

# *i-Croqueta*

## Manual del desarrollador



Versión	1.92
Última edición	15/03/2021
Autor	Alicia Rebordinos Guzmán
Departamento	Programación multimedia y dispositivos móviles

# Contenido

1.	Introducción .....	4
2.	Especificación de requisitos .....	4
2.1.	Requisitos funcionales .....	4
2.2.	Requisitos no funcionales: .....	4
2.3.	Entorno de desarrollo .....	5
2.4.	Lenguaje de programación.....	5
2.5.	Dependencias .....	5
2.6.	Entorno de ejecución.....	5
3.	Diseño de datos.....	5
3.1.	Tablas .....	7
3.1.1.	Persona .....	7
3.1.2.	Teléfono .....	7
3.1.3.	Dirección .....	7
3.1.4.	Tarjeta .....	7
3.1.5.	Persona-Teléfono .....	7
3.1.6.	Persona-Dirección.....	7
3.1.7.	Persona-Tarjeta .....	8
3.1.8.	Pedido.....	8
3.1.9.	Linea.....	8
3.1.10.	Producto .....	8
3.1.11.	Ingrediente .....	8
3.1.12.	Producto-Ingredientes .....	8
3.1.13.	Tipo .....	8
3.1.14.	Tipo-Ingredientes .....	8
4.	Estructura de directorios .....	9
4.1.	Paquete icroqueta .....	9
4.1.1.	ActiveProductActivity.java.....	9
4.1.2.	HistoryActivity.java.....	10
4.1.3.	LineaActivity.java.....	10
4.1.4.	LoginActivity.java .....	10
4.1.5.	MainActivity.jav .....	10
4.1.6.	MapsActivity.java.....	10
4.1.7.	MenuBar.java.....	10
4.1.8.	OptionActivity.java.....	11

4.1.9. ProductActivity.java .....	11
4.1.10. RegisterActivity.java .....	11
4.1.11. RegisterPaymentActivity.java.....	11
4.1.12. ShoppingCarActivity.java.....	11
4.2. Paquete icroqueta/adapter .....	11
4.2.1. IngredientRecyclerViewAdapter.java .....	12
4.2.2. LineRecyclerViewAdapter.java .....	12
4.2.3. OrderRecyclerViewAdapter.java .....	12
4.2.4. ProductRecyclerViewAdapter.java .....	12
4.3. Paquete icroqueta/database .....	12
4.3.1. DBHelper .....	12
4.3.2. DBSource .....	14
4.4. Paquete icroqueta/database/dto .....	14
4.4.1. ProductoCarrito.java .....	14
4.5. Paquete icroqueta/database/entidades .....	14
4.6. Paquete icroqueta/database/tablas .....	15
4.7. Paquete icroqueta/database/utills .....	15
4.7.1. LocalizadorDirecciones.java.....	15
4.7.2. ValidadorDNI.java.....	15

# 1. Introducción

Este es el manual para el desarrollador donde se describe la documentación técnica de programación de la aplicación I-Croqueta.

## 2. Especificación de requisitos

La elección del trabajo conlleva una serie de requisitos funcionales explícitos y otra serie de requisitos no funcionales implícitos que son los siguientes:

### 2.1. Requisitos funcionales

Los requisitos funcionales de esta aplicación son los siguientes:

- El usuario debe registrarse y tener perfil de usuario para poder realizar una compra.
- La web debe mantener los requisitos mínimos de diseño y usabilidad.
- El usuario podrá acceder a su histórico de pedidos, tener seguimiento de el envío de los mismos, así como acceso a políticas de cancelación.
- Los productos se ordenarán por ingredientes y categorías.

