



UNIVERSITAT
POLITÀCNICA
DE VALÈNCIA



Escola Tècnica
Superior d'Enginyeria
Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Diseño e implementación de una aplicación para dispositivos móviles que muestre la guía de vinos de España 2017

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Sergio Vicente Delamo Rizo

Tutor: Lenin G. Lemus Zúñiga

Curso 2017/2018

Resumen

Este proyecto ha consistido en la creación de una aplicación desarrollada para sistemas operativos Android, su finalidad es mostrar la Guía de Vinos y Bodegas de España del año 2017.

Se ha optado Android, en detrimento de iOS, debido a la gran diferencia de cuota de mercado de ambas plataformas en el mercado español: 91,7% Android frente a un 7,9% de iOS en el año 2016, según datos publicados por el periódico 5 días, (Jiménez, 2017).

La aplicación permite obtener los resultados por variedades de uva, por bodegas y por Consejos Reguladores. Estos resultados están enlazados, de tal forma que al mostrar los datos de una bodega se mostrarán todos los vinos de ésta, asimismo al visualizar la ficha de un vino se mostrarán los datos de la bodega que lo embotella.

Incluye una completa descripción de cada variedad de uva, así como una fotografía de la misma e información sobre las sinonimias empleadas en distintas regiones. También dispone de un buscador para localizar cualquier cadena de caracteres en los campos clave. Por último, da al usuario la posibilidad de escribir en la ficha de un vino sus propias anotaciones y otorgarle una valoración numérica.

El proceso de desarrollo se ha realizado en cuatro fases: Una primera fase de análisis para concretar las funcionalidades de la aplicación, las cuales las representamos mediante un diagrama de casos de uso. La segunda fase ha consistido en el diseño de la base de datos mediante un esquema relacional de la misma y normalizándola. En la tercera fase se ha implementado el diseño de la interface de usuario, valiéndonos de diversos bocetos de la aplicación (mockups). Por último, se ha desarrollado la aplicación utilizando Java como lenguaje de programación y SQLite como sistema gestor de base de datos. Para el desarrollo de la app se ha hecho uso de la arquitectura multicapa.

Palabras clave: Aplicación móvil, vino, bodega, guía de vinos.

Abstract

This project consisted of the development of a mobile application to show the Spanish Wine and Wineries Guide for 2017 to smartphones with Android operating system. Android was chosen, at the expense of iOS, due to the great difference between the market shares in both platforms in the Spanish market: 91,7% Android against 7,9% iOS in 2016, according to data published in the newspaper 5 días (Jímenez, 2017).

The mobile application developed allows obtaining results according to grapes varieties, wineries and Governing Councils. These results are linked, in such a way that when showing the information of a winery, all its wines will be shown too. Similarly, when wine sheet is displayed, the information of the winery wich bottled them is shown.

This application includes a complete description of each grape variety, pictures and some information about the synonymies used in different regions. Furthermore, it has a browser to search for any characters in the key fields. Lastly, the user can write its own notes and rate it.

The development process was carried out in four stages: The first one was "analysis stage" where we specify all the functionalities representing them by means of use-case diagrams. The second stage consisted basically of the relational database design. In the third stage, we designed the applicability of our application through various mockups. Finally, the application development itself where we carried out using Java as the programming language and SQLite as database management system and using a multi-layer architecture.

Keywords : Mobile application, wine, winery, wine guide.

Tabla de contenidos

1	Introducción.....	9
1.1	Objetivos	9
1.2	Contenido de la memoria.....	9
2	Búsqueda bibliográfica.....	11
2.1	Guía de vinos de la Organización de Consumidores (OCU)	11
2.2	Guía de vinos Gourmets 2017	12
3	Análisis de requerimientos	13
3.1	Introducción	13
3.1.1	Ámbito del sistema	13
3.2	Descripción general	13
3.2.1	Perspectiva del producto	13
3.2.2	Funciones del producto	14
3.2.3	Características de los usuarios	14
3.2.4	Restricciones generales	14
3.3	Requisitos específicos	14
3.3.1	Interfaces externas	14
3.3.2	Funciones	15
3.3.3	Restricciones de diseño	18
4	Diseño de la aplicación.....	19
4.1	Esquema relacional.....	19
4.2	Casos de uso.....	21
4.3	Mockups.....	22
4.3.1	Menú principal	22
4.3.2	Acerca de	23
4.3.3	Listado de variedades	23
4.3.4	Descripción de la variedad	24
4.3.5	Listado de bodegas	24
4.3.6	Ficha de la bodega	25
4.3.7	Listado de Denominaciones de Origen	25
4.3.8	Buscador	26
4.3.9	Resultados del buscador.....	26
4.3.10	Listado de vinos.....	27
4.3.11	Ficha del vino.....	27



5	Implementación	28
5.1	Base de datos SQLite	28
5.1.1	Normalización de una base datos.....	29
5.1.2	Comentarios	32
5.1.3	Directrices de restauración de la integridad referencial: Borrado y actualización.....	33
5.2	Arquitectura de tres capas	34
5.2.1	Arquitectura multicapa	34
5.2.2	Comunicación entre la capa de Presentación y la capa de Lógica: El controlador.....	35
5.2.3	Comunicación entre la capa de Lógica y la capa de Persistencia: Data Access Layer (DAL).....	35
5.2.4	Patrón Data Access Object (DAO).....	35
5.2.5	Patrón singleton	36
5.2.6	Representación gráfica del modelo de tres capas.....	36
5.2.7	Ventajas de la arquitectura multicapa.....	37
5.2.8	Inconvenientes de la arquitectura multicapa.....	37
5.3	Diseño multicapa en la aplicación	38
5.3.1	Capa de Presentación	39
5.3.2	Capa de Lógica.....	41
5.3.3	Capa de Persistencia.....	43
5.3.4	Capa de control de excepciones.....	47
6	Validación y pruebas	48
6.1	Caso de uso “Listado de variedades”	51
6.2	Caso de uso “Listado de bodegas”.....	52
6.3	Caso de uso “Búsqueda”.....	52
7	Conclusiones	53
8	Bibliografía.....	54

Índice de figuras

Figura 1. Guía de vinos de la OCU	11
Figura 2. Guía de Vinos Gourmets. Menú principal	12
Figura 3. Guía de vinos Gourmets. Búsqueda avanzada	12
Figura 4. Esquema relacional	20
Figura 5. Casos de uso	21
Figura 6. App - Menú principal	22
Figura 7. App – Acerca de.....	23
Figura 8. App - Listado de variedades.....	23
Figura 9. App - Descripción de la variedad	24
Figura 10. App - Listado de bodegas	24
Figura 11. App – Ficha de la bodega	25
Figura 12. App – Listado de Consejos Reguladores	25
Figura 13. App - Ficha del buscador	26
Figura 14. App - Resultados del buscador	26
Figura 15. App - Listado de vinos	27
Figura 16. App - Ficha del vino.....	27
Figura 17. Transformación a primera Forma Normal de un atributo tipo conjunto	30
Figura 18. Transformación a primera Forma Normal de un atributo de tipo registro	31
Figura 19. Transformación a segunda Forma Normal	31
Figura 20. Transformación a tercera Forma Normal.....	32
Figura 21. Representación del modelo de tres capas	37
Figura 22. Estructura de la aplicación por capas	38
Figura 23. Extracto de la clase implementada para testing	48
Figura 24. Resultados del testing	49
Figura 25. Pantalla principal de la aplicación	50
Figura 26. Caso de uso "Listado de variedades"	51
Figura 27. Caso de uso "Listado de bodegas"	52
Figura 28. Caso de uso "Búsqueda"	52



Índice de tablas

Tabla 1. Conversión del esquema relacional a DDL	29
Tabla 2. Layout Consulta_variedades	40
Tabla 3. Uso del controlador	41
Tabla 4. Estructura de la clase Variedad	42
Tabla 5. Definición del patrón singleton	43
Tabla 6. Extracto de la clase ConnectionManager	44
Tabla 7. Interfaz IBodegaDAO	44
Tabla 8. Extracto de la implementación BodegaDAOImp	45
Tabla 9. Extracto de la DAL.....	47

1 Introducción

1.1 Objetivos

General

El objetivo de este trabajo de fin de grado (TFG) es la creación de una aplicación para dispositivos móviles que muestre la Guía de Vinos de España 2017, que, a diferencia de las opciones existentes en el mercado, sea totalmente accesible y fácil de entender para los usuarios sin ningún tipo de restricción de acceso en los contenidos.

Específicos

- Mostrar las fichas de 1.170 vinos con sus características, comentarios, calificaciones y precios.
- Mostrar las características de las variedades de uva, incluyendo una descripción de las bayas, racimos, cepas y aptitudes enológicas básicas
- Mostrar información de contacto de las bodegas embotelladoras.

1.2 Contenido de la memoria

El contenido de esta memoria está organizado de la siguiente forma: en el capítulo 1 – “Introducción”, se describe la estructura del documento y las fases que ha atravesado la aplicación durante su desarrollo.

En el capítulo 2 – “Búsqueda bibliográfica”, se realiza una búsqueda de las herramientas informáticas ligadas a esta temática y cuyos planteamientos se han tenido en cuenta a la hora de realizar este proyecto.

En el capítulo 3 – “Análisis de requerimientos”, se hace una descripción detallada de la funcionalidad del sistema. Se ha estructurado tomando como modelo el estándar de desarrollo de software IEEE-STD-830-1998.

En el capítulo 4 – “Diseño de la aplicación”, se habla de la definición del esquema relacional de la base de datos asociada a la aplicación; de los casos de uso, donde se muestran los potenciales usos que podrían hacerse con la información proporcionada con esta aplicación y de los bocetos utilizados para el diseño de las pantallas. También se mencionarán las diversas herramientas externas utilizadas.

En el capítulo 5 – “Implementación”, se detalla la implementación de la base de datos y se explica la arquitectura por capas empleada para el desarrollo de la aplicación. Se incluye una descripción de las clases Java más importantes.

En el capítulo 6 – “Validación y pruebas”, se mencionan las pruebas realizadas y se muestran pantallas reales de la aplicación completamente operativa.

En el capítulo 7 – “Conclusiones”, se realiza una reflexión sobre el propósito de este trabajo, se da una valoración personal sobre el resultado final y se habla de mejoras a incluir en futuras versiones de la aplicación. Al final del documento podemos encontrar la bibliografía utilizada para realizar este proyecto.

2 Búsqueda bibliográfica

Se ha realizado una búsqueda de aplicaciones equivalentes que existan en el mercado de Android. A fecha hoy (octubre, 2017) en la tienda de aplicaciones de Google “play store” existen solamente dos aplicaciones con una temática similar al objetivo de este proyecto, a continuación, vamos a analizarlas y comentarlas.

2.1 Guía de vinos de la Organización de Consumidores (OCU)

Es una aplicación exclusiva para suscriptores de la Organización de Consumidores (OCU). Las opciones disponibles son “Noticias” y “Videos”, ambas con contenido genérico y poco relacionado con la información de los vinos que se encuentra publicada en la guía.

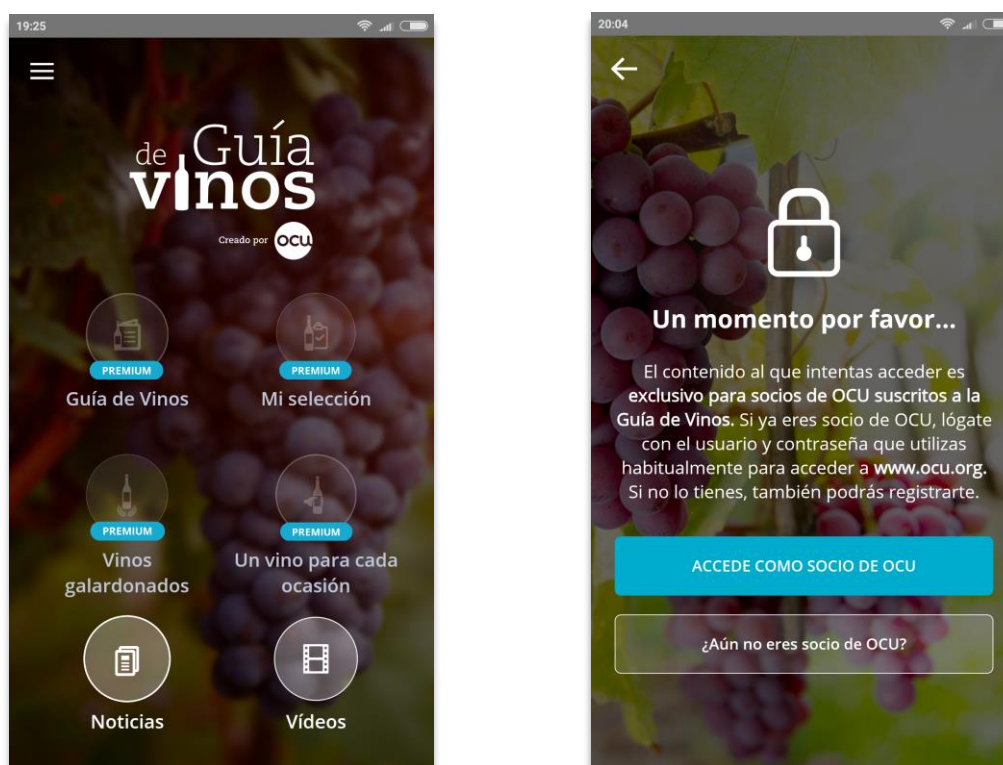


Figura 1. Guía de vinos de la OCU

2.2 Guía de vinos Gourmets 2017

Esta app sí que ofrece gran parte del contenido de forma gratuita, entre sus principales características que ofrece son: búsquedas, recomendaciones, localización en los mapas de Google de las bodegas cercanas.

