

This is a copy of a conversation between ChatGPT & Anonymous.

[Report conversation](#)



Como puedo desconectar el flow meter de la valvula?

Perfecto, con la foto se ve claro 

Te explico **exactamente** cómo desconectar el flow meter de la válvula, sin dañarlo.

Identifica las piezas primero

En tu foto se ve:

- **Válvula solenoide** (caja negra con cable)
- **Flow meter** (pieza de acero inoxidable cilíndrica, con una flecha grabada)
- **Conexión roscada** entre ambos (hexagonal)

 El flow meter NO va a presión, va **roscado**.

Pasos para desconectarlo (recomendado)

1 Cierra el flujo

Antes de tocar nada:

- Apaga el sistema
- Cierra CO₂ o líquido
- Si puedes, libera presión abriendo un faucet

Esto es importante para que no salga líquido a presión.

2 Sujeta el cuerpo de la válvula (MUY importante)

Usa **una llave fija o ajustable** y sujetala:

- El **hexágono metálico de la válvula**, NO el cuerpo negro

 No agarres:

- El cuerpo negro de la válvula
 - El cable
- Eso puede romper la válvula internamente.

3 Afloja el flow meter

Con otra llave:

- Agarra el **hexágono del flow meter**
- Gira en sentido contrario a las manecillas del reloj ⌂

 Consejo:

- Son conexiones NPT, a veces vienen bien apretadas
 - Un pequeño “crack” inicial es normal
-

4 Retíralo a mano

Una vez flojo:

- Termina de desenroscarlo con la mano
- Saldrá completo con su rosca

Ten un trapo por si cae un poco de líquido.

Detalles importantes a revisar al sacarlo

Dirección de la flecha

- La flecha indica el sentido del flujo
- Memorízala para reinstalarlo correctamente

Teflón

- Al volver a montar:
 - Limpia la rosca
 - Pon **2–3 vueltas de teflón**
 - No exageres
-

❓ Si no sale

Si está muy duro:

- Un poquito de **calor leve** (secador de pelo, no pistola industrial)
 - Nunca uses fuerza sobre el cuerpo negro
-

Si quieres, dime:

- ¿Lo vas a mover de lugar?
- ¿Eliminarlo temporalmente?
- ¿Cambiarlo por otro flow meter?
- ¿Ponerlo antes o después de la válvula?

Y te digo **la mejor configuración para BeerBot** según lo que quieras hacer 🍺🤖



Quiero desconectarlo de la parte stainless steel abajo

Perfecto, ahora sí está claro 

Esa parte stainless steel de abajo NO se desenrosca del flow meter.

Lo que tienes ahí es una **conexión tipo push-to-connect (John Guest style)**.

Voy directo y claro:

Cómo desconectarlo del stainless steel (abajo)

Identificación

- El **flow meter** es la pieza redonda (Titan / NSF)
 - Abajo entra un **tubo stainless** en un **conector push-fit**
 - El **aro gris/blanco** alrededor del tubo es el **release collar**
-

Pasos exactos

Asegúrate que NO haya presión

Muy importante:

- Apaga sistema
 - Cierra CO₂ / líquido
 - Abre un faucet si hace falta
-

Empuja el tubo HACIA ADENTRO primero

Esto libera la presión interna del sello.

Empuja el stainless 1-2 mm hacia el flow meter

Presiona el aro blanco/gris

Con la uña o con un destornillador plano pequeño:

- Presiona el aro hacia el cuerpo del flow meter

- **NO lo jales todavía**
-

4 Mientras mantienes el aro presionado, jala el tubo

- Mantén el aro hundido
- Jala el tubo stainless hacia afuera

★ Sale completo, sin girar, sin fuerza bruta.