### 2.2. Requisitos no funcionales:

- Requisitos mínimos de diseño y usabilidad
- Facilidad en el programa con interfaz sencilla e intuitiva.
- Rapidez de respuesta en las operaciones.
- Procedimiento simultáneo de peticiones.
- Estabilidad en el programa a la hora de ir añadiendo contenido a la base de datos.
- Los usuarios destino serán los compradores que harán uso de la aplicación.
- El software debe ser compatible dispositivos portátiles.
- Requerimos de un hosting de calidad.
- Función TPV virtual (Terminal de Punto de Venta) para la realización segura de los pagos online o PayPal.
- Servidor para almacenar los datos.

## 2.3. Entorno de desarrollo

Para trabajar con el proyecto se necesita tener instalados los siguientes programas y dependencias:

- Java JDK 7.
- Android Studio.
- Git.

## 2.4. Lenguaje de programación

Se ha utilizado java para la codificación, ya que para desarrollar con Android Studio se puede utilizar este lenguaje.

## 2.5. Dependencias

Para este proyecto se han implementado Glide para cargar imágenes de manera remota y la api de google maps para poder hacer uso de sus mapas.

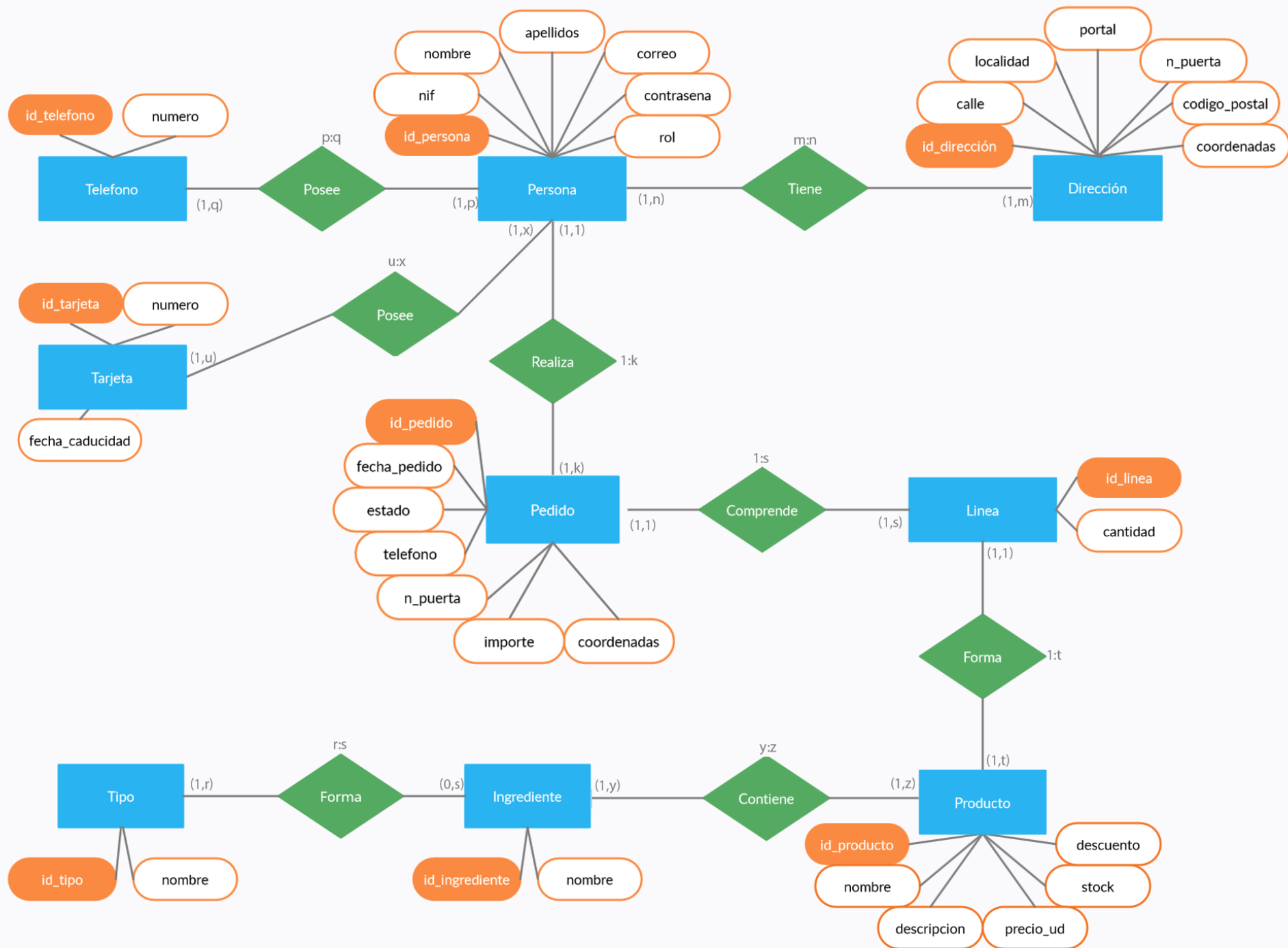
## 2.6. Entorno de ejecución

Se proporcionará un ejecutable en formato apk para poder instalar en dispositivos Android.

# 3. Diseño de datos

Ya tenemos los requisitos necesarios para la ejecución del programa que se plantean, vamos a definir las características del sistema que nos permitirá implementarlo de manera efectiva la aplicación.

Ya que el funcionamiento interno de la información será realizado través de un gestor de base de datos tendremos que dejar constancia de la relación entre las distintas entidades.



## 3.1. Tablas

Una vez hecho esto podemos definir las partes como:

### 3.1.1. Persona

El programa necesitará realizar acciones como crear un nuevo cliente y modificarlos datos o consultar datos.

En la base de datos necesitaremos una tabla con toda la información relevante como: id único, nif, nombre, apellidos, correo, contraseña y rol.

### 3.1.2. Teléfono

El programa guardará todos los teléfonos en una base de datos ajena al de Personas ya que puede existir duplicidad en estos datos

