Appwright — In-App Purchases Guide

Sell things inside your Android app with Google Play. No server, no payment provider, no purchase code to write — you label your HTML and Google handles the money.

This guide walks through a complete working example you can build today (**"Star Jumper"**, a small game), then shows how to do the same in your own app — plain HTML, React or Next.js.

One page, three kinds of purchase

What the buyer gets	Product ID	Type
No ads, forever	`remove_ads_lifetime`	one-time
Levels 11–20 forever	`level_pack_2`	one-time
Levels 21–30 while paying	`all_levels_monthly`	monthly subscription

> **Nothing is ever charged.** As a license tester, Google's payment screen says "Test card, always approves".

> **Follow the steps in the order they are numbered.** Step 1 must come first — Google Play will not let you create any products until it has received a build of your app that declares the billing permission.

Which block in `index.html` does what

#	What it shows	The label that does it
1	"Ads are showing" notice → flips to "Ads removed"	`data-iap-hide="remove_ads_lifetime"` `data-iap-show="remove_ads_lifetime"`
2	Levels 1–10, always free	none — plain HTML
3	Levels 11–20 locked → unlocked	`data-iap-hide="level_pack_2"` `data-iap-show="level_pack_2"`
4	Levels 21–30 locked → unlocked	`data-iap-hide="all_levels_monthly"` `data-iap-show="all_levels_monthly"`

#	What it shows	The label that does it
5	The three buy buttons	`data-iap-buy`, `data-iap-label`, `data-iap-owned`, `data-iap-container`

The page has **no purchase code, no script tag and no purchase-related CSS**. Every locked and unlocked state above is one attribute.

Step 1 — Build the app in Appwright

1. Zip `index.html` — that one file is the whole app.
2. Open Appwright, upload the ZIP, and go to **Step 3 — Integrate**.
3. Set these:

Setting	Value
Ads	ON — so you can show ads disappearing. Use your AdMob IDs (or Google's test IDs).
💰 Sell In-App Products	**ON** ← this is what puts the billing permission in your app
Remove Ads (IAP)	ON
Play Store Product ID	`remove_ads_lifetime`
License Key	paste from Play Console → Monetize → Monetization setup → Licensing

> **" 💰 Sell In-App Products" is a paid capability.** Turning it on needs any paid Appwright plan — Unlock ($2, one-time), Pro, Standard, Max or Agency. On the free tier the switch reverts and the plans screen opens. It is checked only when you *build*: apps you already shipped keep selling regardless of what happens to the plan later.

Only `remove_ads_lifetime` goes in the Product ID box. The two level products must **not** be there — they unlock levels, not ads.

4. Export the **AAB** (not the APK).

Optional check before uploading: open the `.aab` with any zip tool, open `base/manifest/AndroidManifest.xml` and search for `com.android.vending.BILLING`. If it is not there, one of the two IAP switches was off — go back and rebuild. Uploading without it is what makes Play say *"To add one-time products, you need to add the BILLING permission to your APK"*.

Step 2 — Upload it to Internal testing

1. Play Console → your app → Testing → Internal testing → Create new release.
2. Upload the AAB and roll it out.
3. Testers tab → add your Gmail → copy the join link (you will use it in Step 5).

This upload is what unlocks the product screens in Step 3. Play only needs to have *processed* the build — you do not have to publish it anywhere public.

Step 3 — Create the three products

A. Remove ads (one-time)

1. Monetize → Products → In-app products → Create product
2. Product ID: `remove_ads_lifetime`
3. Name "Remove ads forever" · set a price · Save · Activate

B. Level pack (one-time)

1. Same screen → Create product
2. Product ID: `level_pack_2`
3. Name "Levels 11-20" · set a price · Save · Activate

C. Season pass (subscription)

1. Monetize → Products → Subscriptions → Create subscription
2. Product ID: `all_levels_monthly`
3. Save, then add a base plan: type **Auto-renewing**, billing period **Monthly**, set a price
4. Activate the **base plan AND the subscription**. Miss either and the app cannot see it.

Type the IDs exactly as written — they must match the page character for character.

