

**DIGITAL RECIPES**

**Israel Lucas Torrijos Digital Recipes**

**DAM Grupo del profesor D. Francisco J. Pulido Moya**

DESARROLLO DE APLICACIONES MULTIPLATAFORMA

Tabla de contenido

[Proyecto intermodular 3](#_Toc211437298)

[Capítulo 1. Introducción y objetivos 3](#_Toc211437299)

[Capítulo 2. Especificación de Requisitos 3](#_Toc211437300)

[Capítulo 3. Planificación Temporal y Evaluación de Costes 3](#_Toc211437301)

[Capítulo 4. Tecnologías Utilizadas 3](#_Toc211437302)

[Capítulo 5. Desarrollo e Implementación 3](#_Toc211437303)

[5.1. Modelo de datos 3](#_Toc211437309)

[5.2. Interfaces DAO 6](#_Toc211437310)

[Capítulo 6. Conclusiones y líneas futuras 6](#_Toc211437311)

[Capítulo 7. Bibliografía. 6](#_Toc211437312)

[7.1. Páginas web 6](#_Toc211437315)

# Proyecto intermodular

## Capítulo 1. Introducción y objetivos

## Capítulo 2. Especificación de Requisitos

## Capítulo 3. Planificación Temporal y Evaluación de Costes

## Capítulo 4. Tecnologías Utilizadas

## Capítulo 5. Desarrollo e Implementación



### Modelo de datos

La etapa inicial del proyecto aborda la definición del modelo de datos en **Room**. Mediante el plugin **PlantUML** se ha generado el diagrama **Entidad/Relación** que define la estructura de la base de datos:



La **API** de **Room** es una capa de abstracción para acceder a la información de las bases de datos **SQLite**. En Android, tenemos que definir clases **Java** que se correspondan con las entidades/tablas en **Room** mediante la anotación @Entity.

Por eso tengo que crear tres clases java que representan **entidades**:

* **Receta**. Representa la tabla *Recetas*. Incluye un campo autonumérico **idReceta** como clave primaria. El resto de campos permiten almacenar el título, la descripción, la imagen asociada y el tiempo de preparación.

@Entity(tableName = Constantes.*TABLA\_RECETAS*)  
public class Receta {  
  
 @PrimaryKey(autoGenerate = true)  
 private int idReceta;  
  
 @ColumnInfo(defaultValue = "Sin título")  
 private String titulo;  
  
 private String descripcion;  
  
 @ColumnInfo(name = "imagen")  
 private String imagenUri;  
  
 private int tiempo;  
 ...

* **Ingrediente.** Representa la tabla *Ingredientes*. Contiene un campo autonumérico **idIngrediente** como clave primaria. La propiedad **idReceta** se mapea a la columna idReceta\_fk mediante la anotación @ColumnInfo. Lo que implica declarar la clave foránea en la anotación @Entity para garantizar la integridad referencial.

@Entity(tableName = Constantes.*TABLA\_INGREDIENTES*,  
 foreignKeys = {@ForeignKey(  
 entity = Receta.class,  
 parentColumns = "idReceta",  
 childColumns = "idReceta\_fk",  
 onDelete = ForeignKey.*CASCADE*,  
 onUpdate = ForeignKey.*CASCADE*)  
})  
public class Ingrediente {  
  
 @PrimaryKey(autoGenerate = true)  
 private int idIngrediente;  
  
 private String nombre;  
  
 @ColumnInfo(defaultValue = "0")  
 private double cantidad;  
  
 private String unidad;  
  
 @NonNull  
 @ColumnInfo(name = "idReceta\_fk", index = true)  
 private int idReceta;   
 ...

* **Paso**. Representa la tabla *Pasos*. Incluye el campo autonumérico **idPaso** como clave primaria. La propiedad **idReceta** se mapea a la columna idReceta\_fk mediante la anotación @ColumnInfo como en la clase anterior.

@Entity(tableName = Constantes.*TABLA\_PASOS*,  
 foreignKeys = {@ForeignKey(  
 entity = Receta.class,  
 parentColumns = "idReceta",  
 childColumns = "idReceta\_fk",  
 onDelete = ForeignKey.*CASCADE*,  
 onUpdate = ForeignKey.*CASCADE*)  
 })  
public class Paso {  
  
 @PrimaryKey(autoGenerate = true)  
 private int idPaso;  
  
 private int orden;  
 private String descripcion;  
  
 @ColumnInfo(name = "idReceta\_fk")  
 private int idReceta;   
 ...

Para consultar los datos entre **dos entidades** con una relación **1:N**, es necesario **modelar** dicha **relación** mediante una clase Java específica.

El modelado implica crear una clase que contenga una instancia de la clase padre anotada con @Embedded y otra instancia de la clase hija anotada con @Relation. Asignamos a parentColumn el nombre de la clave primaria de la entidad fuerte y a entityColumn el nombre de la entidad débil que hace referencia a la clave primaria de la entidad fuerte.

A tal efecto, se crean las siguientes clases Java que representan las **relaciones**:

* **RecetaIngredientes**. Permite realizar operaciones **CRUD** sobre los ingredientes asociados a una receta.

public class RecetaIngredientes {  
 @Embedded  
 public Receta receta; *// Es la entidad padre* @Relation(  
 parentColumn = "idReceta",  
 entityColumn = "idReceta\_fk" )  
  
public List<Ingrediente> ingredientes;  
}

* **RecetaPasos.** Permite realizar operaciones **CRUD** sobre los pasos vinculados a una receta.

public class RecetaPasos {  
 @Embedded  
 public Receta receta; *// Es la entidad padre* @Relation(  
 parentColumn = "idReceta",  
 entityColumn = "idReceta\_fk" )  
public List<Paso> pasos;  
}

* **RecetaCompleta.** Permite realizar operaciones **CRUD** con los datos completos de la receta.

public class RecetaCompleta {  
  
 @Embedded  
 public Receta receta;  
  
 @Relation(  
 parentColumn = "idReceta",  
 entityColumn = "idReceta\_fk"  
 )  
 public List<Ingrediente> ingredientes;  
  
 @Relation(  
 parentColumn = "idReceta",  
 entityColumn = "idReceta\_fk"  
 )  
 public List<Paso> pasos;  
}

### Interfaces DAO

Las interfaces **DAO** son las responsables de definir los métodos de acceso a la base de datos. En **SQLite** empleamos los objetos **Cursor**. Con la **API Room**, no necesitamos todo el código relacionado con **Cursor**, y simplemente definimos nuestras consultas usando anotaciones en la clase **DAO**.

Definimos tantos métodos como sean necesarios para abarcar las operaciones **CRUD**. Cada método tiene aparejado una anotación con la operación que realiza: @Insert, @Update y @Delete. Estos métodos pueden retornar un valor númerico referente al **rowId** afectado.

Las consultas emplean la anotación @Query que declara la consulta **SQL** asociada, además para implementar correctamente el patrón **MVVM** los valores que retornan deben estar encapsulados con el tipo **LiveData**.

Los métodos que retornan instancias de clases que **modelan una relación** deben anotarse con @Transaction, ya que los campos anotados con @Relation se consultan por separado.

### Base de datos

### Repositorio

## Capítulo 6. Conclusiones y líneas futuras

## Capítulo 7. Bibliografía.



### Páginas web

* <https://developer.android.com/training/data-storage/room?hl=es-419>  
  [14/10/2025]