

Building a HomeView-style wall dashboard on Homey with Claude

Oct 26, 2026 · @Simon

A wall-mounted iPad that shows your Homey dashboards exactly as the Homey app does, with Apple HomeView's clock as a screensaver, motion wake, and the screen off at night, built in about a day with Claude Code.

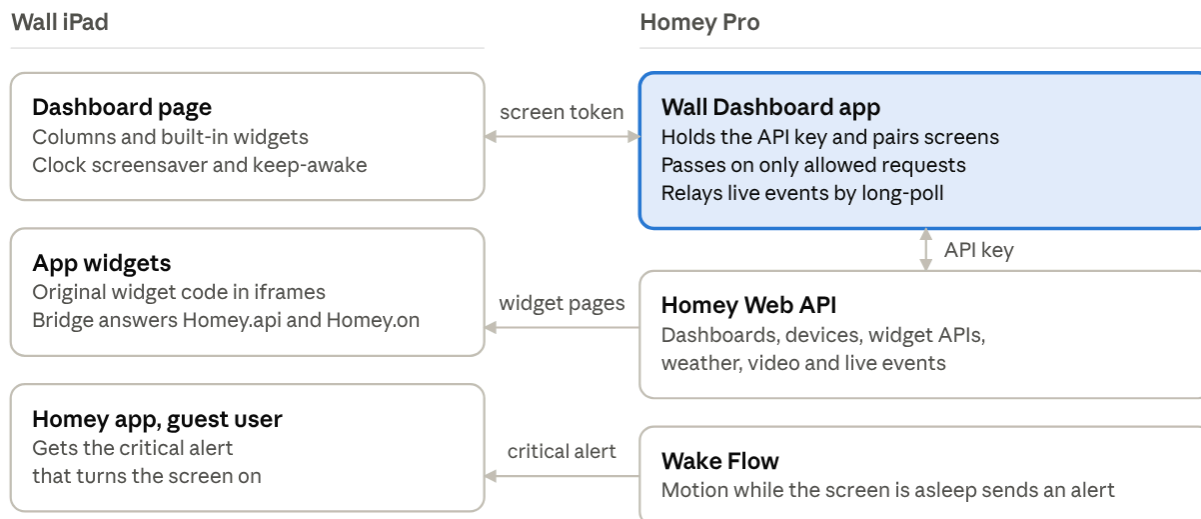
What you end up with

A Homey app of your own (we called it **Wall Dashboard**) that serves your existing Homey dashboards as a web page for a wall tablet, plus the bits the Homey app can't do.

- **Your dashboards, unchanged.** Every app widget (yours and third-party ones such as Porsche Connect) runs its original code, so it looks and works exactly as in the Homey app.
- **Homey's built-in widgets redrawn:** live camera (WebRTC), weather, speaker player, Flow buttons, timeline, low batteries, dashboard shortcuts.
- **A HomeView-style clock screensaver** after a set time without a touch: black dial, white numerals and hands, red seconds hand. A digital face is an option.
- **Motion wake:** a Homey motion sensor clears the clock instantly and switches to a chosen dashboard.
- **Screen off at night** (e.g. 22:00–08:00), woken by a silent Homey critical alert whenever someone walks past while it's asleep, day or night.
- **Live updates:** changes appear on the tablet the moment they happen.
- **Safer than a key in the browser:** the Homey API key stays on the Homey; the tablet pairs with a six-digit code and can only make dashboard requests.

How it fits together

The API key stays on the Homey; the iPad only talks to Wall Dashboard



How it fits together · iPad, Wall Dashboard app, Homey

The iPad never holds the Homey key: it pairs with Wall Dashboard and sends every request through it, while app widgets load from the Homey and run their own code.

What you need

- **Homey Pro** (2023 or later, firmware 12.3+). Widgets and app APIs don't run on Homey Cloud.
- **A Mac or PC** with Node.js and the Homey CLI (`npm install -g homey`), logged in with `homey login`.
- **Claude Code** (desktop app or terminal). It writes the code, installs the app on your Homey and tests pages in its own browser.
- **An iPad** for the wall, in Safari as a Home Screen web app under **Guided Access**.
- **A Homey motion sensor** near the tablet.
- **The Homey app on the wall iPad**, signed in as a guest user, only for the night-time wake alert.
- **A Homey API key** created at my.homey.app → Settings → API Keys. It's saved in the app's settings on the Homey and never reaches the tablet.

What we tried first, and why it didn't work

Skip these unless you want to see for yourself. Each one fails at a hard limit, not a fixable bug.

