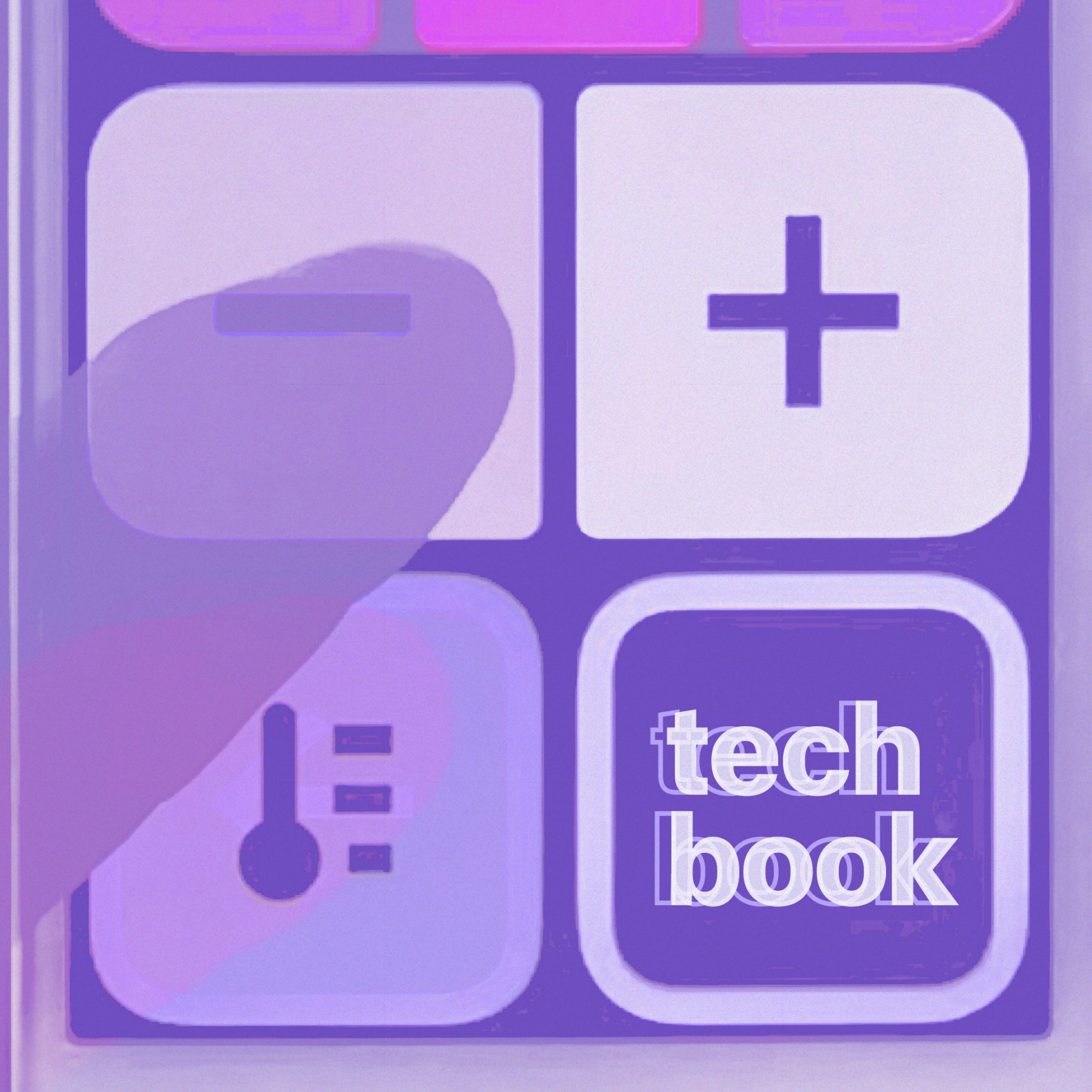

tech
book

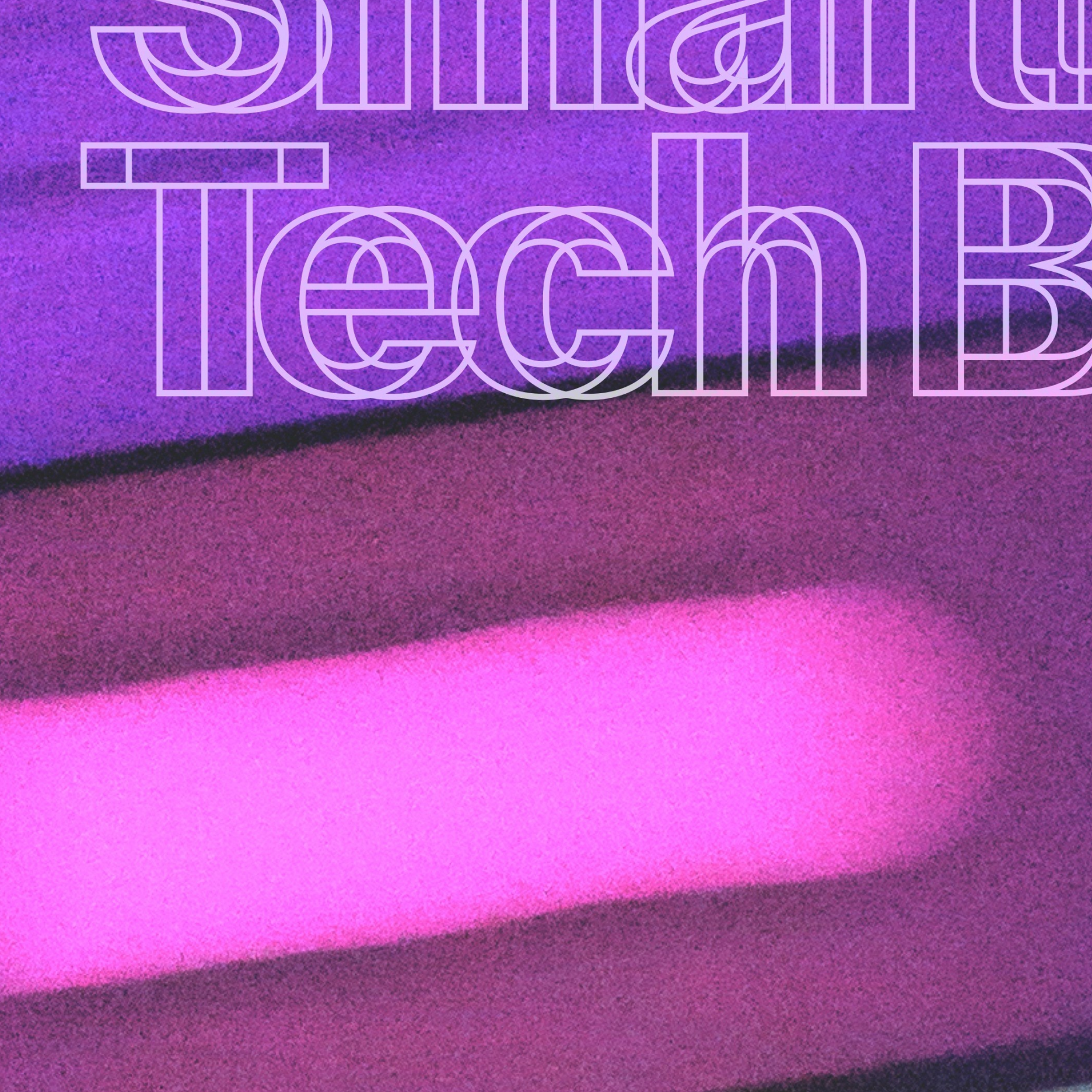
"...At its heart is the idea that the power of a system comes more from the relationships among programs than from the programs themselves."