✗ NO hagas esto

- ✗ No lo gires
 - ✗ No lo desenrosques
 - ✗ No lo jales sin presionar el aro
 - ✗ No uses pinzas en el tubo (lo marcas y luego no sella)
-

🧠 Tip importante (BeerBot / draft)

Cuando lo vuelvas a poner:

- Empuja el tubo hasta el fondo
- Jala ligeramente para confirmar que **trancó**
- Si el tubo tiene marcas o rayones → puede gotear

Si quieres hacerlo **más seguro para vibración**, se puede:

- Cambiar a fitting NPT + flare
 - O usar tubo flexible corto antes del flow meter
-

Si quieres, dime:

- ¿Lo vas a eliminar?
- ¿Moverlo más arriba?

- ¿Cambiar a otro flow meter?
- ¿Pasar todo a flare/barb para draft?

Y te digo **la mejor configuración para que no te dé espuma ni leaks** 🍺🤖

Envíame un link donde puedo comprar ese flowmeter

Aquí tienes opciones **dónde puedes comprar el mismo tipo de flowmeter (Titan Series 300-010)** — el que tienes en tu sistema de cerveza 🍺👉

Opciones disponibles ahora

Titan Enterprises NSF 300-010 Turbine Flow Meter en eBay (venta individual)

Este es el mismo modelo de **flowmeter de cerveza Titan 300-010** que ves en tu sistema. A veces está disponible usado o nuevo según el vendedor.

Otras opciones / distribuidores

 En varios sitios de distribuidores técnicos puedes **pedir este mismo flow meter** o solicitarlo si no está en stock:

- Titan / NSF Beverage Flowmeters en **DirectIndustry** (modelo 300-010, específico para bebidas)
 - Titan flowmeters disponibles vía distribuidores como **flowmeters.co.uk** (pedido directo / cotización)
 - Otra tienda que lo lista con precio aprox. desde ~\$38-39 USD (body PVDF, salida NPN)  [BPCvalves.com](#)
-

Tips antes de comprar

Confirma el tamaño del fitting

Tu flowmeter actual parece usar **3/8" John Guest push-fit**, igual al original Titan 300-010.

Elige el que tenga salida adecuada

Puede tener salida:

- **NPN (negativo)**
- **PNP**

- **Conektor M12** (según qué electronics uses)
- ✓ **Comprueba certificación NSF** si se va a usar en bebidas (requerido en cerveza potable).
-

Si quieres, puedo **encontrarte opciones concretas con compra online directa (Amazon, AliExpress, etc.)** según tu país.

Dime:

- 📍 ¿Dónde estás (país)?
- 📦 ¿Quieres con cable incluido o sin cable?



Sources

Ese tira sirve con mi beerbot arduino?

Sí ✓ ese **Titan flow meter** Sí sirve perfectamente con tu **BeerBot** en **Arduino**.
Te explico por qué, cómo se conecta y qué debes cuidar para que te funcione bien.

✓ **Compatibilidad con Arduino (resumen rápido)**

Elemento **Estado**

Señal eléctrica ✓ Compatible

Voltaje ✓ 5 V (ideal para Arduino)

Tipo de señal ✓ Pulsos digitales

Precisión ✓ Muy buena para cerveza

Uso en bebidas ✓ Certificación NSF

👉 **No necesitas nada especial ni conversores.**

Cómo funciona eléctricamente (importante)

Ese **Titan (300-010)** es un **turbine flow meter** que entrega **pulsos**.

Señales típicas (3 cables)

Normalmente trae:

- **Rojo** → +5 V
- **Negro** → GND
- **Blanco** → Señal (pulse output)

 Cada pulso = una fracción fija de volumen.

Arduino solo:

- Cuenta pulsos
- Calcula volumen o tiempo

Conexión típica a tu Arduino (BeerBot)

Titan Flow Meter → Arduino

Rojo (+5V) → 5V

Negro (GND) → GND

Blanco (Signal) → Pin digital (ej. D2, D3)

Recomendado:

- Usa un pin con **interrupt** (`attachInterrupt`)
- Ej: D2 o D3 en Arduino Mega / Uno

En tu BeerBot, ¿para qué es ideal?

- ✓ Medir volumen servido
- ✓ Limitar pours (12 oz, 16 oz, etc.)

