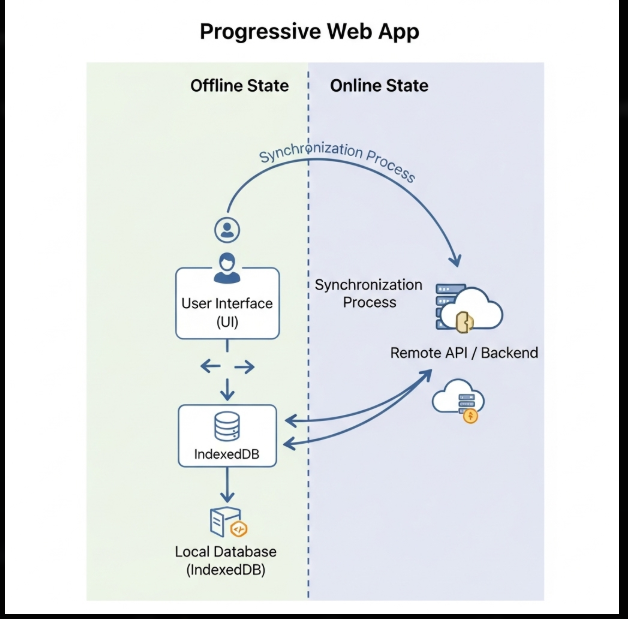
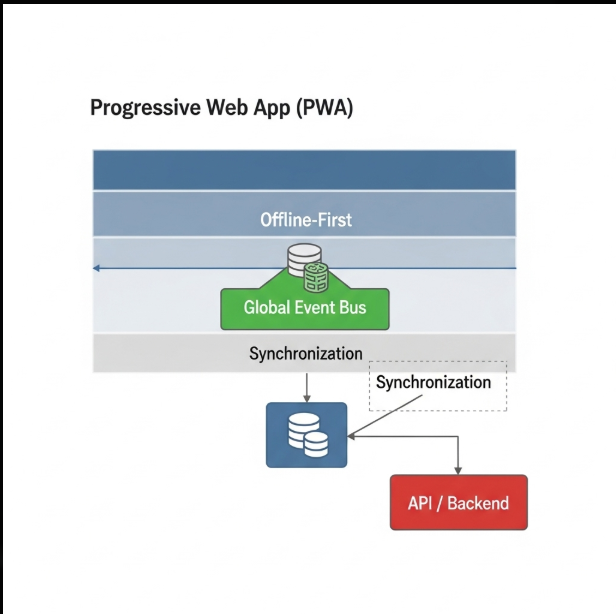
**ARQUITECTURA PWA**

****

**Explicación paso a paso**

1. **Usuario interactúa con la app**
   * Crea, modifica o elimina un recurso (usuario, grupo, tarea, etc.)
   * Todos los cambios se **guardan en IndexedDB** a través del **Repository**.
2. **Repository**
   * Encapsula operaciones CRUD locales.
   * Devuelve datos a la UI y también marca el syncStatus como pending o deleted.
3. **SyncSensor**
   * Monitorea el estado de sincronización de cada entidad.
   * Dispara eventos (itemSynced, itemFailed, itemDeleted) que actualizan la UI o alertas.
4. **NetworkSensor**
   * Detecta cuando el backend está online.
   * Activa el **Worker General**, que decide qué sincronizaciones deben correr.
5. **Worker específico**
   * Por ejemplo: userSensor, groupSensor, taskSensor.
   * Toma elementos pending o deleted y hace los requests correspondientes a la **API**.
   * Actualiza IndexedDB con la versión “synced”.
6. **API Backend**
   * Responde con los datos finales o IDs del backend.
   * Si es un nuevo registro offline, devuelve el id real del servidor.
7. **Sync Switch / Eventos**
   * Permite que la UI y otras partes de la app reaccionen al estado de sincronización.
   * Por ejemplo: marcar un usuario como “sincronizado” o mostrar error si falla.



**Explicación de la Arquitectura de Sincronización de la PWA**

El sistema utiliza una arquitectura **Offline-First** robusta, que prioriza la interacción local y garantiza una experiencia de usuario fluida, independientemente del estado de la conexión a internet. La sincronización con el servidor se maneja de forma asíncrona y transparente para el usuario.

A continuación, se detalla el flujo de trabajo paso a paso:

**1. Interacción del Usuario**

El usuario interactúa con la interfaz de la aplicación para crear, modificar o eliminar cualquier recurso (como un usuario, un grupo o una tarea).

**2. Capa de Repositorio (Offline-First)**

Todo cambio iniciado por el usuario es capturado por un **Repository** (TaskRepository, GroupRepository, etc.). El repositorio es la única capa que interactúa directamente con la base de datos local **IndexedDB**. Su función principal es:

* **Guardar cambios:** Almacena la operación localmente.
* **Marcar estado de sincronización:** Asigna un syncStatus al recurso, que puede ser pending (para creaciones o actualizaciones) o deleted (para eliminaciones).

**3. Control de Sincronización**

La lógica de sincronización se controla mediante dos componentes clave:

* **NetworkSensor**: Monitorea el estado de la conexión. Detecta cuándo la aplicación pasa de estar sin conexión a tener una conexión estable con el backend. Una vez que el backend está en línea, el sensor emite un evento server-online.
* **SyncSwitch / Eventos**: Permite a la interfaz de usuario habilitar o deshabilitar la sincronización. Esto se logra a través de un **Event Bus global** (appEvents), que permite a otros componentes (como el Worker específico) reaccionar al estado de la sincronización.

**4. Sincronización Asíncrona (Worker Específico)**

El evento server-online disparado por el NetworkSensor activa un **Worker específico** (syncTaskWorker, syncUserWorker, etc.). Este worker es el encargado de la sincronización.

1. **Consulta el repositorio:** Obtiene todos los recursos con un syncStatus de pending o deleted.
2. **Envía peticiones a la API:** Para cada recurso, realiza la petición HTTP apropiada (POST, PUT o DELETE) al **Backend API**.
3. **Actualiza IndexedDB:** Si la petición a la API es exitosa, el worker actualiza el recurso en la base de datos local:
   * Cambia su syncStatus a synced.
   * Si era una creación, actualiza el tempId con el id real del servidor.
   * Si era una eliminación, borra el registro de la base de datos.
4. **Emite eventos de sincronización:** El SyncSensor emite eventos (item-synced, item-deleted, etc.) para notificar a la aplicación que una operación de sincronización ha finalizado.

**5. Backend API**

El backend es la **fuente de verdad**. Su única responsabilidad es procesar las peticiones del worker y, en el caso de nuevas creaciones, devolver el ID permanente que se usará para futuras sincronizaciones.

**6. Notificación a la UI**

Finalmente, los eventos emitidos por el SyncSensor y el **Event Bus global** son consumidos por los componentes de la interfaz de usuario, permitiéndoles reaccionar en tiempo real, por ejemplo, mostrando una notificación de "Sincronizado" o un mensaje de error si la sincronización falla.