Approach	What happened	Why
A clock widget opened full screen from the Homey app (<code>Homey.popup</code>)	Opens by itself from a Flow, but as a centred window, about 60% of the screen	The popup's size is fixed and its browser has full-screen mode switched off
A clock dashboard in the Homey app	Works, but you have to switch to it yourself	Widgets can't switch dashboards, and on tablets a widget can't span columns
iPad screensaver or StandBy	Not available	iPadOS has no screensaver; StandBy is iPhone-only
Kiosker (iPad kiosk browser) pointed at my.homey.app	Nothing to show	Homey Dashboards only exist in the mobile app, not at my.homey.app
Community Web Dashboard app	Built-in widgets fine, but some app widgets broke (keypad loading only sometimes, laundry cards misdrawn)	It runs app widgets in sandboxed frames; there Homey's widget scripts fail to load, because the browser holds a full-access key it must protect

The last one pointed the way: run the widgets **unsandboxed** in your own app, and keep the key on the Homey so the page has nothing worth protecting.

Build it: prompts for Claude, stage by stage

Open Claude Code in a new, empty folder and paste these one at a time. Wait for each stage to work on your tablet before the next. Each prompt carries the decisions and pitfalls we found, so your Claude doesn't have to rediscover them. Replace `<you>` with your own name.

Stage 1: the app, pairing and access rules

Build a Homey Pro app (SDK 3, Homey Compose, compatibility $\geq 12.3.0$, plain JavaScript, no build step) with id `com.<you>.walldashboard`, called Wall Dashboard. It will serve my Homey dashboards as a web page for a wall-mounted iPad. Use the homey CLI on this machine to validate and install it, and test with `node test/*.test.js` files (no framework).

Design:

- The Homey API key stays on the Homey. A settings page (`settings/index.html`) takes a user API key from `my.homey.app`, checks it with a request to `/api/manager/dashboards/dashboard` before saving it, and shows the page's address.
- The app makes Homey Web API calls with `fetch` to `homey.api.getLocalUrl()` and that key (Bearer). Don't use the app's own owner token: it gets "Missing Scopes" for dashboards, notifications and other apps' widget APIs.
- Screens pair: the page asks `POST /pair` (public app API route) for a six-digit code; I approve it in the settings page; the screen gets a random token, and only its SHA-256 hash is stored. The settings page lists waiting and paired screens, with Approve, Decline and Forget.
- A public `POST /proxy` route takes `{token, method, path, body}`, checks the token, and passes the request on only if it matches an allow-list in `lib/access.js`: reading dashboards and app widget definitions (`/api/manager/dashboards/appwidget`), app locales, any method on `/api/app/<app>/widget/<widget>/<instance>/...`, reading devices, PUT capability values, weather, notifications, zones, reading and triggering Flows, and the video offer/keep-alive routes. Refuse everything else, plus `..` and encoded slashes in paths. Write tests for the allow-list and for pairing.

Stage 2: the dashboard page and widget bridge

Add the dashboard page at `assets/web/index.html` (served by Homey at `/app/<appId>/assets/web/index.html`), using ES modules and no framework. It pairs if it has no token (code shown full screen), then loads `/api/manager/dashboards/dashboard` and `/api/manager/dashboards/appwidget` through `/proxy` and draws the dashboards in columns, like the Homey app, with the dashboard name as a switcher at the top. Dark style, black background.

App widgets (type `app_webview`): an `iframe` at `/app/<appId>/widgets/<widgetId>/index.html?widgetInstanceId=<widget id>`, NOT sandboxed. Implement the parent side of Homey's CrossFrame protocol: messages are the string `"CF2_MESSAGE"` plus hex-encoded JSON (4 hex digits per character), `{type:"tx", event, data, callbackId}`; reply `{type:"cb", callbackId, args:[error, result]}` posted to `event.source`. Handle `getInitialData` (`settings` = widget defaults merged with `widget.data.settings`, `deviceIds`, `transparent`, `locales` from `/api/manager/apps/app/<id>/locale`, `language`, `devmode:false`, `fontScale:1`, `isDarkMode`), `ready` and `setHeight` (pixels, or a percentage meaning aspect ratio), `api` (proxy to the app's widget route), `popup` (`window.open`), `emit` and `shortPressHapticFeedback` (no-ops). Find the frame by comparing `iframe.contentWindow` to `event.source` when the message arrives. Mount the section before setting the `iframe` so it gets a window.

Pitfalls we hit, handle them up front:

- Don't put `<meta name="color-scheme">` on the page. If it differs from a widget's, the browser paints the `iframe` with an opaque white backdrop (white gaps in a flip clock, white corners on rounded widgets).
- Don't draw a card behind app widgets: Homey's widget stylesheet paints each widget's own background unless it's transparent.
- Add `[hidden]{display:none!important}`, or a panel with its own display rule never hides.
- If loading fails (Homey restarting, app updating), show the error and retry every 5 seconds: a wall tablet must recover by itself.
- Dashboard shortcuts (`dashboards_shortcuts`) switch dashboards.