En la figura 2, observamos la interfaz principal de esta aplicación, que consta de un menú horizontal, cuatro opciones principales con una imagen asociada y un menú vertical que despliega las mismas opciones mencionadas. De todas ellas, la principal la podemos observar en la Figura 3 “Búsqueda avanzada”, en la que se introducen diferentes criterios para la búsqueda de los vinos. Aunque las opciones de búsqueda son variadas, en la actual versión 1.0.4 hemos encontrado los siguientes errores de concepto o de implementación que dificultan su uso:

- Permite seleccionar variedades de uva, pero no existe ningún vino relacionado a las mismas.
- En el desplegable “Todas las denominaciones” no se despliega ninguna denominación.
- Obliga al usuario a memorizar el nombre de un vino para buscarlo posteriormente. Al devolver los resultados de los vinos, se muestra el nombre de su bodega, pero no permite la opción de consultar directamente sus datos.
- La opción “Buscar bodegas” da error y cierra la aplicación, cualquiera que sea el criterio de búsqueda introducido.

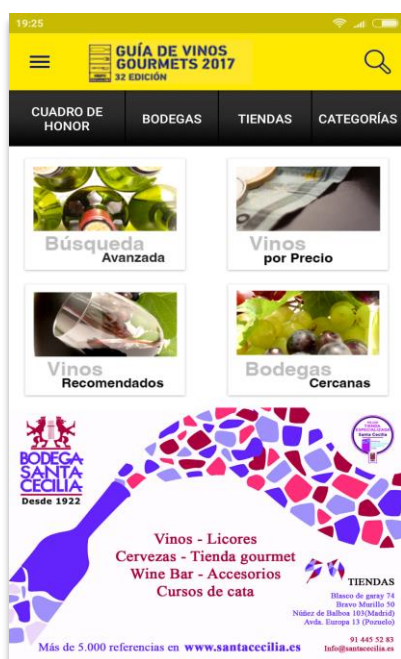


Figura 2. Guía de Vinos Gourmets. Menú principal

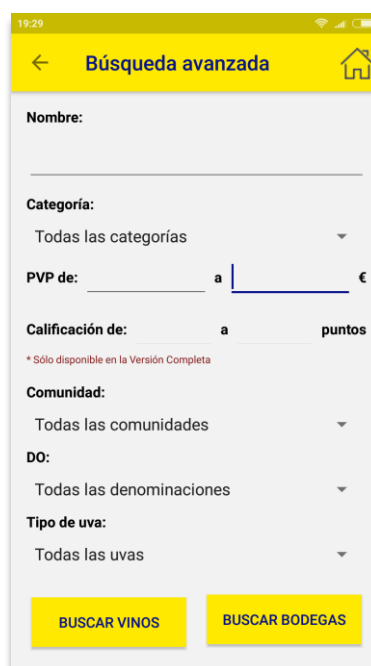


Figura 3. Guía de vinos Gourmets. Búsqueda avanzada

Tomando en cuenta las dificultades que un usuario puede encontrarse al usar estas aplicaciones nos hemos planteado como objetivo de este TFG realizar una aplicación 100% gratuita y robusta en lo que refiere a su diseño e implementación.

3 Análisis de requerimientos

3.1 Introducción

En este apartado se va a proporcionar una apreciación global de la especificación de requerimientos de software (en adelante ERS). El propósito del ERS es obtener un documento completo y formal de los requisitos del sistema. Documento destinado a ser leído por cualquier persona que tenga un interés manifiesto en comprender cómo está desarrollada esta aplicación.

Este ERS se ha estructurado tomando como modelo el estándar de desarrollo de software IEEE-STD-830-1998 (IEEE Standards Association, 2017).

3.1.1 Ámbito del sistema

El nombre asignado a la aplicación es “*Guía de Vinos y Bodegas de España 2017*” destinada a dispositivos móviles con sistema operativo Android.

La información que se mostrará es la que ha sido facilitada en la guía impresa, esta información de normal no se modifica con mucha frecuencia, por lo que la aplicación no necesita acceso a Internet para descargar contenidos. Toda la información permanecerá en la base de datos, y tendrá que ser incluida en el mismo paquete de instalación. De esta manera se asegura un acceso prácticamente instantáneo a la información y su correcto funcionamiento incluso en escenarios sin conexión a Internet.

Beneficios, objetivos y metas:

- Crear una aplicación que mejore en calidad y contenido la escasa oferta actual.
- Facilitar a cualquier persona interesada en el mundo del vino una gran cantidad de información, tanto de bodegas como de vinos.
- Permitir al usuario inexperto tener una apreciación de cada vino, ya que todos ellos incluyen una puntuación sobre 100 obtenida en una cata profesional.

3.2 Descripción general

3.2.1 Perspectiva del producto

La aplicación es totalmente independiente y autónoma, en el sentido de que no depende de otros productos.

3.2.2 Funciones del producto

A continuación, se detallan cuáles serán las principales funcionalidades de la aplicación:

- Listar todas las variedades uva disponibles, mostrando las características de cada variedad y los vinos correspondientes a cada una de ellas, incluyendo sus notas de cata.
- Listar todas las bodegas disponibles, mostrando una ficha con datos de contacto y con todos sus vinos relacionados.
- Listar todos los Consejos Reguladores de las Denominaciones de Origen disponibles, mostrando todos los vinos amparados por cada Denominación de Origen.
- Ofrecer un buscador global completo que devuelva los resultados para todos los campos clave.
- Dar al usuario la opción de valorar y llevar un registro de sus vinos favoritos.
- Facilitar información de contacto de la empresa propietaria de los datos utilizados para creación de la base de datos de la aplicación.

3.2.3 Características de los usuarios

Para la aplicación no existen usuarios con diferentes características, ya que no requiere ningún tipo de identificación para acceder a la totalidad de su contenido. Esto significa que será plenamente operativa para cualquier usuario que proceda a su instalación.

3.2.4 Restricciones generales

Para la instalación y futuras actualizaciones de la aplicación se requiere conexión a Internet y acceso a la tienda de aplicaciones de Google “play store”. Una vez ya lanzada la aplicación al mercado, será necesaria una actualización en los siguientes casos:

- Si se detecta un error en la base de datos
- En caso de detectar un error de programación (*bug*).

Al integrar *Google Play Services* para dar servicio a *Google Maps*, se requiere compilar la aplicación bajo la versión mínima de Android 4.4

3.3 Requisitos específicos

En esta sección se van a analizar los requisitos que debe tener la aplicación de una forma más exhaustiva.

3.3.1 Interfaces externas

Diferenciaremos cuatro tipos de interfaz: usuario, hardware, software y de comunicaciones.

La interfaz de usuario debe ser simple e intuitiva. El menú principal será conciso, visual y de fácil lectura. La interfaz más habitual será una lista en la que se incorporen los diferentes resultados encontrados en la base de datos (lista de bodegas, lista de variedades de uva, etc...). También existirá una interfaz para mostrar los datos del vino y su bodega, asimismo otra interfaz mostrará los datos de la bodega y de todos sus vinos asociados. Por último, habrá una interfaz específica para los campos del buscador global y los resultados de las búsquedas.

Respecto a la interfaz software, es imprescindible que el dispositivo disponga del sistema operativo Android en su versión 4.4 o superior.

En cuanto a la interfaz hardware, es necesario un dispositivo Android, *Tablet* o *Smartphone*, siendo recomendable un mínimo de 4,5 pulgadas de tamaño de pantalla para su correcta visualización.

Por último y referente a la interfaz de comunicaciones, es recomendable, aunque no imprescindible, que el dispositivo disponga de conexión a Internet y acceso a redes de telefonía, ya que la aplicación brindará la opción desde la interfaz de la bodega de enviar un email o realizar una llamada telefónica.

3.3.2 Funciones

En esta subsección se presenta una descripción del sistema a alto nivel, y se describe de forma exhaustiva las funciones que el sistema debe realizar.

Listado de variedades

Entrada: Opción del menú principal.

Proceso: Ordenar alfabéticamente todas las variedades de uva disponibles en la base de datos.

Salida: Listado de variedades.

Listado de vinos por variedad

Entrada 1: Variedad seleccionada por el usuario.

Entrada 2: Resultado de la búsqueda global.

Proceso: Mostrar alfabéticamente todos los vinos relacionados con la variedad

Salida 1: Mostrar información de la variedad.

Salida2: Mostrar la ficha del vino seleccionado.



Información de la variedad

Entrada: Variedad seleccionada por el usuario.

Proceso: Obtener la información técnica de esa variedad, incluyendo sus características organolépticas y una fotografía descriptiva.

Salida: Se muestra la información al usuario.

Listado de bodegas

Entrada: Opción del menú principal

Proceso: Ordenar alfabéticamente todas las bodegas disponibles en la base de datos.

Salida: Listado de bodegas

Ficha de bodega

Entrada 1: Bodega seleccionada por el usuario.

Entrada 2: Resultado de la búsqueda global.

Proceso: Obtener toda la información disponible de la bodega, así como de todos sus vinos que estén incluidos en la guía de vinos.

Salida: Se muestra la información al usuario.

Listado de Consejos Reguladores

Entrada: Opción del menú principal.

Proceso: Ordenar alfabéticamente todos los Consejos Reguladores disponibles en la base de datos.

Salida: Listado de Consejos Reguladores

Listado de vinos por Consejo Regulador

Entrada 1: Consejo Regulador seleccionado por el usuario.

Entrada 2: Resultado de la búsqueda global.

Proceso: Mostrar alfabéticamente todos los vinos relacionados con el Consejo Regulador.

Salida: Mostrar la ficha del vino seleccionado.

Búsqueda global

Entrada: Opción del menú principal.

Proceso: El usuario podrá introducir diferentes valores para realizar una búsqueda personalizada, a continuación, el sistema efectuará una búsqueda en la base de datos, agrupando los resultados en función de la tabla a la que pertenezcan.

Salida: Mostrar los resultados de la búsqueda global.

Resultados de la búsqueda global

Entrada: Filtros de búsqueda introducidos por el usuario en la búsqueda global.

Proceso: El usuario seleccionará uno de los resultados y el sistema, dependiendo del tipo de resultado seleccionado (bodega, variedad, etc..) le remitirá a una pantalla u otra.

Salida: Mostrar la información seleccionada por el usuario en su correspondiente pantalla.

Vinos valorados

Entrada: Opción del menú principal.

Proceso: El sistema buscará en la base de datos todos los vinos que el usuario haya marcado como “valorado”.

Salida: Mostrar un listado con todos los vinos valorados por el usuario.

Ficha del vino

Entrada: Múltiples entradas, ya que ésta es la pantalla donde confluyen todas las opciones de la aplicación, las posibles entradas son:

- Listado de vinos por variedad
- Listado de vinos por Consejo Regulador
- Ficha de Bodega
- Resultados de la búsqueda global
- Vinos valorados

Proceso: Recopilar toda la información del vino seleccionado y de su bodega.

Salida: Mostrar la ficha del vino al usuario.

3.3.3 Restricciones de diseño

Por el tipo de información ofrecida y por cómo se muestra al usuario, se ha decidido que la aplicación sólo esté disponible en formato vertical en la pantalla del dispositivo del usuario.

