Diccionario de Datos — Prueba Técnica Flutter

Este diccionario de datos describe las entidades, atributos y estructuras utilizadas dentro del proyecto, tanto a nivel de **persistencia local (SQLite)** como en la capa de **consumo de API** y **gestión de estado**. Su objetivo es brindar claridad sobre cómo se modelan, almacenan y procesan los datos en la aplicación.

🗄️ 1. Entidad: LocalImageEntity

Representa una imagen guardada por el usuario de forma local, proveniente de la API pública.

Campo	Tipo	Descripción
id	`String`	Identificador único de la imagen (tomado desde la API). Se utiliza como PRIMARY KEY en la base local.
author	`String`	Nombre del autor de la imagen obtenida desde el API de Picsum.
downloadUrl	`String`	URL directa para obtener la imagen. También permite mostrar un fallback si no hay conexión.
customName	`String`	Nombre asignado por el usuario. Editable mediante CRUD local.

Observaciones

- Esta entidad es persistida en SQLite.
- Incluye validación mínima (customName >= 2 caracteres).
- Utilizada por `LocalImagesCubit` para CRUD completo.

🗄️ 2. Tabla SQLite: saved_images

Base de datos local utilizada para almacenar los favoritos del usuario.

Columna	Tipo SQL	Detalles
id	TEXT	Clave primaria. Corresponde al ID de Picsum.
author	TEXT	Nombre del autor. No editable.
download_url	TEXT	URL de descarga.
custom_name	TEXT	Nombre personalizado asignado por el usuario.

Reglas importantes

- Maneja **migración automática** para agregar columnas faltantes.
- Permite búsquedas por autor, nombre personalizado y URL.
- Las operaciones CRUD se realizan mediante `LocalImagesRepositoryImpl`.

🌐 3. Datos provenientes de la API (Picsum Photos)

Cada elemento recibido del endpoint JSON contiene los siguientes campos:

Campo	Tipo	Descripción
id	`String/num`	ID único generado por Picsum. Mapeado como String para consistencia.
author	`String`	Nombre del fotógrafo.
url	`String`	URL de referencia (página).
download_url	`String`	Imagen escalable utilizada en la app.

Procesamiento

- Se mapea a un `Map<String, dynamic>` en el repositorio.
- Solo se almacenan en local los datos del usuario que seleccione “Guardar”.

📦 4. Estados manejados por Cubits

Los datos también se representan mediante estados bien definidos para controlar la UI.

ApiCubit (Consumo de API)

Estado	Descripción
`ApiInitial`	Estado inicial.
`ApiLoading`	Cargando datos desde la API.
`ApiLoaded(items)`	Datos obtenidos correctamente.
`ApiError(message)`	Fallo en la llamada o parseo.
`ApiNoConnection(message)`	Sin internet. Evita llamadas innecesarias.

LocalImagesCubit (CRUD Local)

Estado	Descripción
`LocalImagesInitial`	Estado inicial.
`LocalImagesLoading`	Operación en curso (cargar, guardar, eliminar...).
`LocalImagesLoaded(items)`	Lista cargada desde SQLite.
`LocalImageSaved`	Un elemento fue guardado exitosamente.
`LocalImagesError(message)`	Error en cualquier operación local.
`LocalImageUpdated`	Nombre actualizado correctamente.
`LocalImageDeleted`	Eliminación exitosa.

🧱 5. Estructuras de Navegación (Routing)

Controladas desde `routes.dart`:

Ruta	Pantalla	Descripción
`/api-list`	**ApiListScreen**	Lista de imágenes desde la API.
`/prefs`	**PrefsListScreen**	Lista local guardada.
`/prefs/detail`	**PrefsDetailScreen**	Detalle + edición + eliminación.
`/home`	**HomeScreen**	Menú principal.
`/splash`	**SplashScreen**	Pantalla inicial con animación.

🧩 6. Data Flow (Resumen conceptual)

1. **API Client** → obtiene lista JSON con Dio
2. **ItemsRepositoryImpl** → adapta respuesta y entrega items limpios

3. **ApiCubit** → gestion de estado + validación de conectividad
4. **UI** (ApiListScreen)
    - Muestra datos
    - Permite guardar en SQLite
5. **LocalImagesRepositoryImpl** → CRUD con sqflite
6. **LocalImagesCubit** → emite estados a la UI
7. **Screens y Widgets** → Representan datos con diseño responsive

---

📝 7. Diccionario de Funciones Relevantes

`fetchApiItems()`

- Valida conectividad
- Llama al repositorio
- Maneja estados API Loading / Loaded / Error / NoConnection

`saveImage(LocalImageEntity entity)`

- Inserta/actualiza en SQLite
- Emite `LocalImageSaved`

`updateImageName(id, newName)`

- Actualiza el nombre personalizado

`deleteImage(id)`

- Elimina registro por ID

`queryImagesByText(q)`

- Búsqueda SQL por coincidencia parcial

📚 8. Notas Técnicas Destacables

- Se usan **MediaQuery y porcentajes** para responsive design.
- Imágenes usan fallback si no hay red.

- Migraciones de SQLite están automatizadas para evitar fallos de columnas.
- El proyecto sigue estructura real de producción.
- Estados están correctamente separados y documentados.

✔️ Conclusión

Este diccionario de datos refleja una arquitectura clara, modular y profesional, mostrando dominio en:

- Persistencia local con SQLite
- Modelado de entidades
- Gestión de estado con Cubit
- Consumo de API
- Diseño responsive
- Buenas prácticas de arquitectura en Flutter