemos diferenciar claramente los diferentes elementos necesarios. Está por un lado lo inherente a las recetas, que no será modificable por los diferentes usuarios, y, por otro, lo directamente relacionado con los mismos jugadores, donde se expresará su progreso, que se incrementará mediante se avance en los modos de juego.

Los nombres de los elementos del diagrama están en inglés por ser el lenguaje comúnmente utilizado en el ámbito de la informática. Con esto ganamos que el futuro código sea accesible a más gente. El léxico utilizado será el establecido por la documentación de *SQLite*, puesto que *Android* trabaja con este sistema de gestión de bases de datos.

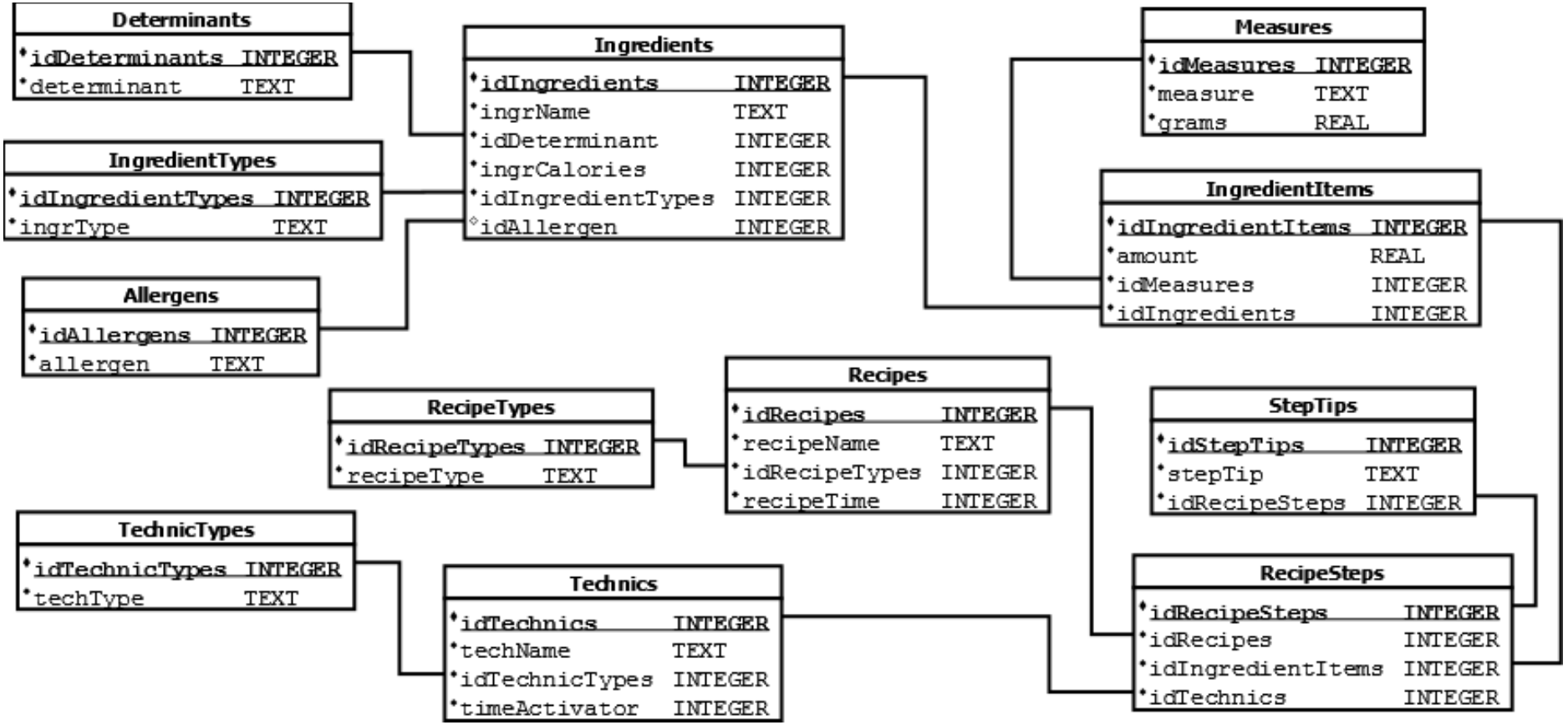


Figura 5.4: Diseño: Diagrama de la lógica de la base de datos para la parte de recetas

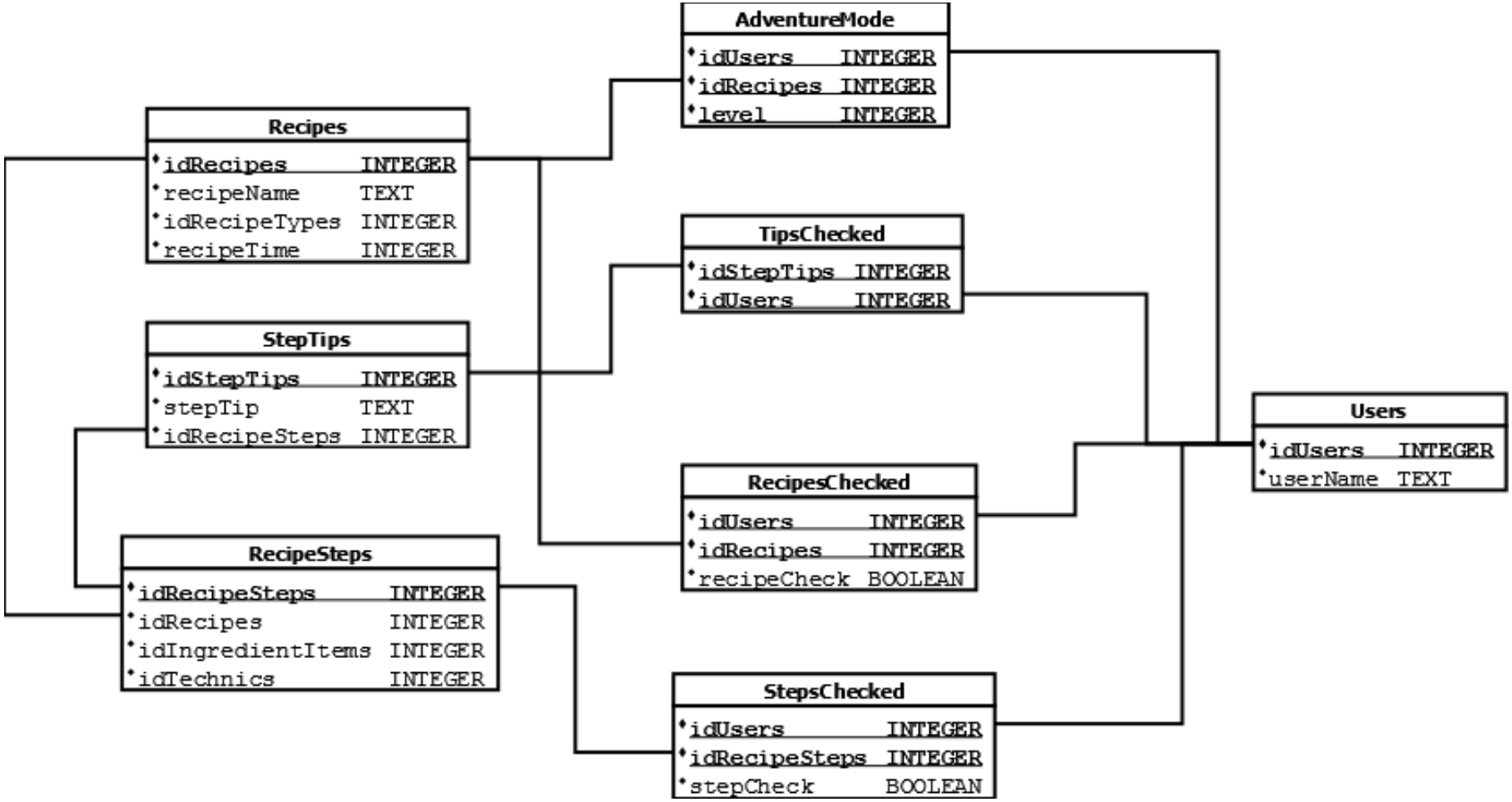


Figura 5.5: Diseño: Diagrama de la lógica de la base de datos para la parte de usuarios

5.4 Diagrama de clases

El siguiente paso en el que se debe concurrir, con conocimiento previo de la forma de almacenar los datos de nuestra aplicación, es examinar cómo estos deben estar organizados para que la información se traspase adecuadamente entre pantallas y sea representada como venimos pretendiendo. La herramienta para ello es el diagrama de clases, que refleja las colecciones de objetos necesarias y los mensajes que se deben existir entre ellas para obtener los resultados deseados en cada momento, en este caso, los que deben ser visualizados.

El léxico utilizado para la expresión de los diagramas será como el tipificado por *Java* para una mejor adaptación a la hora de desarrollar el código.

