

In one line: StoreKit 2 (iOS 15+) is **async/await**: `fetchProducts`, `await product.purchase()`, get back a **JWS-signed** `Transaction` wrapped in `VerificationResult` — **verify** → **grant** → **finish()**. Everything that happens *outside* your purchase call (renewals, refunds, Ask to Buy, other devices) arrives on `Transaction.updates`, which you listen to **from launch**.

How it works

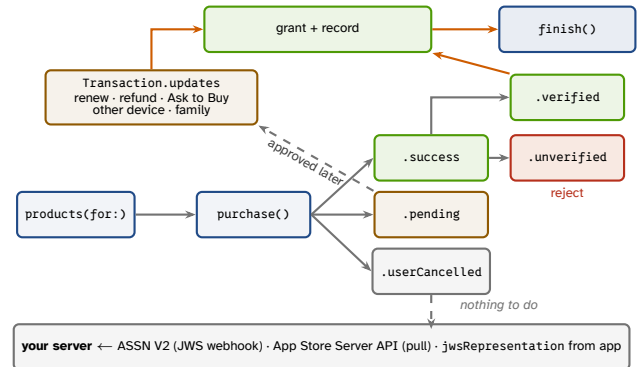
- **Fetch:** `try await Product.products(for: ids) → [Product] (displayName, localized displayPrice, type)`. Empty array, no error = App Store Connect config (IDs, Paid Apps agreement).
- **Types:** consumable · non-consumable · auto-renewable sub · non-renewing sub (you track expiry).
- **Purchase:** `try await product.purchase(options:) → Product.PurchaseResult: .success(VerificationResult) · .userCancelled · .pending (Ask to Buy, SCA — grant nothing; the outcome arrives later on updates)`. Option `.appAccountToken(UUID)` ties it to your user.
- **Verify:** `VerificationResult<T> = .verified(T) or .unverified(T, error)`. StoreKit checked the JWS signature on device; never unlock on `.unverified`. Send `jwsRepresentation` to your server.
- **Finish:** `await t.finish()` only *after* the entitlement is durably recorded; unfinished ones are re-delivered.
- **Entitlements:** `Transaction.currentEntitlements` = latest transaction per non-consumable, active auto-renewable and non-renewing sub. **No consumables**, no refunded (revoked).
- **Subscriptions:** a **subscription group** allows one active plan. Upgrade = immediate, downgrade = next renewal. `product.subscription?.status → state (.subscribed .expired .inGracePeriod .inBillingRetryPeriod .revoked) + renewalInfo (willAutoRenew, autoRenewPreference)`. Promo offers need a server signature.
- **Restore:** entitlements sync by themselves on the same Apple Account; the Restore button calls `try await AppStore.sync()` (prompts sign-in — only on user tap).
- **Server:** **App Store Server Notifications V2** POST a signed JWS (signedPayload; DID_RENEW, EXPIRED, REFUND, DID_FAIL_TO_RENEW...) to your URL; **App Store Server API** (JWT from your in-app purchase key) pulls transaction history / subscription statuses. Device decides UX, server decides truth.
- **Test:** `.storekit config file` (set in the scheme) — offline, refunds, fast renewals, `SKTestSession`. **Sandbox:** sandbox accounts, real server, 1 month ≈ 5 min, limited renewals.

Example

```
func buy(_ p: Product) async throws {
    switch try await p.purchase() {
    case .success(let r):
        guard case .verified(let t) = r else { return } // reject
        await grant(t); await t.finish() // record THEN finish
    case .pending, .userCancelled: break // pending -> updates
    @unknown default: break
    }
}

// App init: start once, keep for the app's lifetime
let listener = Task.detached {
    for await r in Transaction.updates {
        guard case .verified(let t) = r else { continue }
        await grant(t); await t.finish()
    }
}
```

Picture — one purchase, two roads in



Picture — which API answers what

What does the user own now?	<code>Transaction.currentEntitlements</code>
Something changed while I wasn't looking?	<code>Transaction.updates</code>
Will the sub renew? In grace?	<code>subscription?.status</code>
User taps Restore	<code>AppStore.sync()</code>
Did the coin purchase happen?	<code>your own ledger (+server)</code>

Interview traps

- ! Trusting `.unverified` — *the* StoreKit security bug.
- ! No `finish()` → redelivered, double grants; finishing *before* recording → lost purchase.
- ! No updates listener at launch → missed renewals, refunds, approvals.
- ! Counting coins via `currentEntitlements` — consumables never appear.
- ! Granting on `.pending`. Sandbox timing ≠ production.
- ! `AppStore.sync()` at launch — shows a sign-in prompt.

Remember

“V-G-F, and keep ears open”: Verify, Grant, Finish — and `Transaction.updates` runs from launch. Consumables are *yours* to count.

Likely questions

1. Three purchase results? — `.success(verification)`, `.userCancelled`, `.pending`.
2. What arrives on updates? — renewals, refunds, Ask-to-Buy, other-device purchases.
3. Restore in SK2? — automatic via entitlements; button → `AppStore.sync()`.
4. Why a server? — cross-platform access, tamper-proof truth, ASSN V2 events.
5. Test without App Store Connect? — `.storekit config file` + `SKTestSession`.