The UNIX Programming Environment, 1984

JM

JMLLC-SMTBK-20250212
February 2025

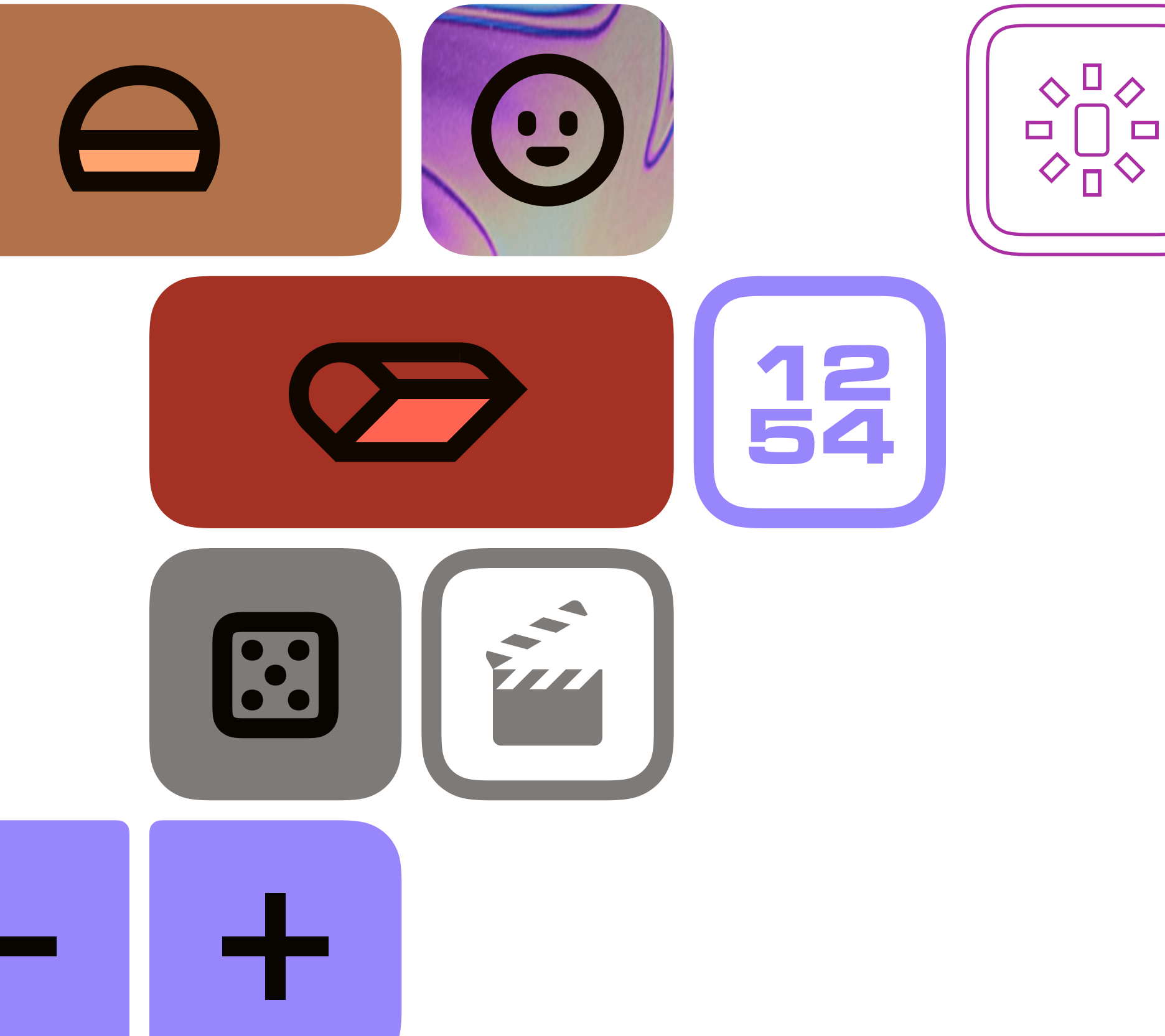
Diagrams were made in Sketch. Page layouts were designed in Sketch and ported to Affinity Publisher for press and qpdf for web.


Tech B

Smarthome Book

Smarthome's inner workings from UNIX to UX

John Matula


12
54

Smarthome is the
connected home system I created
that gives me easy, direct, daily control
over lights, plugs, and the air conditioner.

They fill a need that smart homes struggle with:
a way to fully control smart bulbs and devices,
without using my phone or confusing my friends.

With Smarthome, I tap buttons on the wall.
Those buttons work a lot like wall switches.

In fact, Smarthome has replaced all my switches.
I've only done that because the utility it offers
is solid and simple, for me and for others.
It keeps it up day after day.

Clarity? Longevity? In a smart home?

In this book, I explain the parts, thinking,
design, and strategy behind Smarthome.

Click or tap to jump to a page.

Content

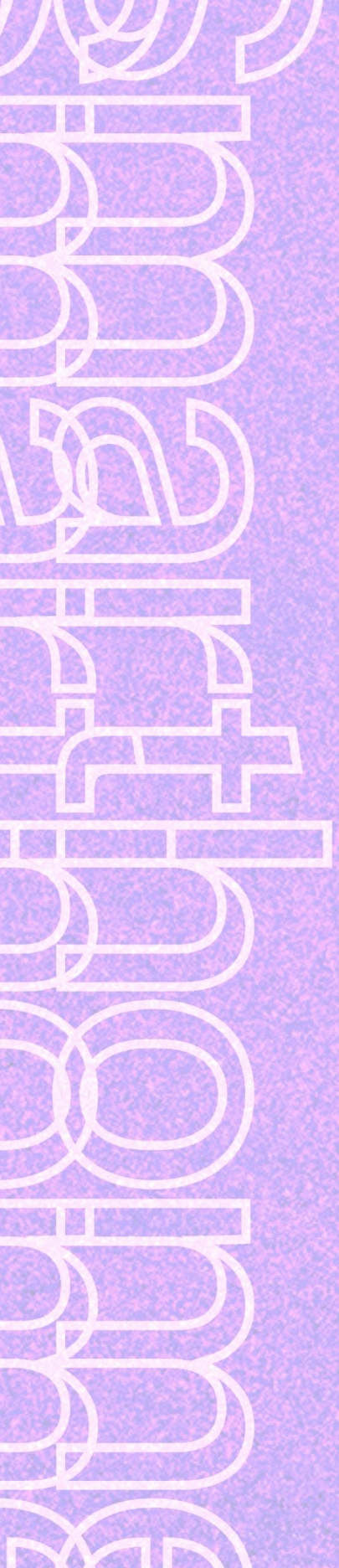
Overview

1 A compliment to life at home

Smarthome should be a system for everyday use. It should support the variety of life, handling joyful and stressful times alike. The system should remain neutral and unobtrusive. It should match the reliability of plastic wall switches. It should also work as calmly as those switches, presenting trustworthy control of the system.

2 For use by friends

Smarthome should be as clear to my guests as it is to me. The capability it offers should be rich enough for daily use without being cumbersome. It should offer control without apps, logins, or instructions. It shouldn’t need a voice assistant to work.