✓ Protección contra over-pour

✓ Integrarlo con:

- RFID
- QR
- App
- POS

⚠ OJO (importante para espuma):

- El flow meter **NO controla espuma**
 - Solo mide volumen
 - Ubícalo **después de la válvula**, no antes
-

Código básico (concepto)

Solo para que sepas que es trivial:

```
volatile unsigned long pulses = 0;
```

```
void pulseCounter() {  
    pulses++;  
}
```

```
void setup() {  
    pinMode(2, INPUT_PULLUP);  
    attachInterrupt(digitalPinToInterrupt(2), pulseCounter, FALLING);  
}
```

```
void loop() {  
    // pulses → convertir a oz / ml
```

}

(Ya tú tienes algo parecido en tu sistema)

Cosas IMPORTANTES a verificar antes de comprar

1 Salida NPN / Open Collector

- ✓ Es la correcta para Arduino

2 Voltaje de trabajo

- ✓ 5–24 V → perfecto

3 Tamaño

- ✓ El tuyo es ideal para draft (3/8")
-

Recomendación honesta (BeerBot Pro Tip)

Para tu setup:

-  **Titan = perfecto para control**
-  No soluciona espuma
-  Combínalo con:
 - Tilt del vaso
 - Solenoide rápida
 - Cierre antes del final

Eso tú ya lo estás haciendo 

Si quieres, dime:

- ¿Lo usarás para **cortar por volumen** o solo **contar pours**?
- ¿Arduino Mega o GIGA?
- ¿Cuántos taps?

Y te digo **la configuración exacta ideal para tu BeerBot**  

No es para la espuma, es para cambiar el servido por oz en vez de tiempo

Perfecto  entonces estás pensando **correctamente**.

 **Ese Titan es EXACTAMENTE para eso:** pasar de **servido por tiempo → servido por onzas (oz)**.

Voy al grano y luego fino detalles para BeerBot.

Respuesta corta

Sí, ese Titan flow meter es ideal para servir por oz en Arduino.

Es uno de los mejores para draft cuando quieres **volumen real**, no tiempo.

Cómo cambia tu lógica (importante)

Antes (por tiempo)

Abrir válvula → esperar X ms → cerrar

Afectado por:

- Presión
 - Temperatura
 - Espuma
 - Restricción de línea
-

Ahora (por volumen / oz)

Abrir válvula → contar pulsos → llegar a oz → cerrar

Independiente de:

- Presión
- Velocidad
- Variaciones pequeñas

Esto es **nivel comercial real**.

Conversión pulsos → oz (clave)

El Titan típicamente entrega algo como:

- ~6,800 – 7,200 pulsos por litro
(depende del modelo exacto)

 1 oz = 29.573 ml

Ejemplo:

12 oz ≈ 355 ml

355 ml ≈ ~2,500 pulsos (aprox)

 **NO uses valores genéricos**

 Se calibra UNA VEZ con agua o cerveza.

Configuración ideal en BeerBot

Ubicación física

- ✓ Después de la válvula
 - ✓ Línea recta antes y después (si puedes)
-

Lógica recomendada (anti-overpour)

1 Reset pulsos

2 Abre válvula

3 Cuenta pulsos

4 Al llegar a:

- (**target - margen**) → pre-cierre
- **target exacto** → cierre duro

Ejemplo:

Target: 12 oz

Cierre suave: 11.7 oz

Cierre total: 12.0 oz

Esto compensa:

- Inercia
 - Goteo
 - Espuma residual
-

Pseudocódigo simple

```
float targetOz = 12.0;  
float pulsesPerOz = CALIBRATED_VALUE;  
  
if (dispensing) {  
    if (pulses >= targetOz * pulsesPerOz) {  
        closeValve();  
        dispensing = false;  
    }  
}  
  
(Ya tú estás listo para esto)
```

Detalles CRÍTICOS que mucha gente falla

-  Usar delay()
 -  Usar interrupciones
 -  Cerrar justo en el número
 -  Cerrar **antes** y compensar
 -  No filtrar rebotes
 -  Usar ISR limpia
-

Recomendación BeerBot PRO

Para tu sistema:

- Sirve **por oz**
- Mantén **tiempos solo como backup**
- Guarda pulsesPerOz en:
 - EEPROM
 - SD
 - IoT Cloud

Así puedes:

- Cambiar tamaño sin tocar código
 - Ajustar desde el dashboard
-

Conclusión

- ✓ Ese Titan es el correcto
- ✓ Arduino lo maneja perfecto
- ✓ Es el paso correcto para escalar BeerBot

Si quieres, en el próximo mensaje:

- Te doy **el bloque de código exacto** listo para integrar
- O hacemos **rutina de calibración paso a paso**
- O lo integramos con tu sistema actual (A, B, C pour)

Dime cómo seguimos 

Pregunta, se pudiese convertir el tiempo de servido(lo q tengo ahora) a oz. Estoy usando una manga 3/16 y 10psi con una manga de 6.5 pies

Sí, se puede **convertir tu servido por tiempo a “oz”**, pero con una condición importante:

-  **Funciona como aproximación si mantienes casi igual:** presión (10 psi), temperatura, largo/diámetro de línea (3/16" y 6.5 ft), tipo de faucet/solenoid, y rutina de tilt.

- **✗ No será exacto** cuando cambien esas variables (o si hay más/menos espuma, CO₂, etc.). Por eso el flow meter sigue siendo lo “pro”.
-

La idea: crear tu “oz por segundo” (calibración)

Lo que haces es medir cuántas onzas sirves en **X segundos**, y de ahí sacas:

$$\text{oz_per_sec} = \text{oz_medidas} / \text{segundos}$$

Luego, para servir por oz:

$$\text{tiempo} = \text{oz_deseadas} / \text{oz_per_sec}$$

Cómo calibrarlo bien (rápido y práctico)

1. Enfría el sistema como normalmente sirves.
2. Usa un vaso medidor (o balanza):
 - **Balanza:** 1 oz (fl oz) ≈ 29.57 g de agua; en cerveza es parecido (un poco más denso), pero sirve para calibrar.
3. Sirve con tu rutina normal (incluyendo tilt) por un tiempo fijo, por ejemplo **10.0 s**.
4. Mides cuántas oz salieron.

Ejemplo:

- 10.0 s → 16 oz
⇒ oz_per_sec = 1.6 oz/s

Entonces:

- 12 oz → 12 / 1.6 = **7.5 s**
 - 16 oz → **10.0 s**
 - 21 oz → **13.1 s**
-

Con tu setup específico (3/16”, 6.5 ft, 10 psi)

Eso suena bastante “estándar” para draft y debería darte un flujo relativamente estable **si el keg está frío**.

Pero ojo con estas 3 cosas que te cambian el “oz/sec” aunque no cambies nada:

1. **Temperatura del keg/torre** (caliente = espuma = menos líquido real por segundo)

2. **Nivel del keg** (a veces cambia el comportamiento)
 3. **Restricciones** (check valves, flow meter, quick connects, codos)
-

Truco que te va a quedar brutal (sin flow meter)

Haz **una tabla de calibración** por cada beer/tap:

- Tap A: oz_per_sec_A
- Tap B: oz_per_sec_B

Y ya tu sistema sirve por “oz” usando el tiempo calculado.

 Ventaja: rápido, sin hardware extra

 Desventaja: si la condición cambia, se descalibra

Lo más importante si lo vas a usar “en producción”

Para que no se vaya largo/corto:

- Calcula tiempo para (**oz objetivo – margen**)
- Y deja un “top-up” por tiempo cortito si quieres afinar

Ejemplo para 12 oz:

- Sirve 11.5 oz por tiempo calculado
 - Luego top-up: 0.5 oz con pulso corto (o 2-3 pulsos de 150 ms)
-

Si me dices **cuántas oz te salen en 10 segundos** (o en 5), yo te saco:

- oz/seg exacto
- tiempos para 12/16/21 oz
- y un margen recomendado para que no se pase

¿Quieres medirlo con **vaso medidor** o con **balanza**?