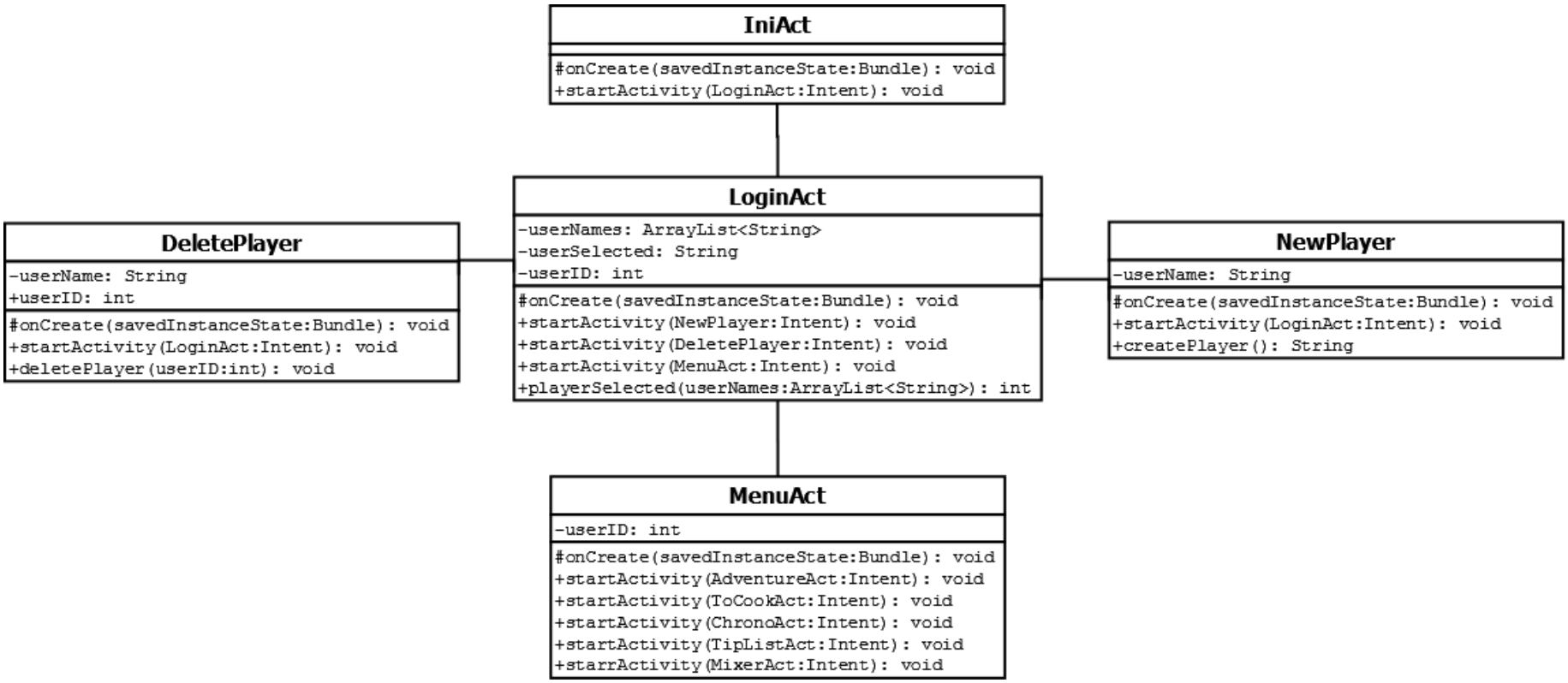


Figura 5.6: Diseño: Diagrama de clases del inicio de la aplicación

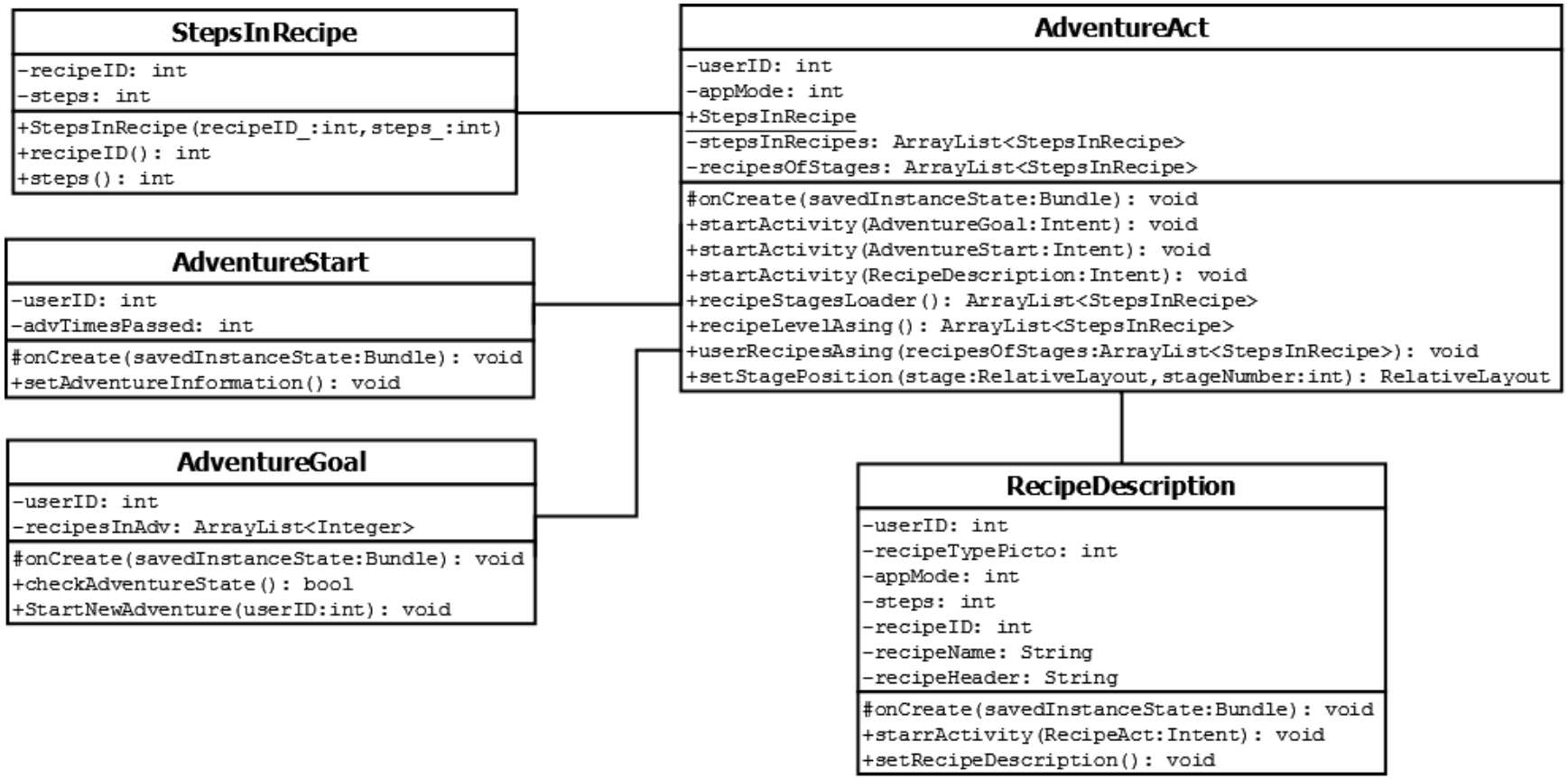


Figura 5.7: Diseño: Diagrama de clases del modo aventura

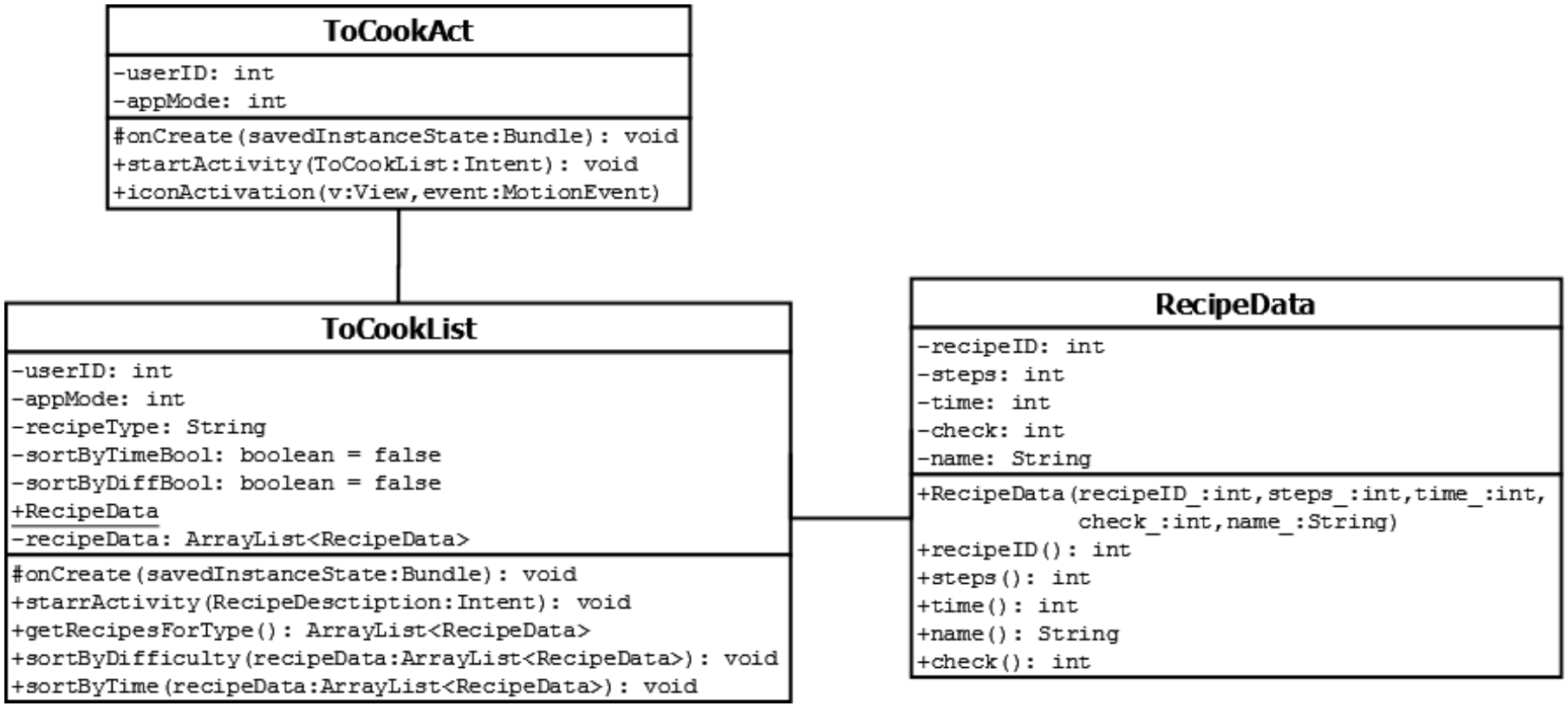


Figura 5.8: Diseño: Diagrama de clases del modo a cocinar

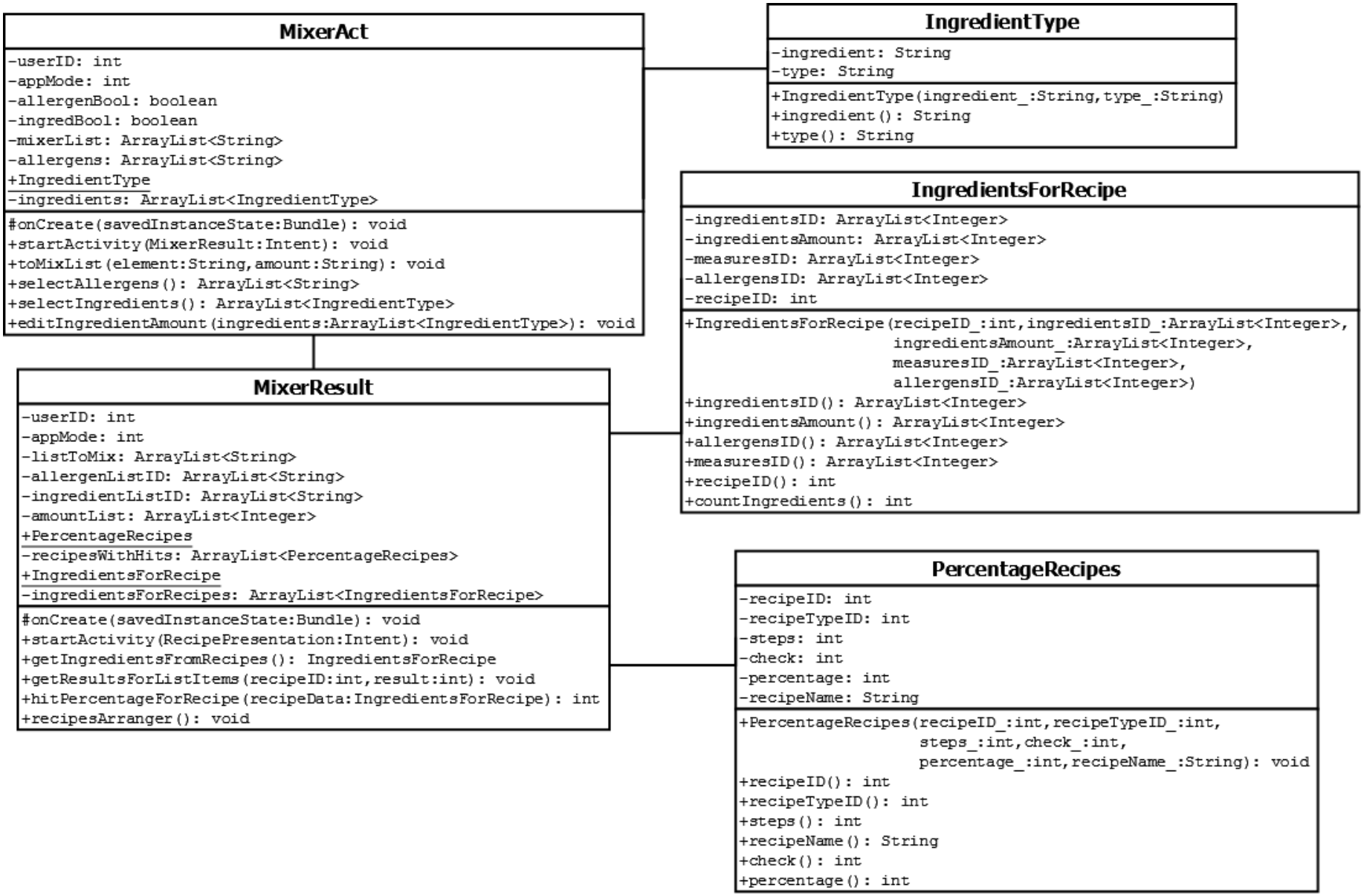


Figura 5.9: Diseño: Diagrama de clases del modo batidora

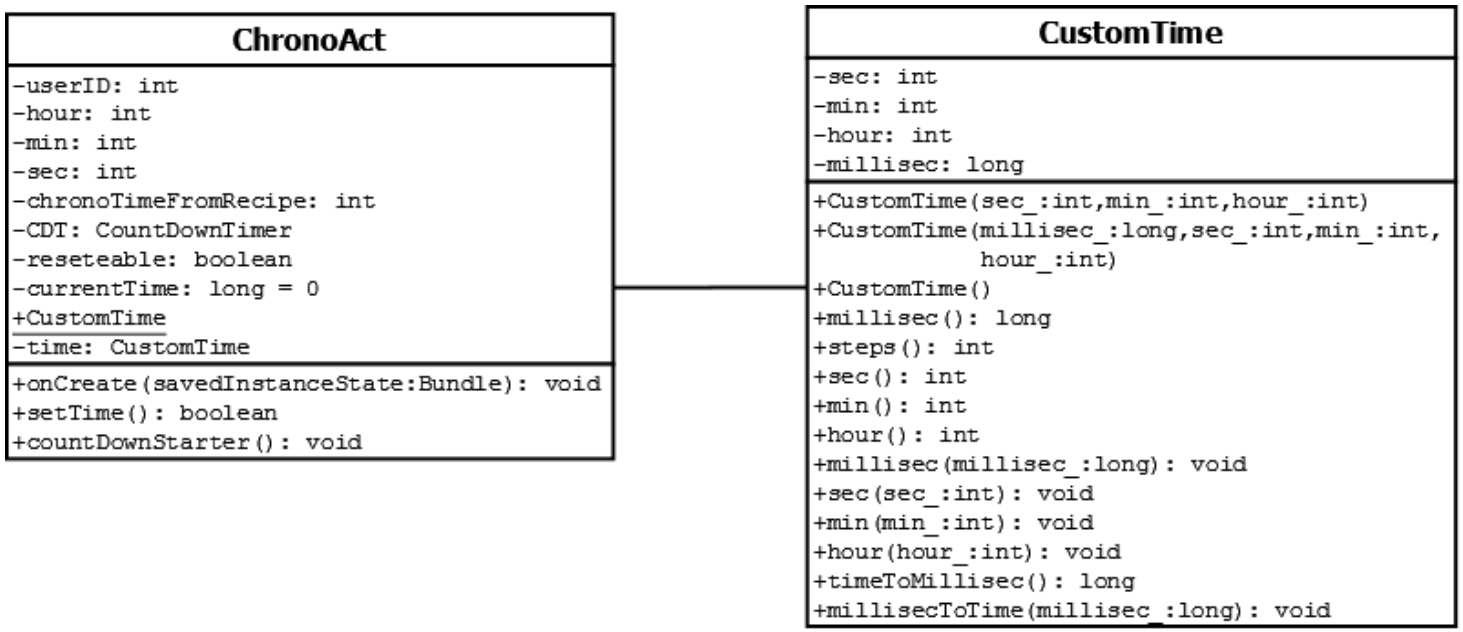


Figura 5.10: Diseño: Diagrama de clases del modo cronómetro

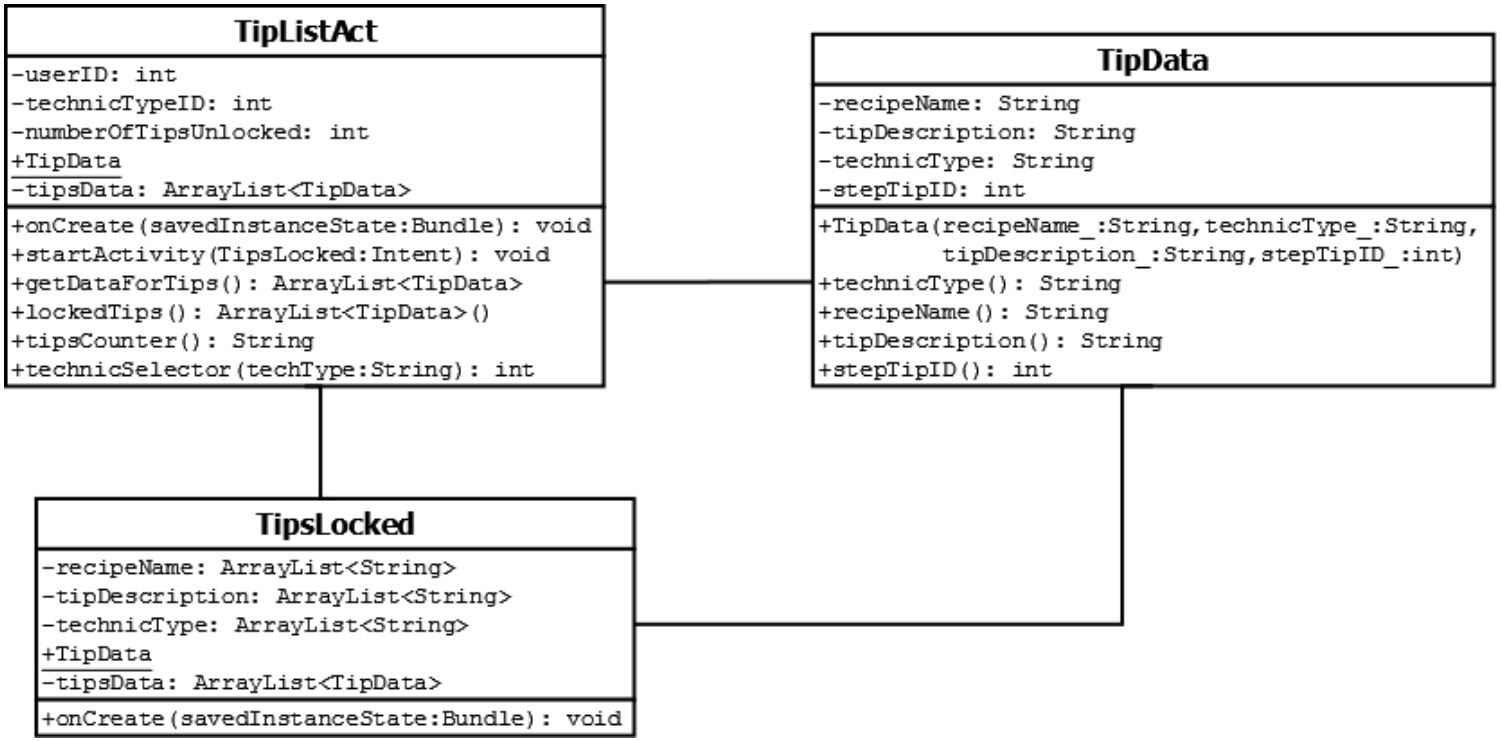


Figura 5.11: Diseño: Diagrama de clases del modo consejos

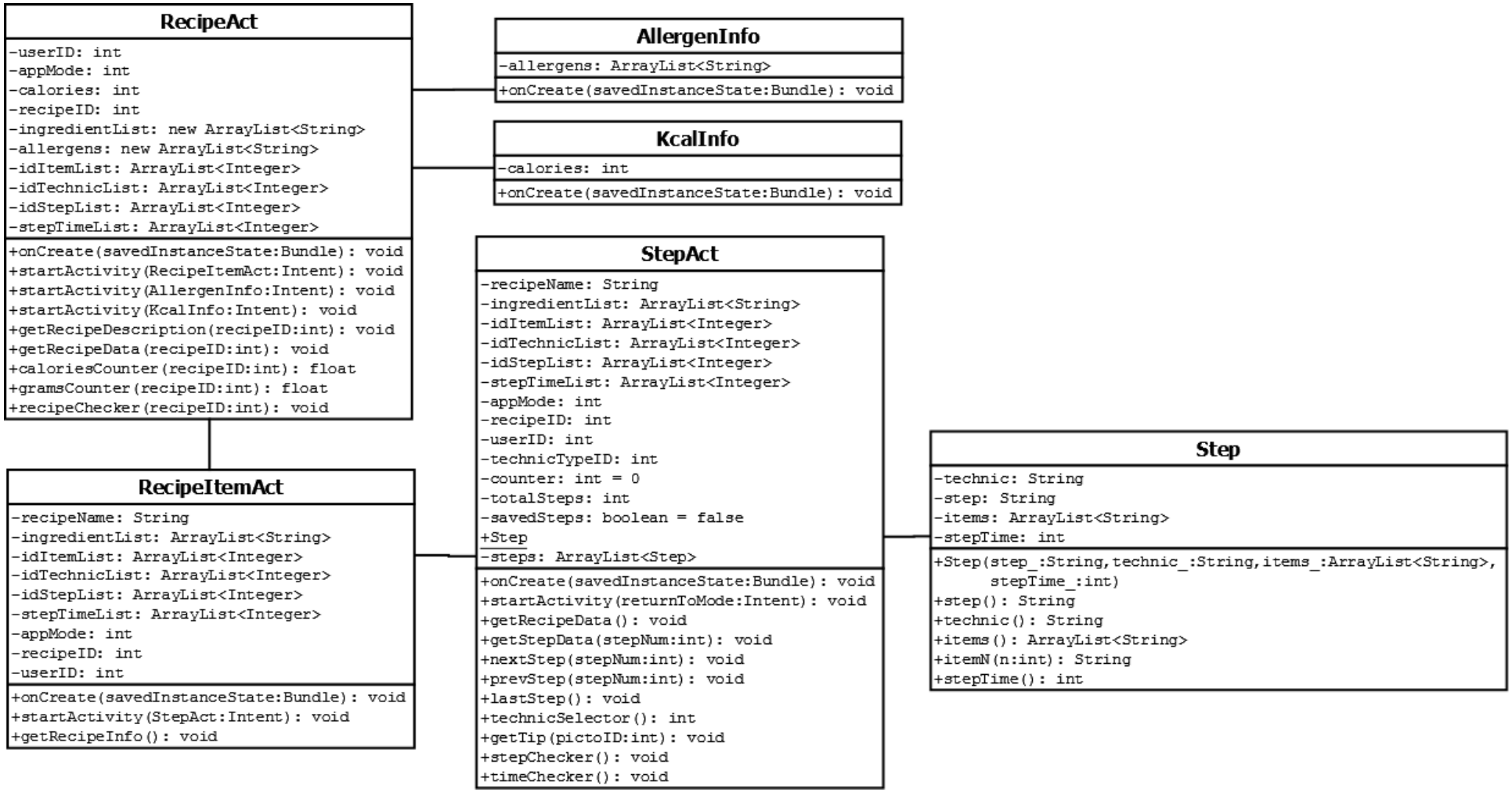


Figura 5.12: Diseño: Diagrama de clases de presentación de recetas

A continuación vamos a detallar un poco más el cometido de cada clase, junto con otras de apoyo que no aparecen en los diagramas. Centraremos las definiciones en las clases que van a ser parte de la aplicación para el funcionamiento de las pantallas. Dentro de estas, existirán otras varias cuya intención es encapsular datos para después poder movilizarlos mejor entre diferentes pantallas.

Es importante recordar que las actividades en *Android* son clases de *Java* creadas por el programador que heredan el funcionamiento de la clase *Activity*, por lo tanto, también son clases. La nomenclatura indicará que las actividades que articularán el funcionamiento de un gran apartado, como puede ser un modo de juego, finalizarán con la contracción "*Act*", haciendo referencia a que son actividades troncales, y, además, se respetará en todo momento la norma de estilo de que su primera letra sea mayúscula.

IniAct: Actividad que se ejecuta al abrir la aplicación y que muestra la pantalla de bienvenida.

LoginAct: Actividad que gestiona la selección, creación y eliminación de jugadores por parte del usuario.

NewPlayer: Actividad dependiente del login que crea un nuevo jugador mediante un nombre introducido por el usuario.

DeletePlayer: Actividad dependiente del login que permite eliminar el usuario que haya seleccionado en ese momento.

MenuAct: Actividad encargada de mostrar el menú de opciones de juego y de habilitarlas para que se puedan acceder a ellas.

AdventureAct: Actividad que muestra la pantalla del modo aventura con los iconos y recetas disponibles, chequeadas o no chequeadas, y gestiona primeros accesos y estados de última aventura para el jugador seleccionado en el login.

StepsInRecipe: Clase que permite crear duplas para contabilizar cuántos pasos tiene cada receta mediante la ID de cada receta y su número de pasos totales, para poder ordenarlas, compararlas y seleccionarlas.

AdventureStart: Actividad donde se muestran las estadísticas del modo aventura dependiendo del jugador que haya sido seleccionado en el login.

AdventureGoal: Actividad la cual permite iniciar una nueva aventura al jugador seleccionado en el login, siempre y cuando dicho jugador tenga todas las recetas del modo actual chequeadas.

RecipeDescription: Esta actividad mostrará un pequeño resumen de una receta seleccionada en cualquiera de los modos de juego habilitados para ello, visualizando su tipo, el tiempo de ejecución, la dificultad, el nombre y permitirá acceder a ella para realizarla.

ToCookAct: Actividad que muestra el listado de iconos de tipos de recetas y permite seleccionar cualquiera de ellos para listar las recetas existentes del tipo escogido.

ToCookList: Actividad resultante de seleccionar un tipo de receta, donde aparecerán en una lista las recetas de dicho tipo seleccionado, chequeadas o no chequeadas, y se permitirá acceder a ellas.

RecipeData: Clase auxiliar que almacena todos los datos de las recetas para poder ordenarlas por tiempo o dificultad y que se puedan reorganizar en la lista del tipo de receta.

MixerAct: Actividad para gestionar el modo batidora, que permite seleccionar alérgenos e ingredientes, visualizarlos en una lista para posteriormente calcular su porcentaje de aproximación entre todas las recetas disponibles dependiendo de lo que se haya seleccionado.

IngredientType: Clase auxiliar que permite crear duplas con el nombre de ingrediente junto con su tipo para poder clasificarlos, listarlos y seleccionarlos.

MixerResult: Actividad donde se calcula y se muestra el resultado de haber comparado todas las recetas con el listado creado mediante el modo batidora. Ordena las recetas por porcentaje de aproximación de mayor a menor, y permite seleccionar la que el usuario permita conveniente para su posterior realización.

IngredientsForRecipe: Clase auxiliar que almacena para cada receta los ingredientes con las cantidades, las medidas y los alérgenos, para poder compararlos con la lista de la batidora y descartar recetas que no contengan ninguno de los ingredientes o que contengan alguno de los alérgenos.

PercentageRecipes: Clase auxiliar usada para guardar los datos de cada receta con algún acierto y poder representarlas con el porcentaje al lado, además de saber si están chequeadas o no.

ChronoAct: Actividad que lanza el cronómetro de cocina integrado en la aplicación, que permite empezar una cuenta atrás añadiendo un tiempo en horas, minutos y segundos, permitiendo pausarla o resetearla, y emitiendo una señal sonora y visual cuando se llegue a cero.

CustomTime: Subclase que da soporte de tiempo al cronómetro de cocina en horas, minutos y segundos, permitiendo hacer conversiones de milisegundos a tiempo en hora minutos y segundos y viceversa.

TipListAct: Actividad para el inicio y visualización del modo de juego consejos. Permite saber cuántos consejos tenemos desbloqueados del total y leerlos junto con sus datos comunes (receta y tipo de técnica).

TipData: Clase auxiliar que permite encapsular los elementos necesarios para procesar consejos y mostrarlos.

TipsLocked: Actividad dependiente del modo consejos que lista los consejos que aún no se han desbloqueado, mostrando una breve descripción directamente en la lista y sin permitir seleccionarlos.

RecipeAct: Actividad inicial de la representación de recetas que carga y secuencia todos los datos necesarios para la visualizarla y para conocer sus detalles. Además ofrece comenzar la navegación y consultar kilocalorías y alérgenos.

AllergenInfo: Actividad dependiente del inicio de una receta que muestra en un listado los alérgenos junto con los alimentos que lo contienen de entre todos los ingredientes que conforman la receta.

KcalInfo: Actividad que revela qué cantidad de calorías tiene la receta en 100 gramos de alimento en su conjunto.

RecipeItemAct: Actividad donde se muestra el listado de ingredientes necesarios junto con las cantidades que hacen falta para llevar a cabo la receta.

StepAct: Actividad principal de navegación entre los pasos de la receta, encargada de controlar en qué paso se encuentra el usuario y de cargar los datos pertinentes de cada uno de ellos. Además, llegados al

último paso, ofrece la posibilidad de tomar una foto, y volviendo al primero, se puede volver a visualizar el listado de ingredientes. Ofrece también la posibilidad de desbloquear consejos en cada paso de la receta, dependiendo de que este tenga ligado uno o no.

Step: Clase auxiliar que secuencia y prepara las cadenas de cada paso para que sean usadas dependiendo del paso en el que nos encontremos y del total de pasos de la receta.

SQLHelper: Clase de vital importancia, que hace de interfaz entre la base de datos que lleva la aplicación localmente y la misma aplicación. Se encarga de crear la BD copiándola al sistema, gestionar las conexiones, aplicar cualquier tipo de consulta y de extraer los datos de las mismas.

Capítulo 6

Implementación

Llegados a este punto, es el momento de plasmar todo el desarrollo previamente realizado en el código que dará vida a la aplicación. Previamente a la implementación se comentarán temas también vitales para esta fase, como la versión de la API, el IDE usado y otras herramientas que se han usado.

Siguiendo las líneas anteriormente marcadas, abordaremos este apartado separando por módulos basados los diferentes modos de juego y pantallas, comentando de cada uno las vicisitudes más complejas que han tenido que sortearse para obtener el producto final.

6.1 Medios, herramientas y lenguaje

Como ya se comentó en el apartado de requisitos no funcionales, el desarrollo del proyecto se va a realizar para una versión mínima de *API* 16 de *Android*. A partir de esta, casi cualquier dispositivo podrá instalar y utilizar la aplicación, puesto que a mes de abril de 2015 hay un 87,5% de dispositivos que usan esa versión o superior, según la página oficial de *Android* [11].

Version	Codename	API	Distribution
2.2	Froyo	8	0.4%
2.3.3 - 2.3.7	Gingerbread	10	6.4%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	5.7%
4.1.x	Jelly Bean	16	16.5%
4.2.x		17	18.6%
4.3		18	5.6%
4.4	KitKat	19	41.4%
5.0	Lollipop	21	5.0%
5.1		22	0.4%

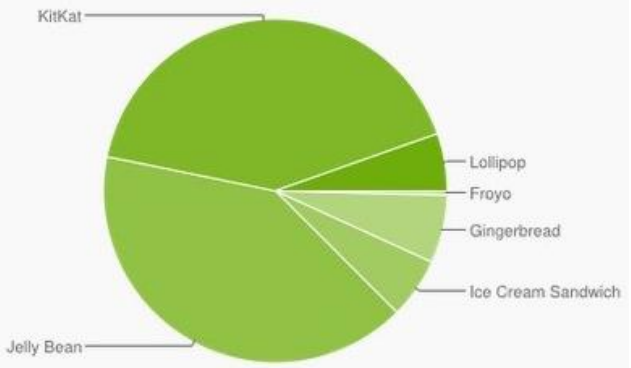


Figura 6.1: Android Dashboards: Porcentaje de uso de versiones de Android en abril de 2015

El entorno de desarrollo usado es *Eclipse Luna* enlazado con el *SDK* oficial de *Android*, mediante la *ADT* existente para ello. A través de él podremos lanzar la máquina virtual (*AVD*) para hacer pruebas e ir depurando el programa, a la vez que se va construyendo, mediante la *DDMD* (*Dalvik Debug Monitor Service*). Además, para llevar un control adecuado de las versiones y tener una copia de seguridad de todo

el proyecto en tiempo real, nos serviremos del *plugin Subversive*, particularización de *Apache Subversion*, que podemos descargar desde el gestor de *Eclipse* para dicho cometido. La dirección web es la siguiente:

https://sourceforge.net/projects/*aCubierto/*

Usaremos *Java* como lenguaje de programación por ser el recomendado, junto con el lenguaje de marcas *xml*, que da soporte a los *layouts*. Para poder crear y manipular la base de datos trabajaremos con *SQLite Manager*, que es un *plugin* del explorador de internet *Firefox* que nos permite realizar todo lo que se va a necesitar.

6.2 Creación de la base de datos

Primeramente, necesitaremos crear una tabla para los metadatos de idiomas, con el nombre *Android*_metadata, y añadirle el atributo de lenguaje en el que vamos a tener los datos almacenados para que así la aplicación pueda usar la base de datos de forma adecuada.

```
CREATE TABLE "Android_metadata" ("locale" TEXT DEFAULT 'es_ES');
INSERT INTO "Android_metadata" VALUES ('es_ES');
```

Partiendo del diagrama lógico del capítulo anterior, es el momento de proceder a la implementación del código SQLite que va a comprender las necesidades de almacenamiento que debemos cubrir. Para ello, nos serviremos de una serie de secuencias CREATE TABLE para cada una de las tablas existentes con su nombre, sus atributos, los tipos de estos y sus configuraciones. A continuación se muestra el fragmento de código necesario para crear la tabla de tipos de receta:

```
CREATE TABLE RecipeTypes(
  idRecipeTypes    INTEGER   PRIMARY KEY AUTOINCREMENT NOT NULL,
  recipeType    TEXT      NOT NULL);
```

Como se puede observar, la tabla tendrá el atributo idRecipeTypes como clave primaria que se incrementa automáticamente, según la documentación de SQLite, a partir de 1 en adelante cuando esta está vacía. Las tablas que no tengan una clave primaria declarada, obtendrán una de forma mecánica que no formará parte de los atributos visibles, pero sí se podrá consultar mediante rowid.

Existirán tablas con atributos previamente establecidos que también serán declarados en el script mediante la cláusula INSERT INTO, para así tener todo el esqueleto de la base de datos creado. Para la tabla anterior introduciríamos las siguientes filas:

```
INSERT INTO RecipeTypes (recipeType)
VALUES ("carne"), ("pescado"), ("arroz"), ("pasta"), ("huevo"), ("postres"), ("sopa"), ("verdura"),
      ("salsa"), ("guiso");
```

Los datos correspondientes a las recetas y todo lo que entrañan (ingredientes, técnicas, consejos, etc.) también estarán predefinidos en la base de datos. El problema es que hacer esto a base de sentencias es algo extremadamente tedioso. Con la idea de ahorrar tiempo y esfuerzo, se ha creado una pequeña aplicación en Java que hace de capa superior y que pregunta cada elemento de la receta, para así introducirla de manera correcta en la base de datos, creando los enlaces necesarios entre tablas y dando como resultado una receta totalmente estructurara con todos sus componentes.

La idea es que, para aumentar el volumen de recetas, la base de datos se actualice rellenando las nuevas filas de las tablas con la información pertinente, y que estas sean incluidas mediante actualizaciones periódicas de la aplicación a través de *Google Play*.

6.3 Elementos gráficos comunes de los *layouts*

Es conveniente empezar por valorar y analizar cómo va a repercutir el trabajo realizado por el diseñador gráfico de la aplicación en el desarrollo. Existen varios componentes recurrentes que se hallan en la gran mayoría de las pantallas.

La tipografía seleccionada por el diseñador ha sido *Roboto*, que desde la *API 16* viene como predefinida. En versiones anteriores, la tipografía estándar era *Noto*, con lo que estaríamos obligados a incluir la fuente a usar en el proyecto y crear una clase que gestionara el uso. Los dos estilos de fuente serán el *light* y el *bold* para todos los elementos de texto de la aplicación y el tamaño dependerá del modo o pantalla en el que nos encontremos.

El juego de colores de los elementos, sobre todo del texto que es lo que más nos interesa a la hora de programar, también será común en todas las pantallas. Con idea de reflejar un estilo sencillo y agradable, hemos acordado usar tres colores para todos los elementos de diseño (obviando iconos de tipos de receta), que son, en código de color hexadecimal, el #CC3333 (rojo), #666666 (gris) y #FFFFFF (blanco).

Derivado de los dos elementos de diseño anteriores, tenemos la composición de los botones. Los que tengan un borde rojo con texto en su interior, como por ejemplo los del menú, estarán configurados mediante una vista de tipo *Button*, acompañados del texto en cuestión y con un *background* que será el mismo recuadro del botón, insertado a través de un recurso de imagen. Este recurso de imagen variará dependiendo de la anchura del botón, con lo que tendremos dos formatos, uno más ancho y otro menos ancho.

Lo mismo que ocurre con la asignación de un *background* de fondo blanco con marco rojo mediante un recurso de imagen para los botones, pasará con los marcos de algunas actividades como la visualización de kilocalorías o alérgenos.

Las cabeceras de la representación de recetas donde viene el título también será un componente recurrente, con la única diferencia de que cambiará el icono de modo dependiendo desde donde se haya accedido. Se puede ahorrar código modificando sólo los elementos que serán diferentes dependiendo del modo de juego, implementando un método que cambie lo necesario dentro de un único conjunto de actividades para representar las recetas.

Por último, hay que prestar especial atención a los *layouts* horizontales y a la rotación en general, puesto que *Android* destruye y reconstruye las actividades en este cambio y podríamos llegar a perder información o a provocar errores.

6.4 Desarrollo de las pantallas y modos de juego

Vamos a dar paso a la explicación sobre cómo se han desarrollado los diferentes módulos de la aplicación. Nos centraremos sobre todo en los puntos que han requerido un mayor esfuerzo, ya sea por el desarrollo del concepto en código o por una mayor exigencia a la hora conocimientos en programación.

6.4.1 *SQLHelper*, la clase interfaz de la base de datos

Para poder manejar y comunicarnos con la base de datos necesitamos una clase especial que sea capaz de verificar su existencia, copiarla a la carpeta donde se almacenan los datos de la aplicación, establecer una conexión y hacer consultas en ella. Esto se consigue mediante una capa intermediaria que hace de interfaz entre los elementos almacenados y los requerimientos que necesite el sistema en cada momento. En la clase SQLHelper.java tenemos todos estos mecanismos implementados, gracias a la herencia de la clase *SQLiteOpenHelper* del paquete de la *API Android.database*.

La base de datos se incluye dentro de la carpeta *assets*, creada cuando se comienza un proyecto en *Eclipse* y destinada a almacenar ficheros de cualquier tipo que podrán ser usados por la *app*. El constructor de la clase se encarga de copiar este archivo al directorio predeterminado de datos dentro del sistema si este no ha sido copiado antes. Si ya se hubiera procedido anteriormente, el constructor llama al método *openDatabase()*, que establece la conexión para su manipulación.