En la base de datos necesitaremos una tabla con toda la información relevante como: id único y el número.

### 3.1.3. Dirección

El programa guardará toda la dirección en una base de datos ajena al de Personas ya que puede existir duplicidad en estos datos

En la base de datos necesitaremos una tabla con toda la información relevante como: id único, calle, localidad, portal, numero de puerta, código postal y una coordenada autogenerada para poder hacer uso de los mapas de google.

### 3.1.4. Tarjeta

El programa guardará todos las en una base de datos ajena al de Personas ya que puede existir duplicidad en estos datos

En la base de datos necesitaremos una tabla con toda la información relevante como: id único, número, fecha de caducidad.

### 3.1.5. Persona-Teléfono

Como una persona puede tener más de un teléfono, o varios pueden hacer uso del mismo, necesitaremos una tabla que relacione a ambas tantas veces como distintos sean los datos que posea.

### 3.1.6. Persona-Dirección

Como una persona puede tener más de una direccion, o varios pueden hacer uso del mismo, necesitaremos una tabla que relacione a ambas tantas veces como distintos sean los datos que posea.

### 3.1.7. Persona-Tarjeta

Como una persona puede tener más de una tarjeta, o varios pueden hacer uso de la misma, necesitaremos una tabla que relacione a ambas tantas veces como distintos sean los datos que posea.

### 3.1.8. Pedido

El programa necesitará realizar acciones como crear un nuevo pedido y modificarlos datos o consultar datos.

En la base de datos necesitaremos una tabla con toda la información relevante como: id único, fecha de realización del pedido, el estado, teléfono del cliente, importe total, las coordenadas y numero de puerta del cliente que ha solicitado un pedido.

### 3.1.9. Linea

Como un pedido está formado por uno o más productos, es necesario tener una tabla con toda esta información que relacione el pedido con los productos y con la cantidad de cada uno.

### 3.1.10. Producto

El programa necesitará realizar acciones como búsqueda de un producto, por ello necesitamos en la base de datos una tabla con toda la información relevante como: id único, nombre, descripción, precio por unidad, stock y descuento.

