**1) Industry 4.0 — Telemetría de máquinas**

**Colecciones nuevas**

**i40\_stream (snapshot “en vivo” por máquina)**

* **Doc ID**: el id de la máquina (el mismo de machines)
* **Campos**:
  + machineId (string)
  + rpm (number)
  + feed (number)
  + temp (number)
  + parts (number)
  + job (string o int)
  + running (bool)
  + alarms (array<string>)
  + updatedAt (timestamp)

**Ejemplo**

i40\_stream / alHmrAh2XHqKzYHbjkrn

{

machineId: "alHmrAh2XHqKzYHbjkrn",

rpm: 1520,

feed: 245.3,

temp: 34.8,

parts: 57,

job: "CHM-1022-C1842",

running: true,

alarms: ["Coolant low"],

updatedAt: (serverTimestamp)

}

**i40\_history (histórico para gráficas)**

* **Doc ID**: auto
* **Campos**:
  + machineId (string)
  + ts (timestamp)
  + rpm, feed, temp, parts, job, running, alarms (igual que arriba)

Tip: si luego quieres agrupar por día, agrega dayKey: "2025-08-29".

**Reglas (añade a tus reglas actuales)**

// INDUSTRY 4.0

match /i40\_stream/{machineId} {

allow read: if isAuthed(); // todos los autenticados leen

allow write: if isSupervisorOrAdmin(); // escribe sup/admin (o tu simulador)

}

match /i40\_history/{docId} {

allow read: if isAuthed();

allow write: if isSupervisorOrAdmin();

}

Si quieres que el operador también lea/vea el tablero, ya lo cubre read: isAuthed().

**Conexión en Flutter (pantalla Industry40)**

En tu industry40\_screen.dart, si ves job en rojo es porque **no existe la variable**. Vamos a leer **en vivo** de i40\_stream:

// Stream del snapshot en vivo de UNA máquina

Stream<Map<String, dynamic>?> i40Stream(String machineId) {

return FirebaseFirestore.instance

.collection('i40\_stream')

.doc(machineId)

.snapshots()

.map((d) => d.data());

}

// En build():

final machineId = 'alHmrAh2XHqKzYHbjkrn'; // CNC-2 (tu doc de machines)

return StreamBuilder<Map<String, dynamic>?>(

stream: i40Stream(machineId),

builder: (context, snap) {

if (snap.hasError) return Center(child: Text('Error: ${snap.error}'));

if (!snap.hasData) return const Center(child: CircularProgressIndicator());

final d = snap.data!;

final job = (d['job'] ?? '-').toString();

final rpm = (d['rpm'] ?? 0).toString();

final feed = (d['feed'] ?? 0).toString();

final temp = (d['temp'] ?? 0).toString();

final parts = (d['parts'] ?? 0).toString();

final running = (d['running'] ?? false) as bool;

final alarms = List<String>.from(d['alarms'] ?? const []);

return Column(

crossAxisAlignment: CrossAxisAlignment.start,

children: [

Row(children: [

\_chip('Job', job),

\_chip('RPM', rpm),

\_chip('Feed', feed),

\_chip('Temp °C', temp),

\_chip('Parts', parts),

]),

const SizedBox(height: 8),

Text(running ? 'RUN' : 'IDLE'),

if (alarms.isNotEmpty) ...[

const SizedBox(height: 8),

Text('Alarms: ${alarms.join(', ')}', style: const TextStyle(color: Colors.red)),

],

],

);

},

);

Con eso, **las variables dejan de estar rojas** porque las estamos leyendo del stream.

**Simulador local (desde la app, para pruebas)**

Agrega un botón “Simular” que cada 2 s actualice i40\_stream/{machineId} y **append** en i40\_history:

Timer? \_sim;

void startSim(String machineId) {

\_sim?.cancel();

\_sim = Timer.periodic(const Duration(seconds: 2), (\_) async {

final now = DateTime.now();

final data = {

'machineId': machineId,

'rpm': 1400 + Random().nextInt(400),

'feed': 200 + Random().nextDouble() \* 100,

'temp': 30 + Random().nextDouble() \* 8,

'parts': Random().nextInt(200),

'job': 'JOB-${now.minute}${now.second}',

'running': Random().nextBool(),

'alarms': Random().nextBool() ? ['Coolant low'] : [],

'updatedAt': FieldValue.serverTimestamp(),

};

final fs = FirebaseFirestore.instance;

await fs.collection('i40\_stream').doc(machineId).set(data, SetOptions(merge: true));

await fs.collection('i40\_history').add({...data, 'ts': Timestamp.fromDate(now)});

});

}

void stopSim() => \_sim?.cancel();

**2) Gantt — Proyecto → Parte → Operación (plan vs real)**

No necesitas colecciones nuevas todavía. Usa tu production\_daily y añade **dos campos plan**:

* planInicio (timestamp)
* planFin (timestamp)
* (ya tienes inicio y fin como **real**)
* agrega proyectoId, parteId, operacionId (si no los tienes)
* opcional: opSecuencia (para ordenar operaciones de una parte)