- SQLHelper.java

```
1  public SQLHelper(Context context) {
2       super(context, DB_NAME, null, 1);
3       this.myContext = context;
4       boolean dbExist = checkDataBase();
5       if(!dbExist) {
6             this.getReadableDatabase();
7             copyDataBase();
8       }
9       try {
10            openDataBase();
11      } catch (Exception e) {
12            System.out.println("FAIL DATABASE OPENING");
13            System.out.println(getException(e));
14      }
15 }
```

Cuando se conecte adecuadamente con la base de datos podremos proceder a realizar cualquier tipo de consulta. Para facilitar esto, se han definido una serie de métodos dependiendo del tipo de consulta que queramos realizar, para así no tener repetir infinidad de código cada vez que tengamos que ejecutar una. Por el formato de las clases que nos son proporcionadas mediante el paquete *Android.database.sqlite*, particularmente la clase *SQLiteDatabase*, necesitaremos recurrir a varios métodos para conseguir nuestros objetivos. Las consultas que tienen carácter modificador (*INSERT*, *UPDATE* y *DELETE*) no necesitarán devolver ningún valor, con lo que sólo tendremos que adjuntarle el parámetro de cadena con la estructura adecuada al formato de la sentencia en *SQLite*. Veamos un gráfico de cómo vienen esto expresado en la documentación oficial [12] para cada caso.

Para llevar a cabo estos comandos nos serviremos del procedimiento *execSQL()* dentro de nuestras funciones personalizadas, que recibirá una cadena con la consulta y la ejecutará sobre la base de datos, sin devolver nada por ser de tipo modificadoras. No obstante, la consulta no puede contener ningún fallo sintáctico, de lo contrario, se lanzará una excepción que cancelará la ejecución de la aplicación. Por esto mismo, las consultas son directamente formateadas en cada función personalizada correspondiente mediante la colocación correcta de los símbolos de *parsing* necesarios, como suelen ser comillas, paréntesis, espacios de separación, operadores lógicos y de comparación, etc. El resultado sería el siguiente:

- SQLHelper.java

```
1  public static void update(String table, String column, String var) {
2       query = "UPDATE " + table + " SET " + column + " WHERE " + var;
3       db.execSQL(query);
4  }

5  public static void insert(String table, String column, String var) {
6       query = "INSERT INTO " + table + "(" + column + ") VALUES (" + var + ")";
7       db.execSQL(query);
8  }

9  public static void delete(String table, String var) {
10      query = "DELETE FROM " + table + " WHERE " + var;
11      db.execSQL(query);
12 }
```

Terminando con el tema de las consultas, comentemos las de tipo *SELECT*. A diferencia de las anteriores, esta cláusula está ampliamente expandida por toda la aplicación, ya que en casi todas las pantallas necesitaremos conocer algún dato relacionado con las recetas o con los jugadores, siendo simplemente consultados. Además, tendremos la obligación de recoger los datos en una variable afín a estilo de representación, filas y columnas, para poder manipular los resultados.

No sólo por la notoria complejidad en comparación con el resto de tipos de consulta, sino también porque va a comunicarse con la sección más densa de la base de datos, se vaticina que el planteamiento será más arduo que los anteriores. Nos serviremos del método *rawQuery()* de la clase *SQLiteDatabase*, que recibe una cadena con la consulta y una serie de argumentos reemplazables, también mediante *String*, que no usaremos. El gran problema de este método es que el objeto que devuelve es de tipo *Cursor*, una clase muy poco manipulable para lo que nos interesa, por tener, entre otras cosas, que aplicar siempre *moveToFirst()*porque el cursor empieza siempre en la posición antes que la primera, por la necesidad de saber nombre y tipos de las columnas, y otros inconvenientes más que pueden ocasionarnos problemas en el futuro. En pos de salvar este problema, se ha creado un método auxiliar para convertir el resultado de *rawQuery()* a un tipo de dato compuesto más intuitivo, como es un *ArrayList<String[]>*. La función encargada de esta conversión se llamará *getDataOut()*.

Habiendo precisado el formato que se va a manejar, nos falta definir cómo se van a gestionar el *parsing* dentro de los *SELECT*. Se dará el caso en el que sólo necesitemos conocer una id a partir del nombre de un elemento, como por ejemplo en los jugadores, que irán entre comillas por ser de tipo cadena. Habrá consulta mucho más complejas, donde se deberán unir tablas a través de ids y cláusulas *AND*, y luego agruparlas con *GROUP BY*. El caso es que, estaremos obligados a hacer una sobrecarga del método mediante diferenciación de parámetros para que, de una manera elegante, tengamos todas las formas de

ejecución bajo un mismo nombre de función. Como resultado, tendremos que *SELECT* se podrá invocar de las siguientes maneras:

- SQLHelper.java

```
1  public static ArrayList<String[]> select(String columns, String tables, String
2       conditions) {
3       query = "SELECT " + columns + " FROM " + tables + " WHERE " + conditions;
4       resultSet = db.rawQuery(query, null);
5       return getDataOut();
6  }
7
8  public static ArrayList<String[]> select(String table, String column, int var) {
9       query = "SELECT * FROM " + table + " WHERE " + column + " = " + var;
10      resultSet = db.rawQuery(query, null);
11      return getDataOut();
12 }
13
14 public static ArrayList<String[]> select(String selected, String table, String
15      column, String var) {
16      query = "SELECT " + selected + " FROM " + table + " WHERE " + column + " = '" +
17      var + "'";
18      resultSet = db.rawQuery(query, null);
19      return getDataOut();
20 }
21
22 public static ArrayList<String[]> select(String idColumn, String table) {
23      query = "SELECT " + idColumn + " FROM " + table + " ORDER BY " + idColumn + "
24      DESC";
25      resultSet = db.rawQuery(query, null);
26      return getDataOut();
27 }
```

6.4.2 Inicio de la aplicación y login

Comenzamos por la pantalla de bienvenida de la aplicación, cuyo desarrollo no tiene mayor trascendencia que la de mostrar el logotipo con un mensaje para guiar al usuario en la acción de pulsar en cualquier sitio de la pantalla para avanzar. Esto se consigue haciendo que la vista padre sea *clickable* y aplicándole un método escuchador de tipo *onClick*.

Pasamos a la pantalla de login, compuesta por tres botones: para crear un jugador nuevo, para eliminar un jugador seleccionado y para poder acceder al menú del juego con un jugador seleccionado. La selección del jugador se acomete usando un elemento de vista llamado *Spinner*, que no es más que una lista desplegable. Este se rellena con ayuda de un adaptador (una clase que hace de puente entre una lista de elementos y una vista) y que, en este caso, está personalizado mediante la herencia de la clase *ArrayAdapter* con el nombre *CommonListAdapter* para poder modificar la fuente de los ítems individuales del *Spinner*. *CommonListAdapter* nos será útil para usarla como adaptador en otros apartados de la aplicación.

Para crear un nuevo jugador, el usuario introduce en un *EditText* una cadena la cual es validada antes de ser insertada como registro en la tabla de *Users*. Las comprobaciones de validez a las que se somete son

sobre la no existencia previa del mismo nombre, la longitud (20 caracteres o menos), ningún carácter y que no haya espacios al principio y al final.

La opción de eliminar usuario borra todo registro del jugador en la base de datos mediante su número de identificación.

- DeletePlayer.java: onCreate()

```
1 SQLHelper.delete("TipsChecked", "idUsers = " + userID);
2 SQLHelper.delete("StepsChecked", "idUsers = " + userID);
3 SQLHelper.delete("RecipesChecked", "idUsers = " + userID);
4 SQLHelper.delete("AdventureMode", "idUsers = " + userID);
5 SQLHelper.delete("Users", "idUsers = " + userID);
```

Recibe la id del jugador mediante la recogida de los datos enviados de una actividad a otra a través de los elementos incluidos en el *Intent* que la inicia. Para ello se usa una clase llamada *Bundle*, que aglutina el lote parámetros recibidos y hace de enlace para poder obtenerlos usando el método recuperador oportuno. Esto es un comportamiento muy común en *Android* para compartir información entre actividades.

6.4.3 Menú principal de juego e inicio de los diferentes modos

Esta pantalla no revisa de demasiada complejidad. Se compone de cinco botones que nos trasladan a cada uno de los modos de juego y nos da la opción de acceder a un menú simple, todo gestionado mediante la clase *MenuAct.java*.

Cada uno de los botones va acompañado de un código de modo que es insertado en el *Intent* correspondiente, junto con la id del jugador que haya sido seleccionado, para saber en todo momento desde qué modo se ha accedido. Esto servirá para que las cabeceras de los modos sean representadas mediante el icono y título adecuado.

Como información extra, comentar que los botones tienen aplicados un estilo personalizado, que para selectores mutables se almacenan en la carpeta *drawable*, el cual hace que cuando cualquiera botón sea presionado el color cambie a un tono grisáceo para que se vea reflejada la selección. Este estilo se llama *menu_bt_long.xml* y se les aplica a todos los botones que contengan texto dentro. Para los que son menos alargados el estilo aplicado es el *menu_bt_long.xml*, que es exactamente igual pero adaptado a tamaños más pequeños.

En el menú de opciones, que es creado fácilmente mediante el método *onCreateOptionsMenu()* con ayuda de la clase MenuInflater (línea 4), se nos muestran dos apartados: uno para conocer más información sobre la aplicación y otro para saber las estadísticas del usuario que ha sido seleccionado, ambas iniciadas desde su propia actividad *MainMenuAbout* y *MainMenuUserInfo* (líneas 13 y 17) . La primera sólo muestra un texto informativo y la segunda usará consultas a la base de datos y métodos que se han sacado de otras actividades, cuya explicación irá implícita en apartados sucesivos.

- MenuAct.java

```
1 @Override
2 public boolean onCreateOptionsMenu(Menu menu) {
```

```
3       super.onCreateOptionsMenu(menu);
4       MenuInflater inflater = getMenuInflater();
5       inflater.inflate(R.menu.main_menu, menu);
6       return true;
7   }
8
9   @Override
10  public boolean onOptionsItemSelected(MenuItem item) {
11      switch (item.getItemId()) {
12          case R.id.aboutApp:
13              startActivity(new Intent(MenuAct.this, MainMenuAbout.class));
14              break;
15
16          case R.id.userInfo:
17              Intent intent = new Intent(MenuAct.this, MainMenuUserInfo.class);
18              intent.putExtra("userID", userID);
19              startActivity(intent);
20              break;
21      }
22      return true;
23  }
```

6.4.4 Modo aventura: la selección aleatoria de recetas y la colocación de *layouts* compuestos

Este modo es, junto con el modo batidora, el más complejo de implementar por la cantidad de requerimientos que conlleva y los problemas que iremos comentando.

Tras analizar la composición gráfica de la actividad, a priori los dos elementos destacables son los iconos que indican los *stages* o fases, que combinan varios elementos como son el tipo de receta (dependiente de la receta seleccionada al azar), la bandera de chequeo (también dependiente de la receta) y el número de fase, y la línea discontinua sobre la que descansan.

Comencemos por la línea discontinua. Esta compone el fondo del mapa de fases, que tiene la propiedad de ser horizontalmente desplazable (*scrollable*). Aunque al ser una imagen fija pueda dar una primera impresión de que quizás sea simple colocar los *stages* encima de forma centrada y proporcionalmente espaciada, el que no siga un patrón convierte esto en un problema, sobre todo para hallar el punto en el eje y sobre la línea en sí. Para solventarlo, se ha aplicado la teoría aprendida en la asignatura de Procesamiento de Imágenes, convirtiendo la imagen en una matriz booleana (línea 4) donde tendremos *false* en los píxeles blancos y *true* donde se detecte el color negro (líneas 9 y 11). El método que realiza este cometido es el siguiente:

- AdventureAct.java

```
1   public void getImageValues(Bitmap bitmap) {
2       int width = bitmap.getWidth();
3       int height = bitmap.getHeight();
4       matrix = new boolean[height][width];
5
6       for(int x = 0; x < height; ++x) {
7           for(int y = 0; y < width; ++y) {
8               if(bitmap.getPixel(y, x) == Color.WHITE)
9                   matrix[x][y] = false;
10              else
```

```
                        matrix[x][y] = true;
                }
        }
}
```

A continuación, dividimos equitativamente la matriz respecto al eje x para obtener los puntos proporcionales para el correcto espaciado entre *stage* y *stage*. Esto se consigue tomando la anchura total de la imagen de fondo dividiendo entre 6, localizando el primer punto de la matriz de booleanos que sea true en ese rango en altura y guardándolo mediante una *Array* bidimensional de enteros de 2 x 7, coordenadas (x,y) para cada una de las fases, la salida y la meta. Para evitar una cantidad excesiva de accesos a la matriz de booleanos, acotamos la búsqueda en un rango lógico donde se hallen los valores verdaderos. Veamos un esquema gráfico para dejar más clara la idea:

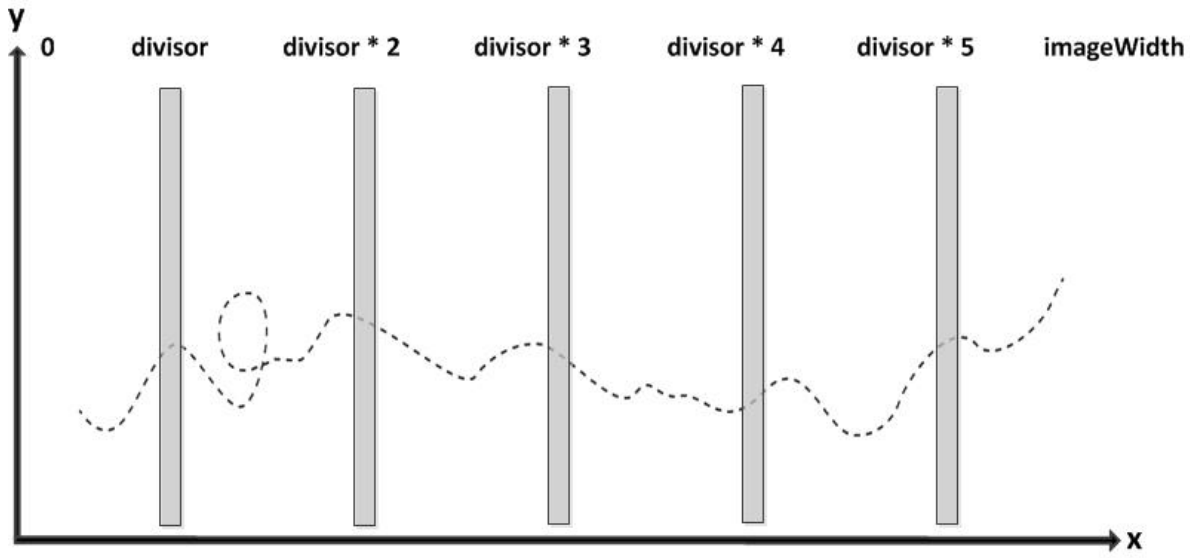


Figura 6.2: AdventureAct.java: Diagrama representativo de la localización de los stages

El código para proceder con este planteamiento, que forma parte de la función *getStagesPosition()*, quedaría así:

- AdventureAct.java: getStagesPosition()

```
public void run() {
      boolean getCoordenate = true;
      imageHeight = bitmap.getHeight();
      imageWidth = bitmap.getWidth();
      divisor = imageWidth / (float)6.0;
      stageXY = (int)divisor;
      stageCounter = 0;
      coef = (float)adventureMap.getWidth() / imageWidth;

      for(int x = stageXY; (x < stageXY + divisorRange) && (stageCounter < 5); x++) {
            for(int y = topHeight; (y < imageHeight - bottonHeight) &&
                        getCoordenate == true; y++)
                  if(matrix[y][x] == true) {
                        coordenates[0][stageCounter] = (int) (x * (coef));
                        coordenates[1][stageCounter] = (int) (y * (coef));
                        getCoordenate = false;
                        stageXY += (int)divisor;
```

```
                        x = stageXY;
                        stageCounter++;
            }
            getCoordenate = true;
            startGoalCreator();
            recipeStagesLoader();
      }
}
```

Como el sistema adapta la imagen a la vista, tendremos que reajustar las coordenadas hallando el coeficiente que surge de dividir alto y ancho del *layout* (línea 8) donde está la imagen entre el tamaño real de la misma. Las variables *topHeight* y *bottonheight* acotan por arriba y abajo la lectura de la matriz booleana para reducir el número de operaciones de lectura (línea 11), y *divisionRange* evita que tengamos la mala suerte de caer en un hueco entre raya y raya de la línea discontinua. Los valores en este caso son 150, 50 y 10 respectivamente. En un futuro, podríamos introducir la mejora para que el fondo del mapa (el recorrido de la línea) cambiara a diferentes niveles, jugando con las cotas para que siguiera teniendo la misma utilidad.

El gran problema derivado de este planteamiento es que los elementos de los *layouts* se cargan más rápido que lo que se tarda en asignarles su posición. Entonces, necesitamos esperar a que terminen los cálculos de coordenadas para que se emplacen el resto de vistas en el mapa. Esto se soluciona aplicando a dicha vista el método *post()*, que crea un hilo de subproceso que obliga al programa a esperar a que finalice para continuar con el resto de la ejecución de la actividad.

Para terminar de colocar las fases y los iconos de salida y meta, calculamos el centro de cada una de las vistas de botones para situar el punto medio de los elementos centrado con la línea del mapa.

- AdventureAct.java: stageViewCreator()

```
RelativeLayout stage = (RelativeLayout) findViewById(R.id.stage1);
stageWidth = stage.getWidth() / 2;
stageHeight = stage.getHeight() / 2;
```

El procedimiento encargado de esto y de dotar de funcionalidad de botón con la capacidad de mostrar el resumen de la receta adecuado para cada una se llama *stageViewCreator()*. Este a su vez se compone de dos subfunciones, *setStagePosition()* y *recipeTypeSelector()*, que se encargan de lo que sus propios nombres en inglés dicen, colocar adecuadamente la vista del *stage* chequeada o no, y cargar el icono correcto de tipo de receta para cada uno de los 5 *stages* del mapa. Para darlee también importancia a la parte de diseño en *xml*, vamos a mostrar y a explicar un poco la composición de una fase y cómo se incluye en el mapa de la aventura:

- adventure_stage.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:Android="http://schemas.Android.com/apk/res/Android"
      android:id="@+id/adventureStage"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:background="@Android:color/transparent"
      android:orientation="vertical" >

      <FrameLayout
```

```
10              android:id="@+id/adventureFlat"
11              android:layout_width="wrap_content"
12              android:layout_height="wrap_content"
13              android:layout_centerHorizontal="true" >
14
15      <ImageView
16              android:id="@+id/checkFlat"
17              android:layout_width="35dp"
18              android:layout_height="50dp"
19              android:layout_gravity="left"
20              android:layout_marginRight="40dp"
21              android:contentDescription="@null"
22              android:gravity="left"
23              android:src="@drawable/adventure_icon_checkedflat" />
24      </FrameLayout>
25
26      <ImageButton
27              android:id="@+id/picto_recipe"
28              android:layout_width="60dp"
29              android:layout_height="60dp"
30              android:layout_below="@id/adventureFlat"
31              android:layout_centerHorizontal="true"
32              android:layout_gravity="center"
33              android:background="@drawable/recipetype_picto_meat"
34              android:contentDescription="@null"
35              android:gravity="center" />
36
37      <TextView
38              android:id="@+id/stageNum"
39              android:layout_width="wrap_content"
40              android:layout_height="wrap_content"
41              android:layout_below="@id/picto_recipe"
42              android:layout_centerHorizontal="true"
43              android:layout_gravity="center"
44              android:gravity="center"
45              android:textColor="@color/principalRed"
46              android:textSize="20sp"
47              android:textStyle="bold" />
48  </RelativeLayout>
```

Como se puede ver, este *layout* tiene tres elementos gráficos, como son la bandera de chequeado (líneas 15 a 23), el icono de tipo de receta (líneas 26 a 35) y el texto de la fase (líneas 37 a 47). Los incluiremos dentro del mapa de la aventura mediante el comando para inclusión de vistas *include*. Hay que tener uno por cada fase con este formato:

- adventure_act.xml

```
1  <include
2       android:id="@+id/stage1"
3       android:layout_width="wrap_content"
4       android:layout_height="wrap_content"
5       android:clickable="true"
6       layout="@layout/adventure_stage" />
```

Habiendo explicado ya cómo se ha solventado la representación gráfica mediante código, vamos a pasar ahora a comentar cómo se ha procedido a la selección aleatoria de las cinco recetas. Los dos casos que pueden ocurrir son que accedamos por primera vez al modo aventura o que ya se haya accedido y existan

recetas asignadas, verificándose mediante el procedimiento *recipeStagesLoader()*. En el primero, contaremos los pasos que tiene cada receta y crearemos un objeto por cada una de ellas del tipo *StepsInRecipe*, que ayudará a almacenar en un *ArrayList* (*stepsInAllRecipes*) dichos datos (número de pasos e id de la receta) para después poder ordenarlo de recetas con menos pasos a recetas con más pasos. Esto se consigue ordenando la colección de instancias indicando la forma de diferenciarlas a través de un comparador:

- AdventureAct.java: recipeLevelAssing()

```
1  Collections.sort(stepsInAllRecipes, new Comparator<StepsInRecipe>() {
2       @Override
3       public int compare(StepsInRecipe recipe1, StepsInRecipe recipe2) {
4             Integer recipeObj1 = Integer.valueOf(recipe1.steps());
5             Integer recipeObj2 = Integer.valueOf(recipe2.steps());
6             return  recipeObj1.compareTo(recipeObj2);
7       }
8  });
```

El problema de la dificultad es que es una característica sujeta a gran subjetividad. No a todas las personas les resulta igual de complejo hacer una misma acción, por lo que se ha tomado como indicador la cantidad de pasos de cada receta, lo que quizás convierta más este concepto en trabajoso en vez de dificultoso. Tomando el número total de recetas mirando el tamaño del *ArrayList* (línea 2) y dividiéndolo en cinco tramos tendremos la diferenciación de los cinco niveles diferentes (líneas 4 a 8), de manera que siempre habrá la misma cantidad de recetas o, a lo sumo, una más en cada tramo. Esto se realiza en el procedimiento *userRecipesAsing()*, que a su vez invoca al método *setRecipeInLevel()* para obtener aleatoriamente tan sólo una receta de cada rango, que preferiblemente no haya sido realizada con anterioridad y que el tipo, a ser posible, no esté repetido. Esto se consigue con una serie de condiciones que priman el desechar las recetas que no cumplan los requisitos que nos interesan, pero siempre intentando que existan ciertas posibilidades de que no sea exactamente así (líneas 36 a 44), requerimiento que puede modificarse aumentando la cantidad de vueltas del bucle para eliminar las recetas que preferimos no seleccionar (*loopCounter*, línea 40). Tras realizar esto, el nivel del modo aventura queda guardado en la base de datos (líneas 10, 13, etc.).

- AdventureAct.java

```
1  public void userRecipesAsing() {
2       int level = stepsInAllRecipes.size();
3
4       recipesOfStages.add(setRecipeInLevel(1, level/5));
5       recipesOfStages.add(setRecipeInLevel(1 + (level/5), (level/5)*2));
6       recipesOfStages.add(setRecipeInLevel(1 + (level/5)*2, (level/5)*3));
7       recipesOfStages.add(setRecipeInLevel(1 + (level/5)*3, (level/5)*4));
8       recipesOfStages.add(setRecipeInLevel(1 + (level/5)*4, level));
9
10      SQLHelper.insert("AdventureMode", "idRecipes, idUsers, level",
11            recipesOfStages.get(0).recipeID + ", " + userID + ", " +
12            recipesOfStages.get(0).steps);
13      SQLHelper.insert("AdventureMode", "idRecipes, idUsers, level",
14            recipesOfStages.get(1).recipeID + ", " + userID + ", " +
15            recipesOfStages.get(1).steps);
16 //...
17 }
```

```
18 public StepsInRecipe setRecipeInLevel(int start, int end) {
19      ArrayList<StepsInRecipe> recipeForLevel = new ArrayList<StepsInRecipe>();
20      for(int i = start - 1; i <= end - 1; i++)
21            recipeForLevel.add(stepsInAllRecipes.get(i));
22
23      ArrayList<Integer> typesOfRecipes = new ArrayList<Integer>();
24      for(int i = 0; i < recipesOfStages.size(); i++)
25            typesOfRecipes.add(recipesOfStages.get(i).recipeType());
26
27      Random rand = new Random();
28      int loopCounter = 0;
29      while(recipeForLevel.size() > 1) {
30            StepsInRecipe recipeToRemove =
31                  recipeForLevel.get(rand.nextInt(recipeForLevel.size() - 1));
32            queryResult = SQLHelper.select("recipeCheck", "RecipesChecked",
33                  "idRecipes = " + recipeToRemove.recipeID() + " AND idUsers = " +
34                  userID);
35
36            if(queryResult.get(0)[0].equals("1"))
37                  recipeForLevel.remove(recipeToRemove);
38            else if(typesOfRecipes.contains(recipeToRemove.recipeType()))
39                  recipeForLevel.remove(recipeToRemove);
40            else if(loopCounter >= 5) {
41                  recipeForLevel.remove(recipeToRemove);
42                  loopCounter = 0;
43            } else
44                  loopCounter++;
45      }
46      return recipeForLevel.get(0);
47 }
```

Para finalizar este modo nos queda explicar cómo se llevan a cabo las funcionalidades de los botones de salida y de meta. El procedimiento que tiene asignada dicha tarea es *startGoalCreator()*, y coloca las vistas de ambos en donde deben ubicarse además de dotarlos de la propiedad seleccionable. En la salida, haremos un par de consultas a la base de datos que nos ayudarán a rescatar toda la información que queremos representar para después formatearla, que no es más que el nombre de usuario, el estado de la aventura actual (recetas chequeadas de las cinco totales) y veces que se ha terminado un modo aventura. Estos datos se guardarán en un *Intent* que iniciará la actividad que configura la visualización, llamada *AdventureStart*, cuyo único cometido es formatear los *TextView* con dicha información.

La base del funcionamiento del botón de meta es exactamente la misma, sólo con un par de variantes. Esta vez no nos hará falta saber cuántos modos aventura se han llegado a terminar, habiendo chequeado todas las recetas de dicho modo, y la actividad oportuna, *AdventureGoal*, será llamada mediante un *startActivityForResult()* para saber si se ha elegido crear una nueva aventura, en caso de que fuera posible. La actividad lanzada permitirá cargar una nueva aventura si todas las recetas de la actual están chequeadas (línea 12). En caso negativo, avisará de que falta alguna por chequear mediante un *Toast*. En caso afirmativo, se cerrará la actividad devolviendo un resultado correcto, que será gestionado por *onActivityResult()*, invocando los métodos necesarios encargados de la gestión de las partes de una aventura (creación de nueva aventura y asignación de nuevas vistas, líneas 1 a 7).

- AdventureGoal.java: onCreate()