### 3.1.11. Ingrediente

Como nuestro programa realiza búsquedas por tipo de ingrediente, necesitamos una tabla con toda esta información: id único, nombre, si es vegetariano, si tiene gluten y si contiene lactosa.

### 3.1.12. Producto-Ingredientes

Cada producto está compuesto de determinados ingredientes y por ello necesitamos una tabla que haga la relación entre el producto y lo los ingredientes que tiene.

### 3.1.13. Tipo

Tabla que agrupa los distintos tipos de alimentos para agrupar los ingredientes en el correspondiente.

### 3.1.14. Tipo-Ingredientes

La relación que existe entre los tipos de alimentos y los ingredientes que están dentro de estas categorías.



## 4. Estructura de directorios

El repositorio del proyecto se distribuye de la siguiente manera:

- `/`: contiene los ficheros de configuración de Gradle, de los servicios de integración continua.
- `/app/`: módulo correspondiente a la aplicación.
- `/app/src/`: código fuente de la aplicación.
- `/app/src/main/`: contiene todas las clases comunes.
- `/app/src/main/res/`: recursos de la aplicación (layouts, menús, imágenes, cadenas de texto, estilos, etc.).
- `/app/src/main/java/com/example/icroqueta/`: todas las activities del proyecto.
- `/app/src/main/java/com/example/icroqueta/adapter/`: todas las activities usadas como adaptadores.
- `/app/src/main/java/com/example/icroqueta/database/`: las clases destinadas a la base de datos.
- `/app/src/main/java/com/example/icroqueta/database/dto/`: los objetos usados como auxiliares para la transferencia de datos.
- `/app/src/main/java/com/example/icroqueta/database/entidades/`: los objetos relacionados con la base de datos.
- `/app/src/main/java/com/example/icroqueta/database/tablas/`: las clases de creación y tratamiento de tablas.
- `/app/src/main/java/com/example/icroqueta/database/utills/`: las clases axiliares que servirán de apoyo en diversas ocasiones.
- `/app/src/androidTest/`: Android UI test.
- `/doc/javadoc/`: documentación *javadoc*.
- `/doc/manuales/`: manual operativo y manual del usuario.

### 4.1. Paquete icroqueta

Dentro de este paquete están las clases explicadas en detalle a continuación.

- |   |   |
|---|---|
| ✓ <code>ActiveProductActivity.java</code> | ✓ <code>MenuBar.java</code>                 |
| ✓ <code>HistoryActivity.java</code>       | ✓ <code>OptionActivity.java</code>          |
| ✓ <code>LineaActivity.java</code>         | ✓ <code>ProductActivity.java</code>         |
| ✓ <code>LoginActivity.java</code>         | ✓ <code>RegisterActivity.java</code>        |
| ✓ <code>MainActivity.java</code>          | ✓ <code>RegisterPaymentActivity.java</code> |
| ✓ <code>MapsActiviy.java</code>           | ✓ <code>ShoppingCarActivity.java</code>     |

#### 4.1.1. `ActiveProductActivity.java`

En esta clase está todo lo referido a los pedidos que tenga el usuario que estén activos, es decir, que no se hayan ni cancelado ni entregado.

Si no hay pedidos nos notificara con un Toast de que no hay nada. En el caso de que haya pedidos, entonces esta clase hará uso de un adaptador para hacer uso del RecyclerView, es decir, reutilizará un fragmento como pedidos activos tenga el usuario.

#### 4.1.2. HistoryActivity.java

En esta clase está todo lo referido a los pedidos que tenga el usuario que se hayan ni cancelado o entregado.

Si no hay pedidos archivados nos notificara con un Toast de que no hay nada. En el caso de que haya pedidos, entonces esta clase hará uso de un adaptador para hacer uso del RecyclerView, es decir, reutilizará un fragmento como productos activos tenga el usuario.