Step 4 — Make yourself a license tester

Play Console → Setup → License testing → add your Gmail → License response: **RESPOND_NORMALLY**. This is what makes the purchases free.

Step 5 — Install on your phone

Open the join link from Step 2, accept, then install from the Play Store.

> ⚠ **Sideloading the APK will not work** — Play billing requires a Play install. This is the single most common reason a demo fails.
>
> ⚠ A product created minutes ago may not be visible yet. If a button says "not available on Google Play yet", wait a while and reopen the app before assuming something is broken.

Step 6 — The demo itself

1. **Open the app.** Ads are showing, the orange "Ads are showing" notice is at the top, levels 11–30 are greyed out, and the three buttons show **real prices in your currency** — those came from Google, not from the page.
2. **Tap "Remove ads forever".** Google's screen says *Test card, always approves*. Complete it. The ads disappear immediately, the orange notice flips to green, and the button turns green. Levels stay locked — point that out.

3. **Tap “Unlock levels 11–20”.** Buy it. That card flips to unlocked. The season-pass card is untouched: different product, different unlock.
4. **Tap “Season pass”.** Buy it. Levels 21–30 unlock.
5. **Force-close the app and reopen.** Everything is still unlocked, no ads, no sign-in. Google restored it at launch.
6. **Turn on airplane mode and reopen.** Still unlocked — ownership is remembered on the device.
7. **(optional, the strongest part)** Cancel the subscription in Play Store → Subscriptions. Test subscriptions expire in minutes rather than a month. Reopen: levels 21–30 are locked again by themselves, while the ad removal and levels 11–20 stay — those were paid once.

Now do it in your own app

Three things. Nothing else.

1. Decide what you sell, and invent an ID for each. Lowercase letters, numbers and underscores. You can never rename or reuse an ID later, so keep it plain: `remove_ads`, `pro_unlock`, `level_pack_2`, `coins_100`, `premium_monthly`. Do not put the price in the name — prices live in Play Console and can change any time.

2. Label the HTML you already have. Copy this and change the words:

```
<!-- the button that sells it -->
<button data-iap-buy="pro_unlock"
        data-iap-label="Unlock Pro — {price}"
        data-iap-owned="✅ Unlocked">Unlock Pro</button>

<!-- appears only after they buy that product -->
<div data-iap-show="pro_unlock" hidden>Thanks! Everything is unlocked.</div>

<!-- disappears once they buy it -->
<div data-iap-hide="pro_unlock">Upgrade to remove the limits.</div>
```

`{price}` becomes Google’s real price in the buyer’s own currency. Add another button for another product — that is the whole API.

3. In Step 3 — Integrate, turn on 💰 Sell In-App Products. Add **Remove Ads (IAP)** too, with the ID in the Product ID box, only if that purchase should switch the ads off.

The same labels work in React, Next.js and Vue — they are only markup, so no `useState`, no `useEffect` and no `'use client'`.