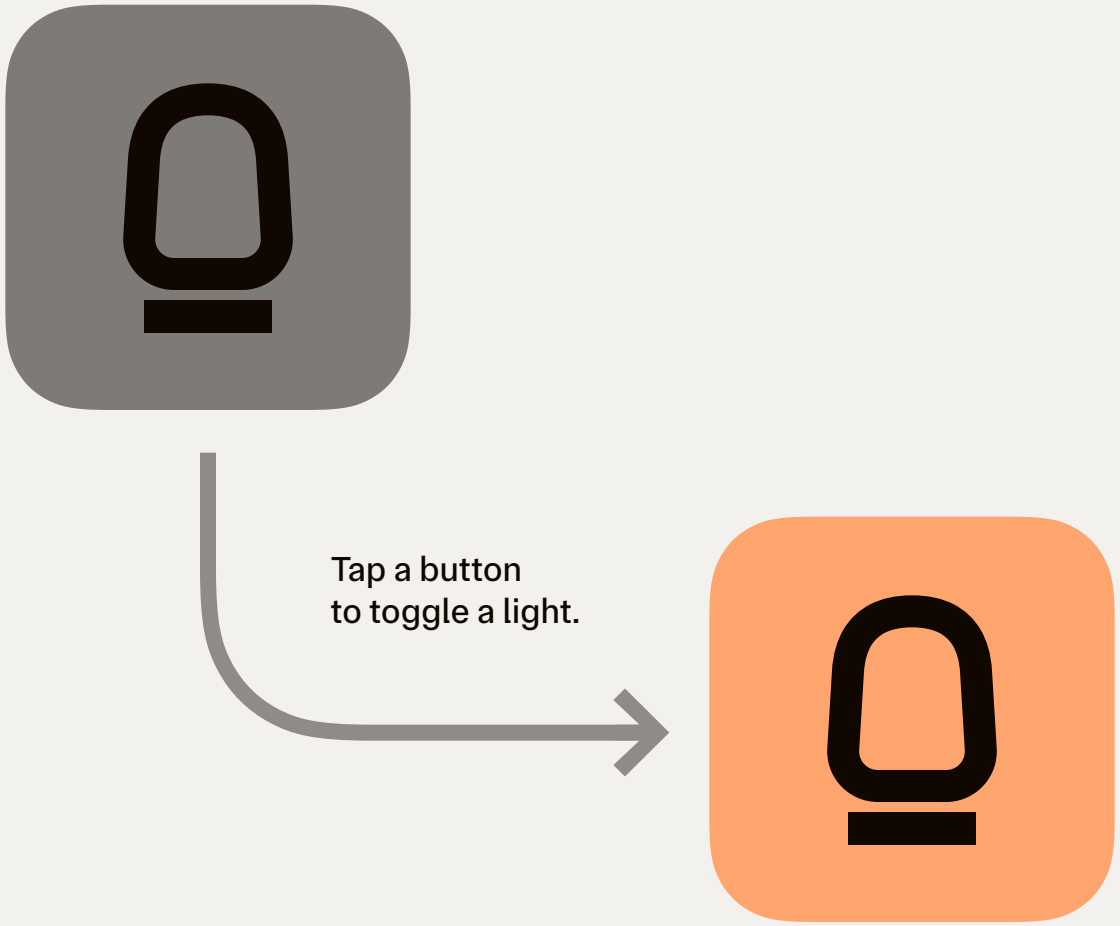
Calmly unbreakable

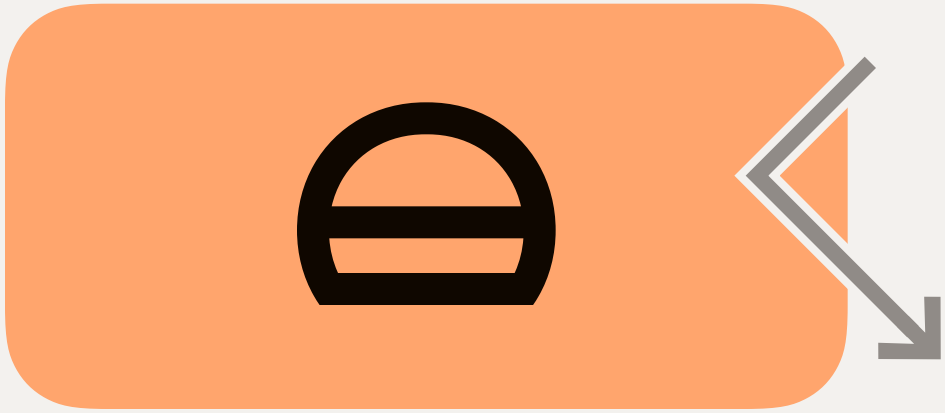
Smarthome should continue working when other things at home are not. The system should be as helpful as possible, even during outages. It should interpret problems and try to correct them. When a situation is resolved, the system should resume working without resets, reboots, or repairs.

Ease without edicts

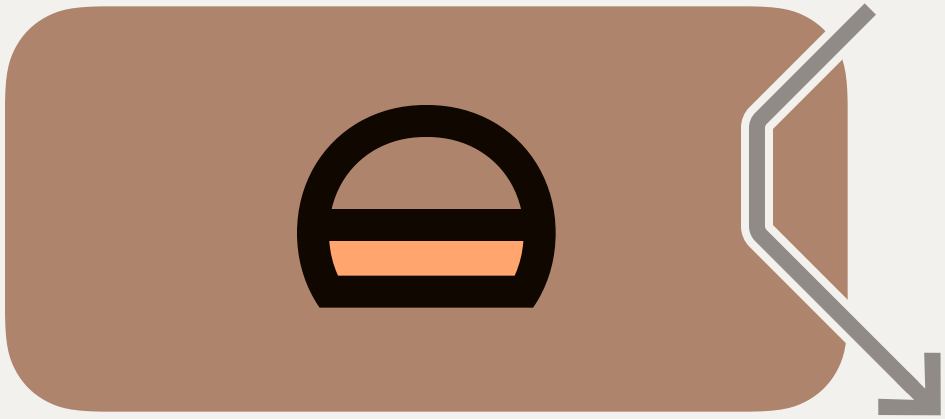
Smarthome should be as flexible as it is capable. It should do so without relying on rigid, overeager, or excessive automations. It should respond quickly and efficiently, even if its controls have sat idle for hours on end. Things should just work.

Buttons are the interface of Smarthome.
They control lights and devices across the apartment.
Each one responds to taps in a uniform way.

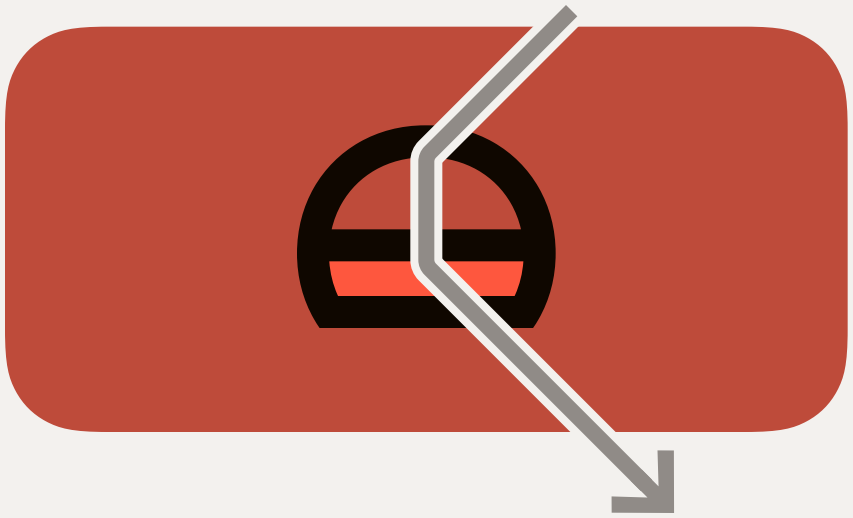




Tap a button
to toggle a light.



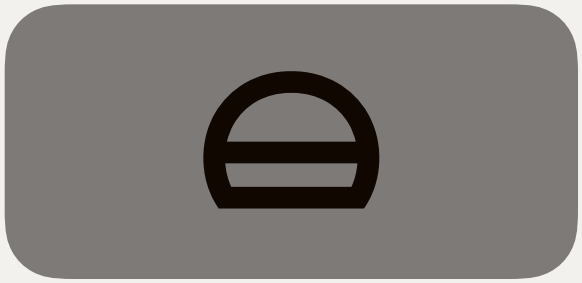
Tap and hold
a button to brighten
or dim a light...



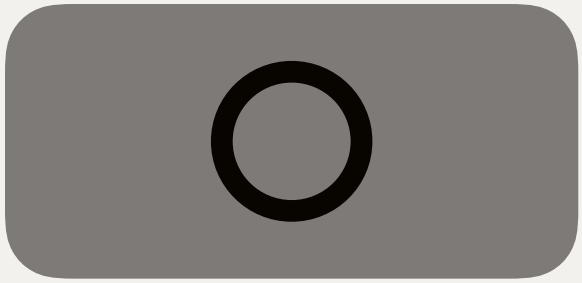
...or change its color.