#### 4.1.3. LineaActivity.java

En esta activity se recogen las líneas que tenga un pedido y las muestra en pantalla para que el usuario pueda ver en detalle el producto o añadir todo al carro para repetir el mismo pedido.

Esta clase también hace uso del RecyclerView, reutilizará un fragmento tantos productos hubiese en el pedido.

#### 4.1.4. LoginActivity.java

En esta clase tendremos la pantalla de inicio de sesión del usuario, donde se comprobará a través de la base de datos si existe el usuario registrado con un correo y su contraseña.

#### 4.1.5. MainActivity.jav

Es la pantalla principal, en ella se mostrarán los productos disponibles y se podrán añadir al carro con tan solo un clic en los botones de + o – para ajustar la cantidad.

También tendremos un menú lateral con todos los ingredientes y preferencias para actualizar los productos mostrados.

#### 4.1.6. MapsActivity.java

Esta es la pantalla donde aquellos usuarios con el rol de Repartidor, tendrán acceso a todos los pedidos que aún no hayan sido entregados por el momento.

Desde aquí se podrá confirmar la entrega y visualizar en el mapa dónde se encuentran estos clientes.

#### 4.1.7. MenuBar.java

Es la clase padre de la demás activities, ya que todas tienen el mismo menú, es más eficiente que todas hereden esta característica a que se repita el mismo código en todas las pantallas.

Aquí están los métodos que hacen que el menú funcione y se muestre.

#### 4.1.8. OptionActivity.java

Actividad de ajustes, se podrán modificar los siguientes valores: Nombre, apellido, NIF, teléfono, dirección.

Al pulsar en el botón Guardar, se realizará la verificación de los distintos datos.

Desde aquí también tendrá la opción de eliminar la cuenta, para el cual necesitará que el usuario pulse dos veces seguidas para poder llevar acabo esta acción.

#### 4.1.9. ProductActivity.java

Pantalla que muestra toda la información de un producto en detalle, tanto su nombre, una descripción y precio por unidad. Desde aquí se pueden añadir al carro con los botones + o – y se puede finalizar la compra, lo que nos derivará a ShoppingCarActivity.java para ello.

#### 4.1.10. RegisterActivity.java

La pantalla de registro de los usuarios, donde podrán sus datos personales y una vez finalizado les llevará directamente a LoginActivity.java.

#### 4.1.11. RegisterPaymentActivity.java

Cuando se finaliza la compra les deriva a esta pantalla con los datos requeridos para realizar el pago del pedido. Cuando se finaliza con éxito se deriva a MainActivity.java.

#### 4.1.12. ShoppingCarActivity.java

Es el carrito del usuario donde tendrá todos los productos agregados y desde donde podrá finalizar la compra. Esto hará que le lleve a RegisterPaymentActivity.java.

Desde esta pantalla también podrá limpiar todo el carro o eliminar los productos que no quiere.

Para cargar los productos se hace uso de una RecyclerView que va cargando todos los productos que tenga el usuario en su carrito.

### 4.2. Paquete icroqueta/adapters

Dentro de este paquete están las clases explicadas en detalle a continuación.

- ✓ IngredientRecyclerViewAdapter.java
- ✓ LineRecyclerViewAdapter.java
- ✓ OrderRecyclerViewAdapter.java
- ✓ ProductRecyclerViewAdapter.java

#### 4.2.1. IngredientRecyclerViewAdapter.java

Esta es la clase que se utilizará para mostrar todos los ingredientes, agrupados por tipos de alimentos, en el menú lateral del MainActivity. Uniendo esta información con el `androidx.recyclerview.widget.RecyclerView` y lo una con un layout, que en nuestro caso es un `CardLayout`.

#### 4.2.2. LineRecyclerViewAdapter.java

En esta clase nos sirve para la aplicación reutilice un número finito (cantidad de líneas que tiene un pedido) nuestro `androidx.recyclerview.widget.RecyclerView` y lo una con un layout, que en nuestro caso es un `CardLayout`.