4 Diseño de la aplicación

Para realizar la fase de diseño se han utilizado dos herramientas externas de apoyo; primera para la creación de los bocetos (mockups) se ha utilizado *Balsamiq Mockups*¹ en su versión 3.5.14 y luego la aplicación *StarUML*² en su versión 2.8.0 para realizar el esquema relacional y los casos de uso.

Balsamiq Mockups, es una aplicación muy completa con una curva de aprendizaje baja. Además, permite instalar bibliotecas de símbolos externas diseñadas por la comunidad de usuarios.

La aplicación *StarUML* facilita la creación de un amplio abanico de diagramas, esquemas relacionales, casos de uso, entre otros.

4.1 Esquema relacional

Una vez analizado y estudiado el origen de datos facilitados por la empresa propietaria de los mismos, se ha llegado a la conclusión de que los requisitos que debe cumplir la base de datos son los siguientes:

- Cada vino obligatoriamente debe tener un número identificador, un nombre, una puntuación y una nota de cata en español e inglés.
- Un vino puede tener una imagen de su etiqueta asociada, el precio de la botella, su graduación alcohólica y la producción total de ese vino. Además, los usuarios podrán marcarlo como “valorado”, al hacerlo, podrán valorar ese vino con una puntuación de 1 a 5 estrellas. También se dará opción a los usuarios de incluir en cada vino sus propias observaciones.
- Un vino debe estar asociado a un tipo de vino (blanco, tinto, etc..) y podría tener asociadas unas características concretas (joven, crianza, reserva, etc...).
- Cada vino estará asociado a una bodega, la cual tendrá obligatoriamente un número identificador, un nombre y el código postal, población y provincia. Podrá tener una dirección, teléfono, email, página web, nombre del enólogo responsable y el número de registro de envasador de la bodega.
- Todos los vinos deben estar asociados a una variedad principal de uva. De cada variedad obligatoriamente debemos tener su número identificador, su nombre y una descripción de sus características. Podrá tener una fotografía identificativa.
- Los vinos pueden estar adscritos a una Denominación de Origen, de la que necesitaremos conocer su número identificador y su nombre.

¹ <https://balsamiq.com/products/mockups/>

² <http://staruml.io/>

- Por último, cada variedad de uva puede tener una o varias sinonimias de su nombre “oficial” (las sinonimias son los nombres con los que se conoce la variedad en diferentes regiones). De cada sinonimia necesitamos conocer su número identificador y su nombre.

Con esta información, se ha procedido a crear el siguiente esquema relacional:

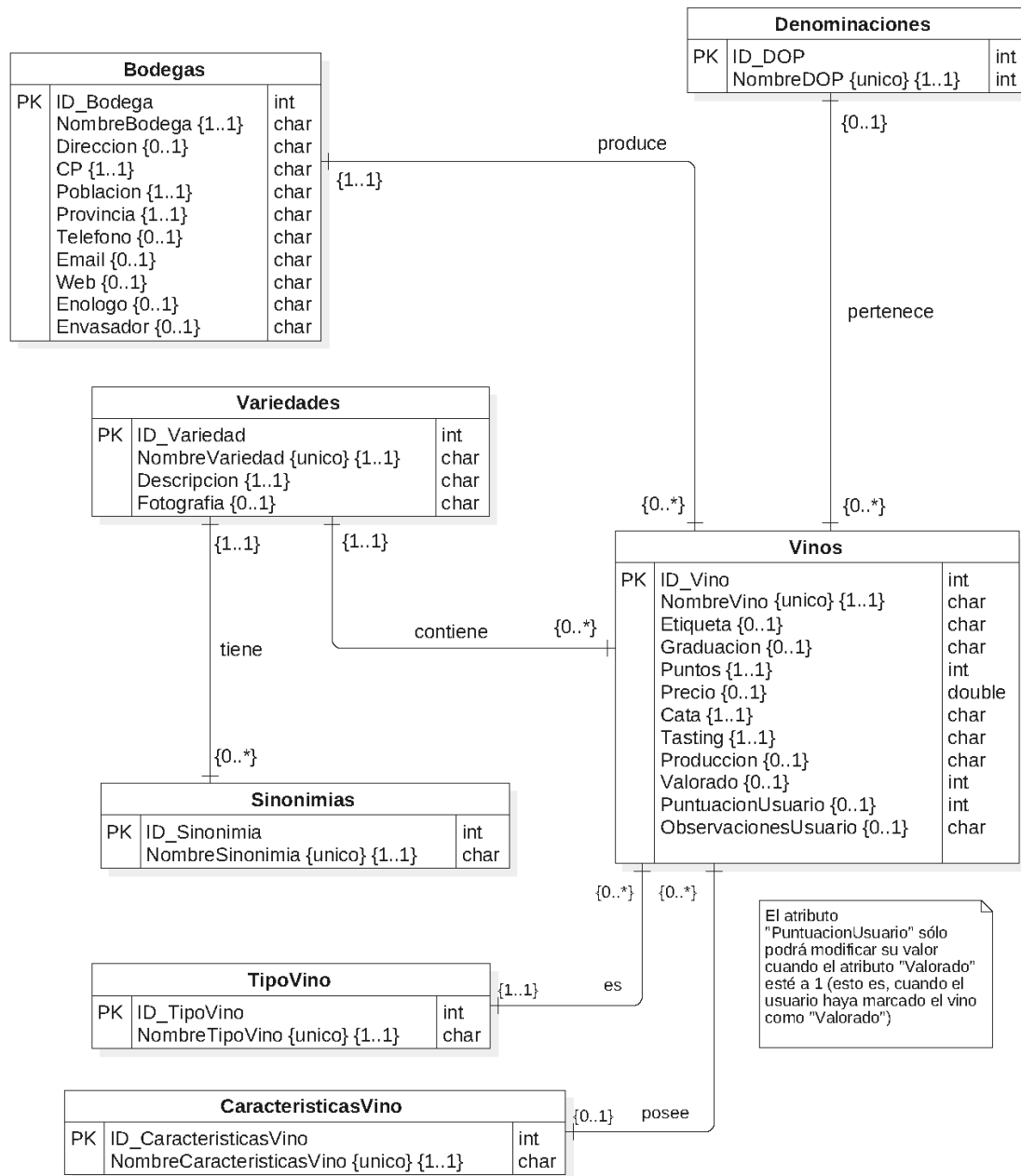


Figura 4. Esquema relacional

4.2 Casos de uso

Los pasos a seguir para realizar una determinada acción quedan descritos en los casos de uso. Cada caso de uso es una interacción entre un actor y el sistema. Las personas o entidades que participan en un caso de uso se denominan actores.

En el caso de esta aplicación existe un único actor, ya que no hay diferencias entre tipos de usuarios, cualquier usuario que utilice la aplicación tendrá los mismos privilegios.

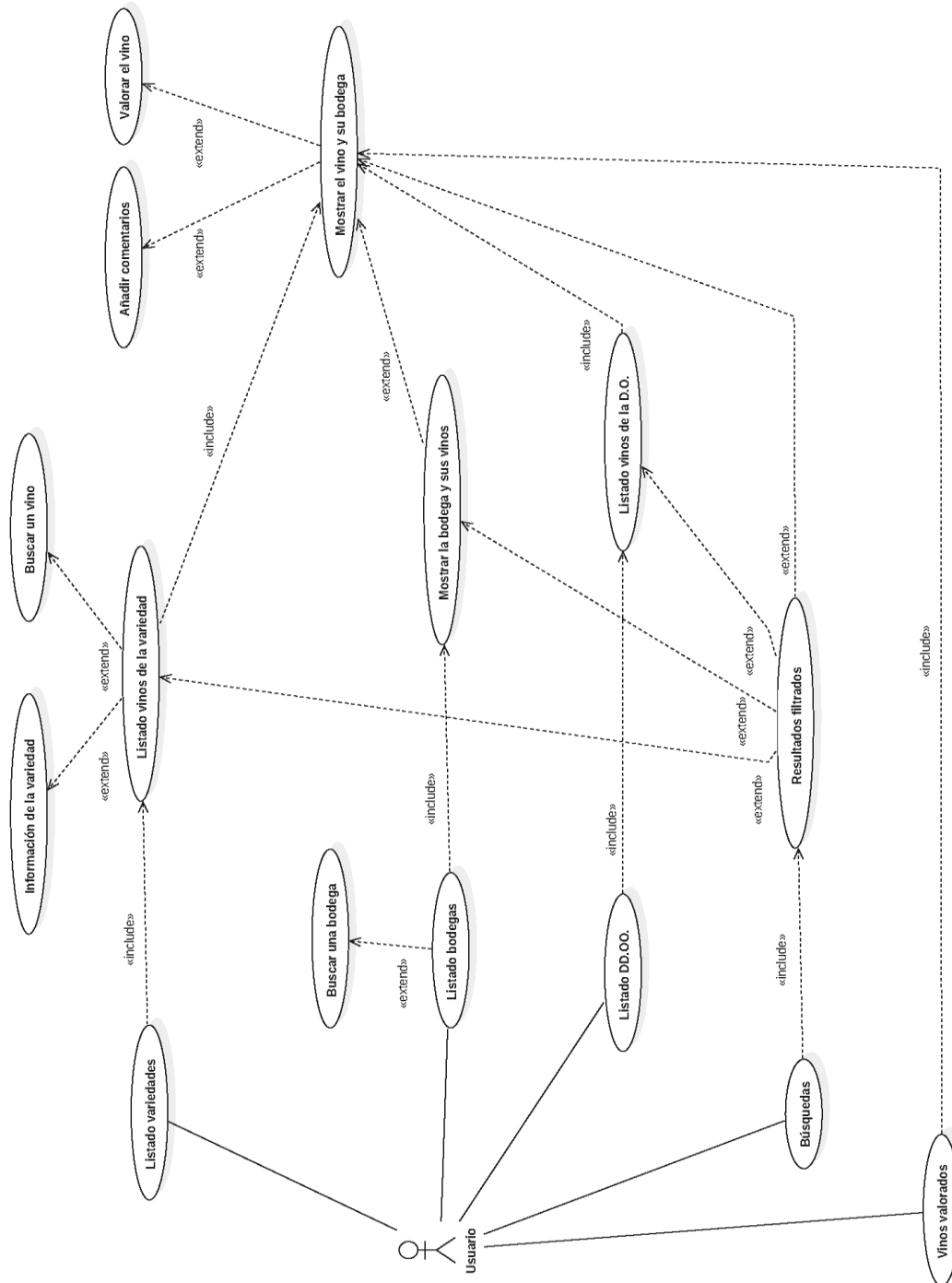


Figura 5. Casos de uso

4.3 Mockups

Un Mockup es un boceto a escala o tamaño real de una aplicación o de una parte de ella. Es muy útil para mostrar al cliente o al equipo de desarrollo la forma que va adquiriendo el producto. Se muestran a continuación los mockups de todas las pantallas de la aplicación.

4.3.1 Menú principal

En esta pantalla encontramos un menú superior (que estará presente en la mayoría de las pantallas posteriores), las cinco opciones principales de la aplicación (variedades, bodegas, ...) junto con una imagen descriptiva y finalmente el logotipo de la empresa.



Figura 6. App - Menú principal

4.3.2 Acerca de

Aquí encontraremos los datos de contacto de la empresa: Logotipo con enlace a su página web, posición en los mapas de Google, dirección y enlaces para Facebook y Twitter. En la parte inferior de la pantalla figura un pequeño banner publicitario.



Figura 7. App – Acerca de

4.3.3 Listado de variedades

En esta pantalla observaremos el listado alfabético de las variedades de uva. Se reserva un pequeño espacio para un banner publicitario.

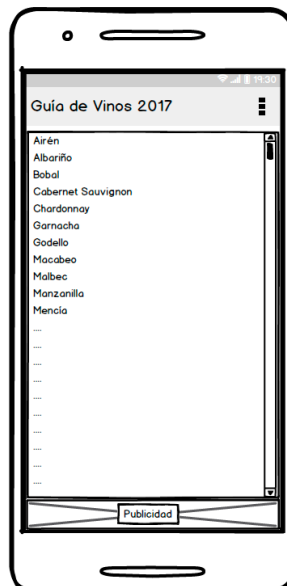


Figura 8. App - Listado de variedades

4.3.4 Descripción de la variedad

En esta pantalla se mostrará una imagen de la variedad, todas las sinonimias asociadas, la morfología de bayas, racimos y cepas y sus aptitudes enológicas básicas.