```
1 TextView userNameText = (TextView) findViewById(R.id.userName);
2 TextView advStateNumText = (TextView) findViewById(R.id.advStateNum);
```

```
3
4  userNameText.setTypeface(Typeface.create("sans-serif", Typeface.BOLD));
5  userNameText.setText(userName);
6  advStateNumText.setText(" " + advStateNum + "/" + "5");
7
8  Button newAdvB = (Button) findViewById(R.id.startNewAdv);
9  newAdvB.setOnClickListener(new OnClickListener() {
10 @Override
11       public void onClick(View v) {
12             if(advStateNum.equals("5")) {
13                   setResult(RESULT_OK);
14                   finish();
15             }
16             else
17                   Toast.makeText(getApplicationContext(), R.string.noNewAdv,
18                               Toast.LENGTH_SHORT).show();
19       }
20 });
```

- AdventureAct.java: onActivityResult()

```
1  if(requestCode == REQUEST_CODE_NEW_ADV)
2        if(resultCode == RESULT_OK) {
3              advTimesPassed++;
4              recipeLevelAssing();
5              userRecipesAssing();
6              recipeStagesLoader();
7        }
```

6.4.5 El modo a cocinar, recetas ordenadas por tipo, tiempo y dificultad

En este apartado las dificultades no serán demasiadas. Representar elementos seleccionables en filas y columnas es algo que se convierte en algo trivial usando la vista *TableRow* combinada con *RelativeLayout*. Como tendrá la característica de poder deslizarse en el *layout* vertical, también necesitaremos que la vista del modo sea *Scrollable*. Veamos el ejemplo de una de las filas:

El método para poder dotar de la propiedad seleccionable a cada botón y no tener que repetir tanto código se basa en el uso de la función *onTouch()* (línea 5) partiendo de un escuchador de cada botón y en la asignación de *tags* o etiquetas a cada una de las vistas de botón. Este método vigila los eventos (pulsar y soltar, líneas 8 y 13) que le ocurren a cada uno de los *ImageButton* (línea 1) y, con el uso de *setTag()* (línea 2), marca cada uno con el tipo de receta asignado, que luego se recogerá para poder cargar los listados oportunos. Además, gestiona que si se pulsa en un botón y luego el puntero de selección se desplaza fuera, no se accederá al listado de dicho tipo, y también aplicará un filtro para colorear de gris el que esté elegido para que sea más cómodo para el usuario visualmente hablando.

- ToCookAct.java: onCreate()

```
1  final ImageButton meatRecipes = (ImageButton) findViewById(R.id.meatIcon);
2  meatRecipes.setTag("meat");
3  meatRecipes.setOnTouchListener(new OnTouchListener() {
4       @Override
5       public boolean onTouch(View v, MotionEvent event)
6                   { return iconActivation(v, event);}});
7  public boolean iconActivation(View v, MotionEvent event) {
8       if (event.getAction() == MotionEvent.ACTION_DOWN) {
```

```
9               ((ImageView) v).setColorFilter(Color.argb(150, 155, 155, 155));
10              rect = new Rect(v.getLeft(), v.getTop(), v.getRight(), v.getBottom());
11              return true;
12        }
13        else if(event.getAction() == MotionEvent.ACTION_UP) {
14              ((ImageView) v).setColorFilter(null);
15              if(rect.contains(v.getLeft() + (int) event.getX(), v.getTop() + (int)
16                                    event.getY())) {
17              ((ImageView) v).setColorFilter(null);
18                    String tag = (String) v.getTag();
19                    Intent intent = new Intent(ToCookAct.this, ToCookList.class);
20                    intent.putExtra("userID", userID);
21                    intent.putExtra("appMode", appMode);
22                    intent.putExtra("recipeType", tag);
23                    finish();
24                    startActivity(intent);
25              }
26        }
27        return false;
28  }
```

Como se puede observar, para aplicar el filtro gris de selección se usa una clase llamada *Rect*, perteneciente al paquete *Android.graphics* que da soporte para guardar las coordenadas del rectángulo que forman el botón y después compararlas para saber si seguimos dentro. Esta es una práctica muy extendida en el desarrollo de la aplicación, sobre todo en los elementos de tipo *ImageButton* como *toHome*, que aparece en casi todas las pantallas.

De esta actividad pasamos a la siguiente, que configura los resultados de un tipo de receta en una lista. Necesitamos guardar de cada una la id, el número de pasos, el tiempo de realización, el nombre y si está chequeada o no, para poder representarlas tanto como ítem de la *ListView*, que será personalizada, como para después mostrar los datos de la descripción. Esto se consigue creando una clase de apoyo llamada *RecipeData* y creando un *ArrayList* de objetos de esta, que se rellena mediante el procedimiento *getRecipesForType()*.

Debemos tener en cuenta que, para representar los elementos de las listas cuando estos están formados por algo más que una cadena de caracteres, el adaptador que coordina la vista debe ser definido por el programador. En este caso, además del título de la receta, tendremos que poder visualizar si está chequeada o no (líneas 13 a 16). Entonces, cada elemento estará formado por un texto de título y una imagen de una banderita que será visible o no (líneas 8 a 11) dependiendo de lo comentado.

- ToCookAdapter.java

```
1   public View getView(int position, View convertView, ViewGroup parent) {
2         RecipeData recipeDataItem = getItem(position);
3
4         if (convertView == null)
5               convertView = LayoutInflater.from(getContext()).inflate(
6               R.layout.tocook_list_adapter, parent, false);
7
8         TextView recipeTitle = (TextView) convertView.findViewById(R.id.recipeTitle);
9         recipeTitle.setText(getSetTitle(recipeDataItem.name()));
10        ImageView recipeFlatChecked = (ImageView)
11              convertView.findViewById(R.id.recipeFlatChecked);
12
```

```
13        if(recipeDataItem.check() == 1)
14              recipeFlatChecked.setVisibility(View.VISIBLE);
15        else
16              recipeFlatChecked.setVisibility(View.GONE);
17
18        return convertView;
19  }
```

En este caso aparece otro elemento nuevo importante para la configuración de los títulos de receta: el método *getSetTitle()*. Como es común a otras actividades se explicará en el apartado oportuno, el dedicado para elementos comunes.

Por último, para este modo se da la posibilidad de ordenar por dificultad o tiempo las recetas de un cierto tipo, cosa que se consigue aplicando el método *sort()* y un comparador adaptado para la clase *RecipeData* de forma similar a como se ordenaban las recetas de la aventura, ayudándose de un procedimiento refrescador de la lista, *adapterCharger()*. El problema aparece cuando hacemos un giro de la pantalla, porque los botones destinados a mostrar la selección pierden el filtro gris del estado y la lista el orden seleccionado porque se vuelve a cargar la actividad. Esto se solventa declarando las variables booleanas que manejan la situación de la selección como estáticas y añadiendo otra bandera, también *static*, que indica cuándo la pantalla se ha volteado, situación que se recoge mediante *onSaveInstanceState()*, y comprobando que el *ArrayList* de *RecipeData* se haya rellenado. Dependiendo de todo esto, se establecerá cómo deben estar.

6.4.6 El modo batidora, selecciones excluyentes y cálculo de porcentajes

Como ya se ha comentado anteriormente, este es otro de los modos más complejos por tener que lidiar con la inclusión de la selección de ingredientes con cantidades y alérgenos en un mismo listado y por tener que implementar las funciones necesarias que den sentido y forma a los porcentajes de aproximación de las recetas. Esto requerirá del uso de varias actividades auxiliares para poder visualizar todo lo que necesitamos, con un nivel de conexión entre ellas bastante apreciable.

La vista principal se compone de dos botones para añadir alérgenos o ingredientes, una zona donde se irán añadiendo las selecciones y un botón para obtener los resultados. Se nos permitirá seleccionar elementos que se sacarán del listado de seleccionables una vez añadidos a la lista a batir, en formato de cadenas, y diferenciándose internamente mediante un símbolo inicial '#'. Dependiendo de si lo que se quiere insertar en la lista es un alérgeno o un ingrediente, se iniciará una actividad diferente para cada caso, que después serán gestionadas a la vuelta a *MixerAct.java* mediante *onActivityResult()*.

Concretemos la explicación desgranándola paso a paso. Cuando el usuario quiere introducir un alérgeno, si no se han seleccionado todos (los 14 reconocidos oficialmente por el Consejo Europeo de Información sobre la Alimentación [10]), se proseguirá con la actividad *MixerAllergSelect*. Lo único que hace esta última es mostrar los elementos en una lista, dependiendo de lo que reciba en el *ArrayList* que se incluye en el *Intent* (línea 21), y devolver la cadena de caracteres que compone el ítem seleccionado.

- MixerAct.java

```
1  Button allergenSelector = (Button) findViewById(R.id.allergenSelector);
2  allergenSelector.setTypeface(Typeface.create("sans-serif-light", Typeface.NORMAL));
3  selectAllergens();
```

```
4  allergenSelector.setOnClickListener(new View.OnClickListener() {
5       @Override
6       public void onClick(View v) {
7             if(allergens.isEmpty()) {
8                   Toast msg = Toast.makeText(getApplicationContext(),
9                                     R.string.noItemMixer, Toast.LENGTH_SHORT);
10                  msg.show();
11            }
12            else {
13                  ArrayList<String> allergenList = new ArrayList<String>();
14                  for(int i = 0; i < allergens.size(); i++)
15                        if(!mixerList.contains("#" + "· " +
16                              getResources().getString(R.string.noContain)
17                              + " " + allergens.get(i)))
18                              allergenList.add(allergens.get(i));
19
20                  Intent intent = new Intent(MixerAct.this, MixerAllergSelect.class);
21                  intent.putStringArrayListExtra("mixerList", allergenList);
22                  allergenBool = true;
23                  db.close();
24                  startActivityForResult(intent, REQUEST_CODE);
25            }
26      }
27 });
```

Aquí ya se ve que los alérgenos irán precedidos de '#' y que se utilizará el carácter punto '·' como viñeta para listar los elementos de los que queremos obtener resultados (línea 15 a 17), indiferentemente sean alérgenos o ingredientes. Tendremos en cuenta estos detalles para trabajar convenientemente con las cadenas.

El resultado de la selección del alérgeno en *MixerAllergSelect* como ítem del listado queda guardado en un *Intent* (línea 6), que es gestionado al volver a la actividad *MixerAct* por medio del método *onActivityResult()* (línea 7).

- MixerAllergSelect.java

```
1  mixerListView.setOnItemClickListener(new OnItemClickListener() {
2       @Override
3       public void onItemClick(AdapterView<?> adapter, View v, int position, long l) {
4             String itemSelected = (String) mixerListView.getItemAtPosition(position);
5             Intent intent = new Intent();
6             intent.putExtra("itemSelected", itemSelected);
7             setResult(RESULT_OK, intent);
8             finish();
9       }
10 });
```

En el caso de los ingredientes, lo primero que debemos hacer es crear una clase propia para poder manejar los datos que nos interesan. En este caso se llamará *IngredientType* y nos serviremos de ella para mostrar los listados debidamente, completo o separado por tipos, almacenando tipo y nombre de ingrediente seleccionado. Iniciamos la selección pulsando el botón oportuno, que hará que se inicie la actividad *MixerIngredSelect*, encargada de gestionar los diferentes escenarios de listados a través de la espera de resultados de la finalización de las actividades que ejecuta. Para reutilizar código, las diferentes listas de elementos serán representadas de la misma forma que los alérgenos, usando la misma actividad

(líneas 11 y 12). Dependiendo del código que se le haya asignado en *startActivityForResult()*, se mostrarán los tipos disponibles, se acotarán los ingredientes por dichos tipos, se mostrarán todos independientemente del tipo o se pedirá una cantidad de un ingrediente concreto (líneas 1 a 14).

- MixerIngredSelect.java, onActivityResult()

```
1  @Override
2  protected void onActivityResult(int requestCode, int resultCode, Intent data) {
3       if(requestCode == REQUEST_CODE_TYPE)
4             IngredientListSelection(requestCode, resultCode, data);
5
6       if(requestCode == REQUEST_CODE_TYPE_SELECTED)
7             getAmountForIngredient(requestCode, resultCode, data);
8
9       if(requestCode == REQUEST_CODE_ALL)
10            getAmountForIngredient(requestCode, resultCode, data);
11
12      if(requestCode == REQUEST_CODE_AMOUNT)
13      forActivityFinish(requestCode, resultCode, data);
14 }
```

- MixerIngredSelect.java

```
1  public void IngredientListSelection(int requestCode, int resultCode, Intent data) {
2       if(resultCode == RESULT_OK) {
3             ArrayList<String> ingredientsWithType = new ArrayList<String>();
4             String ingrType = data.getStringExtra("itemSelected");
5
6             for(int i = 0; i < ingredientList.size(); i++)
7                   if(typeListForIngredients.get(i).equals(ingrType))
8                         ingredientsWithType.add(ingredientList.get(i));
9
10            Collections.sort(ingredientsWithType);
11            Intent intent =
12                        new Intent(MixerIngredSelect.this, MixerAllergSelect.class);
13            intent.putExtra("mixerList", ingredientsWithType);
14            startActivityForResult(intent, REQUEST_CODE_TYPE_SELECTED);
15      }
16      else {
17            setResult(RESULT_CANCELED);
18            finish();
19      }
20 }
```

Cuando se ha seleccionado finalmente un ingrediente, se iniciará una nueva actividad auxiliar que rescata de la base de datos el tipo de medida de dicho ingrediente y la almacena junto con el nombre del ingrediente y el número insertado por el usuario. Se ha tenido en cuenta que, si el usuario no establece ninguna cantidad o escribe la cifra cero, el sistema mostrará que al menos es necesario añadir una cantidad contable, puesto que tener cero de algo no tiene sentido en este ámbito.

Una vez seleccionados diferentes elementos, se le permitirá al usuario eliminarlos si son alérgenos borrándolos de la lista y eliminarlos o editar la cantidad si son ingredientes, función que lanza de nuevo la actividad *MixerIngredAmount*. Estas variaciones son gestionadas desde la clase creada para el adaptador personalizado de la lista para batir, llamada *MixerItemAdapter*, que es totalmente necesario, ya que, además del texto con el elemento a batir, contiene dos elementos de tipo *ImageButton* (líneas 1 y 10)

para editar y eliminar. Si no configuráramos dichas funciones dentro del adaptador, al pulsar cualquiera de los dos botones el sistema no podría diferenciar, y permitiría hacer clic en cualquier punto del ítem, dando resultados indeseados.

- MixerItemAdapter.java

```
removeItem = (ImageButton) convertView.findViewById(R.id.removeItem);
removeItem.setOnClickListener(new OnClickListener() {
      @Override
      public void onClick(View v) {
            String item = getItem(position);
            remove(item);
      }
});

editItem = (ImageButton) convertView.findViewById(R.id.editItem);
editItem.setOnClickListener(new OnClickListener() {
      @Override
      public void onClick(View v) {
            v.setTag(position);
            String item = getItem(position);
            tring[] splitted = item.split(" / ");
            Intent intent = new Intent(getContext(), MixerIngredAmount.class);
            intent.putExtra("itemSelected", splitted[0].substring(2));
            ((Activity) getContext()).
                  startActivityForResult(intent, REQUEST_CODE_AMOUNT);
}
});
```

Cuando haya al menos un ingrediente en la lista, independientemente de los alérgenos, podremos dar paso a la visualización de resultados a través de la actividad *MixerResult*. Esta actividad obtiene el listado de los ingredientes y cantidades para cada receta y lo compara con la petición del usuario, seleccionando sólo las recetas que tengan algo en común y eliminando las que contengan los alérgenos seleccionados. Para llevarlo a cabo, se apoya en dos clases implementadas: *IngredientsForRecipe*, que obtiene para cada receta su id, la id de los ingredientes, la cantidad, el tipo de medida y alérgenos que pueda contener, y *PercentageRecipes*, que guarda datos necesarios para el porcentaje de acierto y la representación de cada receta como son su id, la id del tipo, los pasos que tiene, el nombre, si está chequeada o no y el porcentaje de aproximación a lo requerido por el usuario, que explicaremos en breve de dónde sale.

El primer paso es crear un *ArrayList* de *IngredientsForRecipe*, llamado *ingredientsForRecipes*, donde se obtendrá toda la información necesaria de las recetas para poder compararlas con la lista a batir. Esto se consigue obteniendo de la base de datos los datos de todas las recetas (líneas 7 a 13) y creando *ArrayList* de cada uno de los tipos para añadirlos a la lista de la clase personalizada. El procedimiento *getIngredientsFromRecipes()* es el encargado de ello, y la consulta usada es la siguiente, adaptada para la correcta ejecución mediante la clase *SQLHelper*:

- MixerResult.java: getIngredientsFromRecipes()

```
ArrayList<Integer> ingredientsID  = new ArrayList<Integer>();
ArrayList<Integer> ingredientsAmount = new ArrayList<Integer>();
ArrayList<Integer> measuresID  = new ArrayList<Integer>();
ArrayList<Integer> allergensID = new ArrayList<Integer>();
int recipeID;
```

```
queryResult = SQLHelper.select("RecipeSteps.idRecipes, IngredientItems.idIngredients,
     amount, IngredientItems.idMeasures, idAllergen", "RecipeSteps, IngredientItems,
     Measures,    Recipes,    Ingredients",    "RecipeSteps.idIngredientItems    =
     IngredientItems.idIngredientItems  "  +  "AND  IngredientItems.idIngredients  =
     Ingredients.idIngredients    "    +    "AND    IngredientItems.idMeasures    =
     Measures.idMeasures  "  +  "AND  Recipes.idRecipes  =  RecipeSteps.idRecipes  "  +
     "GROUP                  BY                  RecipeSteps.idIngredientItems");
```

Usando un bucle *for* con una serie de sentencias condicionales, conseguimos guardar los datos de las recetas de forma correcta para su posterior comparación.

El procedimiento *mixerListIDs()* se encarga de hallar las ids de alérgenos e ingredientes para poder cotejarlas con las instancias de la lista *ingredientsForRecipes*. Como anteriormente se dijo, las cadenas de la lista a batir tienen un formato definido internamente para poder representarlas y diferenciarlas (alérgeno o ingrediente), por lo que en este punto debemos tener esto muy en cuenta. Obtendremos cada id con ayuda de los métodos *split()*, *charAt()* e *indexOf()*, el primero para separar el nombre del ingrediente de la cantidad, el segundo para saber si es alérgeno y el tercero para obtener la id teniendo en cuenta que al hacer una consulta, según nuestra base de datos usando *SELECT*, los índices salen ordenados, con lo que equivaldrán al índice de la lista de elementos de la BD más uno (*SQLite* asigna al primer elemento id 1 y no 0).

En el siguiente paso se eliminarán las recetas que no contengan ningún ingrediente o que contengan algún alérgeno (líneas 12 y 13) de los especificados en la actividad principal. Para ello, recorremos el *ArrayList ingredientsForRecipes*, comparando los elementos de cada receta con lo que estamos buscando. Si contuviera algún alérgeno de los seleccionados o no tuviera ningún ingrediente de los requeridos, será eliminada del resultado (líneas 22 a 25). El procedimiento encargado de esto es *recipesArranger()*:

- MixerResult.java

```
public void recipesArranger() {
     boolean recipeToRemoveAllergen, recipeToRemoveNoIngredient;
     IngredientsForRecipe recipeItem;

     for(int i = 0; i < ingredientsForRecipes.size(); i++) {
           recipeItem = ingredientsForRecipes.get(i);
           recipeToRemoveAllergen = false;
           recipeToRemoveNoIngredient = false;

           for(int j = 0; j < allergenListID.size()
                      && recipeToRemoveAllergen == false; j++)
                 if(recipeItem.allergensID().contains(allergenListID.get(j)))
                      recipeToRemoveAllergen = true;

           if (!recipeToRemoveAllergen)
                 for(int j = 0; j < ingredientListID.size()
                                  && recipeToRemoveNoIngredient == false; j++)
                      if(recipeItem.ingredientsID()
                                       .contains(ingredientListID.get(j)))
                            recipeToRemoveNoIngredient = true;

           if(recipeToRemoveAllergen == true
                            || recipeToRemoveNoIngredient == false) {
                 ingredientsForRecipes.remove(i);
```

```
                  i--;
            }
            else
                  getResultsForListItems(recipeItem.recipeID(),
                              hitPercentageForRecipe(recipeItem));
      }
}
```

Como se puede ver en la última línea con código, para finalizar la búsqueda de los aciertos y obtener los resultados finales nos ayudamos de dos métodos, uno para cada cometido. La obtención de los porcentajes de coincidencia tiene una lógica puramente matemática, aplicando reglas de tres sucesivas, centrada en subdividir primero por el total de ingredientes y segundo por la cantidad de cada uno. Veamos de una forma más grafica este concepto:

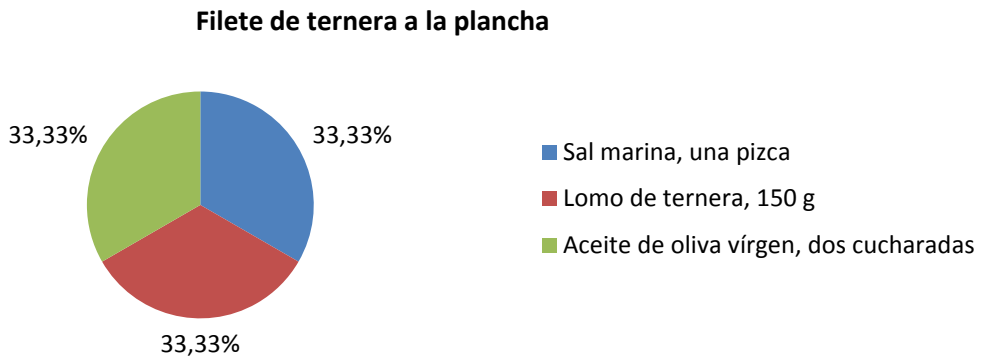


Figura 6.3: Funcionamiento de hitPercentageForRecipe(): Diagrama de subdivisión de ingredientes

En el gráfico anterior tenemos el 100% de una receta. Si, por ejemplo, en vez de dos cucharadas de aceite tuviéramos sólo una, la ponderación de ese ingrediente se vería subdividida por la cantidad de la que disponemos en contraposición de la necesitada, con lo que en vez del 100% de ese ingrediente (un 33,33% en el total de la receta) tendríamos un 50% (16,66% en el total de la receta). Entonces, el porcentaje de acierto para esta receta sería de un 83,33%. La función se construye de la siguiente forma:

- MixerResult.java