Buttons are grouped together to form
a button panel. It is shown on a touch screen,
placed on the wall to serve as a touch wall switch.

cupboards



dining table



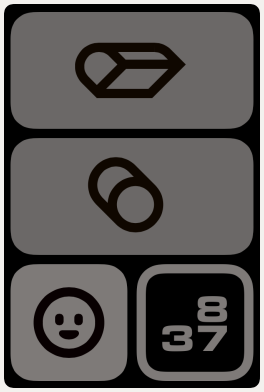
kitchen

countertop

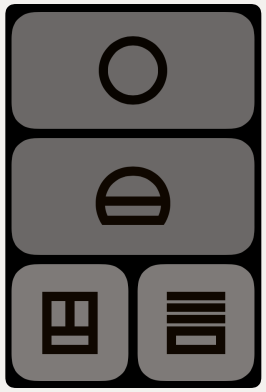
button panel for
kitchen and dining with
corresponding lights

Each room has one button panel, offering quick and direct control for the devices in that room.

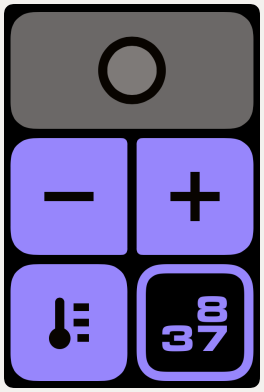
1 Bedroom



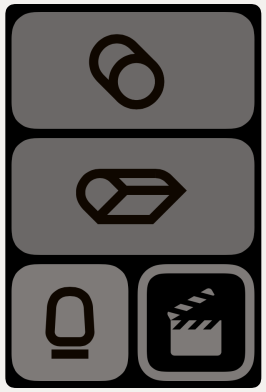
2 Kitchen and dining



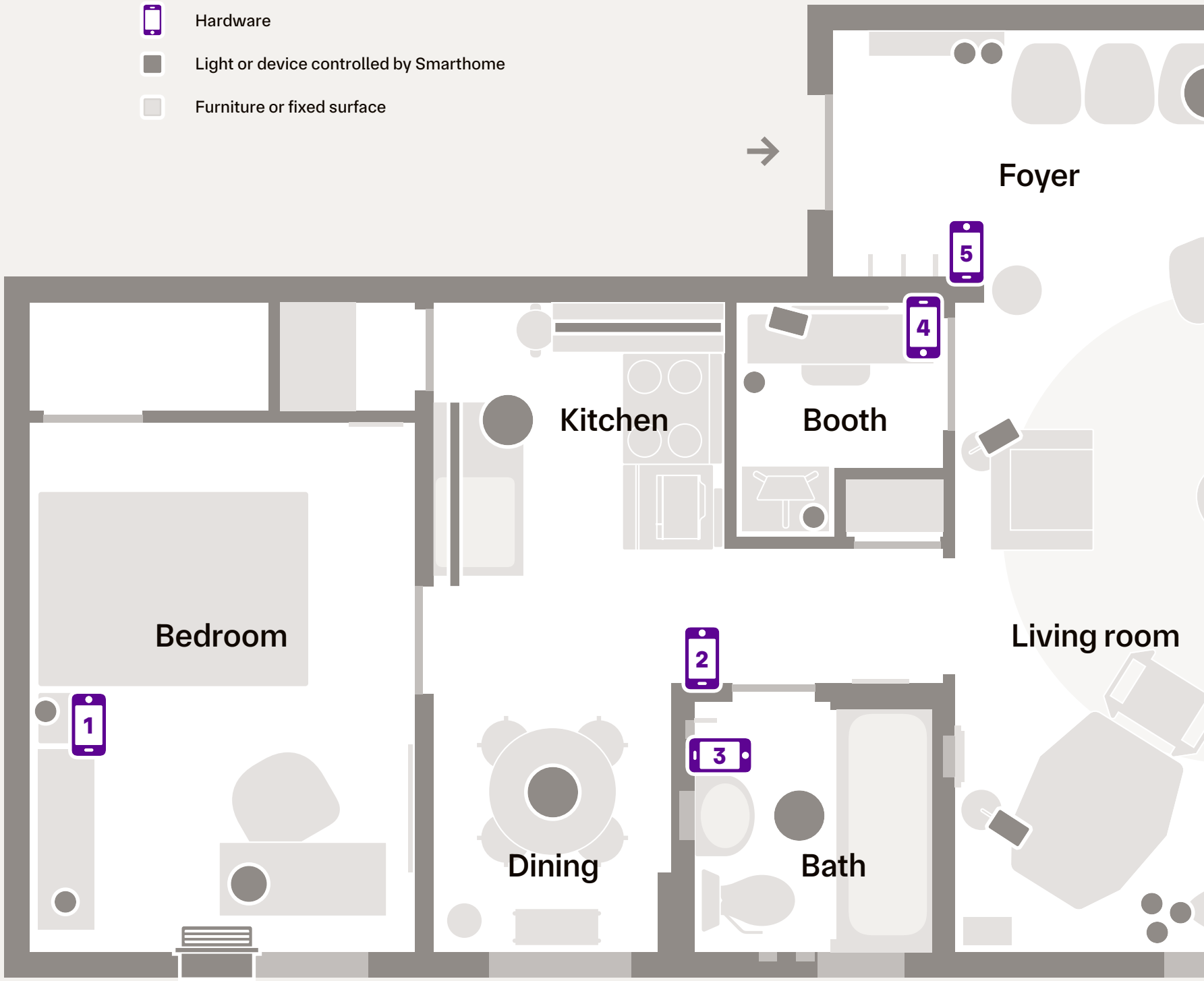
3 Bathroom



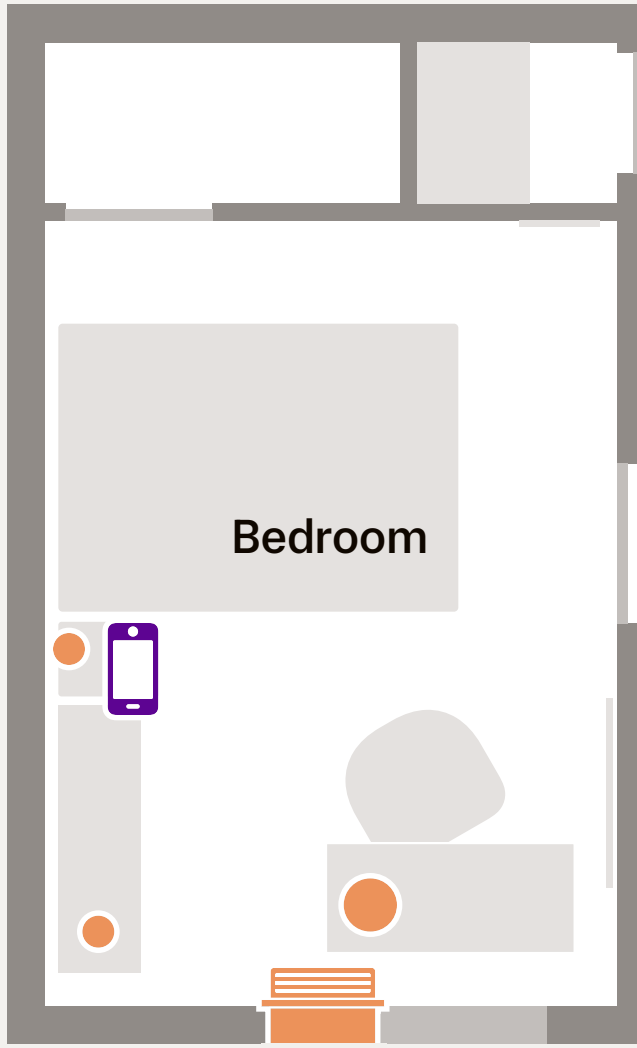
4 Recording booth



5 Foyer and living room


Hardware
Light or device controlled by Smarthome
Furniture or fixed surface
Foyer
Kitchen
Booth
Bedroom
Living room
Dining
Bath
1
2
3
4
5

Together, the five panels control the complete set of lamps, ceiling lights, smart plugs, and window air conditioner in the apartment.