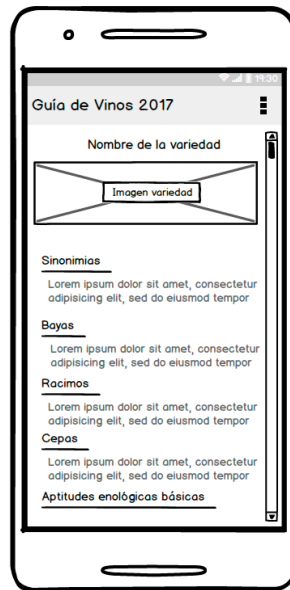


Figura 9. App - Descripción de la variedad

4.3.5 Listado de bodegas

En esta pantalla se mostrará el listado de bodegas ordenadas alfabéticamente. El usuario podrá buscar una bodega tecleando una parte de su nombre. Se reserva un pequeño espacio para un banner publicitario.

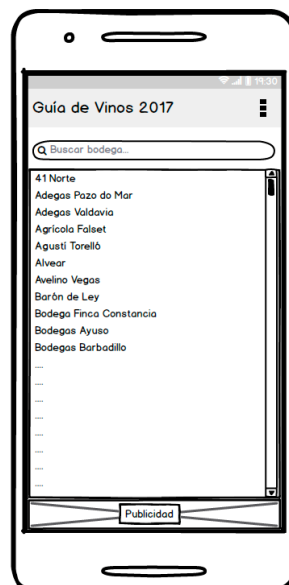


Figura 10. App - Listado de bodegas

4.3.6 Ficha de la bodega

En esta pantalla se mostrarán los datos de contacto de la bodega, su enólogo y su número de embotellador. También se incluirá un listado con todos sus vinos.

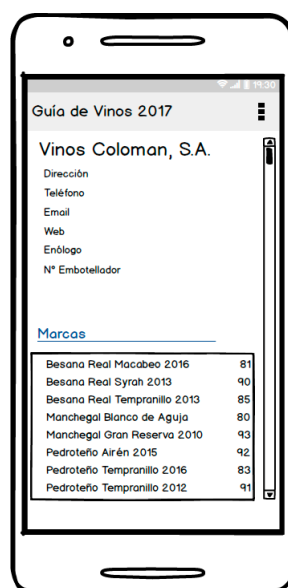


Figura 11. App – Ficha de la bodega

4.3.7 Listado de Denominaciones de Origen

Se mostrará el listado de los Consejos Reguladores de las Denominaciones de Origen ordenado alfabéticamente. Se reserva un pequeño espacio para un banner publicitario.

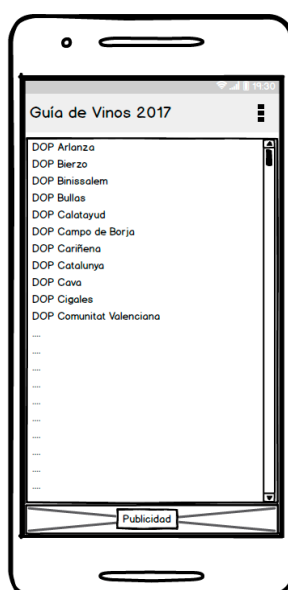


Figura 12. App – Listado de Consejos Reguladores

4.3.8 Buscador

En esta pantalla el usuario introducirá los datos que estime necesarios para realizar una búsqueda acotada en la base de datos.

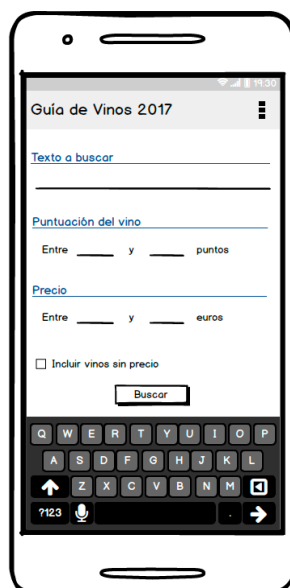


Figura 13. App - Ficha del buscador

4.3.9 Resultados del buscador

Los resultados de la búsqueda del usuario se mostrarán en esta pantalla, en la cual aparecen los resultados en sus respectivos grupos.

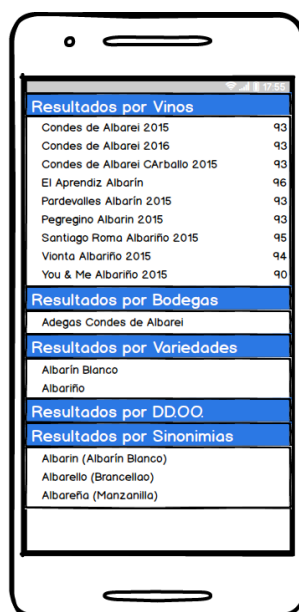


Figura 14. App - Resultados del buscador

4.3.10 Listado de vinos

A esta pantalla se puede llegar desde diferentes lugares de la aplicación, ya que es la que muestra el listado de vinos conforme al caso de uso del que procedan. Por ejemplo, aparecerá el listado de vinos de una bodega o de una variedad de uva. El usuario podrá buscar un vino tecleando una parte de su nombre.

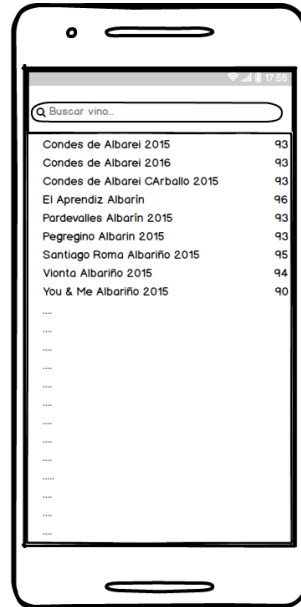


Figura 15. App - Listado de vinos

4.3.11 Ficha del vino

Esta es la pantalla donde se muestran todos los datos del vino seleccionado, su puntuación, precio, notas de cata, anotaciones y puntuación del usuario, así como la información de la bodega que lo embotella.



Figura 16. App - Ficha del vino

5 Implementación

En este apartado se va a detallar la implementación de la base de datos y se explicará la arquitectura por capas empleada para el desarrollo de la aplicación. Finalmente se describirán las clases más importantes.

Como entorno de desarrollo se ha utilizado la plataforma Android Studio en su versión 2.3.3. Se ha elegido esta plataforma por ser un estándar en el desarrollo de aplicaciones Android.

5.1 Base de datos SQLite

La elección del Sistema Gestor de Bases de Datos (SGBD) SQLite se debe a que Android incorpora de serie todas las herramientas necesarias para la creación y gestión de este tipo de base de datos.

Para la creación y actualización de la base de datos, se ha empleado *SQLite Expert*³, versión 5.1.2.140. El cual es un completo sistema gestor de bases de datos SQLite con muchas características de administración y desarrollo.

Una vez obtenido el esquema relacional (Figura 4) se procede a la conversión del mismo al Lenguaje de Definición de Datos (DDL), quedando las tablas definidas de la siguiente forma:

```
CREATE TABLE [Bodegas](  
  [ID_Bodega] INTEGER PRIMARY KEY,  
  [NombreBodega] TEXT NOT NULL,  
  [Direccion] TEXT,  
  [CP] TEXT NOT NULL,  
  [Poblacion] TEXT NOT NULL,  
  [Provincia] TEXT NOT NULL,  
  [Telefono] TEXT,  
  [Email] TEXT,  
  [Web] TEXT,  
  [Enologo] TEXT,  
  [Envasador] TEXT);  
  
CREATE TABLE [Denominaciones](  
  [ID_DOP] INTEGER PRIMARY KEY,  
  [NombreDOP] TEXT NOT NULL UNIQUE);  
  
CREATE TABLE [Sinonimias](  
  [ID_Sinonimia] INTEGER PRIMARY KEY,  
  [NombreSinonimia] TEXT NOT NULL UNIQUE),  
  [ID_Variedad] INTEGER NOT NULL REFERENCES Variedades([ID_Variedad])  
ON DELETE CASCADE ON UPDATE CASCADE;
```

³ <http://www.sqliteexpert.com/>

```

CREATE TABLE [Variedades](
  [ID_Variedad] INTEGER PRIMARY KEY,
  [NombreVariedad] TEXT NOT NULL UNIQUE,
  [Descripcion] TEXT NOT NULL,
  [fotografia] TEXT);

CREATE TABLE [CaracteristicasVino](
  [ID_CaracteristicasVino] INT PRIMARY KEY,
  [NombreCaracteristicasVino] TEXT NOT NULL UNIQUE);

CREATE TABLE [TipoVino](
  [ID_TipoVino] INT PRIMARY KEY,
  [NombreTipoVino] TEXT NOT NULL UNIQUE);

CREATE TABLE [Vinos](
  [ID_Vino] INTEGER PRIMARY KEY,
  [ID_Variedad] INTEGER NOT NULL REFERENCES Variedades([ID_Variedad])
  ON DELETE RESTRICT ON UPDATE CASCADE,
  [ID_Bodega] INTEGER NOT NULL REFERENCES Bodegas([ID_Bodega]) ON
  DELETE CASCADE ON UPDATE CASCADE,
  [ID_DOP] INTEGER REFERENCES Denominaciones([ID_DOP]) ON DELETE
  SET NULL ON UPDATE CASCADE,
  [ID_TipoVino] INTEGER NOT NULL REFERENCES TipoVino([ID_TipoVino])
  ON DELETE RESTRICT ON UPDATE CASCADE,
  [ID_CaracteristicasVino] INTEGER REFERENCES
  CaracteristicasVino([ID_CaracteristicasVino]) ON DELETE RESTRICT ON UPDATE
  CASCADE,
  [Etiqueta] TEXT,
  [NombreVino] TEXT NOT NULL UNIQUE,
  [Graduacion] TEXT,
  [Puntos] INTEGER NOT NULL,
  [Precio] DOUBLE,
  [Cata] TEXT NOT NULL,
  [Tasting] TEXT NOT NULL,
  [Produccion] TEXT,
  [Valorado] INTEGER DEFAULT 0,
  [PuntuacionUsuario] INTEGER DEFAULT 0,
  [ObservacionesUsuario] TEXT);

```

Tabla 1. Conversión del esquema relacional a DDL

5.1.1 Normalización de una base de datos

A la hora de plantearse la creación de una nueva base de datos, uno de los factores decisivos para una buena gestión de la misma es que se encuentre normalizada. Hay varios niveles de normalización y cada nivel superior incluye los requisitos de su nivel inferior. Aunque existen niveles superiores, vamos a explicar la normalización hasta la

tercera Forma Normal, teniendo en cuenta que formalmente vamos a llamar *relación* a una tabla y *atributo* a un campo.

Primera Forma Normal

Una relación está en primera Forma Normal si sus atributos son atómicos.

Se pueden dar dos circunstancias para que esto no se cumpla:

- Que un atributo sea un conjunto de valores similares, por ejemplo, un conjunto de varios números de teléfono.
- Que un atributo sea un registro, esto es que contenga valores que no son similares entre ellos, por ejemplo, que contenga una dirección, un número de calle y una población.

En el primer caso (que sea un conjunto), habrá que eliminar el atributo no atómico de relación y a continuación crear una nueva relación con ese atributo y su clave primaria correspondiente.

En el segundo caso (que sea un registro), habrá que “desglosar” ese atributo en tantos atributos como sean necesarios.

En la figura 17 vemos un ejemplo de transformación en caso de que un atributo sea un conjunto y en la figura 18 un ejemplo de transformación en caso de que un atributo sea un registro.

The diagram illustrates the transformation of a table with a multi-valued attribute into a table with a single-valued attribute and a separate table for the attribute values. A blue arrow points from the 'teléfonos' column of the first table to the 'teléfonos' column of the second table.

vcod	nombre	teléfonos
V1	Pepe	(96 3233258, 964 523844, 979 568987, 987 456123)
V2	Juan	(96 3852741, 910 147258)
V3	Eva	(987 456 312)

vcod	teléfonos
V1	96 3233258
V2	96 3852741
V3	987 456 312
V1	964 523844
V1	979 568987
V1	987 456123
V2	910 147258

Figura 17. Transformación a primera Forma Normal de un atributo tipo conjunto

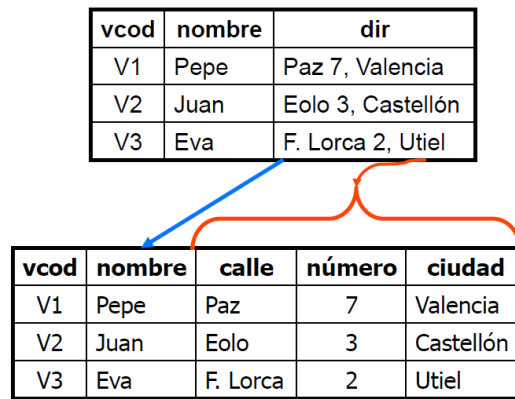


Figura 18. Transformación a primera Forma Normal de un atributo de tipo registro

Segunda Forma Normal

Una relación está en segunda Forma Normal si está en primera Forma Normal y, además:

- Se evalúa la dependencia de los atributos que no son clave con respecto a una clave primaria compuesta. Es decir, no hay ningún atributo no clave que sea dependiente sólo de una parte de una clave primaria compuesta.

