DEVCHALLENGE 2

# 1. introducción

El proyecto “Llista de compra compartida” se desarrolla en el marco del *DevChallenge 2* y tiene como finalidad la creación de una aplicación web colaborativa que permita a los usuarios gestionar de manera eficiente sus listas de compra.  
La aplicación busca facilitar la organización de productos y categorías dentro de cada lista, al mismo tiempo que ofrece la posibilidad de compartirlas con otros usuarios, de forma que varios participantes puedan visualizar, editar y marcar productos como completados en tiempo real.

Cada usuario dispone de un perfil personal y puede crear múltiples listas: tanto las que haya generado él mismo como aquellas que le hayan sido compartidas por otros. Para acceder a la aplicación, se implementan al menos dos métodos de autenticación diferentes (por ejemplo, registro propio y acceso mediante una cuenta externa como Google o GitHub), garantizando así flexibilidad y seguridad.

Desde el punto de vista técnico, el proyecto incluye la gestión de usuarios, listas, categorías y productos mediante una base de datos relacional. Además, se ha añadido una funcionalidad extra denominada “comentarios” un usuario puede dejar comentarios en la lista de los demás y también en la suya.

Modificar línea de abajo dependiendo de lo que utilizemos alfinal:

En este proyecto se han usado las siguientes herramientas: Github, Trello, Laravel, PHP, SQLite, Drawio, VSCode, Tailwind.

# 2. ORGANIZACIÓN

Nos hemos organizado de varias formas, pero principalmente ha sido a través de Trello y GitHub. En Trello lo que hacíamos era crear tarjetas que actuaban como tareas diarias y con etiquetas íbamos teniendo constancia de cómo iba esa parte del proyecto en concreto. También es cierto que, si era algo muy necesario de saber, nos enviamos un WhatsApp como, por ejemplo: “Oye, Juan ya he terminado esta parte y he hecho commit, nada más para informar.”

Por otro lado, GitHub lo utilizábamos como herramienta de organización para código. Es decir, usamos Trello para organización de tareas y tener constancia de cómo van las tareas y GitHub para organizarnos con el código y tener en cuenta que vamos haciendo en el código.

# 3. BASE DE DATOS

La base de datos utilizada en el proyecto sigue un modelo relacional que permite gestionar de manera estructurada los usuarios, listas, categorías, productos, y comentarios, además de las relaciones N:N necesarias para compartir listas entre usuarios.

## 3.1. Diseño conceptual (modelo E/R)

El modelo entidad–relación define las entidades principales del sistema:

* **USER**: representa a cada usuario registrado.
* **LIST**: define una lista de compra creada por un usuario.
* **CATEGORY**: agrupa productos de una lista o de varias listas.
* **PRODUCT**: elemento que puede marcarse como completado y pertenece obligatoriamente a una categoría.
* **COMMENT**: permite a los usuarios dejar mensajes dentro de las listas.
* **LIST\_USERS**, **LIST\_CATEGORIES** y **LIST\_PRODUCTS**: tablas intermedias que gestionan las relaciones N:N (usuarios ↔ listas, listas ↔ categorías y listas ↔ productos).

***Enlace de descarga del modelo (Draw\_Io):***[*Diagram\_DevChallenge\_2*](../diagram/DevChallenge2.drawio)

## 3.2. Diseño lógico (modelo relacional)

En el modelo relacional cada entidad del diagrama E/R se convierte en una tabla con sus claves primarias (PK), foráneas (FK), atributos y reglas de integridad.

**USERS**

* Id\_user (PK), name, lastname, email (unique), phone, address, password, remember\_token, created\_at, updated\_at.
* Cada usuario puede tener varias listas y comentarios.

**LISTS**

* Id\_list (PK), user\_id (FK→user.id), name, created\_at, updated\_at.
* Un usuario puede tener muchas listas; una lista pertenece a un usuario.

**LIST\_USERS**

* Id\_list\_users (PK), id\_list (FK→lists.id), id\_user (FK→users.id), role, created\_at, updated\_at.
* Representa la relación N:M entre usuarios y listas.

**CATEGORIES**

* Id\_category (PK), name, created\_at, updated\_at.
* Asociadas a las listas mediante LIST\_CATEGORIES.

**LIST\_CATEGORIES**

* Id\_list\_categories (PK), id\_list (FK→lists.id), id\_category (FK→categories.id), created\_at, updated\_at.
* Relación N:M entre listas y categorías.

**PRODUCTS**

* Id\_product (PK), id\_category (FK→categories.id, NOT NULL), name, completed (INTEGER DEFAULT 0), created\_at, updated\_at.
* Cada producto pertenece a una categoría y puede estar asociado a varias listas mediante LIST\_PRODUCTS.

**LIST\_PRODUCTS**

* Id\_list\_products (PK), id\_list (FK→lists.id), id\_product (FK→products.id), created\_at, updated\_at.
* Relación N:M entre listas y productos.

**COMMENTS**

* Id\_comments (PK), id\_list (FK→lists.id), id\_user (FK→users.id), content, created\_at, updated\_at.
* Un usuario puede escribir varios comentarios en una lista.

## 3.3. Reglas de integridad

* **ON DELETE CASCADE** en todas las relaciones principales (listas, productos, comentarios) para mantener consistencia.
* **ON DELETE CASCADE** Se ha configurado la relación entre products y categories con ON DELETE CASCADE, de modo que, al eliminar una categoría, se eliminan automáticamente todos los productos asociados a ella.
* Atributos booleanos representados como INTEGER (0/1) en SQLite.
* Timestamps gestionados automáticamente por Laravel.

## 3.4. Consideraciones

* Las tablas pivot (list\_users, list\_categories, list\_products) garantizan flexibilidad y escalabilidad.
* Se evita la duplicidad con índices únicos definidos en Laravel (UNIQUE (id\_list, id\_user) y similares).
* La estructura cumple el enunciado del DevChallenge2: manejo de múltiples usuarios, listas compartidas, categorías, productos y comentarios.