Bno, acuerdate que tengo ABC en el A son 5s y el C 3s el prenpour es de .4seg

Perfecto, me acuerdo: **pre-pour 0.4 s**, y tu servido está en **fases A, B, C** ($A=5s$, $C=3s$). Con eso **sí puedes convertir tu lógica actual a oz**, manteniendo el “perfil” ABC.

La idea es simple:

Cómo convertir ABC (tiempo) → oz (sin flow meter)

1. Tu receta actual tiene un **tiempo total de válvula abierta**:

$$T_{total} = T_{pre} + T_A + T_B + T_C$$

donde $T_{pre} = 0.4$, $T_A = 5$, $T_C = 3$, y T_B es el que tengas.

2. Mides cuántas **oz reales** salen con esa receta completa (una sola vez) y sacas tu factor:

$$\text{oz_por_seg} = \frac{\text{oz_medidas}}{T_{total}}$$

3. Luego, cuando pidas “12 oz”, calculas el tiempo total requerido:

$$T_{req} = \frac{12}{\text{oz_por_seg}}$$

4. Y mantienes tu “forma” ABC escalando **A, B, C** proporcionalmente (pre-pour lo puedes dejar fijo o también escalarlo; yo recomiendo **dejarlo fijo** en 0.4s para consistencia de arranque).

$$k = \frac{T_{req} - T_{pre}}{(T_A + T_B + T_C)}$$
$$T'_A = T_A \cdot k, \quad T'_B = T_B \cdot k, \quad T'_C = T_C \cdot k$$

Así sigues teniendo tu comportamiento anti-espuma (ABC), pero ahora el usuario “elige oz”.

Paso 1 (lo único que necesito para darte los números exactos)

Haz **un servido completo con tu receta actual** y dime:

- **¿Cuántas oz te dio ese servido?** (usa vaso medidor o pesa en gramos y lo convertimos)

Y dime también **cuánto es tu B ahora** (en segundos), porque solo me diste A=5s y C=3s.

Con esas 2 cosas, te calculo:

- oz/seg real de tu sistema
- tiempos nuevos para **12oz / 16oz / 21oz**
- y un margen recomendado para que no se pase (por goteo/inercia).

Ok, aunque este por tiempo, yo puedo ver en el giga display las oz sirviéndose?

Sí puedes ver “oz sirviéndose” en el GIGA Display aunque el servicio siga siendo por tiempo.

Lo que verás son **oz estimadas en tiempo real**, no medidas reales (eso solo con flow meter), pero funciona muy bien visualmente y a nivel UX.

Te explico **cómo y qué tan preciso es**.

Concepto clave

Aunque la válvula cierre por **tiempo**, tú puedes:

Convertir tiempo → oz estimadas en vivo

y mostrarlas como si fuera un contador de volumen.

Esto se basa en un **factor calibrado**:

oz_por_segundo

Cómo se ve la lógica

Tienes tu receta actual

- Pre-pour = 0.4 s
- A = 5 s
- B = (el que tengas)
- C = 3 s

Tiempo total:

$$T_{\text{total}} = 0.4 + A + B + C$$

Calibras UNA VEZ

Sirves un vaso completo con esa receta y mides:

oz_totales_servidas

Entonces:

oz_por_seg = oz_totales / T_total

Ese valor lo guardas (SD, variable, IoT Cloud, etc.).

Mostrar oz en el GIGA Display (en tiempo real)

Mientras la válvula está abierta:

oz_actuales = tiempo_transcurrido * oz_por_seg

Y eso lo refrescas en pantalla, por ejemplo cada 50–100 ms.

Visualmente el cliente ve:

-  “Sirviendo...”
-  **3.2 oz → 6.8 oz → 9.4 oz → 12.0 oz**
-  “Servido completo”

Aunque por debajo sigas usando tiempo.