Si sucediera, se desglosaría la relación original en tantas relaciones como fueran necesarias hasta que todos los atributos no clave dependan por completo de la clave primaria.

En el ejemplo de la figura 19, vemos que en la relación original, con clave primaria {dni,cod}, *nom_asig* no depende de *dni*, ni *nom_alu* de *cod*, por lo que es necesario realizar la transformación.

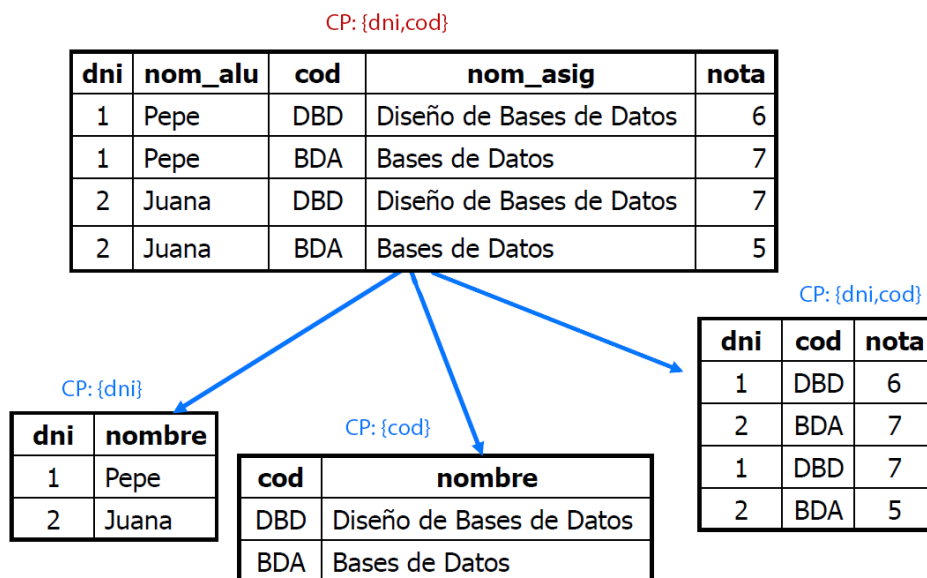


Figura 19. Transformación a segunda Forma Normal

Tercera Forma Normal

Una relación está en tercera Forma Normal si está en segunda Forma Normal y, además:

- Ningún atributo no clave depende de ningún otro atributo no clave.

Si sucediera, extraeríamos el atributo dependiente de la relación original y crearíamos una nueva relación cuya clave primaria será el atributo del que depende.

En el ejemplo de la figura 20 observamos que los atributos no clave *nom_centro* y *director* dependen del atributo no clave *centro* y no del atributo *dni*, que figura como clave primaria, por lo tanto, es necesario realizar la transformación.

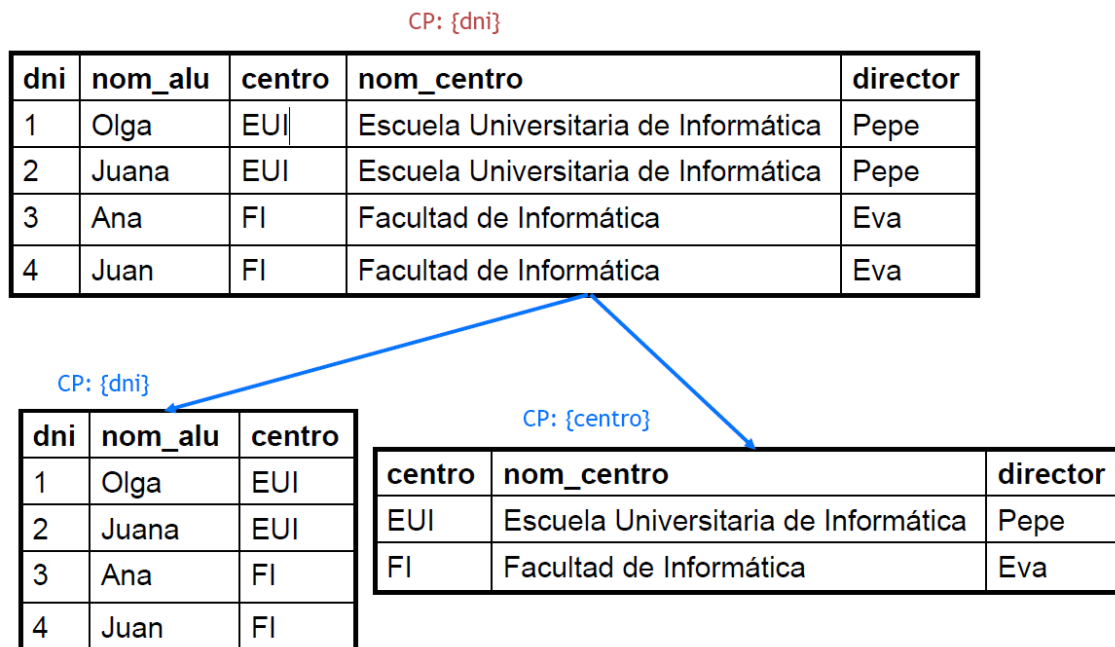


Figura 20. Transformación a tercera Forma Normal

5.1.2 Comentarios

La base de datos ha sido normalizada hasta alcanzar la tercera Forma Normal.

Aunque al usuario de la aplicación no le está permitido eliminar o añadir registros, sí que ha sido necesario normalizar la base de datos y crear las directrices de restauración, ya que es la misma base de datos utilizada por el encargado de su mantenimiento y su futura actualización.

Por otra parte, no ha sido necesario indicar el tipo de integridad referencial de clave ajena debido a que todas las claves constan de un único atributo.

Vinos no es una clase débil de *Bodegas*, ya que no puede existir el mismo nombre de vino en dos bodegas diferentes.

Tampoco *Sinonimias* es una clase débil de *Variedades*, ya que cada sinonimia se identifica con una única variedad de uva.

TipoVino no es clase débil, ya que no hay varios tipos de vino con el mismo nombre.

CaracteristicasVino no es clase débil, ya que no hay varias características con el mismo nombre.

5.1.3 Directrices de restauración de la integridad referencial: Borrado y actualización

En todos los casos, la actualización se realizará en cascada, sin embargo, el borrado variará en cada relación.

Clase Vinos con respecto a clase Denominaciones

Borrado a nulos. Un vino puede perder su Denominación de Origen y seguir existiendo.

Clase Vinos con respecto a clase Bodegas

Borrado en cascada. *Vinos* tiene restricción de existencia respecto a *Bodegas*, por lo que no puede tener un valor nulo la clave foránea *ID_Bodega*.

Clase Vinos con respecto a clase Variedades

Borrado restrictivo. No permitimos que un vino no tenga una variedad principal de uva.

Clase Vinos con respecto a clase TipoVino

Borrado restrictivo. Un vino siempre debe tener un tipo de vino asociado.

Clase Vino con respecto a CaracterísticasVino

Borrado restrictivo. Si un vino tiene una determinada característica (reserva, crianza...), esta característica no puede ser eliminada.

Clase Sinonimias con respecto a clase Variedades

Borrado en cascada. *Sinonimias* tiene restricción de existencia respecto a *Variedades*, no puede existir una sinonimia si no existe su variedad “oficial”.



5.2 Arquitectura de tres capas

Se ha decidido seguir el modelo de tres capas (Presentación, Lógica y Persistencia) visto en la asignatura “Ingeniería del Software”.

Antes de pasar a explicar el contenido de cada una de las capas de la aplicación, es necesario comprender la arquitectura multicapa, que recordamos a continuación.

5.2.1 Arquitectura multicapa

Un sistema multicapa es un modelo de desarrollo de software que consiste en la separación de las partes que forman el sistema. Cada una de estas partes es lo que llamamos una capa.

Existe una relación cliente/servidor entre las capas inferiores (que proporcionan servicios) y las capas superiores (que son usuarios de estos servicios).

Las arquitecturas basadas en capas pueden ser abiertas o cerradas según la dependencia existente entre las capas.

- **Abiertas:** Una capa puede utilizar características de cualquier capa a cualquier nivel.
- **Cerradas:** Una capa sólo utiliza características de su capa inmediatamente inferior.

La arquitectura más recomendable y la que se ha utilizado en este TFG es la cerrada, ya que de esta manera se logra un buen aislamiento de las diferentes etapas del código fuente.

Una forma habitual de implementar este modelo es mediante la creación de tres capas: Presentación, Lógica y Persistencia.

Capa de Presentación: Es la interfaz gráfica que ve el usuario. Incluye los controles de la interfaz (botones, imágenes, etc...) y el código fuente necesario para la interacción del usuario con el sistema. Esta capa se comunica exclusivamente con la capa de Lógica.

Capa de Lógica o de Negocio: Esta capa recibe las peticiones de la capa de Presentación, procesa la petición y le devuelve el resultado.

Si la petición no requiere el acceso a los datos, la misma capa de Lógica devolverá directamente el resultado a la capa de Presentación. En el caso de que la petición sea un acceso a datos, la capa de Lógica hará de “puente” entre la capa de Presentación y la de Persistencia.

Capa de Persistencia: Es la capa que tiene acceso a los datos, recibe las peticiones de la capa de Lógica y devuelve a ésta los resultados. Puede contener uno o varios gestores de bases de datos.

5.2.2 Comunicación entre la capa de Presentación y la capa de Lógica: El controlador

La comunicación entre estas dos capas se consigue mediante uno o varios objetos de control que están situados en la capa de Lógica.

Se puede crear un objeto de control para cada caso de uso o bien agrupar todos los servicios de los objetos de control en una sola clase, lo cual por practicidad es recomendable si la aplicación no es muy extensa.

Para esta aplicación se ha creado un único objeto de control, cuya clase ha recibido el nombre de *controlador*.

Este objeto controlador contiene las llamadas a los métodos definidos en la DAL (Data Access Layer), que está implementada en la capa de Persistencia.

5.2.3 Comunicación entre la capa de Lógica y la capa de Persistencia: Data Access Layer (DAL)

Para la comunicación entre estas dos capas es necesario utilizar una capa de unión llamada DAL (Data Access Layer, Capa de acceso a datos). Está situada en la capa de Persistencia y proporciona las llamadas a todos los métodos de acceso a datos implementados en la capa de Persistencia.

La ventaja de esta capa DAL es evidente: Si por algún motivo cambiase el gestor de base de datos este cambio no afectaría a la capa de Lógica, ya que las llamadas a los métodos de la DAL seguirían siendo los mismos y el cambio se produciría únicamente en la capa de Persistencia.

5.2.4 Patrón Data Access Object (DAO)

Como hemos explicado, la DAL proporciona las llamadas de todos los métodos de datos a la capa de Lógica, pero no contiene su implementación.

Para obtener esta implementación hacemos uso del patrón DAO (Data Access Object), que consta de dos elementos: Una interfaz que detalle los métodos disponibles y una clase que implemente dicha interfaz. Este patrón se ubica en la capa de Persistencia.



Se creará un DAO para cada objeto del que se quiera tener persistencia. Por ejemplo, si tuviéramos una tabla en la base de datos llamada “ciclista”, su implementación sería la siguiente:

- Una clase *Ciclista*, con sus métodos set y get en la capa de Lógica
- Una clase *ICiclistaDAO*, que será la interfaz, en la capa de Persistencia
- Una clase *CiclistaDAOImp*, que será la implementación de *ICiclistaDAO*, también en la capa de Persistencia.

5.2.5 Patrón *singleton*

El patrón *singleton* permite definir clases en Java de las que únicamente existirá una instancia en tiempo de ejecución. En otras palabras, de una determinada clase sólo puede existir un único objeto.

Las ventajas de este patrón, en lo referente al acceso a datos, es que permite que sólo usemos una instancia de conexión para todas las veces que necesitemos acceder a la base de datos durante la ejecución del programa.

El patrón *singleton* se encarga de levantar el driver, conectarse a la base de datos (SQLite en este caso) y devolvernos un objeto del tipo *Connection*. De esta manera, ahorramos recursos al evitar la sobrecarga del sistema.