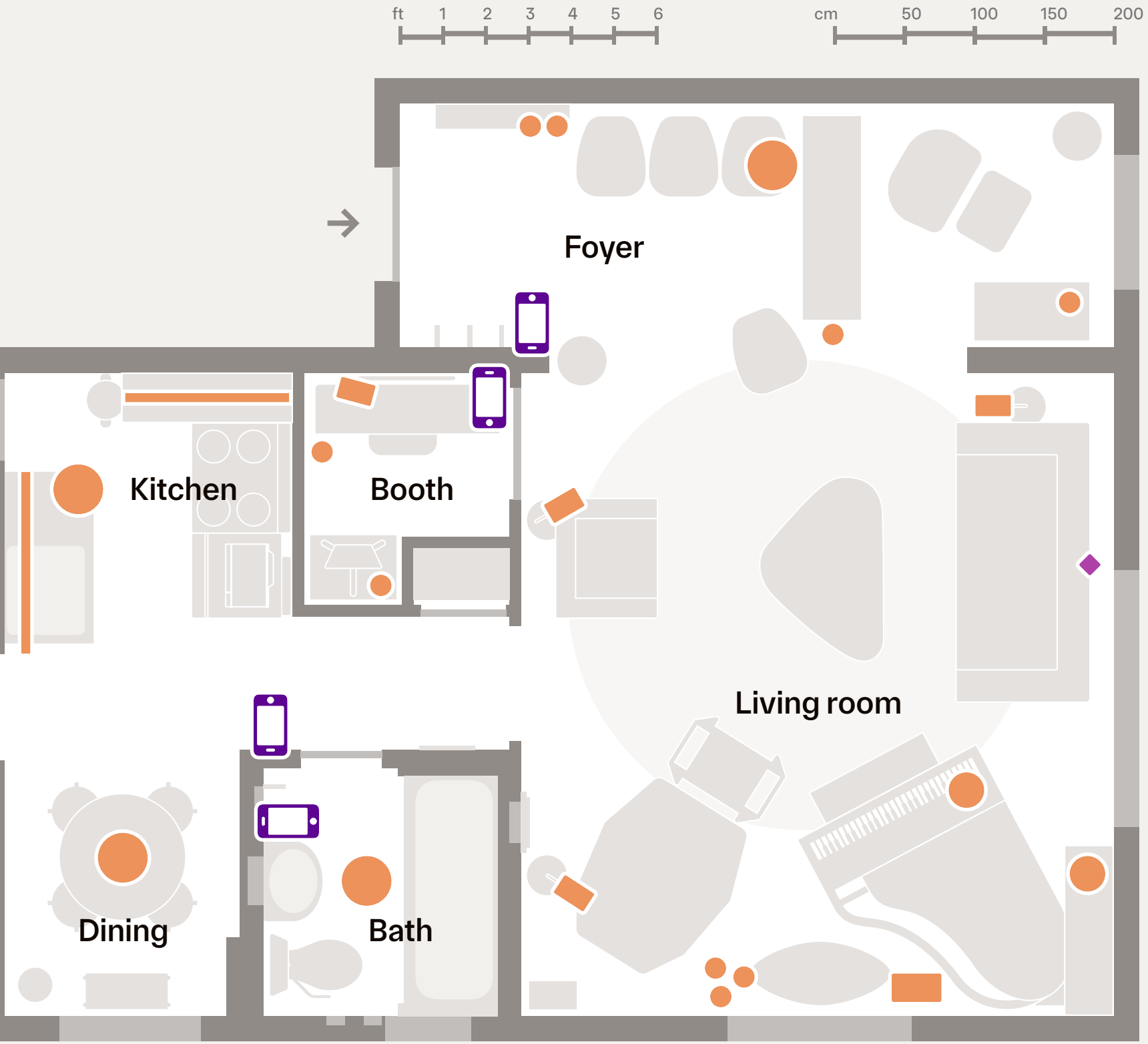


Hardware

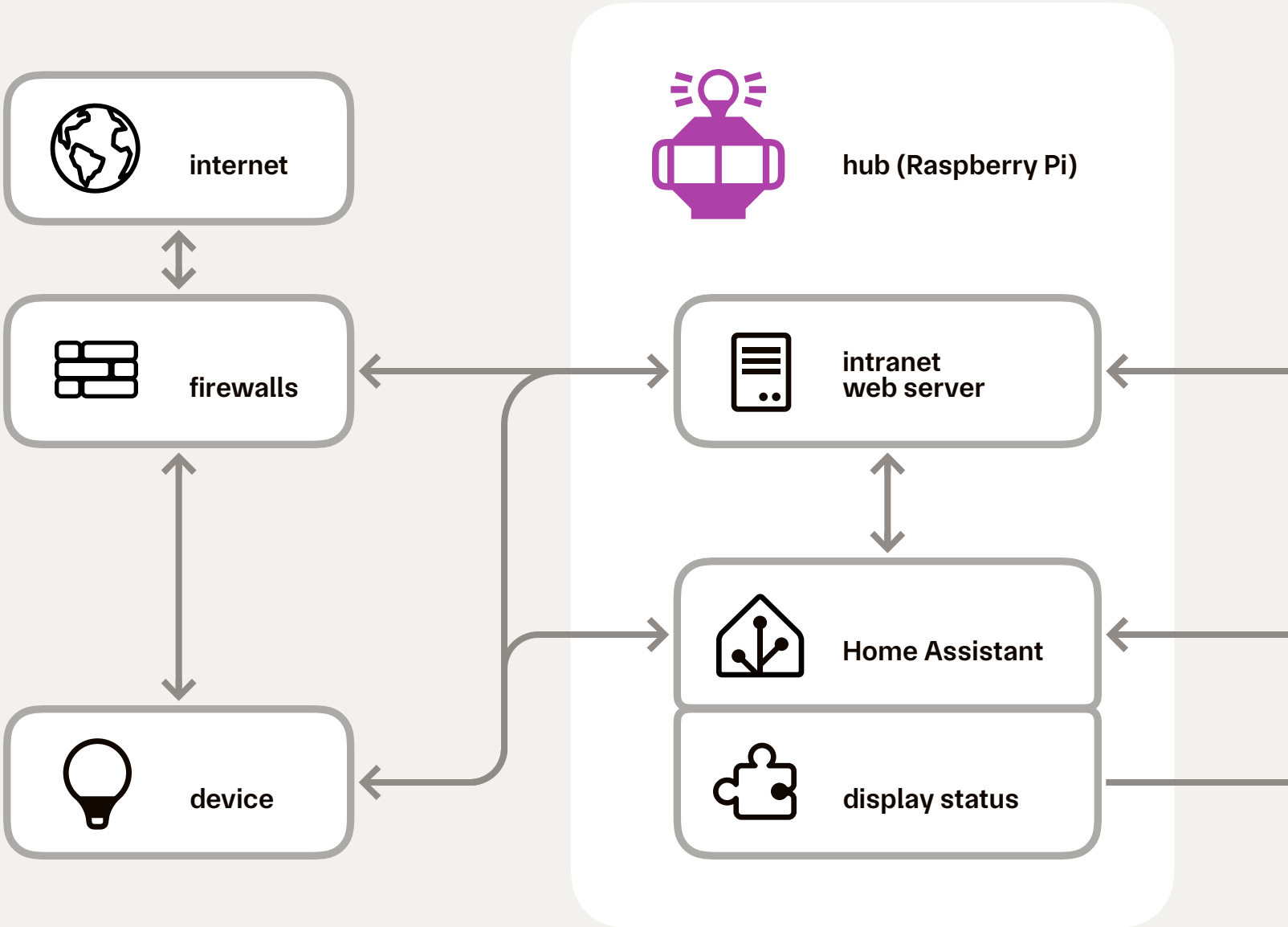
Hub

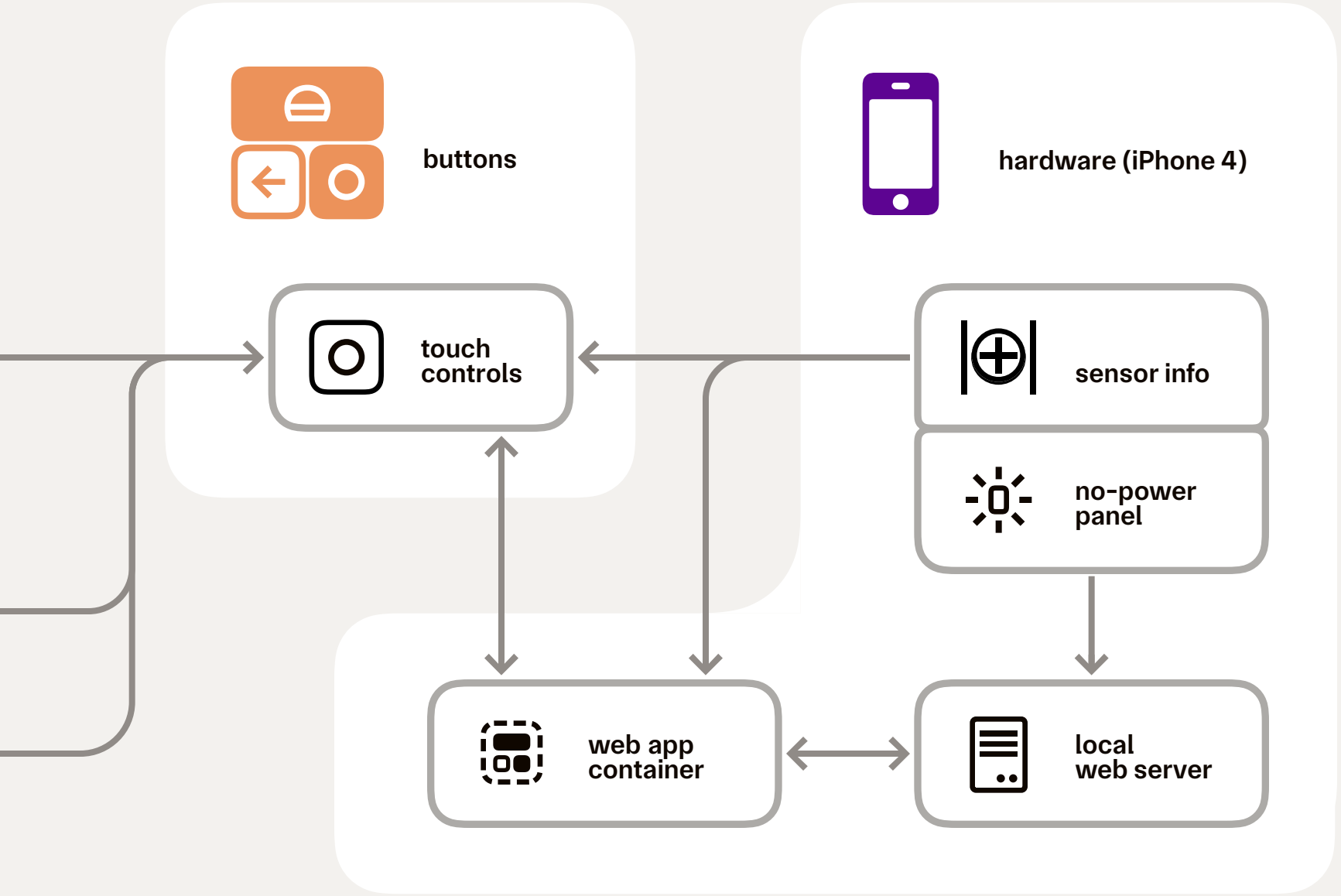
Light or device controlled by Smarthome

Furniture or fixed surface

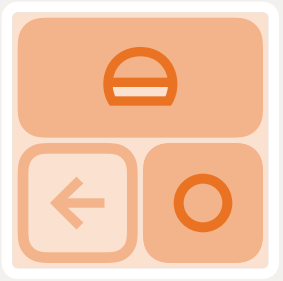

ft 1 2 3 4 5 6
cm 50 100 150 200
Foyer
Kitchen
Booth
Living room
Dining
Bath

The button panels send commands to a hub, which administers devices on behalf of buttons. Along with the hardware, each part works to support a responsive and clear system.




buttons
hardware (iPhone 4)
touch controls
sensor info
no-power panel
web app container
local web server

I designed Smarthome like a product, using goals to determine how it would be built. The result is a system that provides rich, robust control that's still easy enough for guests to use.



Buttons are presented in panels, and those panels are web apps. The panels are designed to be always available. Like their plastic switch counterparts, the buttons take a single action — a tap — to control light and device settings. Auxiliary control is offered with tap-and-hold gestures. A button is designed to be legible and respond quickly, clearly indicating the device it controls and the status it currently has.

Button panels are view controllers. They connect to a back-end hub, which relays commands on behalf of buttons and returns a full system status back to them. The panels interpret that raw status and present a simplified version of it through button states and indicators.

The web apps are built for long-term use. Feedback and rendering is pushed to the GPU, leaving the CPU to handle timers and requests. Repaints occur only when needed, mimicking one-way binding found in heavier front-end frameworks. As simple HTML pages with SVG controls, the apps are very performant. The apps smoothly handle interaction after hours of sitting idle.

The button panels also act as household tools. Depending on the panel, buttons open up air conditioner controls, clocks, and recording tools. These tools exist as separate panel apps that are injected into the main app instance.

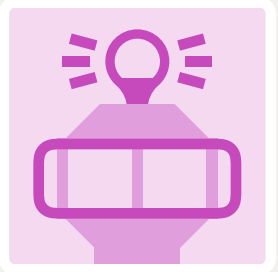