Qué tan exacto es

Con tu setup fijo (3/16”, 6.5 ft, 10 psi):

-  Bastante consistente si:
 - Keg frío
 - Presión estable
 - Misma cerveza
-  Puede variar si:
 - Cambia temperatura

- Hay más espuma ese día
- Cambias línea / faucet

👉 Para display y experiencia de usuario es **PERFECTO**

👉 Para facturación exacta: **flow meter**

🧠 Recomendación PRO (muy BeerBot)

Haz esto:

- 📈 Usa **tiempo** para controlar válvula (como ahora)
- 📺 Usa **oz estimadas** solo para:
 - Display
 - App
 - Feedback visual
- 💡 Mantén un margen interno (ej. cerrar a “11.7 oz” estimadas)

Más adelante:

- Cambias backend a flow meter
- El display **no cambia nada**
- Solo cambia de “oz estimadas” → “oz reales”

Arquitectura limpia 🌟

✅ Resumen corto

- ✓ Sí, puedes mostrar oz en el GIGA Display
- ✓ Aunque el sistema cierre por tiempo
- ✓ Es una conversión tiempo → volumen
- ✓ Ideal como paso intermedio antes del flow meter

Siquieres, próximo paso puedo:

- 📈 Hacer el **bloque exacto de cálculo** para el GIGA

- Diseñar cómo se ve en pantalla (barra + número)
- Integrarlo con tu lógica ABC sin romper nada

Dime cómo seguimos.

Ok, lo hago luego.. la pregunta clave ahora es... me estan haciendo un app que va a tener el app del teléfono, un kiosko y un monitor pequeño sobre el tap. Eso va a controlar todo incluyendo las oz cuando tenga el flow meter, tambn la temperatura del barril y si no esta a temp no sirve... esa info esta en el giga, como puedo hacer que se vea desde allá y se maneje todo desde allá y el giga solo sea el plc para activar

Perfecto: lo que tú quieras es una arquitectura “**App/Kiosk = cerebro (UI + reglas + órdenes)**” y **GIGA = PLC/IO (sensores + actuadores)**.

Eso se hace bien limpio con un **API local** en el GIGA + un **canal de eventos** para que el app vea en vivo (oz, temp, estados).

1) Arquitectura recomendada (la que usan kioskos de verdad)

Roles

- **App / Kiosko / monitor tap (Frontend)**
 - Muestra menú, disponibilidad, temp, oz sirviéndose
 - Cobra / valida orden
 - Envía comando: “Sirve 12 oz en Tap 2”
 - Recibe estado en vivo: *temp, oz, errores, “cup present”*, etc.
- **Arduino GIGA (PLC/Controller)**
 - Lee sensores: flow meter (pulsos), temp (DS18B20 u otro), cup sensor, switches
 - Controla salidas: solenoide, motor tilt, LEDs, etc.
 - Aplica **interlocks de seguridad** (ej. “si temp > límite → NO sirve”)
 - Reporta estado en tiempo real

Así aunque el app falle, el GIGA **nunca servirá** si algo está fuera de regla.

2) Cómo “ver y manejar todo desde el app” (y que el GIGA sea el PLC)

Tienes 2 patrones. Te digo cuál usaría yo para BeerBot:

Opción A (RECOMENDADA): Backend local (Raspberry Pi o mini PC)

- El app/kiosko habla con un **servidor local** (Node.js)
- El servidor local habla con el GIGA (WiFi) por HTTP/WebSocket
- Ventaja: robusto, logs, reintentos, colas, multi-dispositivos fácil

Ideal porque tú ya has usado Pi/Node en BeerBot.

Opción B: El app habla directo con el GIGA

- El GIGA expone un API HTTP/WebSocket y ya
- Más simple, pero:
 - Más difícil manejar muchos clientes
 - Menos “business logic” central
 - Menos control de seguridad de red

Si vas a tener 3 frontends (app + kiosko + tap monitor), la A es mejor.