```
public int hitPercentageForRecipe(IngredientsForRecipe recipeData) {
      double accuracy = 0.0;
      int numberOfIngredients = recipeData.countIngredients();
      double ingredientTotalPercentage = 100.0 / numberOfIngredients;

      for(int i = 0; i < numberOfIngredients; i++)
            for(int j = 0; j < ingredientListID.size(); j++)
                  if(recipeData.ingredientsID().get(i) == ingredientListID.get(j))
                        if(recipeData.ingredientsAmount().get(i)
                                          <= amountList.get(j))
                              accuracy += ingredientTotalPercentage;
                        else
                              accuracy +=
                                    ingredientTotalPercentage * amountList.get(j)
                                    / recipeData.ingredientsAmount().get(i);

      return (int) accuracy;
}
```

Se ha decidido que esto sea así porque es una forma lógica de acometer el problema. Otra hubiera sido hacer los porcentajes dependiendo de los gramos totales de los ingredientes, pero ese punto de vista hace que se pierda semántica, además de dificultar la compresión del resultado final. La ventaja es que el usuario solo tendrá que hacer tres clics de pantalla para saber qué ingredientes necesita para la receta. Si faltara algún ingrediente por completo, la mayoría de las veces podría sustituirse por uno similar, lo que promueve perspicacia del usuario.

Los datos concretos de cada receta con algún porcentaje de acierto se obtienen con ayuda del método *getResultsForListItems()*. Teniendo la información podremos representar la lista de resultados con todos los detalles. Esta vez se ha decidido implementar el adaptador personalizado dentro de la actividad por ser la única que va a hacer uso de él, llamado *MixerResultAdapter*, que va a contener ítems con el porcentaje de acierto a la izquierda, siendo no seleccionable, y el recuadro con el nombre de la receta, seleccionable, chequeada o no chequeada. La función *adapterCharger()* ordena según porcentajes de acierto de mayor a menor e inicializa el adaptador para su visualización. Esto se consigue englobando los elementos que queremos que tengan la capacidad de ser pulsados en una vista padre, y dándole dicha propiedad

6.4.7 El reloj de cocina: modo cronómetro y uso de *CountDownTimer*

La funcionalidad de esta actividad es aparentemente básica, pero encierra ciertos desarrollos que revisan de una notoria complejidad. Durante la implementación han ido apareciendo problemas como cuál es la mejor forma de acometer una cuenta atrás en *Android*, qué clase es la más conveniente para manejar objetos de tipo tiempo y cómo conservar adecuadamente las instancias de clases y variables al girar la pantalla del dispositivo. Iremos comentando en profundidad cada uno de los casos.

Lo primero que se debe plantear es cómo poder realizar una cuenta atrás con la funcionalidad de comenzarla, pausarla y resetearla. Tras la búsqueda y análisis de diferentes clases, se ha decidido que lo más oportuno es usar la clase *CountDownTimer*, perteneciente al paquete *Android.os*. Los métodos que acompañan a esta clase serán más que necesarios para poder crear el cronómetro de cuenta atrás que dará vida a nuestro reloj de cocina. El usuario introducirá unos valores de tiempos en horas minutos y segundos correctos en las casillas de tipo *EditText* destinadas para ello, que serán verificados mediante el método *setTime()*. El sistema avisará de cuál es el fallo en caso de que haya algún error mediante un mensaje en ventana flotante gestionado por la clase *Toast*.

Si todo es correcto se almacenan los valores de tiempo en un objeto llamado *time*. Esto ha sido otro punto de inflexión a la hora de implementar este apartado de la aplicación, puesto que las clases nativas de *Java* no hacían exactamente lo demandado por los requisitos. El problema surge por el uso de *CountDownTimer*, que necesita un valor en milisegundos para funcionar, lo que nos obliga a hacer conversiones constantemente para pasar de milisegundos a horas, minutos y segundos y viceversa (líneas 21 y 24). *Time* o *Calendar*, pertenecientes a *Java*, conseguían realizar una cuenta atrás acometiendo las conversiones de tiempo adecuadamente, pero al estar subyugadas por las diferentes franjas horarias no se terminaba de conseguir que la cuenta atrás finalizara cuando el contador digital marcara cero, sino que lo hacía una hora antes. Por ello y por requerir solamente de la conversión de unidades de tiempo, que es algo que no precisa de cálculos complejos, se tomó la decisión de crear una clase conversora de tiempo personalizada, llamada *CustomTime*, que única y exclusivamente diera soporte a dicha necesidad, obviando el resto de funcionalidades que ofrecen las clases propias.

- ChronoAct.java

```
1  public class CustomTime {
2       private int sec, min, hour;
3       private long millisec;
4
5       public CustomTime(int sec_, int min_, int hour_) {
6                    sec = sec_; min = min_; hour = hour_; millisec = 0;}
7       public CustomTime(long millisec_, int sec_, int min_, int hour_) {
8                    millisec = millisec_; sec = sec_; min = min_; hour = hour_;}
9       public CustomTime() {millisec = 0; sec = 0; min = 0; hour = 0;}
10
11      public long millisec() {return millisec;}
12      public int sec() {return sec;}
13      public int min() {return min;}
14      public int hour() {return hour;}
15
16      public void millisec(long millisec_) {millisec = millisec_;}
17      public void sec(int sec_) {sec = sec_;}
18      public void min(int min_) {min = min_;}
19      public void hour(int hour_) {hour = hour_;}
20
21      public long timeToMillisec() {
22                   return millisec + 1000 * ((hour * 3600) + (min * 60) + sec);}
23
24      public void millisecToTime(long millisec_) {
25             int milliMod, secMod, minMod, milliDiv, secDiv, minDiv;
26             milliMod = (int)millisec_ % 1000; milliDiv = (int)millisec_ / 1000;
27             secMod = milliDiv % 60; secDiv = milliDiv / 60;
28             minMod = secDiv % 60; minDiv = secDiv / 60;
29             this.millisec(milliMod);
30             this.sec(secMod);
31             this.min(minMod);
32             this.hour(minDiv);
33      }
34 }
```

Una vez definido cómo se va a gestionar el tiempo vamos a comentar de qué forma funcionan los métodos para dar vida a la cuenta atrás. El botón encargado para comenzar es de tipo *ToggleButton*, lo que quiere decir que cambiará su estado al hacer clic en él, dependiendo de si el cronómetro está en marcha o pausado. Si es la primera vez que se inicia, deberá existir un valor válido de tiempo introducido por el usuario, que al pulsar el botón se traspasará al contador y cambiará a durmiendo imagen del pájaro. Si está en marcha, se ofrecerá la opción de pausar, cancelando la cuenta atrás mediante el método *cancel()* y guardando la instancia de los datos en ese momento para poder reanudarla cuando se pulse de nuevo continuar. Estas acciones dan paso a la iniciación de *CountDownTimer*, que se encarga de gestionar la cuenta regresiva en milisegundos y que muestra en pantalla el estado del tiempo mediante el formateo del *TextView* llamado *chrono*, además, cuando llegue al final todos los valores se resetearán, haciendo que el pájaro despierte y emitiendo una señal sonora, acciones definidas en *onFinish()*. Todo esto se procesa con el procedimiento *countDownStarter()*.

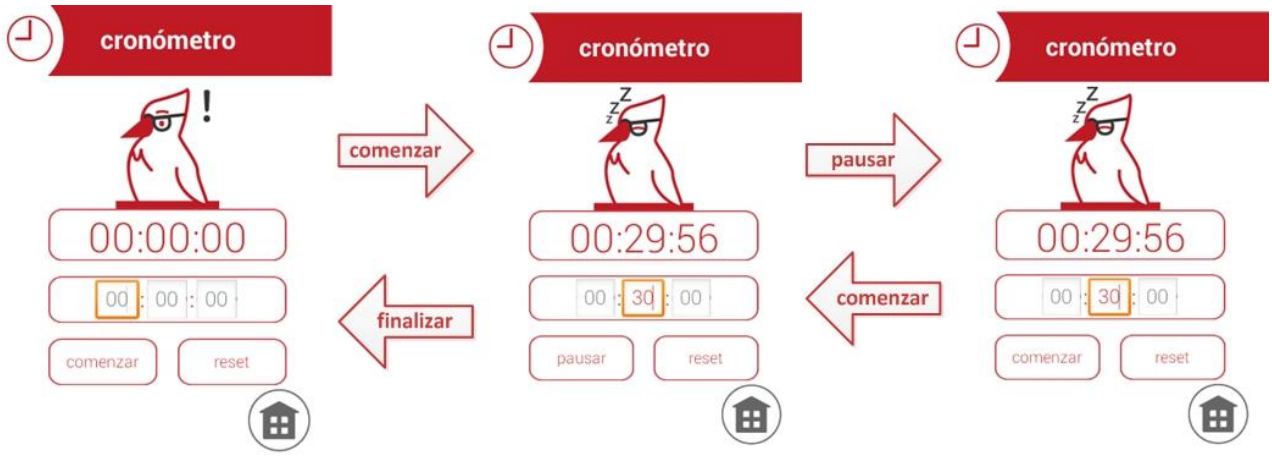


Figura 6.4: Vistas de ChronoAct.java: Pasos por los diferentes estados

Hay que resaltar un detalle en los parámetros de la función *CountDownTimer()*. Se le añaden 50 milisegundos a la cuenta atrás y los intervalos se contabilizan cada medio segundo para que cuando se muestre el tiempo por pantalla no haya problemas de saltos de segundos y cuando llegue a cero con algún decimal se contabilice siempre un poco más para que la visualización sea la correcta.

Para finalizar, debemos tener en cuenta que, al ser una actividad dinámica que cambia cada segundo, tendremos que asegurar la reconstrucción de la misma cuando se gire la pantalla, recuperando el estado interno y visual de los elementos. Esto se consigue haciendo las variables estáticas para que no sean destruidas junto con el proceso de vida de la actividad, aplicando unas banderas (línea 7) para diferenciar los estados de pausado y activo y manejando todo esto con ayuda de *onSaveInstanceState()* (línea 21).

- ChronoAct.java

```
1  @Override
2  protected void onCreate(Bundle savedInstanceState) {
3
4       if(rotateFlat) {
5             if (currentTime != 0) {
6                   time.millisecToTime(currentTime);
7                   if(checkButton)
8                         countDownStarter();
9                   else {
10                        birdImg.setImageResource(R.drawable.chrono_icon_birdsleep);
11                        chrono.setText(String.format("%1$02d", time.hour())
12                              + ":" + String.format("%1$02d", time.min()) + ":"
13                              + String.format("%1$02d", time.sec()));
14                  }
15            }
16            rotateFlat = false;
17      }
18 //...
19 }

20 @Override
21 public void onSaveInstanceState(Bundle savedInstanceState) {
22      rotateFlat = true;
23      if(CDT != null)
24            CDT.cancel();
```

```
25      super.onSaveInstanceState(savedInstanceState);
26  }
```

Una vez acabada la actividad principal, surgió un problema que, más que ser un error, podría considerarse una necesidad inherente a la naturaleza de un cronómetro. Lo lógico es que, mientras el cronómetro está activo, el usuario pueda pasarlo a segundo plano abriendo otra aplicación, pulsando el botón *home* del dispositivo o bloqueándolo. Si esto se lleva a cabo, la aplicación se pausa y se guarda una instancia del estado en el que se encuentra, y, en ningún caso, continuaría la cuenta atrás. Para darle solución, se deberá usar lo que en *Android* se llama *Service*, que no es más que una tarea que se ejecuta en segundo plano, sin necesidad de que el usuario interactúe con ella. Como la actividad tendrá un canal de comunicación con el servicio para transmitir el estado de la cuenta atrás, el tipo que tendremos que usar es el de servicio ligado (*bindService*) para poder acceder a los métodos integrados en *ChronoService*, que así se llamará, y conocer su situación. En vez de extender de *Activity*, para implementar nuestra clase hay que hacerlo de esta vez de *Service*, que provee los métodos necesarios para su funcionamiento que tendremos que sobrescribir. La clase que nos ayuda a comunicar la actividad y el servicio es Binder, que también se deberá extender en una personalizada para poder devolver los objetos. El método *onBind()* (línea 2) es el encargado de recoger la información de la actividad invocadora, de procesarla y de devolver el correspondiente objeto.

- ChronoService.java

```
1   @Override
2   public IBinder onBind(Intent intent) {
3       milliseconds = (Long) intent.getExtras().get("milliseconds");
4       notifyIntent = new Intent(ChronoService.this,
5                   ChronoAct.class).setFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP
6                                           | Intent.FLAG_ACTIVITY_CLEAR_TOP);
7       notifyIntent.putExtra("chronoTimeFromRecipe",
8                   (Integer) intent.getExtras().get("chronoTimeFromRecipe"));
9       notifyIntent.putExtra("userID", (Integer) intent.getExtras().get("userID"));
10
11      CDT = new CountDownTimer(milliseconds + 50, 500) {
12            public void onTick(long countDown) {
13                  time.millisecToTime(countDown);
14                  isFinished = false;
15            }
16
17            public void onFinish(){
18                  mediaPlayer = MediaPlayer.create(ChronoService.this,
19                                        R.raw.bird_full);
20                  mediaPlayer.setLooping(true);
21                  mediaPlayer.start();
22                  if(!mediaPlayer.isPlaying())
23                        mediaPlayer.release();
24                  isFinished = true;
25            }
26      }.start();
27
28      ChronoAct.CustomTime timeText = chronoInstance.new CustomTime();
29      timeText.millisecToTime(milliseconds);
30      chrono = (String.format("%1$02d", timeText.hour()) + ":"
31                        + String.format("%1$02d", timeText.min()) + ":"
32                        + String.format("%1$02d", timeText.sec()));
33
```

```
34      PendingIntent pendingIntent = PendingIntent.getActivity(this,
35                                  0, notifyIntent, PendingIntent.FLAG_UPDATE_CURRENT);
36      NotificationCompat.Builder builder = new NotificationCompat.Builder(this);
37      notify = builder.setContentIntent(pendingIntent)
38                          .setSmallIcon(R.drawable.chrono_icon_mode)
39                          .setTicker("aCubierto: cronómetro en curso")
40                          .setWhen(System.currentTimeMillis())
41                          .setAutoCancel(true).setContentTitle("aCubierto")
42                          .setContentText(chrono).build();
43
44  notifyManager.notify(1, notify);
45
46  return iBinder;
47  }
```

Varios detalles a comentar. El primero es el establecimiento de *flags* (líneas 5 y 6) en el *Intent* que se encargará de volver a la actividad cuando se seleccione la notificación que se crea y que en breve explicaremos. Con esto se logra que la actividad *ChronoAct* esté una vez abierta al mismo tiempo y que no se lancen nuevas instancias si ya existe una en la pila de actividades, si no, la actividad se iría duplicando cada vez que saliéramos del cronómetro y pulsáramos en la notificación. Lo siguiente es el uso de nuevo de *CountDownTimer* (líneas 10 a 26) para poder seguir con la cuenta atrás, con una bandera en su interior (líneas 14 y 24) para poder comunicarle a la actividad principal cuándo ha finalizado mediante el método *isFinished()*. Ya al final de la función, se crea un *PendingIntent* (línea 34)para indicarle a la notificación, que se crea mediante la clase *Notification*, cómo se debe actuar cuando esta es seleccionada en la barra de notificaciones, que alojará un indicador con un título, la hora en la que fue lanzada y el tiempo que faltaba cuando se salió de la aplicación para que terminara la cuenta atrás (líneas 37 a 42).

Por último, se deberán sobrescribir los métodos *onPause()* y *onResume()* de ChronoAct para que actúen adecuadamente. Es aquí donde se dan las órdenes para vincular el cronómetro con el servicio y poder comunicar con sus métodos, además de desligar el servicio cuando este haya acabado, acción que es necesaria para que el sistema reproduzca el comportamiento deseado. Para que el funcionamiento cuando se lanza el cronómetro desde un paso de receta sea correcto se debe tener cuidado con desligar un servicio cuando aún no ha sido ligado, puesto que, para volver al paso del que procedemos, si usamos *System.exit(0)*, que elimina el valor de las variables estáticas, volveremos al primer paso de la receta, y no del que se provenía. Se usa una bandera, *returnToRecipe*, para saber cuándo se ha vuelto al paso y que *onResume()*, si se accede de nuevo al cronómetro, no aplique *unBindService()* como si la procedencia fuera el haber pulsado en la notificación del servicio. El tiempo se establecerá de forma correcta mediante el procedimiento *setTimeFromRecipe()*, donde las únicas medidas con sentido dentro del contexto serán horas y minutos, mediante la conversión de los minutos requeridos:

6.4.8 Modo consejos, desbloqueados y bloqueados

La implementación del último de los modos es simple, estando compuesta de una actividad principal que visualizar el listado personalizado de consejos desbloqueados seleccionables, y otra secundaria que permite ver los consejos que aún no han sido desbloqueados en forma de previsualización, o sea, no seleccionables.

Necesitaremos una clase que englobe los datos que queremos recuperar de la base de datos. Esta se llama *TipData*, y encapsula el tipo de técnica de la que deriva el consejo, el nombre de la receta de la que

ha sido desbloqueada, la ID del consejo y su descripción completa. Esto se almacena secuencialmente en un *ArrayList* de nombre *tipsData*. Cuando se inicie el modo, el procedimiento *getDataForTips()* se encargará de obtener toda la información requerida para inicializar cada uno de los objetos que comprenderán los consejos.

El adaptador personalizado se establece dentro de la actividad por ser la única que va a hacer uso de él. La actividad secundaria para consejos aún bloqueados tendrá el mismo adaptador personalizado con la diferencia sustancial de que el layout tendrá un diseño emergente, por lo que tendremos que volver a incluirlo en dicha actividad para poder enlazarlo con su respectiva vista. Además de esto, los elementos no serán seleccionables en contraposición de la actividad principal. La visualización del listado gestionada por el adaptador será la siguiente:

- TipListAct.java: TipAdapter()

```
@Override
public View getView(int pos, View convertView, ViewGroup parent) {
     TipData tipListItem = getItem(pos);

     if (convertView == null)
     convertView = LayoutInflater.from(getContext())
                 .inflate(R.layout.tip_list_adapter, parent, false);

     TextView recipeTitle = (TextView) convertView.findViewById(R.id.recipeTitle);
     recipeTitle.setText(ToCookAdapter.getSetTitle(tipListItem.recipeName()));

     TextView tipTechnicType = (TextView)
                  convertView.findViewById(R.id.tipTechnicType);
     tipTechnicType.setText(tipListItem.technicType());
     tipTechnicType
                 .setTypeface(Typeface.create("sans-serif-light", Typeface.NORMAL));

     TextView tipDescription =
                 (TextView) convertView.findViewById(R.id.tipDescription);
     tring description = tipListItem.tipDescription().substring(0, 30) + "...";
     tipDescription.setText(description);
     tipDescription
                 .setTypeface(Typeface.create("sans-serif-light", Typeface.NORMAL));

     return convertView;
}
```

La selección de cualquier ítem de la lista iniciará la actividad *TipUnlocker*, que mostrará una ventana emergente con toda la información del consejo en cuestión, obteniendo los datos que faltan realizando una segunda consulta a la base de datos. Dicha actividad será explicada con mayor detenimiento en el apartado de representación de recetas, puesto que es la misma que se usa para desbloquear los consejos de los pasos.

Tendremos un *TextView* que contabilizará los consejos que tenemos desbloqueados sobre el total existente para que el usuario conozca su recorrido. Esto es tan sencillo como aplicar el método *size()* a la lista de consejos desbloqueados y después realizar una consulta *SELECT* para obtener todos los consejos, a la que le aplicaremos de nuevo *size()*.

Para conocer los consejos que siguen bloqueados existe el botón con forma de candado. Si se pulsa en él, el sistema realizará una consulta a la base de datos obteniendo toda la información necesaria de cada uno de los consejos para poder mostrarlos como ítems de una lista en la que sólo tendremos una previsualización de los mismos, sin posibilidad de conocer la descripción completa del consejo. Si todos hubieran sido ya desbloqueados, mostraría un mensaje emergente comunicando al usuario que no existen consejos por desbloquear. En caso contrario, se iniciaría la actividad *TipsLocked*, que recibe la secuencia con mismos índices (nombre de receta, descripción de consejo y tipo de técnica) mediante *ArrayList* adheridos al *Intent* que se le envía.

6.4.9 Representación de las recetas

Pasemos a explicar cómo se ha implementado la parte que se considera la piedra angular de la aplicación. Se puede adelantar que estará compuesta de un buen número de actividades, cada una con un propósito bien diferenciado:

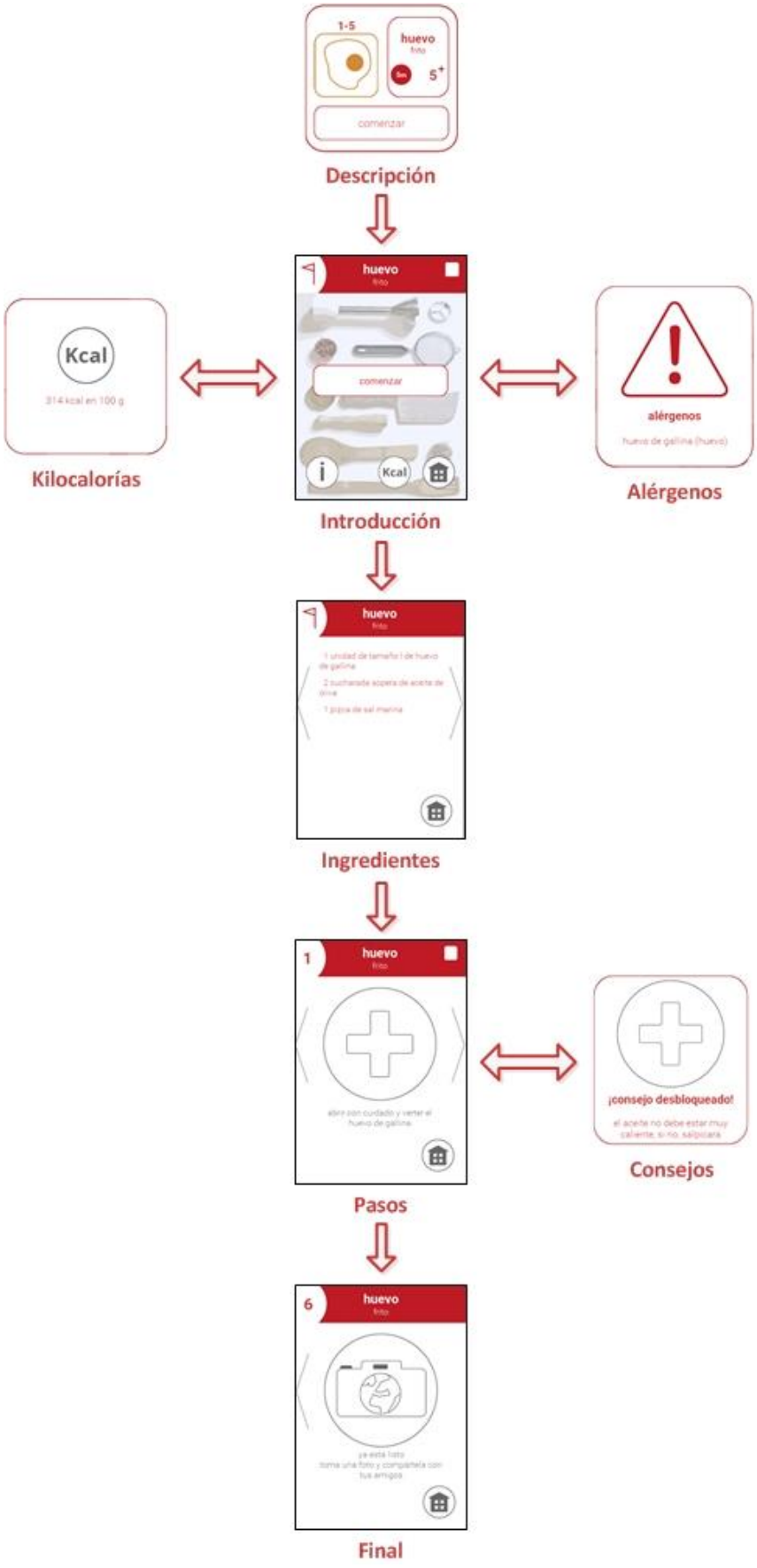


Figura 6.5: Representación de las recetas: Actividades implicadas

Cualquier modo que tenga como objetivo invocar una receta cualquiera usará de enlace una actividad que mostrará una descripción de la receta, en formato de ventana flotante, con información como el tiempo de realización, numero de pasos, pictograma de tipo, y, si se accede desde la aventura, el número de fase, desde otro modo cualquiera, el título de tipo de receta. Esta actividad no tiene mayor misterio que una serie de *vistas* para cada elemento a representar, modelados por los resultados que se recogen del *Intent* que es enviado desde el modo que haya sido seleccionado por el usuario. Como curiosidad, podría comentarse la forma en la que se obtiene el tiempo de realización de la receta, que es extraído de la base de datos y modelado en su vista, un botón no clicable con un background redondo rojo obtenido de una fuente de imagen, y que contiene el texto del tiempo como título del mismo, dependiendo de si se sobrepasa la hora o no(líneas 9 a 13).

- RecipeDescription.java

```
1  public void setRecipeTime() {
2       Button timeB = (Button) findViewById(R.id.timeButton);
3       timeB.setTypeface(Typeface.create("sans-serif", Typeface.BOLD));
4
5       queryResult = SQLHelper.select("recipeTime",
6                            "Recipes", "idRecipes = " + recipeID);
7       int time = Integer.parseInt(queryResult.get(0)[0]);
8
9       if(time < 59)
10            timeB.setText(String.valueOf(time) + "m");
11      else
12            timeB.setText(String.valueOf(time / 60) + ":"
13                         + String.valueOf(time % 60) + "h");
14 }
```

Pulsando el botón comenzar damos paso a la ficha de introducción de la receta que haya sido seleccionada. Mediante esta actividad, *RecipeAct*, podremos visualizar y acceder a diferente información de interés que podrán ser visualizadas mediante las actividades secundarias *KcalInfo* y *AllergenInfo*, donde se podrán consultar las kilocalorías y los alérgenos de la receta en cuestión. Los datos se almacenan bajo una serie de *ArrayLists* de cadenas y enteros, en los que se hacen coincidir los índices para posteriormente codificarlos en objetos de una clase personalizada que contengan la información de cada paso y del resto de elementos.

El esquema del diseño de la pantalla tendrá una estructura común para todo el conjunto de actividades encargadas de la representación de recetas, jugando con el cambio de algunos elementos, englobados en diferentes *layouts*, y jugando con su visibilidad mediante diferentes métodos que gestionan las vistas.

Figura 6.6: RecipeAct.java: Elementos informativos de la actividad

Esta actividad es la encargada de obtener toda la información necesaria para reproducir la receta en su totalidad, sin contar las cadenas que formarían la descripción de los pasos, pero sí las ids necesarias y el listado de ingredientes ya configurado. El procedimiento que tiene esta tarea encomendada se llama *getRecipeData()*.

Además de la información relativa a la realización de la receta, se obtiene también lo que podría llamarse el contexto mediante el procedimiento *getRecipeContext()*. Esto no es más que parte de la información de la receta que no será variable durante su realización, como son el título de la misma y el icono que representará el modo desde el que se ha accedido para efectuarla (líneas 3 a 26). Los tres modos que tienen acceso directo a las recetas son el modo aventura, el a cocinar y la batidora (líneas 10 a 26), cada uno con su icono representativo. Esto se repetirá también en las siguientes actividades de la misma forma.

- RecipeAct.java