Este patrón también será utilizado en el diseño de la aplicación.

5.2.6 Representación gráfica del modelo de tres capas

En la figura 21 podemos ver la representación del modelo:

1. El usuario interactúa con la capa de presentación.
2. La capa de Presentación se comunica con la capa de Lógica y le envía la solicitud recibida por la capa de Presentación.
3. La capa de Persistencia resuelve la consulta y devuelve el resultado a la capa de Lógica, quien a su vez devuelve el resultado a la capa de Presentación, que es la que interactúa con el usuario.

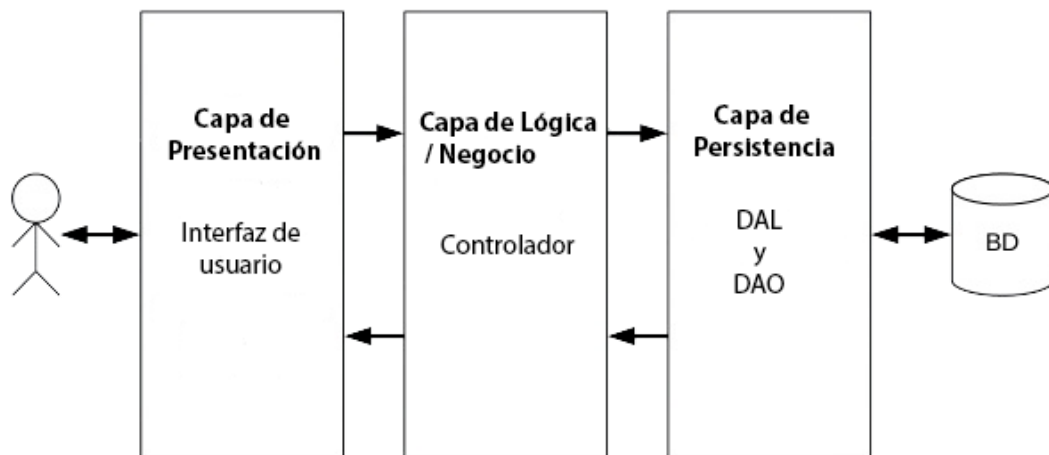


Figura 21. Representación del modelo de tres capas

5.2.7 Ventajas de la arquitectura multicapa

- Aislar la lógica de la aplicación en componentes separados.
- Distribución de capas en diferentes máquinas o procesos.
- Permite la reutilización del código.

5.2.8 Inconvenientes de la arquitectura multicapa

- Requiere un mayor planteamiento en el desarrollo de la aplicación.
- Necesidad de crear clases específicas de unión entre las diferentes capas.
- Cierta redundancia en la declaración de métodos.

5.3 Diseño multicapa en la aplicación

En Android existe un problema, derivado del hecho de que las actividades están íntimamente acopladas tanto con la interfaz como con los mecanismos de acceso a datos. Aun así, se ha conseguido organizar las capas de manera satisfactoria. El esquema resultante puede apreciarse en la Figura 22.

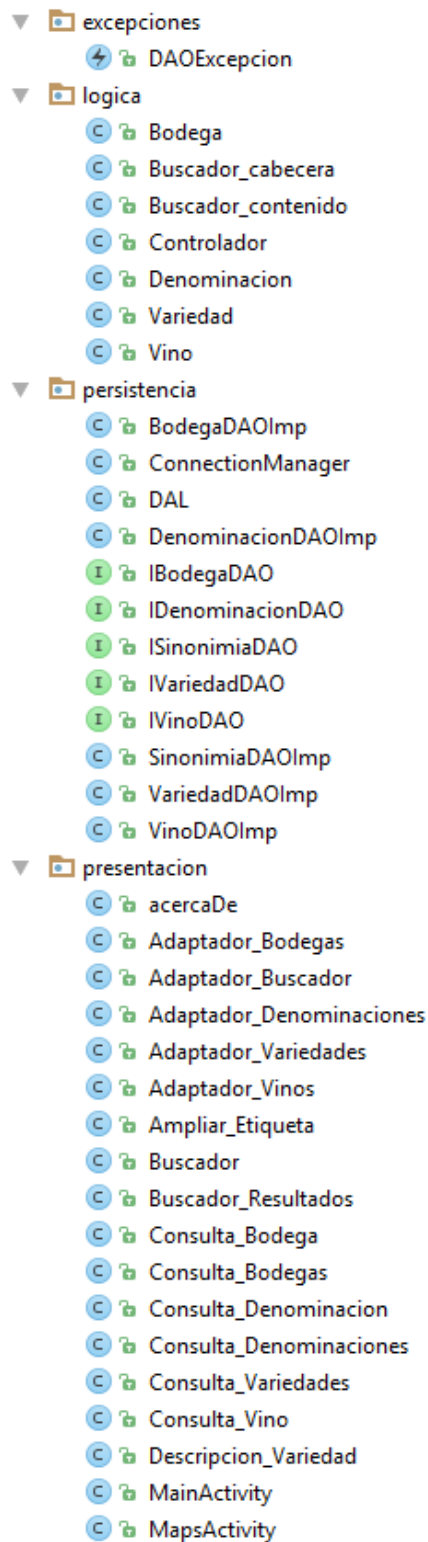


Figura 22. Estructura de la aplicación por capas

A continuación, se explica con detalle la metodología de acceso a los datos en cada una de las capas.

5.3.1 Capa de Presentación

En Android, para mostrar una pantalla al usuario se necesitan dos componentes:

- Un *layout*, que es un contenedor donde se incorporan todos los controles que visualizará el usuario y cuyo código está escrito en XML (Extensible Markup Language). Estos layouts, por imperativo de los sistemas Android, tienen que estar localizados en la ruta “*res/layout*” del proyecto Android.
- Una clase Java vinculada al layout que “infla”, esto es, parsea, los elementos del layout XML para generar el código necesario.

Por este motivo, en esta capa figuran las clases Java vinculadas a los layouts.

En la tabla 2 se muestra el código del layout perteneciente a la actividad *Consulta_variedades.xml*

```
<?xml version="1.0" encoding="utf-8" ?>

<TableLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:id="@+id/tabla_variedades"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:weightSum="5"

tools:context="com.apps.sevi.guiavinos_2017.presentacion.Consulta_Vari
edades">

    <TableRow
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_margin="0dp"
        android:layout_weight="5"
        android:alwaysDrawnWithCache="false"
        android:padding="0dp">

        <ListView
            android:id="@android:id/list"
            android:layout_width="wrap_content"
            android:layout_height="match_parent" />
    </TableRow>

    <LinearLayout
        android:id="@+id/separador6"
        android:layout_width="match_parent"
        android:layout_height="1dp"
        android:layout_toLeftOf="@+id/imgBut_Settings"
```



```
        android:background="#37000000"
        android:orientation="horizontal"></LinearLayout>

<TableRow
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_margin="0dp"
    android:layout_weight="0"
    android:padding="0dp">

    <ImageView
        android:id="@+id/anunciosvariedades"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom"
        android:adjustViewBounds="true"
        app:srcCompat="@drawable/_anuncio_barcolobo" />

</TableRow>
</TableLayout>
```

Tabla 2. Layout Consulta_variedades

A continuación, en la Tabla 3 mostramos el uso del controlador en la clase Java asociada al layout, la clase *Consulta_Variedades.java*

Como se puede observar, una vez definido el controlador, le asignamos dentro del método *onCreate* el patrón singleton.

Seguidamente, asignamos a la variable *consulta*, que es un *ArrayList* del tipo *Variedad*, el método de *controlador* correspondiente a la llamada a la base de datos, en este caso *listadoVariedades*.

El valor obtenido en la variable *consulta* es asignado al instanciar un objeto del tipo *Adaptador_variedades*, que hemos llamado *adaptador*, que será el encargado de mostrar la información obtenida en la consulta del usuario.

```
package com.apps.sevi.guiavinos_2017.presentacion;

import ...

public class Consulta_Variedades extends AppCompatActivity {

    ...

    private ArrayList<Variedad> consulta;

    //definimos el controlador
    private Controlador controlador;
```



```

@Override
protected void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_consulta_variedades);

    lanzarVariedades = (ListView) findViewById(android.R.id.list);

    try {
        controlador= Controlador.dameControlador(this);
    } catch (DAOExcepcion e) {
        System.out.print("DAOExcepcion: "+e);
    }

    consulta=controlador.listadoVariedades(null);

    Adaptador_Variedades adaptador = new Adaptador_Variedades(this,
    consulta);
    lanzarVariedades.setAdapter(adaptador);

    ...

}

```

Tabla 3. Uso del controlador

Lo que acabamos de ver es la forma en que, a través de *controlador*, se comunican la capa de Presentación y la capa de Lógica.

5.3.2 Capa de Lógica

Contiene las clases Java que representan cada una de las tablas de la base de datos que serán utilizadas por los objetos DAO, la clase *Controlador*, utilizada para comunicar la capa de presentación con la capa de persistencia y las clases *Buscador_cabecera* y *Buscador_contenido*, que serán instanciadas cuando se realice una búsqueda global en la aplicación.

Las clases Java que representan las tablas de la base de datos contienen una serie de variables que se corresponden con los campos de la tabla, así como de los correspondientes *sets* y *gets*.

Vemos la estructura de la clase Variedad en la tabla 4.

```
public class Variedad {
    String nombreVariedad;
    String ID_Variedad;
    String descripcion;
    String fotografia;
    String sinonimias;

    public String getNombreVariedad() {
        return nombreVariedad;
    }

    public void setNombreVariedad(String nombreVariedad) {
        this.nombreVariedad = nombreVariedad;
    }

    public String getID_Variedad() {
        return ID_Variedad;
    }

    public void setID_Variedad(String ID_Variedad) {
        this.ID_Variedad = ID_Variedad;
    }

    public String getDescripcion() {
        return descripcion;
    }

    public void setDescripcion(String descripcion) {
        this.descripcion = descripcion;
    }

    public String getFotografia() {
        return fotografia;
    }

    public void setFotografia(String fotografia) {
        this.fotografia = fotografia;
    }

    public String getSinonimias() {
        return sinonimias;
    }

    public void setSinonimias(String sinonimias) {
        this.sinonimias = sinonimias;
    }
}
```

Tabla 4. Estructura de la clase Variedad

En lo que respecta al patrón singleton empleado, está definido en el método *dameControlador*, de la clase *Controlador*. Se muestra en la tabla 5.

```
private static Controlador controlador;
private DAL dal;

// Creación del controlador
private Controlador(Context context) throws DAOExcepcion{
```

```

try{
    //Objeto para comunicarse con la capa de persistencia
    dal = DAL.dameDAL(context);
} catch (DAOExcepcion e){
    throw new DAOExcepcion (e);
}

}

//Patrón Singleton
public static Controlador dameControlador(Context context) throws
DAOExcepcion {
    if (controlador==null)
        controlador = new Controlador(context);
    return controlador;
}

```

Tabla 5. Definición del patrón singleton

Por último, en esta capa encontramos dos clases, Buscador_cabecera y Buscador_contenido, que son utilizadas para la opción de búsqueda global de la aplicación.

5.3.3 Capa de Persistencia

Contiene todas las interfaces DAO y sus respectivas implementaciones, también incluye la clase *ConnectionManager*, que realiza la conexión con la base de datos y un objeto DAL que trabaja directamente con los DAO y será el que se comunique con la capa de Lógica.

Seguidamente analizamos los componentes citados:

ConnectionManager

Esta clase es la que realiza la conexión real con la base de datos. En la llamada se incluye el nombre de la base de datos y el número de su versión actual.

Si posteriormente cambiásemos el número de la versión, se ejecutaría el método onUpgrade, que es el encargado de actualizar la base de datos.

```

public class ConnectionManager extends SQLiteAssetHelper {

    public static final int DATABASE_VERSION = 1;
    public static final String DATABASE_NAME = "guia2017.db";

    public ConnectionManager(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    ...
}

```



```
}

```

Tabla 6. Extracto de la clase *ConnectionManager*

Interfaces DAO

Contienen los métodos de acceso a datos que deberán ser implementados. A modo de ejemplo, mostramos el contenido de la interfaz *IBodegaDAO*.

```
package com.apps.sevi.guiavinos_2017.persistencia;

import com.apps.sevi.guiavinos_2017.logica.Bodega;
import com.apps.sevi.guiavinos_2017.logica.Buscador_contenido;

import java.util.ArrayList;

public interface IBodegaDAO {

    ArrayList<Bodega> listadoBodegas(Integer ID_Bodega);

    Integer ObtenerID_Bodega (Integer ID_Vino);

    ArrayList<Buscador_contenido> busquedaGlobalBodegas(String
cadena);

}
```

Tabla 7. Interfaz *IBodegaDAO*

Implementaciones DAO

Contienen la implementación de las interfaces. Siguen el mismo patrón *singleton* que el visto anteriormente en la clase *controlador*.