Or hand it to an AI

Building with ChatGPT, Claude, Cursor, Lovable, or Appwright's own AI? Copy everything between the lines below and paste it in along with your page. It contains every rule the AI needs, and — just as importantly — tells it what *not* to write, so it does not invent a broken purchase script.

```
_____________________________________________________________________________
ADD IN-APP PURCHASES TO THIS PAGE (Appwright apps)

This page will be turned into an Android app by Appwright. The app already
provides the purchase engine as window.AppMintIAP. Your job is ONLY to label
my HTML.

DO NOT:
- do not write any purchase JavaScript, listeners, fetch calls or state handling
- do not add <script src=...>, do not import or bundle any library
- do not write prices into the page — Google supplies them per country
- do not unlock anything when a button is clicked

DO: add these attributes to the HTML.
  data-iap-buy="<product id>"    put on the button that buys it   (required)
  data-iap-label="… {price} …"   button text; {price} = Google's real price
  data-iap-owned="…"             button text after it is bought
  data-iap-text                  put on the <span> holding the text, if the
                                 button also contains icons
  data-iap-show="<product id>"   element that appears once THAT product is owned
                                 (give it the hidden attribute in the HTML)
  data-iap-hide="<product id>"   element that disappears once it is owned
  data-iap-container             a wrapper that appears only inside the app
  data-iap-group="<name>"        ONLY for products that are alternatives of each
                                 other (monthly vs lifetime of the same thing);
                                 buying either then hides the other

RULES
- Different products are independent. Buying one must not change another.
  Do not give unrelated products the same group.
- Always name the product in data-iap-show / data-iap-hide. A bare
  data-iap-show fires as soon as ANY product is owned.
- Product IDs are lowercase letters, numbers and underscores, and must match
  Google Play Console exactly. They can never be renamed or reused.
- One-time purchases and subscriptions use the SAME attributes. Which one it is
  gets decided in Play Console, not in the page.
- In a normal web browser nothing appears — that is correct and intended.

REACT / NEXT.JS / VUE
- The same attributes work as plain JSX. No useState, no useEffect, no
  'use client' needed. Buttons mounted later are picked up automatically.
- Only if I ask for React to render the price itself:

      const [iap, setIap] = useState({ available: false, products: {} });
      useEffect(() => {
        if (typeof window === 'undefined' || !window.AppMintIAP) return;
        window.AppMintIAP.start({ plans: [{ id: 'pro_unlock' }] });
        return window.AppMintIAP.onChange(setIap);
      }, []);
      const pro = iap.products?.pro_unlock;      // ALWAYS optional-chain
      if (!iap.available) return null;
      if (pro?.owned) return <p>Thanks!</p>;
```

```
      // buy with window.AppMintIAP.buy('pro_unlock')

  Read ownership PER PRODUCT via products?.<id>?.owned — never the top-level
  iap.owned, which means "anything is owned" and changes meaning as soon as a
  second product exists. Never touch window.AppMintIAP outside useEffect.

EXAMPLE OF THE WHOLE INTEGRATION
  <button data-iap-buy="pro_unlock"
          data-iap-label="Unlock Pro — {price}"
          data-iap-owned="✅ Unlocked">Unlock Pro</button>
  <div data-iap-show="pro_unlock" hidden>Thanks! Pro is unlocked.</div>
  <div data-iap-hide="pro_unlock">Upgrade to remove the limits.</div>

Now apply this to my page. Tell me the list of product IDs you used, so I can
create them in Google Play Console.
————————————————————————————————————————————————————————————
```

The last line matters: you need that ID list to create the matching products in Play Console, and to know which ID (if any) goes in **Remove Ads (IAP)** → **Product ID**.

The full JavaScript API

You never need any of this — the `data-iap-*` labels above are a complete integration. It is here for the cases where your own code wants to ask, rather than be told.

`window.AppMintIAP`

Member	What it does
`.available`	true only inside an Appwright app with purchases enabled
`.sell(id, button, opts?)`	the one-liner: give it a product id and the button that buys it. Options: `label`, `owned`, `show`, `hide`.
`.start(config)`	the same thing with every option at once — `plans[{id, button?, label?, owned?, group?, labelEl?}]`, `container`, `show`, `hide`, `hideOtherPlans`, `busyLabel`, `toast`, `onUnlock(id)`, `onLock(id)`. A plan with no `button` is tracked but never drawn — that is the React path.
`.onChange(fn)`	calls `fn({available, adFree, owned, ownedIds, products})` now and on every change; returns an unsubscribe function
`.state()`	the same object, right now
`.isOwned(id)`	true / false — instant, works offline
`.owned()`	array of everything the buyer owns
`.buy(id)`	open Google's payment screen
`.refresh()`	re-scan the page for new `data-iap-buy` buttons

```
// the whole thing, in one line
AppMintIAP.sell('pro_unlock', '#buy-btn');

// or with the optional bits
AppMintIAP.sell('pro_unlock', '#buy-btn', {
  label: 'Unlock Pro — {price}',
  owned: 'Unlocked',
  show:  '#thanks',
  hide:  '.upsell'
});
```