```
1  public void getRecipeContext() {
2       appMode = bundle.getInt("appMode");
3       queryResult = SQLHelper.select("Recipes", "idRecipes", recipeID);
4       recipeName = SQLHelper.getValue(queryResult, 0, 1);
5
6       TextView recipeText = (TextView) findViewById(R.id.recipeTitle);
7       recipeText.setText(getSetTitle(recipeName));
8
9       ImageView pictoMode = (ImageView) findViewById(R.id.pictoMode);
10      switch(appMode) {
11            case 1: //Modo a aventura
12                  pictoMode.setImageResource(R.drawable.adventure_icon_mode);
13                  break;
14
```

```
15              case 2: //modo a cocinar
16                    pictoMode.setImageResource(R.drawable.tocook_icon_mode);
17                    break;
18
19              case 5: //modo batidora
20                    pictoMode.setImageResource(R.drawable.mixer_icon_mode);
21                    break;
22
23              default: //ERROR
24                    Log.d("ERROR DE PICTOGRAMA", "Elección del pictograma del modo");
25                    break;
26        }
27  }
```

En esta *Activity* se debe comprobar que la receta haya sido chequeada completamente. El chequeo de una receta tiene como condición estricta que todos sus pasos hayan sido clicados como realizados en cada una de las respectivas ventanas y se hayan mantenido así en la última realización, aunque en otro momento se haya deseleccionado alguno de ellos. El sistema hará una consulta *SELECT* en la tabla de las recetas chequeadas (*RecipesChecked*) usando la id de la receta, pudiendo dar como resultado tres escenarios diferentes: la consulta se devuelve vacía, o sea, que nunca se ha accedido a esa receta, con lo que se inserta una nueva fila con la id pertinente y la variable booleana de chequeo a cero (en *SQLite* los booleanos se representan mediante un entero 0,1) (líneas 5 a 7); la consulta devuelve cero, lo que quiere decir que hay al menos un paso no chequeado (líneas 9 y 10); se devuelve uno, que significa que la receta está totalmente chequeada (líneas 12 y 13). El método encargado de realizar chequeos en cada paso estará dentro de la actividad que articula dicho apartado.

- RecipeAct.java

```
1  public void recipeChecker() {
2       queryResult = SQLHelper.select("recipeCheck", "RecipesChecked",
3                             "idRecipes = " +  recipeID + " AND idUsers = " + userID);
4
5       if(queryResult.isEmpty())
6             SQLHelper.insert("RecipesChecked", "idUsers, idRecipes",
7                             userID + ", " + recipeID);
8
9       else if(Integer.parseInt(queryResult.get(0)[0]) == 1)
10            recipeCheckB.setImageResource(R.drawable.recipe_icon_checkboxcheck);
11
12      else
13            recipeCheckB.setImageResource(R.drawable.recipe_icon_checkboxnocheck);
14 }
```

Finalmente y antes de avanzar en la consecución de la receta, se nos da la posibilidad de saber qué alérgenos se contienen y la cantidad de kilocalorías. Para ello, nos apoyamos en dos actividades secundarias con formato de ventana emergente, que tienen el único cometido de mostrar ambas informaciones al usuario, llamadas *AllergenInfo* y *KcalInfo* respectivamente. La implementación en sí no revisa de gran dificultad, tan sólo, como dato reseñable, está bien comentar que las actividades se pueden moldear como ventana emergente (línea 4) directamente desde el *AndroidManifest.xml*, aplicando el tema adecuado:

- *Android*Manifest.xml

```
<activity
      android:name=".AllergenInfo"
      android:label="@string/title_activity_recipe_info"
      android:theme="@Android:style/Theme.Dialog" >
</activity>
```

Continuamos con la representación de nuestra receta visualizando el listado de ingredientes. Pulsando el botón comenzar lanzaremos la actividad *RecipeItemAct*. Lo único que realiza es la obtención de nuevo del contexto para la configuración de los elementos generales de la vista y la adición del *ArrayList* que contiene la lista de ingredientes rescatada del *Intent* enviado por la actividad anterior, *ingredientList*, a la vista oportuna mediante un adaptador estándar. Eso sí, el texto está formateado mediante un *layout* definido (*ingredients_textview.xml*) para que su presentación sea perfecta.

Es importante recalcar la aparición de un par de elementos que nos acompañarán en las siguientes pantallas, como son los botones de avanzar y retroceder, y la posibilidad de emularlos mediante el deslizamiento horizontal sobre la pantalla.

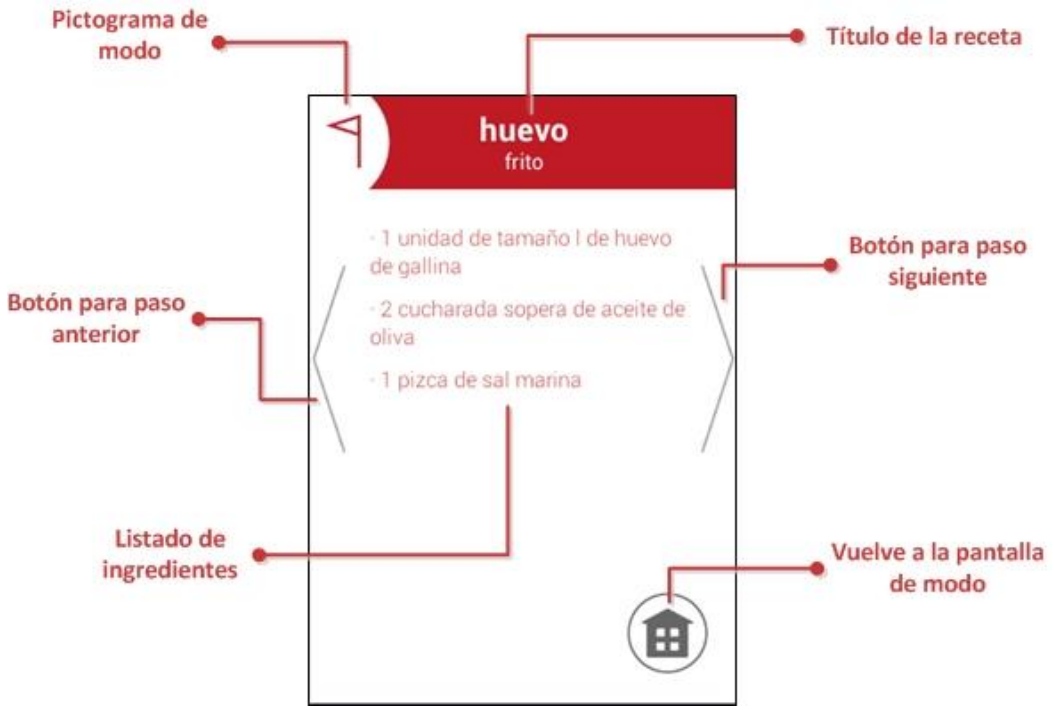


Figura 6.7: RecipeItemAct.java: Elementos informativos y de navegación de la actividad

Para poder aplicar el deslizamiento tenemos que hacer que nuestra clase de la actividad se convierta en una interfaz de la clase abstracta *OnTouchListener*. Esto se consigue añadiendo la palabra reservada *implements* a la definición de nuestra actividad.

- RecipeItemAct.java

```
public class RecipeItemAct extends Activity implements OnTouchListener {
      //Resto del código de la actividad
}
```

Esto nos obliga a implementar el método abstracto heredado *View.OnTouchListener.onTouch*. Dentro del mismo están las directrices de lo que se debe hacer según sea el movimiento, de derecha a izquierda o de

izquierda a derecha, teniendo dos procedimientos que lo resuelven iniciando las actividades adecuadas, *RecipeAct* (*recipeActivity()*, línea 9) o *StepAct* (*stepActivity()*, línea 12) respectivamente. Para que esto sea aplicable a todo el espacio donde es lógico (zona de listado y botones de navegación) que el usuario pueda moverse entre pantallas mediante deslizamiento, aplicamos el método escuchador a la vista padre, de tipo *FrameLayout*, con el nombre identificativo asignado en el *xml* de *movableLayout*, y a la vista de la lista.

- RecipeItemAct.java

```
1  @Override
2  public boolean onTouch(View v, MotionEvent event) {
3       if (event.getAction() == MotionEvent.ACTION_DOWN)
4             prevX = event.getX();
5       else if(event.getAction() == MotionEvent.ACTION_UP) {
6             float postX = event.getX();
7
8             if (prevX < postX)
9                   recipeActivity();
10
11            if(prevX > postX) {
12                  stepActivity();
13      }
14      return true;
15 }
```

Hacer clic en el botón de paso siguiente o deslizar el dedo sobre la pantalla de derecha a izquierda hará que se lance la actividad que representa los pasos de las recetas (línea 12). Lo primero que hay que hacer es dar forma a la secuencia de datos (ingredientes, técnicas, pasos y tiempos asociados) que suponen los pasos. Recordamos que, según la estructura de la base de datos, un paso tiene ligada una técnica y uno o varios ingredientes, por lo que tendremos que diferenciar cuáles son los correspondientes en cada momento. Para esclarecer mejor este asunto podemos observar el siguiente gráfico:

Figura 6.8: Datos de la BD: Representación interna de una receta

Vemos que, para esta receta, existen tres ingredientes a los que conjuntamente se les aplica primero una técnica, añadir en un vaso de batidora, y luego otra, batir con ayuda de unas varillas. La base de datos crea una fila por asignación de técnica a ingrediente en la tabla *RecipeSteps*, independientemente de si son uno o varios los manipulados, pudiéndose diferenciar los que tienen una misma técnica aplicada mediante la ID de la misma, que no cambiará hasta que empleemos una nueva con los mismos o con

otros ingredientes. A esto es a lo que llamaremos un paso, por lo tanto, para encapsular debidamente la información secuenciada que proviene de la actividad anterior, tendremos que separar cada paso, teniendo en cuenta la correspondencia técnica-ingrediente(s) de la forma que se acaba de explicar.

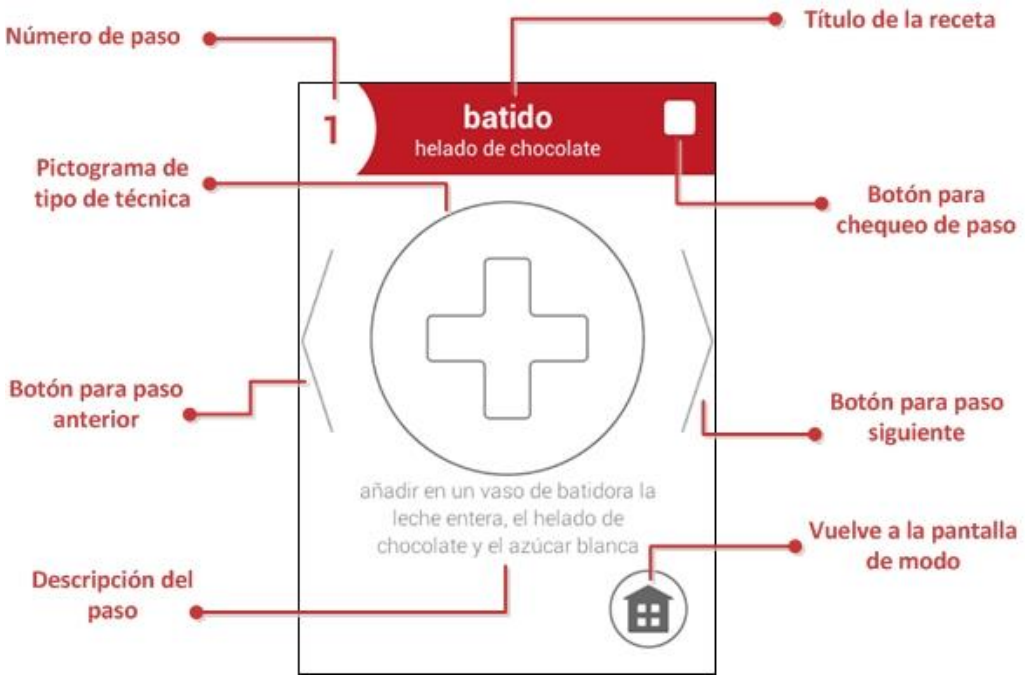


Figura 6.9: StepAct.java: Elementos informativos, de interacción y de navegación de la actividad

La clase implementada se llamará *Step* y, para alcanzar la finalidad planteada, almacenará la id última de *RecipeSteps* que tenga asociada una técnica a ingredientes (línea 9) (será la única que necesitaremos para el chequeo del paso y del consejo, que después explicaremos), la descripción de la técnica (línea 8), una lista con los ingredientes implicados (línea 10) y, si tuviera, su tiempo de realización asociado (si no, la BD devolverá un valor nulo, que para *int* sería cero) (línea 11). Está perfectamente modelada con su constructor y métodos observadores, que son los únicos que se van a usar. Con esta clase personalizada se creará un *ArrayList* llamado *steps*, que guardará todos los pasos de una receta listos para que se configure su descripción en pantalla mediante la unión de cadenas (línea 36). El procedimiento responsable de rellenar dicha lista con los objetos que representarán cada uno de los pasos es *getRecipeData()*, que sólo se invocará en una ocasión cuando se inicie la actividad para así no tener que repetir más de una vez el proceso y evitar que sea excesivamente ineficiente por tener un par de bucles *while* anidados. También hallará la cantidad de pasos conociendo el tamaño *steps* para poder numerar cada uno de ellos (línea 39).

- StepAct.java

```
1  public void getRecipeData() {
2        bundle = getIntent().getExtras();
3        userID = bundle.getInt("userID");
4        appMode = bundle.getInt("appMode");
5        recipeID = bundle.getInt("recipeID");
6
7        idItemList = getIntent().getIntegerArrayListExtra("idItemList");
8        idTechnicList = getIntent().getIntegerArrayListExtra("idTechnicList");
9        idStepList = getIntent().getIntegerArrayListExtra("idStepList");
```

```
10      ingredientList = getIntent().getStringArrayListExtra("ingredientList");
11      stepTimeList = getIntent().getIntegerArrayListExtra("stepTimeList");
12
13      int i = 0;
14      while(i < idTechnicList.size()) {
15            int techStep = idTechnicList.get(i), stepTime = stepTimeList.get(i);
16            queryResult = SQLHelper.select("techName", "Technics",
17                              "Technics.idTechnics = " + techStep);
18            String technic = new String(queryResult.get(0)[0]);
19            ArrayList<String> items = new  ArrayList<String>();
20
21                  while (i < idTechnicList.size() && techStep
22                              == idTechnicList.get(i)) {
23                        queryResult = SQLHelper.select("Determinants.determinant,
24                              Ingredients.ingrName",
25                              "Determinants, Ingredients, IngredientItems",
26                              "Determinants.idDeterminants =
27                              Ingredients.idDeterminant "
28                              + "AND IngredientItems.idIngredients =
29                              Ingredients.idIngredients " + "AND
30                              IngredientItems.idIngredientItems = "
31                              + idItemList.get(i));
32                        items.add(queryResult.get(0)[0] + " "
33                                          + queryResult.get(0)[1]);
34                        i++;
35                  }
36            steps.add(new Step(String.valueOf(idStepList.get(i - 1)),
37                              technic, items, stepTime));
38      }
39      totalSteps = steps.size();
40  }
```

Teniendo organizada toda la información que se va a necesitar, el siguiente paso es mostrarla al usuario por pantalla de forma organizada y coherente. Se usará parte del código que se creó en la actividad anterior para el método que configuraba los elementos comunes y tendrá el mismo tipo de nombre pero adaptado a los pasos, *getStepContext()*. Además de título de la receta, se necesitará modelar la descripción del paso con su número, si tiene consejo adjunto y el pictograma de la técnica que tendrá relacionada. A continuación, se muestra cómo se modela descripción, mediante *stepDescription()*, dependiendo del número de ingredientes que estén enlazados (1, 2, 3 o más) (líneas 6 a 11), y devolviendo directamente la cadena que la formaría y que configurará el *TextView* directamente (línea 13).

- StepAct.java

```
1   public String stepDescription() {
2       String stepToShow = new String(steps.get(counter).technic() + " ");
3       int numItems = steps.get(counter).items().size();
4
5       for(int i = 0; i < numItems; i++) {
6             if(numItems - i == 1)
7                   stepToShow = stepToShow + steps.get(counter).itemN(i);
8             else if(numItems - i == 2)
9                   stepToShow = stepToShow + steps.get(counter).itemN(i) + " y ";
10            else
11                  stepToShow = stepToShow  + steps.get(counter).itemN(i) + ", ";
12      }
```

```
13 return stepToShow;
14 }
```

Existen otros datos que son dependientes del usuario que esté usando la aplicación, como el estado de chequeo del paso o, si existiera un consejo adjunto, que haya sido desbloqueado con anterioridad. Para ello, es necesario consultar la parte de la base de datos que gestiona los usuarios y ver el estado actual. Esto se lleva a cabo mediante sendos procedimientos, cada uno con sus peculiaridades.

El encargado de los chequeos, además de salvaguardar la integridad del estado de la receta (líneas 36 a 50), que puede variar si desclicamos o clicamos un paso, debe dotar de un método escuchador al botón encargado para saber cuándo cambia su condición (línea 19). En los consejos, el visualizador de eventos estará implícito en el procedimiento *getStepContext()*, donde se contemplará el pulsar sin desplazamiento (la pulsación con desplazamiento tendrá las mismas consecuencias que en la actividad anterior) sobre el pictograma de tipo de técnica como el activador de acceso a los posibles consejos, y se lanzará la actividad *TipsUnlocked* (línea 52), la misma que para el modo consejos, con la variante de desbloqueado en el momento o con anterioridad. Estos elementos tendrán como denominador común que, en un paso donde una técnica se le aplique a varios ingredientes, siempre la referencia será la id de la última fila de la tabla *RecipeSteps*, como se comentó anteriormente.

- StepAct.java

```
1  public void stepChecker() {
2       if(counter < totalSteps) {
3             final String lastStepID = steps.get(counter).step();
4             queryResult = SQLHelper.select("stepCheck", "StepsChecked",
5                   "idRecipeSteps = " + lastStepID + " AND idUsers = " + userID);
6
7             if(counter < totalSteps)
8                   if(queryResult.isEmpty()) {
9                         SQLHelper.insert("StepsChecked", "idUsers, idRecipeSteps,
10                                idRecipes", userID + ", "
11                                + lastStepID + ", " + recipeID);
12                        checkStep.setChecked(false);
13                  }
14                  else if(queryResult.get(0)[0].equals("1"))
15                        checkStep.setChecked(true);
16                  else
17                        checkStep.setChecked(false);
18
19            checkStep.setOnClickListener(new View.OnClickListener() {
20                  @Override
21                  public void onClick(View v) {
22                        if(((ToggleButton) checkStep).isChecked())
23                              SQLHelper.update("StepsChecked", "stepCheck = 1",
24                                    "idUsers = " + userID + " AND idRecipeSteps = "
25                                    + lastStepID + " AND idRecipes = " + recipeID);
26                        else
27                              SQLHelper.update("StepsChecked", "stepCheck = 0",
28                                    "idUsers = " + userID + " AND idRecipeSteps = "
29                                    + lastStepID + " AND idRecipes = " + recipeID);
30                        recipeChecker();
31                  }
32            });
33      }
34 }
```

```
35
36  public void recipeChecker() {
37       queryResult = SQLHelper.select("stepCheck", "StepsChecked", "idRecipes = "
38                            + recipeID + " AND idUsers = " + userID);
39
40       ArrayList<String> recipeChecker = new ArrayList<String>();
41       for(int i = 0; i < queryResult.size(); i++)
42             recipeChecker.add(queryResult.get(i)[0]);
43
44       if(recipeChecker.size() == totalSteps && !recipeChecker.contains("0"))
45             SQLHelper.update("RecipesChecked", "recipeCheck = 1", idUsers = "
46                                  + userID + " AND idRecipes = " + recipeID);
47       else
48             SQLHelper.update("RecipesChecked", "recipeCheck = 0", "idUsers = "
49                                  + userID + " AND idRecipes = " + recipeID);
50  }
51
52  public void getTip(int pictoID) {
53       final String lastStepID = steps.get(counter).step();
54       queryResult = SQLHelper.select("StepTips.idStepTips", "StepTips",
55                                  "StepTips.idRecipeSteps = " + lastStepID);
56
57       if(!queryResult.isEmpty()) {
58             Intent intent = new Intent(StepAct.this, TipUnlocker.class);
59             intent.putExtra("tipID", Integer.parseInt(queryResult.get(0)[0]));
60             intent.putExtra("userID", userID);
61             intent.putExtra("technicType", technicTypeID);
62             intent.putExtra("pictoID", pictoID);
63             startActivity(intent);
64       }
65  }
```

Como último elemento adherido a los pasos, cabe la posibilidad de que alguno de ellos tenga un tiempo de realización ligado. Esto activaría la opción, haciendo visible el botón, de acceder el modo cronómetro con el tiempo necesitado para aplicar la técnica. Aparecería el icono pulsable en la esquina inferior izquierda como sugerencia al usuario. En el resto de pasos, este permanecería invisible al usuario, y por tanto, no pulsable. El procedimiento *timeChecker()* será el responsable de dar funcionalidad a esta opción. La variable *rect* establece el área donde el botón tiene la característica de ser clicado, tema que se explicará más a fondo en el apartado de elementos comunes.

El desencadenante de todas estas acciones viene ligado a la transición de un paso a otro, ya sea uno siguiente o anterior, tanto desplazando el dedo de forma horizontal sobre la pantalla como pulsando los botones de navegación.

El ir hacia atrás en el primer paso nos llevará de nuevo a la actividad de la lista de ingredientes, aplicando el procedimiento *firstStep()*. Llegar a la última descripción y avanzar un paso más establecerá la configuración de vista de último paso mediante *lastStep()*, que permitirá hacer una captura mediante la cámara del dispositivo del resultado final y compartirla en redes sociales. Los pasos intermedios estarán reglados por *nextStep()* para avanzar y *prevStep()* para retroceder.

Para finalizar la explicación de esta actividad, comentemos la captura de una imagen como resultado y su compartición mediante las redes sociales disponibles en cada dispositivo. En el último paso, el pictograma de tipo de técnica será especial, puesto que tendrá la propiedad de lanzar la actividad que coordina el contenido multimedia (líneas 3 y 4).

- StepAct.java: getStepContext()

```
1  if(prevX == postX)
2       if(counter >= totalSteps) {
3             Intent cameraIntent =
4                   new Intent(Android.provider.MediaStore.ACTION_IMAGE_CAPTURE);
5             startActivityForResult(cameraIntent, 0);
6       }
7       else if(counter >= 0)
8             getTip(pictoID);
```

La cámara se activará y el resultado de la captura será gestionado mediante el método *onActivityResult()*. Este tendrá la labor de revisar si la cámara ha realzado la captura o si se ha vuelto atrás, y, en caso afirmativo, creará una carpeta en la galería multimedia de imágenes del dispositivo. La captura será tratada con el apoyo de la clase *Bitmap* de *Android*, que hace de contenedor para aplicarle la compresión *jpeg* y para añadirla al flujo de datos que guardará el archivo.

El procedimiento *metaDataPicture()* dota de una descripción al archivo almacenado con ayuda de la clase *ContentValues* y *ContentResolver*, y sharePicture() lanzará el cuadro de diálogo para compartir el archivo con otras aplicaciones de nuestro dispositivo, aplicando una bandera *ACTION_SEND* al *Intent* y creando un seleccionador con el método *Intent.createChooser()* para incluirlas.

6.4.10 Elementos comunes en las actividades

Comentemos varias piezas del entramado de actividades que serán casi idénticas, desechando pequeños matices. La idea de este apartado es exponer dichos elementos, puesto que no pertenecen implícitamente a una única actividad, sino que tienen sentido en varias a la vez.

Por establecimiento del diseñador gráfico, los títulos de las recetas tendrán una configuración especial, que se basa en la primera palabra en negrita y aumentada de tamaño, y el resto en tipografía *light*. Para ello necesitamos separar la cadena del nombre de receta en dos cadenas y formatearlas por separado para que el *TextView* que se encarga de la visualización se muestre adecuadamente. Con la clase *Spannable* y su método *setSpan()* podremos conseguirlo (líneas 7 a 16), que es la que nos permite editar textos con diferentes características para que sean mostrados. Esto se aplicará en todos los textos de nombres de recetas, con el matiz de que los que aparezcan en una cabecera tendrán un salto de línea entre la primera palabra y las siguientes, y los que formen parte de un ítem de listado no, sólo un espaciado simple (línea 4) y un tamaño algo menor en la parte en negrita (1,3 de coeficiente en listados y 1,5 en cabeceras) (línea 12). Cada actividad que necesite invocar este método usará el debido contexto, *ToCookAdapter.getSetTitle()* para los listados y *RecipeAct.getSetTitle()* para las cabeceras.

- ToCookAdapter.java