Estas implementaciones generan una instancia del objeto *connectionManager* visto anteriormente.

En la tabla 8 tenemos un extracto de la implementación de la interfaz *IBodegaDAO*, llamado *BodegaDAOImp*, en el que podemos observar lo siguiente:

- Patrón *singleton*
- Instanciación de *connectionManager*
- Método de apertura de la base de datos en modo lectura
- Método de cierre de la base de datos
- Método para obtener el ID de una bodega en concreto

```

public class BodegaDAOImp implements IBodegaDAO {

private ConnectionManager connManager;
private SQLiteDatabase db;

//Usado para el Patrón Singleton
private static BodegaDAOImp bodega;

//Patrón Singleton
public static BodegaDAOImp dameDAO(Context context) throws
DAOExcepcion{
    if (bodega==null)
        bodega = new BodegaDAOImp(context);
    return bodega;
}

public BodegaDAOImp(Context context) throws DAOExcepcion{
    try {
        connManager = new ConnectionManager(context);
    } catch (Error e){
        System.out.println(e);
    }
}

private void abrirLectura () {db=connManager.getReadableDatabase();}

private void cerrarBD () {db.close();}

public Integer ObtenerID_Bodega (Integer ID_Vino){
    Integer ID_Bodega=0;

    abrirLectura();
    Cursor mCursor;

    mCursor = db.rawQuery("select ID_Bodega from Vinos WHERE ID_Vino
= "+ID_Vino, null);

    if (mCursor.getCount()>0) {
        mCursor.moveToFirst();

        ID_Bodega=mCursor.getInt(mCursor.getColumnIndex("ID_Bodega"));
    }

    db.close();

    return ID_Bodega;
}

...
...
}

```

Tabla 8. Extracto de la implementación BodegaDAOImp

Capa de Acceso a Datos (DAL)

Como vimos en la definición de la Capa de Acceso a Datos (DAL), ésta proporciona el acceso a la totalidad de los servicios creados en las distintas implementaciones DAO, también dispone de su patrón *singleton*.

Mostramos un extracto en la tabla 9 que ilustra la comunicación con las implementaciones DAO, su patrón *singleton* y tres de los servicios facilitados.

```
package com.apps.sevi.guiavinos_2017.persistencia;

import android.content.Context;

import com.apps.sevi.guiavinos_2017.excepciones.DAOExcepcion;
import com.apps.sevi.guiavinos_2017.logica.Bodega;
import com.apps.sevi.guiavinos_2017.logica.Buscador_contenido;
import com.apps.sevi.guiavinos_2017.logica.Denominacion;
import com.apps.sevi.guiavinos_2017.logica.Variedad;
import com.apps.sevi.guiavinos_2017.logica.Vino;

import java.util.ArrayList;

//Data Access Layer. Capa de indirección para gestionar el acceso a
datos

public class DAL {
    private static DAL dal;
    BodegaDAOImp bodegaDAO;
    DenominacionDAOImp denominacionDAO;
    SinonimiaDAOImp sinonimiaDAO;
    VariedadDAOImp variedadDAO;
    VinoDAOImp vinoDAO;

    // Constructor privado
    private DAL(Context context) throws DAOExcepcion {
        //Objetos para comunicarse con las implementaciones DAO

        bodegaDAO = BodegaDAOImp.dameDAO(context);
        denominacionDAO = DenominacionDAOImp.dameDAO(context);
        sinonimiaDAO = SinonimiaDAOImp.dameDAO(context);
        variedadDAO = VariedadDAOImp.dameDAO(context);
        vinoDAO = VinoDAOImp.dameDAO(context);

    }

    //Patrón Singleton
    public static DAL dameDAL(Context context) throws DAOExcepcion {
        if (dal == null)
            dal = new DAL(context);
        return dal;
    }
}
```

```

// ***** Servicios de la DAL *****

// *****
// ***** Métodos de Bodega *****
// *****

public ArrayList<Bodega> listadoBodegas(Integer ID_Bodega) {
    return bodegaDAO.listadoBodegas (ID_Bodega);
}

public int ObtenerID_Bodega (Integer ID_Vino){
    return bodegaDAO.ObtenerID_Bodega (ID_Vino);
}

public ArrayList<Buscador_contenido>
busquedaGlobalBodegas (String Cadena) {
    return bodegaDAO.busquedaGlobalBodegas (Cadena);
}

```

Tabla 9. Extracto de la DAL

5.3.4 Capa de control de excepciones

Además de las tres capas principales, se añadirá una cuarta capa para control de las excepciones, de esta manera las tendremos centralizadas en una sola clase.

6 Validación y pruebas

Una vez finalizada la etapa de implementación, se han realizado diversas pruebas para comprobar el correcto funcionamiento de la aplicación.

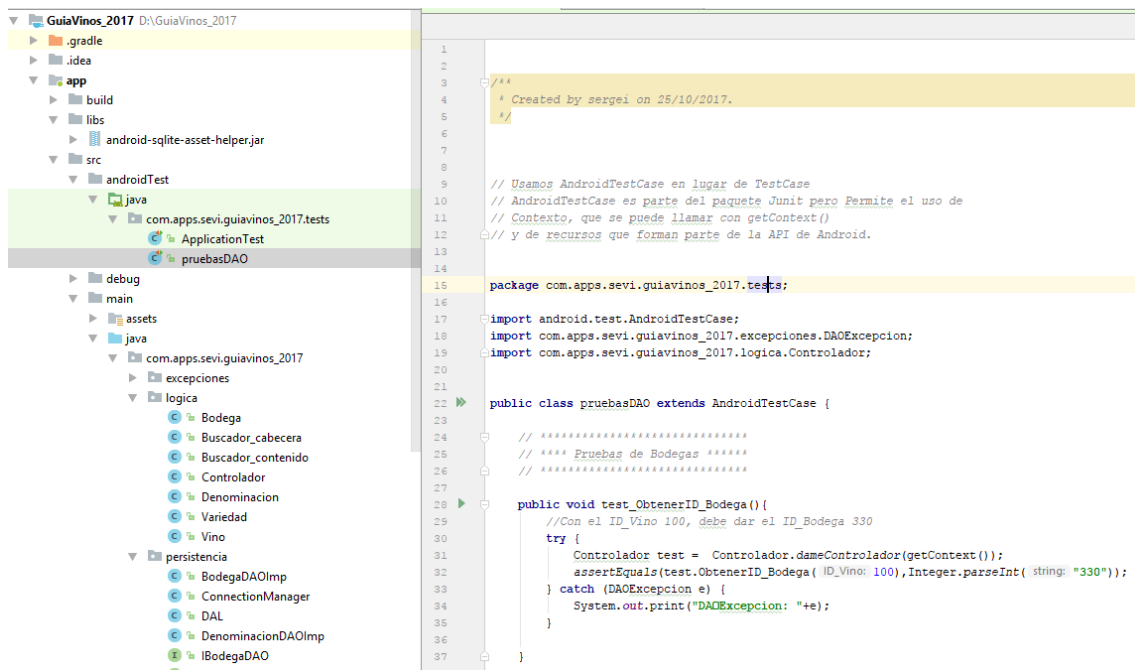
En la presente etapa se han optimizado diversos métodos para hacerlos más eficientes y se han corregido los errores observados en su funcionamiento.

En primer lugar, se ha utilizado las bibliotecas *JUnit* en su versión 4.12, que vienen incorporadas en Android Studio y que permiten realizar pruebas unitarias en las aplicaciones.

Para que Android Studio reconozca que una clase es una clase de pruebas, debe estar ubicada en una de las dos rutas siguientes:

Nombre modulo/src/test/java: Si las clases a probar no tienen dependencias se pueden ubicar aquí. Las pruebas se ejecutarán en una versión modificada de android.jar que permite utilizar bibliotecas de simulación tales como *Mockito*, las cuales no usaremos en este proyecto.

Nombre modulo/src/androidTest/java: Las pruebas aquí ubicadas tienen acceso al Context de la app. Si las pruebas a realizar necesitan acceder al Context (como es el caso) o tienen dependencias de otras clases deberán alojarse aquí. Esta es la ubicación que ha sido elegida para testear la app.



```
1
2
3 /**
4  * Created by sergei on 25/10/2017.
5  */
6
7
8
9 // Usamos AndroidTestCase en lugar de TestCase
10 // AndroidTestCase es parte del paquete Junit pero permite el uso de
11 // Contexto, que se puede llamar con getContext()
12 // y de recursos que forman parte de la API de Android.
13
14
15 package com.apps.sevi.guiavinos_2017.tests;
16
17 import android.test.AndroidTestCase;
18 import com.apps.sevi.guiavinos_2017.excepciones.DAOExcepcion;
19 import com.apps.sevi.guiavinos_2017.logica.Controlador;
20
21
22 public class pruebasDAO extends AndroidTestCase {
23
24     // ***** Pruebas de Bodegas *****
25     // ***** Pruebas de Bodegas *****
26     // ***** Pruebas de Bodegas *****
27
28     public void test_ObtenerID_Bodega() {
29         // Con el ID_Vino 100, debe dar el ID_Bodega 330
30         try {
31             Controlador test = Controlador.dameControlador(getContext());
32             assertEquals(test.ObtenerID_Bodega( Integer.parseInt( string: "100") ), Integer.parseInt( string: "330" ));
33         } catch (DAOExcepcion e) {
34             System.out.println("DAOExcepcion: "+e);
35         }
36     }
37 }
```

Figura 23. Extracto de la clase implementada para testing

En la figura 23, en la parte izquierda, observamos la clase “pruebaDAO”, ubicada en la ruta descrita anteriormente y que es donde figuran todas las pruebas realizadas. En la parte derecha vemos un extracto de la clase.

Podemos comprobar que la clase extiende de *AndroidTestCase*, que es parte del paquete *JUnit* y permite el uso de *Context*, al que se puede llamar con *getContext()*, también permite el uso de recursos que forman parte de la API de Android.

Hay que tener en cuenta que es imprescindible el que todos los métodos comiencen con la palabra “test” para que sean reconocidos como métodos de pruebas.

La forma de evaluar los resultados es mediante los métodos *Assert* que proporciona *JUnit*. Estos métodos son los siguientes:

- `assertArrayEquals()`
- `assertEquals()`
- `assertTrue()`
- `assertFalse()`
- `assertNull()`
- `assertNotNull()`
- `assertSame()`
- `assertNotSame()`
- `assertThat()`

En el extracto de la figura 23 vemos el empleo del método `assertEquals`, que devolverá true si la comparación es verdadera.

Una vez realizados los test, Android Studio genera una página html con los resultados obtenidos. Se pueden observar en la figura 24.



Figura 24. Resultados del testing



Una vez pasados los tests de *JUnit*, la aplicación se distribuyó entre compañeros de trabajo y personas ajenas a la empresa para que pudieran valorarla, tanto en diseño como en lo relativo a la interacción con el usuario. Gracias a estos usuarios se han detectado algunos *bugs* que habían pasado inadvertidos y se han mejorado algunos aspectos de la interfaz.

La aplicación ha sido testada en diferentes dispositivos, desde el emulador de Android que viene integrado en Android Studio hasta diversos Smartphones y Tablets. Se ha podido comprobar que es visualmente idéntica en diferentes tamaños de pantalla y resoluciones.

A continuación, en la Figura 25 se muestran las pantallas reales de la aplicación en tres de los casos de uso mostrados previamente en la figura 5, concretamente los casos de uso “Listado de variedades”, “Listado de bodegas” y “Búsqueda”, partiendo siempre de la pantalla principal de la aplicación.