3) Lo que tiene que exponer el GIGA (como PLC)

Piensa en “tags” como si fuera un PLC:

Estado (lo que el app ve)

- tap[1].tempF
- tap[1].tempOk (true/false)
- tap[1].availableOz (si quieres)
- tap[1].pouring (true/false)
- tap[1].pouredOz (en vivo)
- tap[1].errorCode (si algo falla)
- system.mode (idle/serving/cleaning/fault)
- cupPresent (si aplica)

Comandos (lo que el app manda)

- startPour(tapId, targetOz, orderId)
- stopPour(tapId)
- setMasterSettings(...) (solo admin)
- primeLine(tapId, seconds) (mantenimiento)

📌 Importante: el app manda “**targetOz**”, pero el GIGA es quien cuenta pulsos y cierra.

4) Comunicación: lo mínimo que necesitas

Para que sea “en vivo” (oz sirviéndose en pantalla):

Canal 1: HTTP (órdenes y configuración)

- POST /api/pour/start
- POST /api/pour/stop
- GET /api/status

Canal 2: WebSocket o SSE (telemetría en vivo)

- Enviar cada 100–250 ms:
 - tempF, tempOk, pouredOz, state, errors

WebSocket es lo más flexible.

5) Temperatura “si no está a temp no sirve”

Esto **SIEMPRE debe vivir en el GIGA** como “hard rule”.

Ejemplo de regla:

- tempOk = (tempF <= 38°F) (tú decides el setpoint)
- Si tempOk == false:
 - Rechaza startPour con error: TEMP_NOT_READY
 - En el stream manda tempOk=false + “Cooling...”

Así el app solo muestra, pero el PLC decide.

6) Multi-dispositivo: app + kiosko + tap monitor

Todos consumen el mismo estado.

- Kiosko: interfaz principal + QR + pago
- Tap monitor: solo “Now pouring / oz / temp / logo”
- Teléfono: ordenar/remitir/estado

Todos se sincronizan porque escuchan el mismo WebSocket del backend.

7) Seguridad / anti-trampa (bien importante para BeerBot)

- orderId único por compra
- GIGA valida:
 - order no usado
 - tap correcto
 - volumen permitido
- Si se pierde conexión:
 - GIGA termina el pour de forma segura o cierra

Cómo lo implementas sin reinventar todo

Tu “app team” puede construir esto con un **contrato** (API spec).

Si tú me dices qué stack están usando (Flutter/React Native/Swift + backend Node/Firebase/etc.), yo te doy:

- La lista final de endpoints
- Formato JSON exacto
- Estados y códigos de error
- Diagrama de flujo (order → redeem → pour → complete)

Siguiente paso (sin código todavía)

Dime **solo esto**:

1. ¿El kiosko y el tap monitor corren en qué? (Android tablet? Raspberry Pi con web? Windows?)
2. ¿El app está usando backend en la nube (Firebase/Stripe) o todo local?

Con eso te diseño la arquitectura exacta para tu caso y no pierdes tiempo.

Pero si no hay internet no funciona cierto?

Clave pregunta — y **NO**, bien diseñado **Sí funciona sin internet**.

Aquí va la respuesta clara, sin vueltas.

🔑 Regla de oro

Internet ≠ red local (LAN).

Tu BeerBot **NO debe depender de internet para servir**.

✓ Qué funciona SIN internet (como debe ser)

Mientras exista **red local** (WiFi del router o AP):

- ✓ Kiosko
- ✓ Monitor del tap
- ✓ Arduino GIGA (PLC)
- ✓ Backend local (Raspberry Pi / mini-PC)
- ✓ Flow meter → oz reales
- ✓ Sensor de temperatura → bloqueo si está caliente
- ✓ Conteo, cierres, seguridad

👉 Todo eso corre **localmente**.

✗ Qué NO funciona sin internet (opcional)

- Pagos (Stripe / Evertec)
- Sync con la nube

- Reportes remotos
- Actualizaciones OTA
- Analytics centralizados

💡 Pero **el servicio no se cae.**

🧠 Arquitectura correcta (BeerBot PRO)

(Internet) ← solo para pagos / reportes

|

| Router/AP |

| | |

Kiosko Tap App local

\ | /

\ | /

Backend local (Pi)

|

Arduino GIGA

|

Sensores / válvulas / motor

📌 Si se va el internet:

- El router sigue activo
 - El Pi sigue activo
 - El GIGA sigue activo
 - El sistema sirve normal
-

Caso extremo: sin router

Incluso aquí tienes plan B:

- El Pi o GIGA crea su propio WiFi (AP mode)
- Kiosko se conecta directo
- Todo local

Eso se usa mucho en eventos.