#### 4.2.3. OrderRecyclerViewAdapter.java

En esta clase nos sirve para la aplicación reutilice un número finito (cantidad de pedidos que tiene un usuario) nuestro `androidx.recyclerview.widget.RecyclerView` y lo una con un layout, que en nuestro caso es un `CardLayout`.

#### 4.2.4. ProductRecyclerViewAdapter.java

En esta clase nos sirve para la aplicación reutilice un número finito (cantidad de productos) nuestro `androidx.recyclerview.widget.RecyclerView` y lo una con un layout, que en nuestro caso es un `CardLayout`.

### 4.3. Paquete `icroqueta/database`





















































En este paquete tres carpetas con de entidades, tablas y data transfer object (dto), que se explicarán más adelante. Aquí tenemos únicamente dos clases:

- ✓ DBHelper
- ✓ DBSource

#### 4.3.1. DBHelper

Aquí se encuentran recogidos todos los métodos que hace uso de base de datos. Como consultar, modificar, borrar o añadir. Para buscar todos los métodos de una tabla concreta hay que buscar “Métodos tabla *nombre\_tabla*”

## DBHelper

- m  allProductosCarrito(Context, int): List<ProductoCarrito>
- m  allProductosCarritoByld(Context, int, List<String>): List<ProductoCarrito>
- m  findProductosInCarrito(Context, int): List<ProductoCarrito>
- m  totalProductosEnCarrito(Context, int): double
- m  getProductoCarrito(Context, int, int): ProductoCarrito
- m  allProducto(Context): List<ProductoCarrito>
- m  allProductoByld(Context, List<String>): List<ProductoCarrito>
- m  oneProducto(Context, int): Producto
- m  notExistCarritoProducto(Context, int, int): boolean
- m  deleteCarritoProducto(Context, int, int): void
- m  allCarritoPersona(Context, int): List<Carrito>
- m  addCarrito(Context, int, int, int): void
- m  updateCarrito(Context, int, int, int): void
- m  deleteCarrito(Context, int): void
- m  findProducto(Context, int): Producto
- m  addPersona(Context, String, String, String, String, String): boolean
- m  deletePersona(Context, int): void
- m  findPersonaLogin(Context, String, String): int
- m  findPersonald(Context, int): Persona
- m  isRolPersona(Context, int): int
- m  notExistCorreo(Context, int, String): boolean
- m  updateNombrePersona(Context, int, String): void
- m  updateApellidoPersona(Context, int, String): void
- m  updateNifPersona(Context, int, String): void
- m  updateCorreoPersona(Context, int, String): void
- m  updateContrasenyaPersona(Context, int, String): void
- m  addLinea(Context, int, int): void
- m  allLineasProducto(Context, int): List<Linea>
- m  addPedido(Context, int, String, String, String): boolean
- m  allPedidos(Context): List<Pedido>
- m  allPedidosActivos(Context): List<Pedido>
- m  allPedidosNoActivosUsuario(Context, int): List<Pedido>
- m  allPedidosActivosUsuario(Context, int): List<Pedido>
- m  updatePedido(Context, Integer, Integer, String, String, String, String, ...): void
- m  allIngredientes(Context): List<Ingrediente>
- m  allTipoIngredientes(Context): List<TipoIngrediente>
- m  TiposDelIngredientesDeTipo(Context, int): List<TipoIngrediente>
- m  allIngredientesDeUnTipo(Context, List<TipoIngrediente>): List<Ingrediente>
- m  idProductosIdIngredientes(Context, List<String>): List<String>
- m  idProductosIdIngredientesIsEmpty(Context, int): boolean
- m  updateProducto(Context, Integer, Integer): void
- m  oneTipo(Context, int): Tipo
- m  oneIngrediente(Context, int): Ingrediente
- m  addPersonaTelefono(Context, int, int): void
- m  oneTelefonoByld(Context, int): Telefono
- m  oneTelefonoByNum(Context, int): Telefono
- m  addPersonaDireccion(Context, int, String, String, String, String, String, ...): void
- m  oneDireccionByld(Context, int): Direccion
- m  oneDireccionByCoord(Context, String): Direccion
- m  addPersonaTarjeta(Context, int, String, String): void
- m  oneTarjetaByld(Context, int): Tarjeta
- m  oneTarjetaByNum(Context, String): Tarjeta