Stage 3: Homey's built-in widgets

Draw Homey's built-in widget types ourselves, matched to the Homey app's look, through `/proxy`: `device_camera_live`, `weather_forecast`, `device_speaker_large`, `flows_custom`, `notifications_timeline_small`, `batteries_device_low`. Notes:

- Camera: `RTCPeerConnection` with `recvonly` video and audio transceivers; wait for ICE gathering (3 s at most); `POST /api/manager/videos/video/<videoObj.id>/offer {offer: sdp}` returns `{answerSdp, streamId}`; `POST .../keep-alive {streamId}` every 30 s. Respect `widget.data.autoplay`. When live, tapping the picture shows a stop button for 3 s. Full-screen button. Stop while the page is hidden.
- Device images and icons (`/api/image/<id>`, `/api/icon/<id>`) load without a login, so plain `<img>` works. Use CSS mask for icons.
- Speaker apps replace artwork in place (same URL, same `lastUpdated`), so reload it with a cache-buster on every track change and again 3 s later.
- Weather: use the top-level `iconId` for the current conditions; forecast items have time (Unix seconds), `temperatureMin/Max` and `iconId`.
- Flow buttons: `POST /api/manager/flow/flow/<id>/trigger (or advancedflow)`. All buttons in a widget share one font size, shrunk from 15 px to a 12 px floor until every name fits on one line.
- Keep text modest: widget titles 17 px semibold, nothing heavier than 600.

Stage 4: live updates

Add live updates. The page can't hold the key, so the app subscribes to Homey's realtime events and passes them on by long-poll.

- In the app: `HomeyAPI.createLocalAPI({address: await homey.api.getLocalUrl(), token: <saved key>})` from the `homey-api` package, then `api.subscribe(topic, {onEvent(event, data)})`. Topics allowed: `homey:device:<id>` (event `"capability"`, data `{capabilityId, value}`), `homey:app:<appId>` (an app's own realtime events, for its widgets' `Homey.on`), `homey:manager:notifications`, `homey:manager:images` (`"image.update"`, data `{id}`).
- `lib/events.js`: number every event, keep the last 500, subscribe to topics lazily, unsubscribe after 10 minutes unused. A public `POST /events {token, since, topics}` answers at once if there are events since that number, otherwise waits up to 20 s. A first poll (since null) or a number from before an app restart returns `reset:true`. Careful: `Number(null)` is 0, so check for null before converting. Test it with a fake clock.
- On the page: one long-poll loop; `subscribe(topic, handler)` restarts the open request when a new topic is added. Devices update from capability events, with a full refresh every minute as a safety net. App widgets' `registerRealtimeListener` gets callbacks on its `callbackId`. Artwork reloads on `image.update`. The timeline redraws on new notifications.

Stage 5: the screensaver, motion wake and screen on/off

Add a screensaver with settings in the app's settings page: enabled, minutes idle, up to 5 motion sensors, a dashboard to wake to, dim at night with times, 12/24-hour, clock face ("california" analog or "digital"), and "let the screen sleep at night" with times (default 22:00-08:00). Validate settings in lib/screensaver.js with tests. Screens read them through a paired-only POST /config, which also takes a status report from the screen.

- Idle detection must see touches inside widget iframes: they're same-origin, so add pointerdown/touchstart/keydown/wheel listeners in each iframe's contentWindow on load.
- Dismiss on click (not pointerdown), so the tap can't reach a widget underneath. Stop the camera stream while the clock is up and return it to its autoplay setting afterwards.
- Motion: watch alarm_motion on the chosen sensors (live). Motion clears the clock and switches to the wake dashboard; the first reading is just the current state, not someone arriving. A visibilitychange to visible (the screen coming back on) also clears it.
- Keeping the screen on: Safari only grants navigator.wakeLock after a touch, so request it on load AND again on every touch while it isn't held. Also loop a tiny silent muted inline video out of sight (generate a 2-second black 64x64 H.264 mp4 with a Swift AVAssetWriter script, under 1 KB); a playing video stops Auto-Lock. During the night sleep hours release both, so the iPad's own Auto-Lock turns the screen off. Report wake lock and video status and errors to the app each minute and show them per screen in the settings.
- Add a Flow condition card "The wall screen is asleep" (argument: any wall screen, or one paired screen). A screen counts as awake while it polls /events or reports to /config; one not heard from for 45 s has its display off, because the iPad suspends the page. Put the logic in lib/presence.js with tests. The wake-up Flow must use this condition, never a time window: a page can't turn its own screen back on, so if the screen is still asleep when the night hours end, only a Flow that fires whenever it's asleep can wake it.
- The California face: an SVG filling the screen. Rounded-square (superellipse, exponent 7) track for the minute ticks; bold white 12, 3, 6 and 9 with a short mark at the edge; thick white bars at the other hours; Apple-style hands (thin stem, then a wide rounded blade) in white; a red seconds hand and red centre pin; black background. Recompute on resize. Drift the face slightly each minute. At night, dim and warm the colours.