Figura 25. Pantalla principal de la aplicación

6.1 Caso de uso “Listado de variedades”

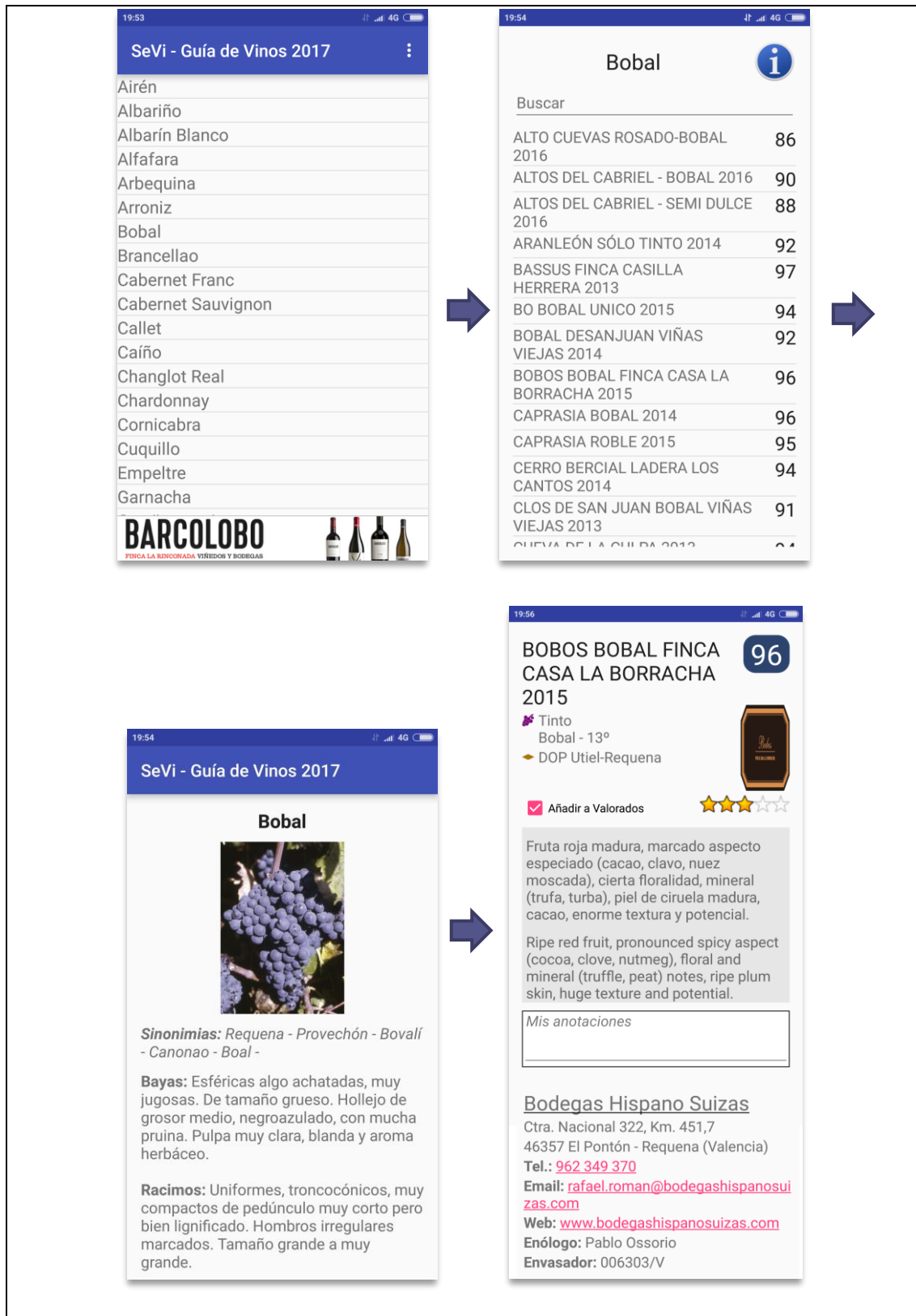


Figura 26. Caso de uso "Listado de variedades"

6.2 Caso de uso “Listado de bodegas”

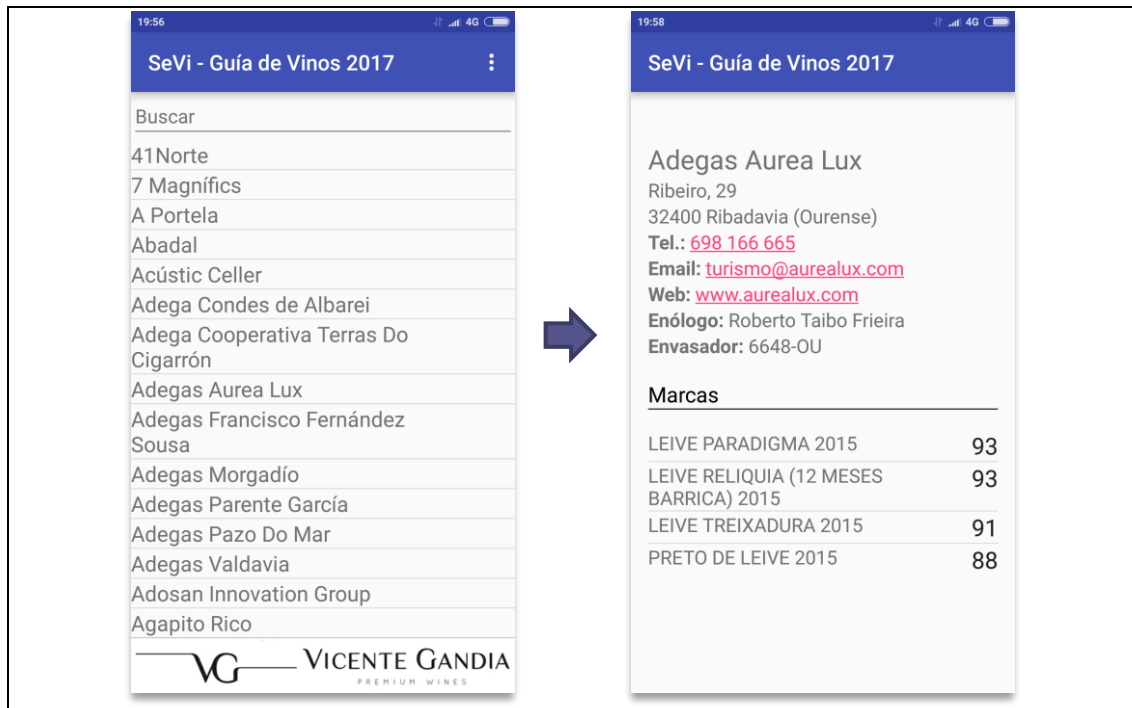


Figura 27. Caso de uso "Listado de bodegas"

6.3 Caso de uso “Búsqueda”

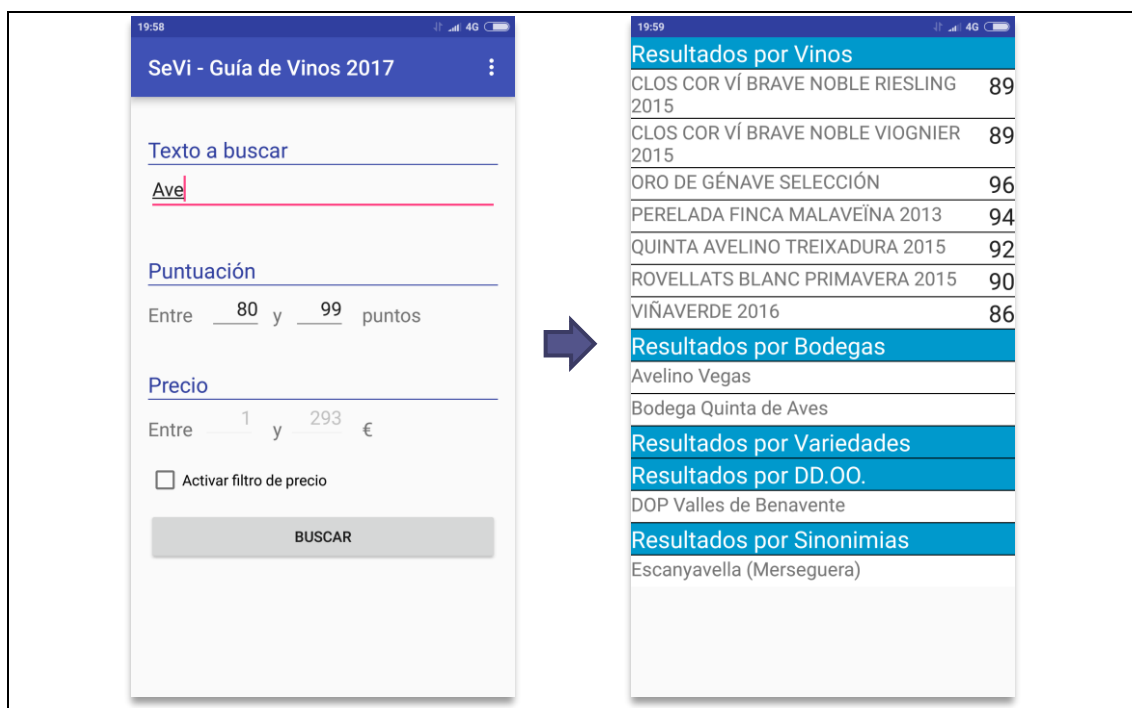


Figura 28. Caso de uso "Búsqueda"

7 Conclusiones

La empresa donde trabajo, *La Semana Vitivinícola, S.L.*, edita anualmente la Guía de Vinos y Bodegas de España en formato impreso, pero no dispone de una versión digital. El propósito de este TFG ha sido diseñar una aplicación para dispositivos móviles de dicha Guía.

De esta manera, la aplicación para dispositivos móviles, además de incluir todo el contenido de la versión impresa, proporciona valores añadidos al usuario, tales como realizar búsquedas de todo tipo (por nombre, por bodega, por denominación de origen, etc.), así como la posibilidad de que el usuario pueda incluir sus propias anotaciones en la ficha de los vinos.

Para las fases de diseño y desarrollo se tuvieron en cuenta diversas directrices aprendidas a lo largo de la carrera. Desde la normalización de una base de datos, pasando por el esquema relacional, la conversión al lenguaje de definición de datos (DDL), la especificación de casos de uso, la definición de la aplicación en una arquitectura multicapa, hasta el diseño de los bocetos de la interfaz del usuario (mockups).

La principal fuente de información que contiene la aplicación “Guía de vinos de España 2017” está basada en la versión impresa de la guía del mismo nombre. Se puede mencionar que la tarea del tratamiento de la información ha sido ardua, tareas como la organización de una gran cantidad de datos poco estructurados o el posterior almacenamiento de los mismo en una base de datos consistente. Arduo también ha sido la fase de creación de las interfaces para conseguir una visualización equivalente de todos los componentes, imágenes y textos en cualquier tipo de pantalla y resolución debido a que se requiere mucha dedicación.

Como fruto de este trabajo he intentado y espero haber desarrollado una aplicación robusta, visualmente atractiva y de fácil manejo para cualquier tipo de usuario, que tras haber sido testeada se puede decir que está libre de fallos de programación.

Queda pendiente para futuras versiones el integrar la aplicación con las redes sociales, registrar los tiempos de uso de la aplicación y desarrollar una versión para el sistema iOS de Apple.

8 Bibliografía

- Jiménez, M. (2017). *El iPhone 7 eleva la cuota de Apple a nivel mundial. Cinco Días*. Recuperado el día 5 de Septiembre de 2017, desde https://cincodias.elpais.com/cincodias/2016/12/07/tecnologia/1481141193_499323.html
- Revelo, J. (2017). *Tutorial De Bases De Datos SQLite En Aplicaciones Android. Hermosa Programación: +50 Tutoriales Desarrollo Android*. Recuperado el día 15 de Septiembre de 2017, desde <http://www.hermosaprogramacion.com/2014/10/android-sqlite-bases-de-datos/>
- Interactive and Cooperative Technologies Lab (2017). *Modelo relacional*. Recuperado el día 18 de Septiembre de 2017, desde <http://ict.udlap.mx/people/carlos/is341/bases03.html>
- Revelo, J. (2017). *Configurar Layouts y Views En Android Studio. Hermosa Programación: +50 Tutoriales Desarrollo Android*. Recuperado el día 22 de Septiembre de 2017, desde <http://www.hermosaprogramacion.com/2014/09/android-layouts-views/>
- Universitat Politècnica de València (2017). *Implementación del patrón DAO (Data Access Object)*. Recuperado el día 28 de Septiembre de 2017, desde <http://www.upv.es/visorv/media/d55b1e80-c327-f14f-93f6-e55ee68525fa/c>
- Nolasco Valenzuela, J. (2015). *Desarrollo de aplicaciones móviles con Android*. Paracuellos de Jarama: Ra-Ma.
- Torres Milano, D. (2012). *Android Application Testing Guide*. Packt Publishing
- IEEE Standards Association (2017). *IEEE SA - 830-1998 - IEEE Recommended Practice for Software Requirements Specifications*. Recuperado el día 20 de Octubre de 2017, desde <https://standards.ieee.org/findstds/standard/830-1998.html>