**Ejemplo de doc production\_daily**

{

proyecto: "CHM-1022-C1842",

proyectoId: "...",

numeroParte: "CHM-1022-C1842",

parteId: "...",

operacionNombre: "DIBUJO",

operacionId: "...",

planInicio: 2025-08-29T08:00:00Z,

planFin: 2025-08-29T12:00:00Z,

inicio: 2025-08-29T08:10:00Z,

fin: 2025-08-29T12:05:00Z,

cantidad: 10,

status: "programado"

}

**Query por proyecto para el Gantt**

final q = FirebaseFirestore.instance

.collection('production\_daily')

.where('proyectoId', isEqualTo: selectedProjectId)

.orderBy('planInicio'); // crea índice si te lo pide

Luego agrupas en memoria por parteId y pones cada operacion como una barra (2 tracks: Plan y Real). Ya tienes un gantt\_screen.dart; con esto te doy los **datos correctos**.

**3) Warehouse — existencias y movimientos**

**Colecciones nuevas**

**warehouse\_stock (stock actual por parte/bodega)**

* **Doc ID**: <parteId>\_\_<bodegaId>
* **Campos**:
  + parteId, numeroParte, bodegaId, bodegaNombre
  + qty (int) — **existencia actual**
  + updatedAt (timestamp)

**warehouse\_movements (historial de movimientos)**

* **Doc ID**: auto
* **Campos**:
  + parteId, numeroParte, bodegaId, bodegaNombre
  + type (string: "in" o "out")
  + qty (int, >0)
  + ref (opcional: id de orden / comentario)
  + ts (timestamp)

**Ejemplo movimiento**

warehouse\_movements / abc123

{

parteId: "tCjQhAxs...",

numeroParte: "CHM-1022-C1842",

bodegaId: "NQ7KC0sZ...",

bodegaNombre: "Bodega 12",

type: "out",

qty: 3,

ref: "Consumo orden PD-0001",

ts: (serverTimestamp)

}

**Reglas (añade)**

// WAREHOUSE

match /warehouse\_stock/{docId} {

allow read: if isAuthed();

allow write: if isSupervisorOrAdmin();

}

match /warehouse\_movements/{docId} {

allow read: if isAuthed();

allow write: if isSupervisorOrAdmin();

}

**Operación segura de stock (transacción)**

Future<void> addMovement({

required String parteId,

required String numeroParte,

required String bodegaId,

required String bodegaNombre,

required String type, // "in" | "out"

required int qty,

String? ref,

}) async {

assert(qty > 0);

final fs = FirebaseFirestore.instance;

final stockId = '${parteId}\_\_${bodegaId}';

final stockRef = fs.collection('warehouse\_stock').doc(stockId);

final movRef = fs.collection('warehouse\_movements').doc();

await fs.runTransaction((tx) async {

final snap = await tx.get(stockRef);

int current = 0;

if (snap.exists) current = (snap.data()!['qty'] ?? 0) as int;

final delta = type == 'in' ? qty : -qty;

final next = current + delta;

if (next < 0) {

throw Exception('Stock insuficiente para salida');

}

tx.set(stockRef, {

'parteId': parteId,

'numeroParte': numeroParte,

'bodegaId': bodegaId,

'bodegaNombre': bodegaNombre,

'qty': next,

'updatedAt': FieldValue.serverTimestamp(),

}, SetOptions(merge: true));

tx.set(movRef, {

'parteId': parteId,

'numeroParte': numeroParte,

'bodegaId': bodegaId,

'bodegaNombre': bodegaNombre,

'type': type,

'qty': qty,

'ref': ref ?? '',

'ts': FieldValue.serverTimestamp(),

});

});

}

En warehouse\_screen.dart puedes listar warehouse\_stock y, al tocar un ítem, abrir un diálogo para aplicar Entrada o Salida usando addMovement(...).

**Índices que seguramente te pedirá**

* production\_daily: .where('proyectoId', isEqualTo: ...).orderBy('planInicio')
* warehouse\_movements: .where('parteId', isEqualTo: ...).orderBy('ts')
* i40\_history: .where('machineId', isEqualTo: ...).orderBy('ts')

Cuando salga el error con el link, lo abres y le das *“Crear índice”*.

**Qué crear ahora mismo (checklist rápido)**

1. **Crear colecciones vacías**:
   * i40\_stream (un doc por máquina)
   * i40\_history
   * warehouse\_stock
   * warehouse\_movements
2. **Pegar reglas** de arriba y publicar.
3. **Industry40**:
   * Conectar tu pantalla a i40\_stream (código de StreamBuilder).
   * Agregar botón **Simular** (Timer) para ver datos moverse.
4. **Gantt**:
   * Empezar a guardar planInicio y planFin cuando el supervisor crea la orden (o edítalas manual en Firestore para probar).
   * Leer por proyectoId y pintar barras Plan vs Real.
5. **Warehouse**:
   * Añadir en warehouse\_screen.dart la tabla de stock (lee warehouse\_stock) y un botón “Movimiento” que llama addMovement(...).

Con eso ya tienes **todo enlazado** y funcionando con Firestore, sin necesidad de cambiar de base de datos aún. Más adelante, si quieres SQL para BI pesado, lo sincronizamos (no hay prisa).