Stage 6: finishing touches

Give the app its own icon (assets/icon.svg, black line art on a transparent background, 96x96 viewBox: a landscape tablet with a camera dot and a clock on it) and App Store images: draw a hallway scene with the tablet on the wall showing the clock in dev/store-image.html (1000x700 SVG), and render it with headless Chrome plus sips to assets/images/xlarge.png (1000x700), large.png (500x350) and small.png (250x175).

Setting up the wall iPad

1. In the Homey app: **Apps → Wall Dashboard → Configure**. Save your API key, then copy the **Address**.
2. On the wall iPad, open the address in Safari, then approve the six-digit code under **Waiting for approval**.
3. Safari **Share → Add to Home Screen**, and open it from the icon. It pairs once more, since the Home Screen app has its own storage.
4. **Tap the screen once**, so Safari allows the keep-awake. **Paired screens** in Configure should show wake lock ✓ and video ✓ within a minute.
5. Settings → Accessibility → Guided Access: **Display Auto-Lock = 1 minute**, not Never. The dashboard keeps the screen on by day and lets go at night. Then start Guided Access in the dashboard (triple-click).
6. In Configure → Screensaver, pick the motion sensor, the wake dashboard, the clock face and the night sleep hours.

The night-time wake. Ordinary Homey pushes arrive but never light the screen, and the same went for critical alerts until the iPad allowed them:

1. Install the Homey app on the wall iPad and sign in as a **guest user** for that iPad.
2. In the Homey app's notification settings on that iPad, turn on notifications **and the separate critical alerts toggle**. Allow both iPadOS prompts. Settings → Notifications → Homey should then show **Critical Alerts** on.
3. Create a Flow. **When:** the motion sensor's motion alarm turns on. **And:** Wall Dashboard → The wall screen is asleep (any wall screen). **Then:** Homey → **Send a Critical Alert** to the guest user, text "Wake".

With Sounds off for Homey on that iPad and the volume down, the alert was silent for us and still woke the screen. Critical alerts also get through Sleep Focus. The Homey app doesn't need to be open: the dashboard stays in front.

Gotchas we hit

Problem	Cause	Fix
White gaps and corners around widgets	The page declared a color-scheme, so iframes got an opaque white backdrop	No color-scheme meta on the dashboard page
App widgets never start in Web Dashboard	Sandboxed iframes block Homey's widget scripts	Run widgets unsandboxed and keep the key on the Homey
"Missing Scopes" from the app's own API access	An app's owner token can't reach dashboards or other apps' widget APIs	Use a user API key stored in the app's settings
<code>homey app install</code> fails with ENOENT and <code>***</code> in the path	npm hides ID-like strings in its output, which breaks paths containing UUIDs	Build from a folder whose path has no long IDs, e.g. <code>~/Developer/<app></code>
Screen locks after a minute even by day	Safari refuses the Wake Lock without a recent touch	Retry it on every touch, plus a silent looping video
Screen didn't sleep at night	The page let go correctly; the iPad's Auto-Lock was still Never in Guided Access	Guided Access Display Auto-Lock = 1 minute
Night notification arrives but screen stays dark	Normal Homey pushes don't wake the screen	Homey critical alert, allowed on the iPad itself
Sonos artwork stuck on the old cover	The speaker app replaces the image in place	Reload on track change and on <code>image.update</code>
Pairing panel stayed over the dashboard	<code>display:flex</code> overrode the hidden attribute	<code>[hidden]{display:none!important}</code>
A screen's first live request waited 20 s	<code>Number(null)</code> is 0, read as a real position	Treat null as a first poll
"Unmute Soundbar" cut off	Fixed font size in a narrow column	Shrink all buttons together until every name fits
Dark screen in the morning; walking past didn't wake it	The wake Flow only ran 22:00–08:00, and a page can't turn its own screen back on at 08:00	A "wall screen is asleep" Flow condition in place of the time window

Sources

- [Homey Apps SDK: Widgets](#): the widget API the bridge answers
- [Web Dashboard for Homey \(GitHub\)](#): the community app that showed how Homey's CrossFrame protocol and WebRTC camera API work
- [Homey support: Use the Homey Web App](#): Dashboards are mobile-app only
- [Homey support: Start a Homey Dashboard in kiosk mode](#)
- [Shane Whatley: I Built My Own Apple HomeView... Using an iPad!](#): the Home Assistant version that started this