Button panel apps run on touch screen hardware. That hardware is the iPhone 4. Smarthome uses this specific device and model for its ideal combination of display technology and reliable operating system. They are inexpensive, enabling Smarthome's panel-per-room design.

Each iPhone 4 is jailbroken, transforming it from an iOS device into generic, Unix-like hardware. Button apps are able to read battery and sensor info. Hardware button actions are replaced with custom actions, like clicking the home button to show system-wide controls. There are OS-level optimizations as well, stopping unneeded daemons and hiding UI components like Notification Center and the lock screen.

Screens brighten and dim to match the brightness of a room. The hardware reports its ambient light sensor value, which a custom routine reads in to adjust the backlight according to a custom curve. They are also tuned to reduce blue light and appear even warmer at night. These display adjustments run on the hardware itself, leaving panel web apps performant for interactions.

The hardware performs automatic maintenance regularly. Nightly, panel screens dim. Weekly, the panels restart. Occasionally, they run an image retention refresher. This maintenance happens automatically overnight, and the routines are easily paused and interrupted.

Physically, the hardware is a plain rectangle made of glass and steel. They are light enough to be affixed to surfaces with removable adhesive, making them suitable for rented homes. Even as a non-hardwired, temporary system, the hardware blends in with my home's décor.

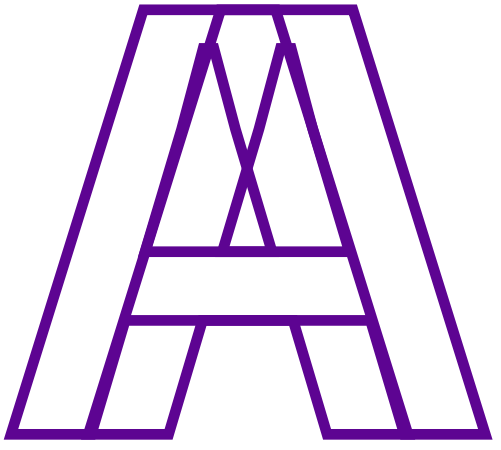


Home Assistant is Smarthome's hub. It is an open-source smart home platform that allows devices of different kinds to work compatibly. The hub gives all those devices a common set of control instructions.

My system relies on the robustness of Home Assistant, particularly the work of its maintainers and contributors, to work as well as it does. Smarthome extends that work with the addition of a custom extension, a display status. This additional status lets buttons appear steady when a brief disconnection or a delayed response occurs.