### 4.3.2. DBSource

Esta clase es donde se crea la base de datos, hereda de SQLiteOpenHelper y aquí tenemos los métodos que se corresponden con la creación de la base de datos, cómo queremos que actúe según aumentemos la versión de la BBDD o la disminuyamos.

## 4.4. Paquete icroqueta/database/dto

En este paquete se guardan los data transfer object, que son objetos auxiliares para la realización de consultas a tablas a través de Joins. En este paquete solo tenemos la clase ProductoCarrito.java

### 4.4.1. ProductoCarrito.java

Une los atributos de la clase Producto y Carrito en una sola para poder obtener toda la información de un producto que esté en el carrito del usuario.

Esta clase es necesaria para poder pasar información de las consultas que requieran un JOIN entre la tabla Producto y Carrito.

## 4.5. Paquete icroqueta/database/entidades

Son todas los objetos que vamos a utilizar para tratar la base de datos.

- |                            |                        |
|----------------------------|------------------------|
| ✓ Carrito.java             | ✓ PersonaTarjeta.java  |
| ✓ Direccion.java           | ✓ PersonaTeléfono.java |
| ✓ Ingrediente.java         | ✓ Producto.java        |
| ✓ IngredienteProducto.java | ✓ Tarjeta.java         |
| ✓ Linea.java               | ✓ Teléfono.java        |
| ✓ Pedido.java              | ✓ Tipo.java            |
| ✓ Persona.java             | ✓ TipoIngrediente.java |
| ✓ PersonaDirección.java    |                        |

Todas las clases tiene los atributos expuestos en el punto [3.1 Tablas](#), excepto Carrito.java que es un objeto auxiliar para guardar los datos de la tabla auxiliar con el mismo nombre, para guardar en la base de datos los productos que haya añadido el usuario.

En todas las clases tiene dos métodos muy importantes para el tratamiento de la base de datos que son:

- ❖ `mapearAContenValues()`: sirve para meter valores en cada columna de la tabla, es decir que sirve para escribir la información de la tabla.
- ❖ `loadFromCursor (Cursor cursor)`: sirve para cargar las columnas que ha recogido una consulta en la base de datos y las carga dentro de los atributos de la clase para tener la información obtenida en objetos.

## 4.6. Paquete icroqueta/database/tablas

Todas estas clases son las que se utilizan para la creación de la base de datos con su método para ejecutar crear o eliminar toda la tabla.

- ✓ CarritoTable.Java
- ✓ DirecciónTabla.java
- ✓ IngredienteProductoTable.Java
- ✓ IngredienteTabla.java
- ✓ Linea Tabla.java
- ✓ PedidoTabla.java
- ✓ PersonaDirecciónTabla.java
- ✓ PersonaTabla.java
- ✓ PersonaTarjetaTabla.java
- ✓ PersonaTeléfonoTabla.java
- ✓ ProductoTabla.java
- ✓ TarjetaTabla.java
- ✓ TeléfonoTabla.java
- ✓ TipoIngredienteTabla.java
- ✓ TipoTabla.java

## 4.7. Paquete icroqueta/database/utils

Aquí van las clases que sirven de apoyo para otras.

Y estos son los archivos que nos encontramos dentro:

- ✓ LocalizadorDirecciones.java
- ✓ ValidadorDNI.java

### 4.7.1. LocalizadorDirecciones.java

Con esta clase podemos sacar las coordenadas de una dirección introducida y la devuelve en coordenadas.

### 4.7.2. ValidadorDNI.java

Esta clase valida un DNI, tanto la longitud, como si el último dígito de este es correcto.