Pagos offline (opción avanzada)

Si quieres:

- Modo “offline tab” (RFID / QR pre-pagado)
- O tokens cargados antes

Pero eso ya es fase 2–3.

Conclusión

- ✓ El servicio **NO depende de internet**
 - ✓ Internet solo es “nice to have”
 - ✓ GIGA manda siempre
 - ✓ Arquitectura correcta evita dolores de cabeza
-

Si quieres, próximo paso puedo:

- Dibujarte el **diagrama final BeerBot (online/offline)**
- Definir **qué corre dónde** (GIGA vs Pi vs App)
- O ayudarte a escribir el **spec técnico** para el developer del app

Dime cómo seguimos.

Que el qr code y el rfid si va a funcionar.. puedo cobrar aparte y usar eso

Sí **QR y RFID pueden funcionar sin internet** y tú puedes “cobrar aparte” (offline) usando un sistema de **créditos/tokens**.

La clave es separar:

- **Pago (internet opcional)**
- **Autorización de servido (local, siempre)**

Cómo hacerlo sin internet (modelo práctico)

Opción 1: RFID como “wallet” local (recomendado)

- Cada tarjeta RFID tiene un **balance de oz o créditos**
- El kiosko/monitor solo muestra y el GIGA (o el Pi local) valida:
 - “tiene \geq 12 oz disponibles”
 - si sí: descuenta y sirve
 - si no: deniega

Ventajas:

- 100% offline
- Rápido en eventos
- Control total tuyo

Para “cobrar aparte”:

- Cobras cash/ATH/whatever en la barra
- El staff usa un “Admin card” o PIN para **recargar balance** en el sistema local

Opción 2: QR “tokens” pre-generados (también sirve)

- Imprimes/vendes QR que representan:
 - “12oz-1”, “16oz-1”, etc.
- El sistema local verifica en una base de datos local (Pi o SD):
 - token existe
 - token no usado
 - tap permitido
- Sirve y marca token como **redeemed**

 Ventajas:

- No necesitas tarjetas RFID
- Bueno para tickets en eventos

 Importante:

- Sin internet, el QR no debe depender de “consultar la nube”
 - Tiene que validarse **local**
-

“Puedo cobrar aparte y usar eso”

Sí. Y hay 3 formas comunes:

A) Cash + RFID wallet (lo más fácil)

- Pagas en caja
- Te encourage: “Te cargo 48 oz al RFID”
- La gente sirve cuando quiera hasta gastar

B) Cash + QR vouchers

- Pagas en caja
- Te dan QR (papel o screenshot)
- Redeem en el tap

C) Híbrido

- Con internet: Stripe/Evertec normal
 - Sin internet: se activa modo “Offline” y solo RFID/QR
-

Detalle CRÍTICO (para que nadie te robe servir)

Si vas offline, necesitas **anti-duplicación**:

Para RFID

- Balance vive en tu sistema local, no en la tarjeta (mejor)
- O si vive en tarjeta, usar MIFARE con sectores + llaves (más complejo)

Para QR

- QR debe ser un **token firmado** o aleatorio (ej. 12–16 caracteres)
 - El sistema local debe guardar “usado / no usado”
-

Lo que yo te recomiendo para BeerBot (simple y robusto)

RFID wallet local como principal offline

QR solo para:

- órdenes del app cuando sí hay internet
- o vouchers para eventos (si quieres)

Así tu operación nunca se detiene.

Si me dices cuál prefieres (RFID wallet o QR vouchers), te escribo el flujo exacto:

- Pantallas del kiosko
- Qué valida el GIGA
- Qué guarda el Pi (o el mismo GIGA)
- Cómo recargar créditos en modo admin
- Y cómo protegerlo contra “reuse” y duplicación