Home Assistant runs locally on a Raspberry Pi, accessible at home over the local intranet. The button panel web apps are hosted there too, also locally accessible. The button apps and the legacy web features they are built with are deliberately hosted separately, which allows the display hardware to run the apps properly.

Hub



At the center of Smarthome is Home Assistant, a smart home platform that prioritizes local control, privacy, and extensibility. The platform is a popular choice for homes big and small. In mine, it's used to take in and reply to Smarthome's button taps.

Part of Home Assistant's appeal is its extensibility. The hub is open and community-built, so unlike some others, it works with devices across different brands. It also lets Smarthome use custom, behind-the-scenes logic that help the system feel snappy.

The hub acts as the single source of data in the system, from light statuses to color presets. Actions and responses are sent over the local network. If the internet goes out, the hub and the devices it manages continue to work.

Smarthome uses web apps that also live on the hub. They are separate from Home Assistant. The apps communicate with the platform, and they respond to the panel hardware's sensors.

Managing the hub

Home Assistant offers its own UI, called Lovelace, and it is beyond robust. It has workflows to add new devices, manage scenes, assign rooms and groups, create automations, and more. You can think of it as the administrator's view of Smarthome.

Lovelace also offers day-to-day control of devices, through dashboards. By default, everything that Home Assistant manages gets thrown into one. That can be a lot, and Lovelace lets you make custom dashboards that are more understandable. Keep going far enough, and you can come up with screens similar to Smarthome's button panels.

Smarthome uses a custom-built UI for its buttons, however, as Lovelace uses modern features that aren't supported on the iPhone 4 (it's understandable). Smarthome instead uses custom web apps I built that run alongside Home Assistant and hook into its API. Smarthome sends a button tap's action and Home Assistant handles the device command.

Display status

Devices occasionally lose their connections to the hub. These short and infrequent "blips" occur when Wi-Fi signal in the house is spotty or out of range. Say a light is on and a blip occurs, where the bulb is still physically shining but Home Assistant can't reach it. In these cases, Home Assistant changes the device's status to "unavailable." When it re-establishes a connection, it snaps its status back to "on" and once again reports the true state of the device.

Home Assistant shows these status changes as they occur, even in typical blips that last for a few seconds. That makes them helpful from a management point of view but results in buttons that "flicker" on Smarthome's panels. Smarthome needs to operate more calmly, even if it's only for show during blips. It adds a "display status" mechanism to support this.

The display status is a timed comparison of a device's actual and intended status. Its intended status comes from the most recent button press, telling it what it should do. Its actual status comes from whatever the device reports to Home Assistant. If something acts up, Smarthome keeps showing the intended state until the mismatch in statuses is resolved. If the timed "grace period" expires, Smarthome gives up and rolls back the display state to the actual "unavailable" state.

Returning to that light, when it's on and suddenly blips, its actual status flips to "unavailable." Its intended status is still "on," so its display status (and the panel) remains "on" as well. Meanwhile, Home Assistant and Smarthome both try to reconnect to the blipped light. When the status returns to "on" the mismatch is resolved, and the buttons continue to show the unflinching state of "on."

This also helps devices that are slow to report their actual status. The air conditioner, for instance, responds to commands instantly but can take up to two minutes to actually reply to Home Assistant. Smarthome shows the intended state immediately as the unit takes immediate action (the audible "clunk" is the real-world confirmation). Eventually, the air conditioner reports its status, matching what Smarthome's buttons are already showing.

Color presets

Home Assistant keeps track of three color presets and calibrated Kelvin temperature values. These settings are a mix of system-wide and per-device values, which I set as I add a new bulb to the hub.

Smarthome uses a daylight and warm white light that has been calibrated, so that a bulb doesn't look flourescent blue or downright orange. If a specially tuned one isn't available for a given light, then it falls back to the generic, system-wide settings I've set.

Similarly, the three color presets are shared across the whole system. These color presets (specialty colors other than white) are setup through Home Assistant as custom variables. Each preset has an actual (XY) color value and a display (RGB) value. Smarthome's buttons show the display color and the actual color gets sent to Home Assistant to relay to the device. Splitting the color into two helps the case where a device's setting looks totally different from its actual color ("it looks purple but the screen says blue?").

Scenes and automations

Most rooms are small enough for Smarthome to offer per-device control. The living room has several lamps, though, so its button panel controls the whole room. Smarthome does this by making use of scenes, capability built into Home Assistant to set multiple devices at once. Just like individual lights, Home Assistant sets all applicable devices according to the preset, and the intended scene is captured by the display status system.

Smarthome reflects my preference for direct control over automations, but a couple are included to fix small annoyances. For instance, each time I turn on the air conditioner, it sets itself to "auto fan" mode (the kind where the fan starts and stops over and over). On the hub, an automation catches this and kicks it back into steady-on fan mode.

Notably, lights are not automated: I don't want to be spooked by scheduled lights suddenly turning on around the house.

Web server

Outside of Home Assistant, the other major component on the Raspberry Pi is the intranet web server. This basic web server hosts the web app files that the display panels connect to. They are not internet-as-in-WWW-hosted but intranet-as-in-locally-hosted.

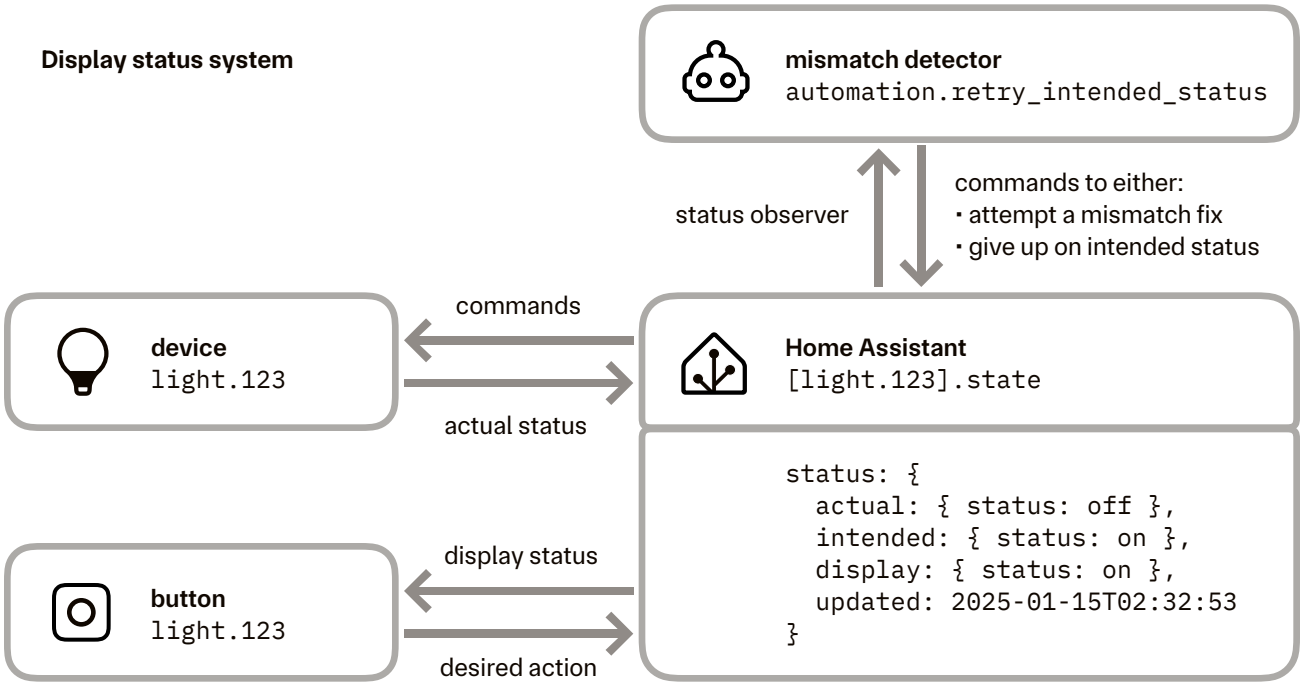
The app files are developed on my laptop, and the production files are uploaded to the server via SSH. I can test the button panels back on my Mac by pointing a browser to the Raspberry Pi and opening one.