```
1  public static Spannable getSetTitle(String recipeName) {
2       String[] separated = recipeName.split(" ", 2);
3       String boldString = separated[0].trim();
4       boldString = boldString + ' ';
5       String lightString = separated[1].trim();
6
7       Spannable recipeTitle = new SpannableString(boldString + lightString);
8       recipeTitle.setSpan(new StyleSpan(Android.graphics.Typeface.BOLD), 0,
```

```
                    boldString.length(), Spannable.SPAN_EXCLUSIVE_EXCLUSIVE);
        recipeTitle.setSpan(new TypefaceSpan("sans-serif"), 0, boldString.length(),
                    Spannable.SPAN_EXCLUSIVE_EXCLUSIVE);
        recipeTitle.setSpan(new RelativeSizeSpan(1.3f), 0, boldString.length(),
                    Spannable.SPAN_EXCLUSIVE_EXCLUSIVE);
        recipeTitle.setSpan(new TypefaceSpan("sans-serif-light"), boldString.length(),
                    boldString.length() + lightString.length(),
                    Spannable.SPAN_EXCLUSIVE_EXCLUSIVE);
        return recipeTitle;
    }
```

Otro componente que actúa de la misma forma en casi todas las actividades es la tecla atrás del dispositivo. El evento de pulsación se recoge mediante el método *onKeyDown()*, y da las directrices de la reacción que se debe dar a nivel de sistema, dependiendo de la actividad en la que nos encontremos. El funcionamiento generalizado es el de volver a la pantalla inmediatamente anterior desde el punto de vista del usuario, pero esto, a nivel interno, requiere del cierre de la actividad en la que nos encontremos y de la creación de un *Intent* con los datos necesarios que circulan en el tránsito entre pantallas (líneas 4 a 10). Deberemos también tener en cuenta el estado de la conexión de la base de datos, que, en gran parte de los casos, deberá cerrarse para volver a abrirse en el lanzamiento de actividad de vuelta (línea 7). Veamos un ejemplo de los más completos, las acciones tomadas al pulsar atrás en la actividad *ToCookList*:

- ToCookList.java

```
1  @Override
2  public boolean onKeyDown(int keyCode, KeyEvent event) {
3      if ((keyCode == KeyEvent.KEYCODE_BACK)) {
4          Intent intent = new Intent(ToCookList.this, ToCookAct.class);
5          intent.putExtra("userID", userID);
6          intent.putExtra("appMode", appMode);
7          db.close();
8          finish();
9          startActivity(intent);
10         System.exit(0);
11     }
12     return false;
13 }
```

No está de más comentar el uso que se hace de *onActivityResult()* en varias actividades. Usamos este procedimiento de la clase *Activity* para gestionar las acciones que se deben realizar cuando se inicia otra actividad de la que esperamos un resultado, a veces variable. Esto nos da la capacidad de poder tomar las determinaciones necesarias dependiendo de un código de respuesta, dependiendo si la salida de la actividad que ha sido llamada con *startActivityForResult()* devuelve un resultado correcto o no. En apartados anteriores tenemos varios ejemplos, como en el del modo aventura, que recargará las fases cuando se vuelve de una receta para modificar los chequeos (volviendo de *StepAct*) o creará una nueva aventura si el usuario lo ha decidido así (retornando desde *AdventureGoal*), y en la batidora mediante la actividad *MixerIngredientSelect*, que gestiona la selección de los diferentes listados y, a su vez, de los ingredientes, analizando el resultado dependiente del *REQUEST_CODE* que se reciba.

El botón de *home* en las vistas es otro elemento altamente recurrente. El esquema del funcionamiento, como viene siendo común en este apartado, es idéntico en todas las vistas. El matiz que cambia siempre, dependiente también de datos en el *Intent* que se tenga que enviar, es el de cuál será la actividad a la que volvamos, que depende del punto del que hayamos partido. Lo estándar es que volvamos a la pantalla

principal anterior. Esto quiere decir que, por ejemplo, si nos hallamos visualizando una receta y pulsamos el botón, se cerrará la ventana y la aplicación se ubicará en la pantalla principal anterior, o sea, la del modo del que se provenga. Como ejemplos, si se pulsa en cualquier pantalla de receta, el sistema nos devolverá a la pantalla de modo desde donde se haya accedido a ella, en la pantalla de modo, si se vuelve a pulsar, se cerrará y aparecerá el menú principal, y así con todas las actividades. Como es un *ImageButton*, necesitaremos saber su área aproximada para aplicarle un filtro de gris que nos ayudará a saber si se ha pulsado o no. Esto se consigue con la clase *Rect*, que almacena las coordenadas enteras de un rectángulo. Recogeremos las acciones de "poner dedo, quitar dedo" mediante el método *onTouch()*. Este ejemplo podría equivaler a la visualización de los ingredientes de una receta que no resultan de nuestro agrado, entonces, decidimos volver:

- RecipeItemAct.java: onCreate()