Events the app fires at your page

Event	When
`appmint:products`	Google's live localized prices have arrived
`appmint:purchase`	a purchase completed
`appmint:purchase-failed`	a purchase did not complete
`appmint:owned-changed`	ownership changed (restore, refund, lapse)
`appmint:premium`	the ad-free entitlement changed

Removing the ads from your own button

The ad-free purchase is separate from the product catalogue and lives on the native bridge. Your page must provide the button — the generated app has no built-in menu to carry one.

```
var iap = window.WebToApk;

if (iap && !iap.isPremium()) {
  showRemoveAdsButton(function onTap() {
    iap.startRemoveAdsPurchase();
  });
}

window.addEventListener('appmint:premium', function (e) {
  if (e.detail.premium) hideRemoveAdsButton();
});
```

Tips worth knowing before you start

Ads are the one special case. Your ads are real Android ads sitting on top of the page, so nothing in your HTML can remove them. Only IDs typed into **Remove Ads (IAP)** → **Product ID** switch them off. Everything else you sell unlocks whatever your page does with it, and the ads keep running. That is usually what you want.

Selling two ad-free plans? Put both IDs in that one box, comma separated: `remove_ads_monthly,remove_ads_lifetime`. Owning either removes the ads. The first ID listed is the one that `window.WebToApk.startRemoveAdsPurchase()` buys, so put your default plan first.

Different products never affect each other. Buying a level pack does not touch your Pro button. Only add `data-iap-group="something"` when two products are alternatives for the same thing — a monthly plan and a lifetime plan — and buying either then hides the other so nobody pays twice.

Name the product in `data-iap-show`. `data-iap-show="level_pack_2"` opens only that content. A bare `data-iap-show` opens as soon as anything is bought, which is rarely what you mean once you sell more than one thing. (A group name works there too, if you gave the products one.)

Never write the price into your page. Google supplies it per country, and you can change it in Play Console without rebuilding.

Nothing unlocks on a tap. Access is granted only when Google confirms the payment, so no one can cheat by tapping. Your page cannot get this wrong — it is handled for you.

No server, no database, no login. The purchase lives in the buyer's Google account. It comes back by itself on a new phone, after a reinstall, and works offline.

Refunds and lapsed subscriptions handle themselves. The content locks again and the offer comes back. Do not write any expiry logic.

One-time vs subscription is decided in Play Console, not in your code. The same `data-iap-buy` works for both; the app asks Google which kind each ID is.

Subscriptions need two activations. The base plan and the subscription itself. Missing one is the most common "my subscription doesn't show up".

Test with a license tester account, never your own money. And always install from a Play testing track — billing simply does not work on an APK you copied across.

Do not send buyers to an outside payment link for anything digital. Google requires it to go through Play, and apps that break this get removed.

Running the demo a second time

One-time purchases stay bought, so on the same account steps 2–3 will already show as owned. Before demoing again, either:

- Play Console → Order management → find the test order → **Refund and revoke**, or
- use a different tester Gmail.

Subscriptions cancel themselves quickly on a test account, so step 4 is repeatable as-is.

If something goes wrong

What you see	What it means
Play Console: "you need to add the BILLING permission to your APK"	No uploaded build declares billing. Redo Step 1 with 💰 Sell In-App Products ON, then Step 2.
"This item is not available on Google Play yet"	ID typo, or the product is not Active. For the subscription, check the base plan is activated too. Or it was created minutes ago — wait.

What you see	What it means
“In-app purchases are not enabled in this app”	💰 Sell In-App Products was OFF when you built.
The 💰 Sell In-App Products switch turns itself back off	In-App Selling needs a paid plan — Unlock, Pro, Standard, Max or Agency. The plans screen opens; buy one and the switch stays on.
“Google Play could not start the payment”	The app was sideloaded, or signed with a different key than you uploaded.
Buttons don’t appear at all	Same as above — purchases were off at build time.
Bought ad removal but ads still show	`remove_ads_lifetime` is missing from Remove Ads (IAP) → Product ID.
Nothing appears when you open the page in a browser	Correct. There is nothing to buy outside the app, so the shop hides itself. Demo on the phone.