```
toHome = (ImageButton) findViewById(R.id.toHome);
toHome.setOnTouchListener(new OnTouchListener() {
      @Override
      public boolean onTouch(View v, MotionEvent event) {
            if (event.getAction() == MotionEvent.ACTION_DOWN) {
                  toHome.setColorFilter(Color.argb(150, 155, 155, 155));
                  rect = new Rect(v.getLeft(), v.getTop(),
                                    v.getRight(), v.getBottom());
                  return true;
            } else if(event.getAction() == MotionEvent.ACTION_UP) {
                  toHome.setColorFilter(null);
                  if(rect.contains(v.getLeft() + (int) event.getX(),
                                    v.getTop() + (int) event.getY())) {
                        toHome.setColorFilter(null);
                        finish();
                        if(appMode == 1)
                              finishActivity(AdventureAct.REQUEST_CODE);
                        if(appMode == 2)
                              finishActivity(ToCookList.REQUEST_CODE);
                  }
            }
            return false;
      }
});
```

Capítulo 7

Pruebas

En este punto vamos a explicar las pruebas que se han llevado a cabo durante la implementación de la aplicación y tras tener el prototipo afianzado. Es deber del programador concretar los fallos de un programa y subsanarlos, tanto errores en tiempo de ejecución como el no cumplimiento de los requisitos iniciales, puesto que la aplicación no estará terminada hasta que cumpla al menos las expectativas que se plantearon en primera instancia y las derivaciones del desarrollo de estas al aplicar la ingeniería de proyectos. Gracias a dichas pruebas, nos aseguramos que la aplicación funciona de forma correcta, lógica, consistente y además han servido para depurar el código haciéndolo más eficiente.

El proceso de creación de módulos por separado que vayan dando vida a la aplicación mediante su interconexión, depende de cómo se vea, tiene una ventaja o inconveniente. La ventaja es que nos permite centrarnos en un solo problema en cada momento, diseñando las piezas que después compondrán el puzle, cosa que a su vez es negativa, porque no podremos cerciorarnos totalmente de la corrección de cada componente hasta que no esté anexado con los que se vaya a comunicar. Esto nos obliga a que, como si de un grafo se tratase, cada creación de un nuevo nodo debe ser revisada en primera instancia, y cuando se trace una arista que una a ese nodo con el resto del grafo se debe revisar desde dicho nodo a todos los demás y volver para saber si la respuesta es la deseada en todos los recorridos.

El proceso descrito anteriormente puede dividirse en pruebas unitarias y pruebas de integración, que, como se ha explicado con el ejemplo del grafo, se han ido alternando durante el crecimiento de la aplicación para todos los componentes. Además, como *Android* tiene dependencias de las versiones y dispositivos, se necesitará saber si las visualizaciones son correctas en diferentes escenarios.

El proceso de testeo se ha llevado a cabo usando la *AVD* que viene junto con el *SDK* en primera instancia. A medida que la aplicación iba creciendo, se han usado también dispositivos físicos, smartphones (*Sony Xperia M*, *BQ Aquaris 5*, *BQ Aquaris M5*) y tablets(*Samsung Galaxy Tab 2*, *Woxter QX 90*), porque el dispositivo virtual no es del todo fiable, y al final donde tiene que funcionar a la perfección es en dispositivos reales.

7.1 Pruebas unitarias

Estas tienen como objetivo asegurar que los módulos que componen la aplicación funcionan adecuadamente de forma individual y sin emitir ningún tipo de error. Esto se consigue probando los métodos y procedimientos que forman cada actividad, revisando que los resultados de sus ejecuciones y verificando que la comunicación entre ellos es la requerida, a medida que se iban implementando.

La idea es, además de revisar su comportamiento común, aplicarles valores extremos o inusuales para ver si la reacción es la esperada. La aplicación tiene que estar reforzada para que cuando los usuarios la manejen se aperciba cualquier posibilidad de uso. Con esto, además de la interacción con el usuario, se debe verificar que las variables que circulan en un módulo tengan diferentes comportamientos dependiendo del estado del sistema en ese momento, como por ejemplo, iniciar una aventura por primera vez o que ya haya sido iniciada con anterioridad.

Otro punto a tener en cuenta es cómo afecta el giro de pantalla al ciclo de vida de una actividad. El sistema la destruye y la vuelve a crear cuando detecta el cambio de vertical a horizontal y viceversa, por lo que se pueden generar errores provocado por el cambio de estado de alguno de los elementos. Por lo tanto, otra de las verificaciones que se han hecho es girar el dispositivo en cada una de las pantallas para comprobar que la destrucción y reconstrucción de cada actividad no genera problemas.

Por último, las consultas a la base de datos existentes en cada módulo deben ceñirse a las especificaciones de la misma para que al ejecutarlas no se lance ningún mensaje de error.

7.2 Pruebas de integración

A medida que se iban completando los módulos, se han ido ensamblando, estableciendo de forma conveniente las interacciones que se existen entre ellos. Cada vez que se ensambla uno nuevo es recomendable verificar todo el conjunto, puesto que existen elementos de los que casi todos los componentes hacen uso, como por ejemplo la base de datos.

Hay que cerciorar que los elementos que requiere cada actividad son correctamente empaquetados y enviados mediante *Intents* que son creados por la actividad iniciante, y que se rescatan de forma correcta con el uso de *Bundles*.

Hay actividades que se encargan de devolver un resultado a otra, que es la que la ha invocado. Se debe revisar la interacción en esos casos y comprobar que la comunicación al abrir y finalizar luego la actividad es la que deseamos.

En otros casos, existen actividades, dos o más, que hacen uso de una concreta (modos que ejecutan la descripción de una receta). Es altamente necesario revisar exhaustivamente esas situaciones, dado que el lanzamiento tiene que estar bien acotado, conociendo las excepciones de cada uno de ellos, y sabiendo resolverlas dependiendo de los requisitos.

Para terminar, la comunicación de la base de datos con los componentes debe ser basculante, abriendo y cerrando la conexión según se necesite, pero sin crear duplicidades, iniciando o clausurando dos o más veces de forma consecutiva, porque genera errores y cierres inesperados.

7.3 Pruebas de visualización

No sólo es importante que el programa no emita errores y que cumpla los requisitos establecidos, también lo es que las vistas que se muestren en pantalla sean las deseadas, que se vean correctamente en cualquier dispositivo que cumpla los requisitos del sistema y que sea una experiencia agradable para el usuario.

El testeo de la aplicación se ha hecho en la *ADV* y en dispositivos físicos diferentes, con lo que se ha podido comprobar que la visualización es correcta en todos ellos. El único punto débil puede encontrarse en la representación en tabletas, donde, aún habiendo aplicado los parámetros correctos en los *layouts* para que sean relativos a las pantallas, el tamaño de los elementos es algo pequeño. Esto viene dado porque es necesario adaptar el tamaño de todos los elementos de imagen a las diferentes densidades de pantalla y guardarlos en sus respectivas carpetas de recursos para que, dependiendo de las densidades, se escojan unas imágenes u otras.

Capítulo 8

Conclusiones y reflexiones

Este apartado se va a dedicar a comentar el resultado del proyecto de manera objetiva, cómo ha sido el trabajar conjuntamente con un diseñador, lo que ha supuesto para mi desarrollo personal y las posibles mejoras que se pueden aplicar en un futuro.

8.1 Alcance de los objetivos primarios planteados

Al ser una idea que se ha elegido y desarrollado desde cero, sin ninguna referencia, el primer paso más importante fue definirla bien. Una vez vislumbrado el hasta dónde se quería llegar habiendo esquematizado la idea, era cuestión de perseverancia y esfuerzo el que el producto final fuera como mínimo el deseado en ese punto.

El resultado de la aplicación supera con creces el del primer boceto concebido, tanto en el apartado gráfico como en el de funcionalidades. En ningún instante me he conformado con hacer sólo lo mínimo, sino que he intentado en todo momento que todas las partes de la aplicación, además de hacer lo que se había planteado, quedaran perfectas dotándolas de la mayor funcionalidad posible de cara al usuario y de toda la flexibilidad plausible en lo que al sistema respecta.

Esto ha lastrado el tiempo de desarrollo, que ha sido mayor de lo esperado, pero siempre en pos de que el sistema móvil interactivo para el desarrollo de las habilidades culinarias, después rebautizado como *aCubierto* de cara al público, tuviera la mayor calidad posible en todos los sentidos.

Resumiendo, se han cubierto satisfactoriamente las proposiciones primigenias y también las derivadas del análisis de requisitos, dando como resultado una aplicación móvil de la que puedo sentirme muy satisfecho.

8.2 Trabajo en equipo multidisciplinar

Desde el primer momento en el que se expuso la idea a la persona que ha desarrollado el diseño visual del proyecto, la evolución fue muy favorable. Siendo mi hermano, el nivel de entendimiento entre ambos es muy alto, lo que propició que la retroalimentación entre los requisitos planteados por mí y su desarrollo casi a tiempo real del concepto gráfico enriquecieran la idea del proyecto.

El tener una buena comunicación, por entendimiento y cercanía, ha producido que, desde el inicio de la creación del boceto, pocas ideas fueran declinadas. El diseñador ha captado en todo momento lo que yo le he querido transmitir a la perfección, dando a luz con gran agudeza lo que se pretendía.

En definitiva, esta filosofía de trabajo con comunicación ininterrumpida ha beneficiado mucho el desarrollo del proyecto, además de demostrarnos a nosotros mismos que trabajamos bien en equipo.

8.3 Conclusiones personales

Gracias a la consecución de este proyecto, quitando la parte irrefutable de la gran cantidad de habilidades y conceptos adquiridos, me he demostrado a mí mismo de lo que soy capaz. Han sido muchas las dificultades que he tenido que ir superando hasta obtener una aplicación que me dejara satisfecho en todos los sentido.

Es la primera vez que me enfrentaba a algo así, y creo que lo más positivo ha sido que me lo he tomado como un reto. Siempre estaba ahí la pregunta de hasta dónde iba a ser capaz de llegar. La planificación inicial me dio a entender que no iba a ser fácil, que incluso podría llegar a fracasar al no ser capaz de reproducir alguno de los objetivos que quería alcanzar. Mi deseo en todo momento fue que el resultado final hiciera sentirme satisfecho y que además pudiera resultarle útil a la gente interesada en la cocina, cosa que siento que he cumplido muy satisfactoriamente.

La primera toma de contacto para adquirir los conocimientos necesarios para llevar a cabo el proyecto no fue tan dura. Pero claro, como toma de contacto para aprender a programar en *Android* lo que se suele hacer es poco más que crear alguna pantalla con botones y textos. Cuando la idea fue tomando forma advertí que iba a necesitar llegar muchísimo más allá y que no solo iba a ser *Android*, sino que también *SQLite*.

Otra cosa que intenté es aprovechar los conocimientos que había adquirido durante la carrera para modelar y desarrollar la aplicación. He necesitado volver a estudiar y recordar asignaturas como Ingeniería del Software, Base de Datos, Programación Orientada a Objetos e incluso Procesamiento de Imágenes. Esto hace que uno se vuelva consciente de que cualquier conocimiento que podamos aprender siempre va a ser poco y de la importancia que tiene el intentar tenerlos presentes o refrescarlos de vez en cuando.

Considero que he adquirido conocimientos muy valiosos para el futuro gracias a este proyecto. *Java* y *SQLite* son los que creo más importantes por ser el primero un lenguaje de programación actual y extendido, y el segundo por conocer con más profundidad cómo funcionan los lenguajes de consulta, indispensables para el mundo laboral.

En resumidas cuentas, siento que el tiempo y el esfuerzo dedicados para crear *aCubierto*, aunque largo y arduo, ha merecido la pena con creces de forma general, tanto en a lo que conocimientos adquiridos se refiere como a superación y satisfacción personal.

8.4 Posibilidad de ampliaciones y mejoras

La idea principal desde el primer momento en el que se comenzó el planteamiento fue dotar de gran flexibilidad a la aplicación para que el abanico de extensiones que se le pudieran aplicar fuera el máximo posible. Esto hace que la aplicación tenga muchas posibilidades de expansión.

A continuación, vamos a listar diferentes características con las que se podría plantear mejorar el programa:

- Conversión del programa auxiliar que gestiona la introducción de las recetas en la base de datos en una aplicación paralela para que los usuarios pudieran crear las suyas propias.
- Alojar los datos en un servidor para que la aplicación pudiera tomar de allí los datos de la receta y de cada usuario, modificando el login para introducir contraseña. Haciendo esto se perdería la propiedad de no necesidad de estar conectado a una red wifi o de datos.
- Implementar el apartado de actualización de la base de datos para que las ampliaciones de recetas fueran actualizaciones del programa desde *Google Play Store*.
- Implementar un modo basado en comida saludable que recomiende sólo recetas bajas en calorías.
- Implementar un modo que dé la posibilidad al usuario de poder crear un calendario de recetas semanal, permitiendo asignarlas a los horarios de comida y que sólo se tenga que comenzar su reproducción cuando el usuario quiera prepararlas.
- Implementar la opción que permita ir tomando capturas de la realización de una receta y que después monte un pequeño vídeo con todas ellas.
- Implementar un modo que, a partir de una receta, comente el tiempo que se debería realizar algún tipo de ejercicio para quemar las calorías que aporte.

Capítulo 9

Herramientas utilizadas

Comentemos los programas y utilidades de las que nos hemos servido para poder crear la aplicación. En su mayoría, son herramientas basadas en el software libre, cosa que se ha intentado siempre para seguir con la filosofía del proyecto.

9.1 Lenguaje de programación

Para trabajar con *Android* lo más recomendable es usar *Java* para la parte programática y *xml* para la gráfica, aunque hay otras posibilidades que fueron desechadas desde el principio. El soporte de paquetes y bibliotecas viene dado por la *SDK*. Las ventajas que tiene son la orientación a objetos, el soporte de referencias y tutoriales, tanto el autóctono de *Google* como el que ha sido creado por desarrolladores mediante blogs, manuales y páginas de exposición de dudas como *stack overflow* [7].

Figura 9.1: Herramientas utilizadas: Logotipo de Android

9.2 Base de datos

El sistema de base de datos integrado por *Android* es *SQLite*.

Figura 9.2: Herramientas utilizadas: Logotipo de SQLite

Para programar y gestionar las bases de datos se ha usado *SQLite Manager*, que se integra en el buscador *Mozilla Firefox* como un complemento. Tiene una interfaz intuitiva y básica, pero eso no quita que revise de una usabilidad más que necesaria para lo que se pretendía.

Figura 9.3: Herramientas utilizadas: Icono de la aplicación de SQLite Manager

9.3 Entorno de desarrollo

Se ha usado *Eclipse Luna*, que nos ha surtido de los *plugins* necesarios que se han necesitado, como el enlace con la *SDK* o *Subversive* para el control de versiones.

Figura 9.4: Herramientas utilizadas: Logotipo de Eclipse

9.4 Creación de diagramas UML

La herramienta utilizada en este apartado ha sido *Dia*, que nos ha provisto de los medios necesarios para desarrollar diagramas de casos de uso, modelos de base de datos, diagramas de clase, etc.

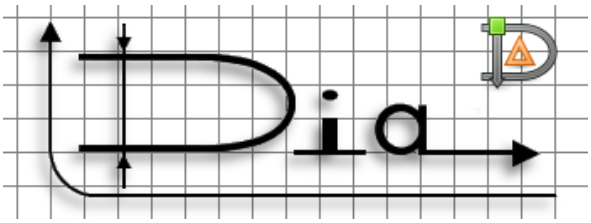


Figura 9.5: Herramientas utilizadas: Logotipo de Dia

9.5 Diagramas de tiempo de trabajo

En este caso, el programa seleccionado ha sido *GanttProject*.

Figura 9.6: Herramientas utilizadas: Logotipo de GanttProject

9.6 Control de versiones y repositorio

Uno de los elementos más importantes usados para dar soporte al desarrollo de la aplicación ha sido el controlador de versiones *Subversive* integrado mediante *plugin* en *Eclipse*, ligado a un repositorio en la web https://sourceforge.net. Se empezó usando la forja de Google, pero el servicio acabó cerrando.

Figura 9.7: Herramientas utilizadas: Logotipo de Subversive

Figura 9.8: Herramientas utilizadas: Logotipo de Sourceforge

9.7 Realización de dibujos explicativos

Una de las únicas herramientas que se ha usado y no es software libre es *Microsoft Visio 2010*, por el simple hecho de que conocía cómo funcionaba con anterioridad.

Figura 9.9: Herramientas utilizadas: Logotipo de Microsoft Visio 2010

9.8 Redacción de la memoria

En primera instancia pensó hacerse mediante *LATEX*, pero por problemas con la descarga de la plantilla de la universidad y falta de tiempo se desestimó la idea a favor de *Microsoft Word 2010*. Podría haber escogido alguna herramienta de código abierto, pero posiblemente hubiera empleado más tiempo en familiarizarme con ella.

Figura 9.10: Herramientas utilizadas: Logotipo de Microsoft Word 2010

Capítulo 10

Manuales de instalación y uso

En pos de esclarecer y facilitar la tarea a los usuarios, vamos a explicar cómo pueden conseguir *aCubierto*, cómo deben instalarla en sus dispositivos y qué tienen que saber para utilizarla a la perfección.

10.1 Cómo conseguir e instalar *aCubierto*

Lo primero que debemos hacer es descargar la aplicación en nuestro dispositivo. Para ello existirá una versión totalmente actualizada que podremos obtener haciendo clic en dicho enlace o copiándolo a un explorador de internet. La descarga comenzará automáticamente en unos segundos:

https://sourceforge.net/projects/*aCubierto*/files/PFC_v1_final.apk/download

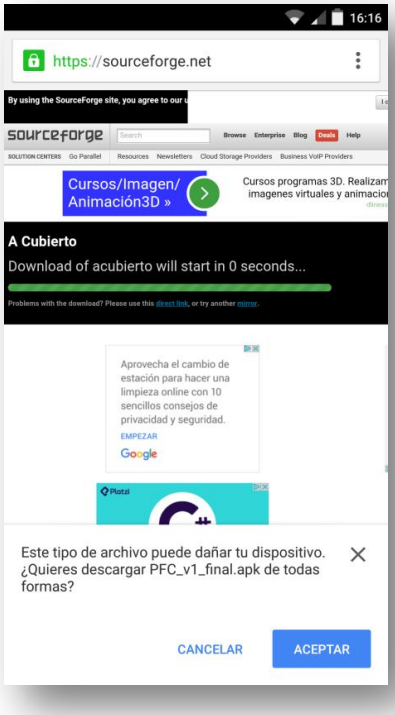


Figura 10.1: Instalación de aCubierto: Pulsamos aceptar para iniciar la descarga

En un futuro estará disponible en *Google Play Store* para facilitar a los usuarios su obtención e instalación.

Cuando la descarga del archivo *.apk* haya finalizado, se podrá comenzar su instalación mediante la selección en la barra de notificaciones. Es posible que se tenga que habilitar la función para poder instalar aplicaciones con orígenes desconocidos.

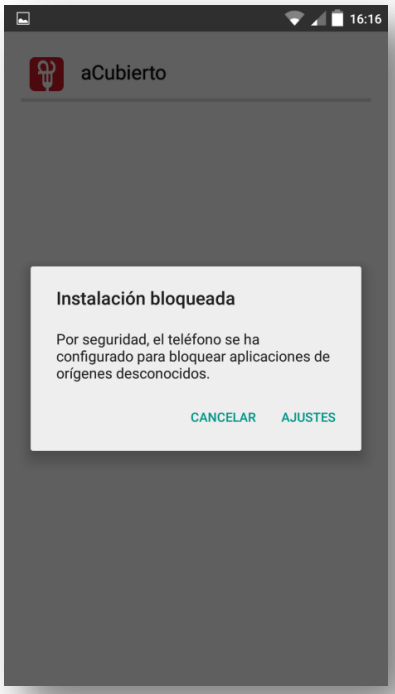


Figura 10.2: Instalación de aCubierto: Aviso de aplicación de origen desconocido

Mediante ajustes, el sistema accederá directamente al apartado de seguridad, donde tan sólo se tendrá que activar la opción. Pulsamos aceptar en el cuadro de diálogo. Se recomienda que una vez instalada la *app* se vuelva a desactivar dicha opción para la seguridad del dispositivo.

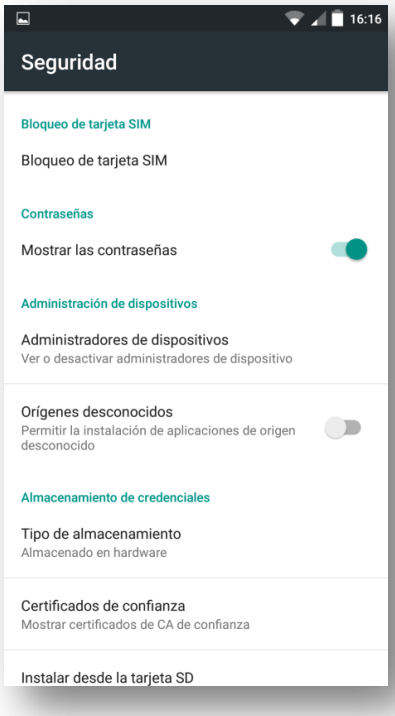


Figura 10.3: Instalación de aCubierto: Activación del permiso de orígenes desconocidos

Acto seguido tendremos que buscar la aplicación en la carpeta de descargas del dispositivo o en el acceso directo del menú. Accederemos y volveremos a pulsar sobre el archivo para seleccionar instalar, y esta vez no habrá problema alguno.

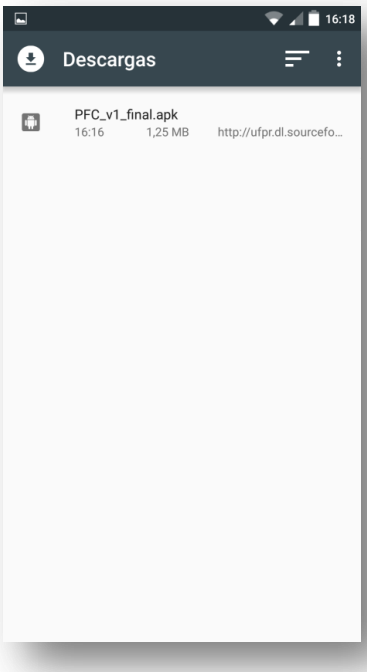


Figura 10.4: Instalación de aCubierto: Archivo visto desde el acceso directo a descargas del menú

Seleccionamos la opción “instalar” y dejamos que el sistema haga el resto. En pocos segundos habrá finalizado el proceso.

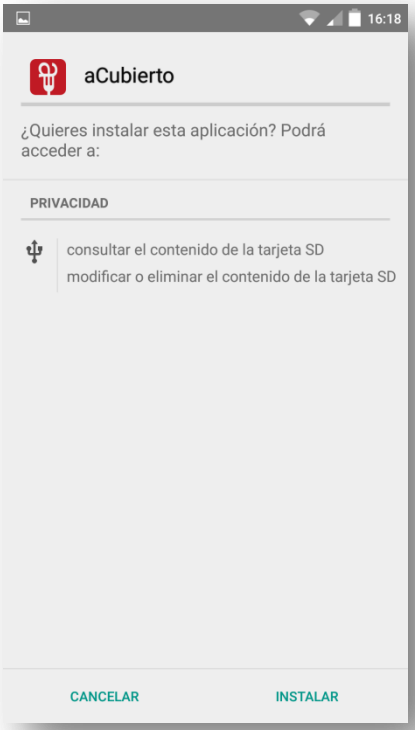


Figura 10.5: Instalación de aCubierto: Pantalla de presentación de la instalación

Una vez terminada, podremos acceder desde la misma pantalla por primera vez a *aCubierto* y comenzar a disfrutar de ella.

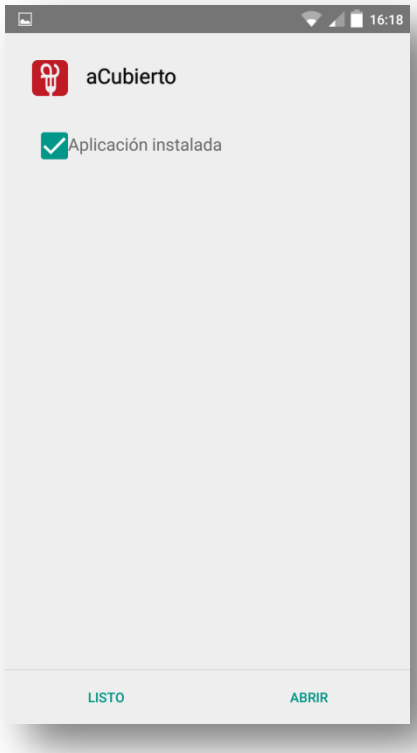


Figura 10.6: Instalación de aCubierto: Pantalla de instalación finalizada

10.2 Cómo desinstalar *aCubierto*

La desinstalación se lleva a cabo como cualquier otra aplicación en *Android*, accediendo en el menú principal a las opciones de ajustes, y luego a aplicaciones. De base vienen ordenadas por orden alfabético, con lo que es probable que aparezca entre las primeras. Tan sólo tendremos que seleccionarla y pulsar "desinstalar" en la ventana de descripción de la aplicación.

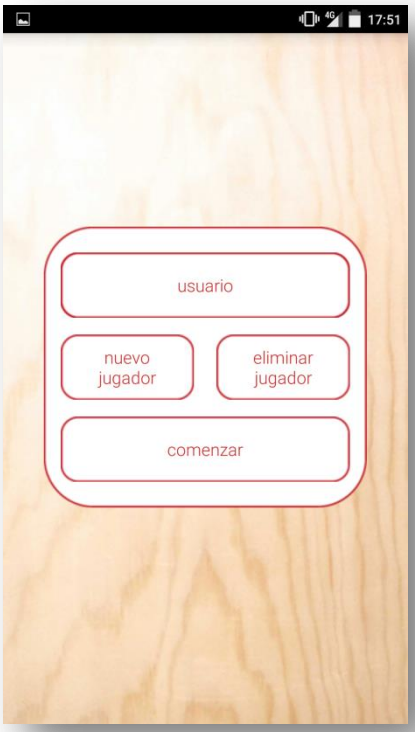
10.3 Manual de Usuario

Empecemos con la diversión y el aprendizaje. Paso a paso se va a ir explicado todo lo que el usuario puede hacer con esta estupenda aplicación. Esperamos que la experiencia sea de su agrado.

10.3.1 Cosas a tener en cuenta previamente

Para poder utilizar aCuierto necesitaremos un dispositivo con sistema operativo *Android* que tenga como mínimo la versión *Jelly Bean (Android 4.1.2)*. Se puede confirmar que se cumple este requisito accediendo desde la pantalla de "Ajustes" a la información del teléfono, donde aparecerá cuál es la versión actual.

10.3.2 Nada más empezar



Al ejecutar *aCubierto* se nos dará la bienvenida mediante la pantalla de inicio, que invita a pulsar sobre ella para acceder al *Login*.

Crea tu usuario aportando un nombre, que será el que te acompañará durante toda la experiencia. Si ya lo has creado con anterioridad búscalo en la lista para proseguir con tu progreso.

Si alguna vez deseas eliminar alguno de los usuarios creados, selecciónalo y confírmalo mediante la opción para ello, teniendo en cuenta que una vez aceptada la confirmación ya no habrá vuelta atrás.

10.3.3 Selecciona un modo de juego

Una vez te hayas logueado ya puedes comenzar a aprender y disfrutar de la cocina.

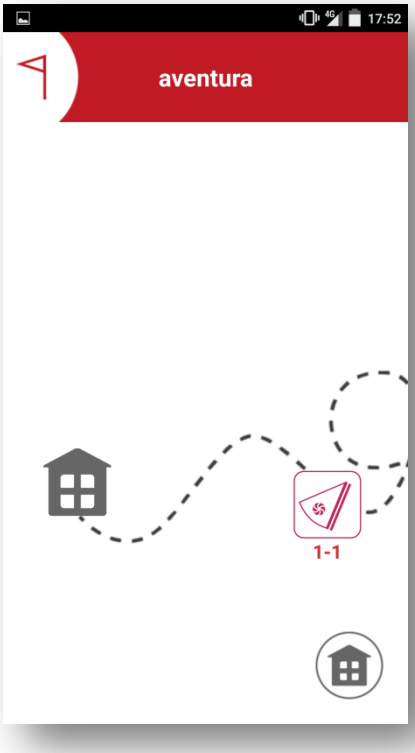
En la pantalla de menú de *aCubierto* puedes elegir entre tres modos para comenzar a realizar recetas y dos utilidades de ayuda.

Mediante la aventura, a cocinar y batidora, podrás acceder a listados de recetas con diferentes características que serán expuestas una a una en las siguientes categorías del manual.

Con crono y consejos tendrás el complemento perfecto para cuando seas lo suficientemente valiente para volar solo. En breve también serán explicadas.

Podrás consultar tus estadísticas y ver algo de información sobre aCubierto pulsando el botón de menú del dispositivo. Si este no existe como tal, puedes probar a dejar pulsado durante un instante el botón que muestra las pantallas recientes.

10.3.4 Aprendiendo desde cero con el modo aventura



En esta sección podrás aprender a hacer recetas que irán incrementando su dificultad dependiendo de la fase que sea, yendo de más fácil a más difícil.

Desliza el dedo sobre la pantalla para recorrer el mapa y conocer las recetas que te ha tocado realizar. Cada mapa tiene cinco recetas, preferiblemente que estén chequeadas y de diferentes tipos entre sí.

Sabrás cuándo has chequeado todos los pasos de una receta porque aparecerá sobre la fase el siguiente icono:

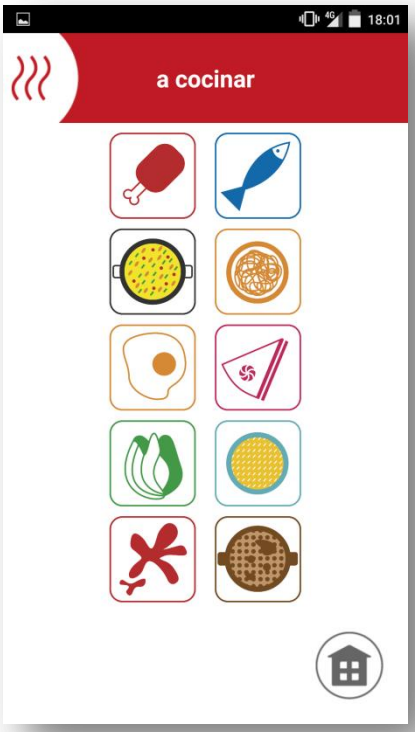
Accede a la descripción de cada receta haciendo clic encima de las fases, que vienen establecidas por el icono de su tipo.

Puedes consultar tus avances pulsando en el botón de salida que se encuentra al principio de la línea discontinua:

Una vez estén todas la recetas de cada fase chequeadas podrás crear una nueva aventura con otras recetas mediante el icono de meta:

Para volver al menú principal se puede pulsar el atrás del dispositivo o el botón home ubicado siempre en la esquina inferior derecha:

10.3.5 Elige qué receta hacer dependiendo de su tipo



¿Qué te apetece? Con el modo a cocinar podrás visualizar recetas de un determinado tipo y ver todas las que Cubierto tiene para ti.

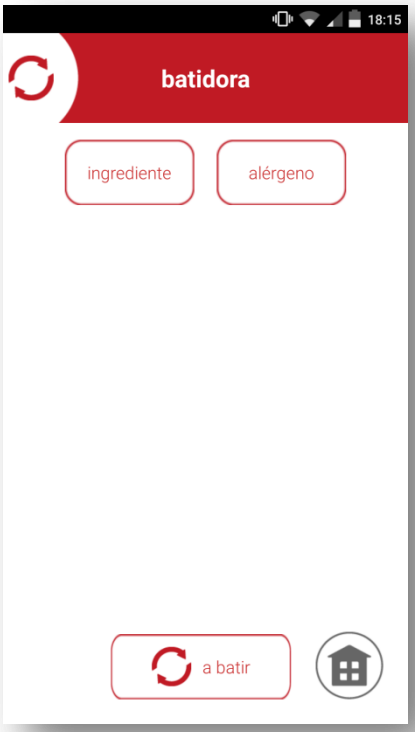
Elige entre carne, pescado, arroces, pasta, huevo, postres, verduras, sopas, salsas y guisos. Verás el listado de recetas de ese tipo, listas para que las realices cuando te apetezca.

Para tu comodidad, podrás elegir entre dos formas de ordenar las recetas una vez hayas elegido el tipo mediante los botones que se hallan en la esquina inferior izquierda.

Selecciona el botón de dificultad para establecer un orden de menor a mayor. Si vuelves a seleccionarlo se invertirá el orden, tú eliges a qué quieres enfrentarte con un solo clic:

Si no temes a la dificultad pero andas corto de tiempo prueba el botón de reloj para ver primero las recetas que requieren de menos. Si deseas deleitarte con una larga sesión culinaria, haz clic de nuevo:

10.3.6 Si no sabes qué hacer con eso que tienes…



Tu modo es la batidora. Selecciona ingredientes con sus cantidades que deseas que tuviera tu receta y alérgenos que quieres evitar.

aCubierto te mostrará lo que tiene disponible en ese momento en su despensa para que tú elijas lo que más te convenga. Puedes ver todos los ingredientes juntos por orden alfabético o en bloques por su tipo.

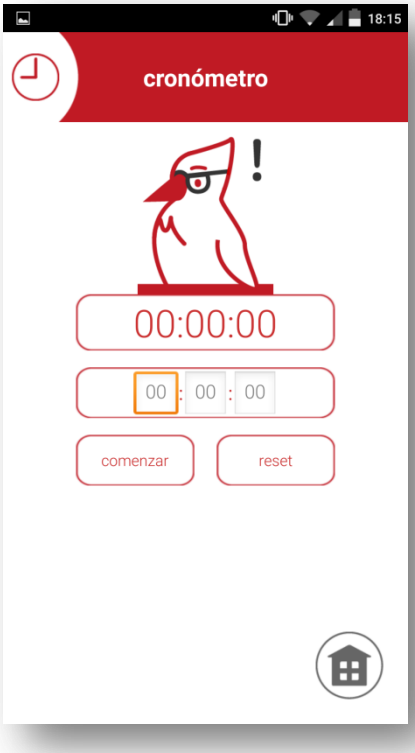
Desecha las recetas que contengan ciertos alérgenos de los catorce reconocidos por el EUFIC.

Si deseas cambiar la cantidad de un ingrediente elegido, será tan fácil como seleccionar el botón de editar del mismo en el listado: 🖍

Cuando lo que pretendes es eliminar un ingrediente o alérgeno, desaparecerá al hacer clic en el botón para borrar: ❌

Obtén los resultados de tu selección mediante “a batir”. Enseguida obtendrás los resultados y sabrás qué recetas se aproxima más a tus especificaciones. Aparecerán ordenadas mediante un porcentaje de aproximación, esperando a que elijas la que más te guste.

10.3.7 Entérate de cuándo ha acabado el paso que estabas haciendo



El cronómetro de cocina se encargará de ayudarte siempre que lo necesites.

Introduce el tiempo que desees que tu guiso esté en el fuego y comienza la cuenta atrás cuando más te convenga. Si lo necesitas, puedes pausarla para continuar con ella en otro momento.

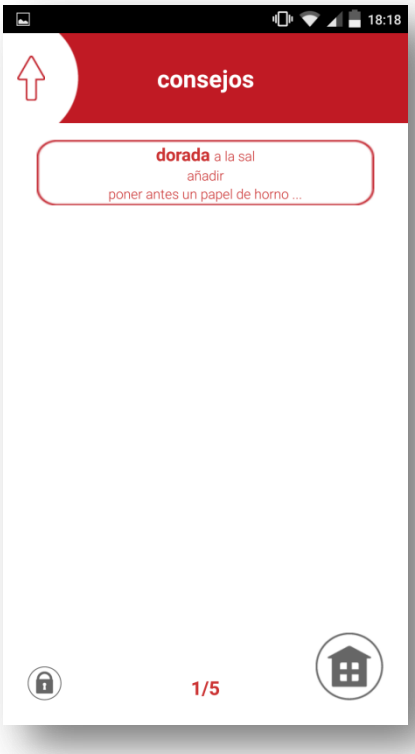
Para volver a poner otro tiempo diferente, aunque esté el cronómetro en marcha, deja pulsado durante unos segundos el botón de reset.

Siempre tendrás la facilidad de poder hacer otra cosa con tu dispositivo abriendo otra aplicación y dejando el reloj en segundo plano. Si el tiempo está corriendo, *aCubierto* se encargará de recordarte cuándo lo quitaste del primer plano de la pantalla y el tiempo que aún quedaba.

Además, se te ofrecerá la opción de activarlo en algunos pasos en los que el rato que se vaya a tardar sea significante.

¡Recuerda! El botón de *home* te devuelve al menú del juego y cancela la cuenta atrás independientemente del estado.

10.3.8 Extrapola y experimenta en la cocina



Usa los consejos que vayas desbloqueando en los pasos de las recetas para improvisar en tus momentos de inspiración.

Para que puedas usar tu imaginación *aCubierto* te recuerda los consejos que conseguiste encontrar escondidos entre los pictogramas de técnicas.

Aplícalos a recetas parecidas para obtener buenos resultados y facilitarte la vida. Selecciónalos para saber todo sobre ellos.

Conocerás en todo momento cuántos consejos llevas desbloqueados y los que aún te quedan por conocer.

Si te da pereza emprender la búsqueda del tesoro y tienes curiosidad por lo que puedes aprender, siempre tendrás la opción de obtener una pista mediante el botón que te sopla lo que aún está oculto:

10.3.9 ¿Cómo funcionan las recetas?

Has llegado hasta tu primera receta. ¡Enhorabuena! Es el verdadero momento para aprender a ser un buen chef. Pero, como buen chef, haz de saber manejarte en la cocina.

Las recetas van acompañadas de su nombre y su tipo, o, en caso de la aventura, de la fase a la que se corresponde.

Podrás saber cuánto tiempo aproximado vas a necesitar para realizarla mediante el indicador circular rojo que se halla bajo el nombre.

El otro detalle importante es la dificultad definida como cantidad de pasos que vas a tener que realizar hasta obtener tu delicatesen. Esto es así porque la dificultad que puede conllevar el aplicar una técnica de cocina no es la misma para una persona que para otra. Esta información se expresa mediante un número y un icono:

El botón comenzar nos conducirá hacia la pantalla donde encontraremos la información básica de la receta.

Aquí podremos conocer cualquier tipo de detalle general.

En la esquina superior derecha encontramos un recuadro blanco, que nos indicará si todos los pasos de la receta han sido chequeados. Esto es sólo informativo, no te empeñes en querer chequear toda la receta de golpe. Recórrela entera y aprende.

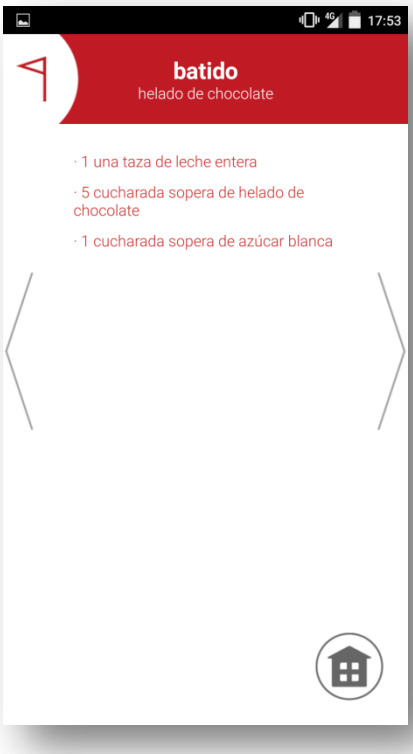
Si estuviera chequeada es que has marcado el *checkbox* en todos sus pasos, con lo que la vista del elemento cambiaría: ✓

Conoce todos los alérgenos que contienen los diferentes ingredientes haciendo clic en el botón de información de la esquina inferior izquierda: ⓘ

Si eres de esas personas que se preocupa por lo que come, puedes conocer una aproximación de la cantidad de kilocalorías que tiene la receta por cada 100 gramos. El elemento para ello es el botón que se encuentra en la esquina inferior derecha, a la izquierda de *home*: Kcal

Para conocer los ingredientes que componen la receta y las cantidades necesarias basta con pulsar comenzar.

Podrás visualizar la lista con lo que necesitarás para poder realizar la receta y todas las cantidades que te harán falta.



A partir de aquí, se transcurre entre ventanas mediante los botones de navegación. Estos son los que se encuentran a los laterales de la pantalla en la parte central.

Como *aCubierto* intenta ser lo más intuitiva posible, ofrece la posibilidad de realizar el mismo desplazamiento que con los botones de navegación a través del deslizamiento del dedo sobre la pantalla, como de un libro se tratara.

En este caso, el botón *home* hará que se regrese al modo del que partió la receta.

Si decides hacer uso del botón atrás del dispositivo a partir de este punto, tendrá el mismo efecto que el botón de navegación izquierdo, conduciéndote a la ventana automáticamente anterior en la línea de tiempo de la representación.

Avanzando hacia la derecha, encontramos el primer paso de la receta. La estructura de los pasos será común, quitando el último, que lo explicaremos al final.

Vuelve a aparecer el *checkbox* de la esquina superior derecha. Este sí es interactivo y cuenta para poder chequear la receta completa, ¡así que no te olvides de dejarlo marcado antes de continuar! Si no lo haces no pasará nada, pero no se verá reflejado en tu progreso.

En la esquina superior izquierda tenemos el número al que equivale el paso dentro de la receta. Sabrás cuánto te queda para acabar recordando la información que aparecía en la descripción de la receta.

En la parte central, con forma redonda, aparece un pictograma de modo. Estos son cruciales si quieres desbloquear consejos. Haz clic siempre encima de ellos evitando deslizar el dedo para saber si tras él se oculta algún truco.

Justo debajo aparece la descripción del paso, que nos dirá qué tenemos que hacer con qué ingredientes.

Si algún paso tiene tiempo ligado, aparecerá un botón pulsable en la esquina inferior izquierda que conducirá al cronómetro:

¡Ya hemos completado nuestra primera receta y qué bien ha quedado!

Para ponerle el broche final a nuestra creación, *aCubierto* te ofrece la posibilidad de inmortalizar el momento gracias al último pictograma, que es muy especial.

Haciendo clic de nuevo sin deslizamiento sobre él, lanzará la cámara de nuestro dispositivo para poder hacer una fotografía. Esta se almacenará en tu galería de imágenes en una carpeta que se creará la primera vez que se tome una y que tendrá de nombre, como no, *aCubierto*.

Una vez realizada, se ofrecerá la oportunidad de mostrarle a tus amigos lo buen cocinero que eres a través de las aplicaciones y redes sociales que tengas instaladas en tu dispositivo.

Para volver al punto de partida, el modo desde el que se accedió a la receta, sólo hará falta pulsar el icono de *home*.

¡Esperemos que *aCubierto* te ayude mucho en la cocina y te convierta en un gran chef!

Capítulo 11

Bibliografía y fuentes

[1] JM. Rodríguez - Izquierdo Flores, *¡Al Buche!*, blog de cocina y recetas:
http://albuche.blogspot.com.es/

[2] A. Rodríguez - Izquierdo Flores, *Manual de Identidad Corporativa de aCubierto*, 2015:
https://sourceforge.net/projects/acubierto/files/identidad%20corporativa.pdf/download

[3] T. Groussard, *JAVA 7 Bases del lenguaje y de la programación orientada a objetos*, Editions ENI, 2013.

[4] S. McCracken, *Android: curso de desarrollo de aplicaciones*, INFOBOOK'S, 2012.

[5] Universidad Politécnica de Valencia, *Android: Introducción a la Programación*, Curso Online:
http://www.androidcurso.com/index.php/mooc-introduccion

[6] Página oficial de *Android* para desarolladores:
http://developer.android.com/

[7] Stack Overflow community:
http://stackoverflow.com/

[8] Varios, *Platos únicos*, PARRAGON, 2008.

[9] H. Arbelaizt, M. de la Rosa, T. Pérez, M. Hofmann, X. Pellicer. C. Craig, *Carnes: la cocina de casa con el toque de los grandes chefs*, Ediciones Primera Plana, 2008.

[10] The European Food Information Council:
http://www.eufic.org/

[11] Android Dashboards:
http://developer.android.com/intl/es/about/dashboards/index.html

[12] SQLite Documentation:
https://www.sqlite.org

[13] I. Sommerville, *Ingeniería del Software*, PEARSON, 2005.

[14] C. Larman, *UML Y PATRONES*, PEARSON, 2003.

[15] Java™ Platform, Standard Edition 7 API Specification:
http://docs.oracle.com/javase/7/docs/api/

[16] SGOLIVER.NET, *Curso de programación Android*:

http://www.sgoliver.net/blog/curso-de-programacion-android/

[17] Vogella, *Android Development*:
http://www.vogella.com/tutorials/android.html

[18] Repositorio de archivos del proyecto:
https://sourceforge.net/projects/*aCubierto*/

[19] Freesound, collaborative database of Creative Commons Licensed sounds:
Electric Whisk Rev: https://freesound.org/people/nebulousflynn/sounds/220938/
cuckoo.ogg: https://freesound.org/people/luis2004lima/sounds/240887/

Capítulo 12

GNU Free Documentation License

Version 1.3, 3 November 2008

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a

specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright (C) 2016 José María Rodríguez – Izquierdo Flores. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with … Texts." line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.