Maintenance

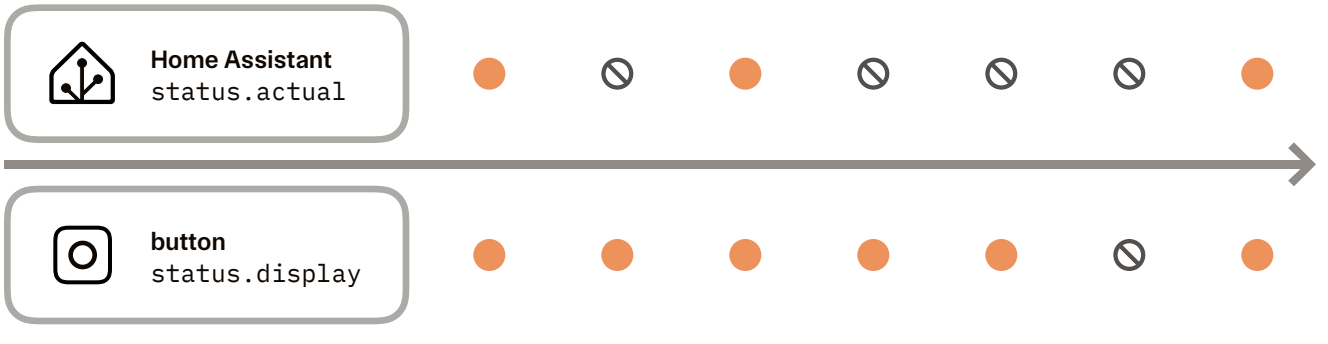
In addition, the web server hosts configuration profiles for the hardware. These are font files and dark interface elements that are downloaded and applied when setting up panel hardware for the system.

As needed, I update and review the hub. Updates remain a manual process in order to give me a chance to review them beforehand.

Display status system



Status for a “blippy” device



Hardware

Smarthome is controlled by buttons in button apps, and those apps run on hardware: multi-touch screens that have excellent display clarity and brightness, wireless connectivity, power management, security, and sound.

Most of us know them by their given name, the iPhone 4. While they no longer connect to cell towers, all of its other specs work well for applications like Smarthome. At eleven years old, there are plenty of still-good ones to be found, and for a good price secondhand.

The phones are jailbroken, which means they can be opened up to do all sorts of non-phone things. It turns the iPhone 4 into an all-in-one Unix-like computer with a touchscreen. I tend to call them “hardware” or “panels” in this book to reflect how Smarthome uses their fully-opened-up abilities.

In daily use, each panel manages itself through the use of its sensors. Reading things like the ambient light sensor and power status, it can adjust itself to look and work as best it can.

Display

The button panel apps make full use of the hardware's high-resolution display. They have good contrast and accurate color, even when viewed at angles.

The displays are backlit LCD panels. Measured, they range from less than a nit of brightness to around 570 nits of brightness. In practice, that means Smarthome will only cast a glow when a room is pitch black, and it will get bright enough to be legible in a sunbeam. The display's brightness diminishes when looking at it from an angle, which slightly hinders daytime legibility but actually improves nighttime use. (In effect, the display "channels" its brightness straight ahead, throwing less unwanted light off to the sides.)

Overall, the image that the display provides is fantastic, even at odd viewing angles. The display is readable at angles up to 80° left or right of center, as well as from diagonal, horizontal-and-vertical viewing angles. The kitchen panel, for instance, is in a hallway; the display technology allows it to remain usable as I walk by it.

The button panel apps and hardware work in concert, working together to create an integrated appearance. One example is the black bezel around the LCD panel's edge. Look closely, and you'll see this black border wrapping around the entire "picture" of the screen. The button apps use this to their advantage by placing controls right against that bezel edge. The background is black, so it blends in. The hardware, meanwhile, is accurate enough to catch taps even when they're right near that bezel.

The display does have one drawback: its tint. The color they give off is uncomfortably blue and needs adjustment for nighttime use. It also needs to adjust its backlight automatically throughout the day. These things are possible with the capabilities provided by the jailbreak, and these things are set up once as part of the hardware's initial setup.

Sensors

The foyer/living room, recording booth, bathroom, kitchen, and bedroom each contain a panel. Each one runs a button panel app, which connects to the hub to talk to devices. It's the hardware's job to support those apps, ensuring the hardware is running efficiently and in a way that suits the room nicely.

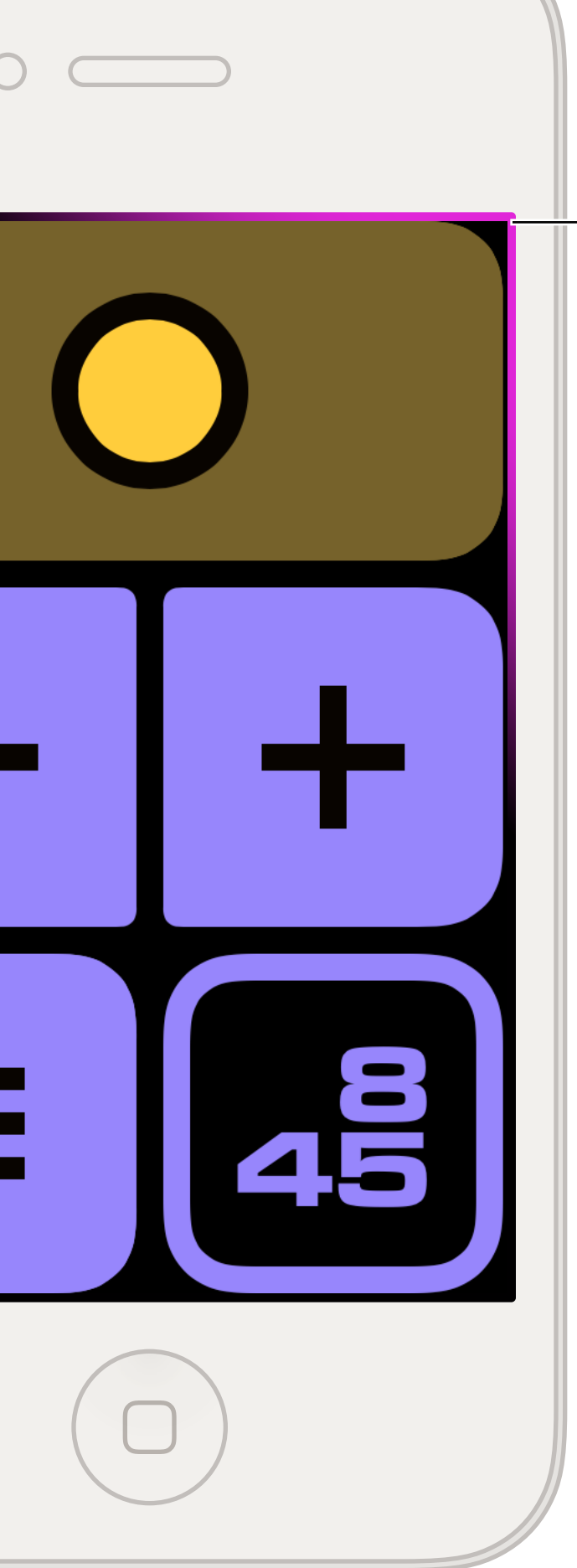
The hardware is able to do that by measuring its surroundings through its sensors. It interprets values like room brightness, orientation, and location, and responds to them as necessary.

For its display, the hardware checks the ambient light sensor, compares it against a mapped value, and applies that value gradually. (Partly cloudy days are no big deal.)

The display's color uses geolocation to find sunrise and sunset times. As sunset approaches, the displays smoothly switch to a tint that's (even) warmer than their daytime setting. As sunrise times adjust throughout the year, the displays adjust with it.

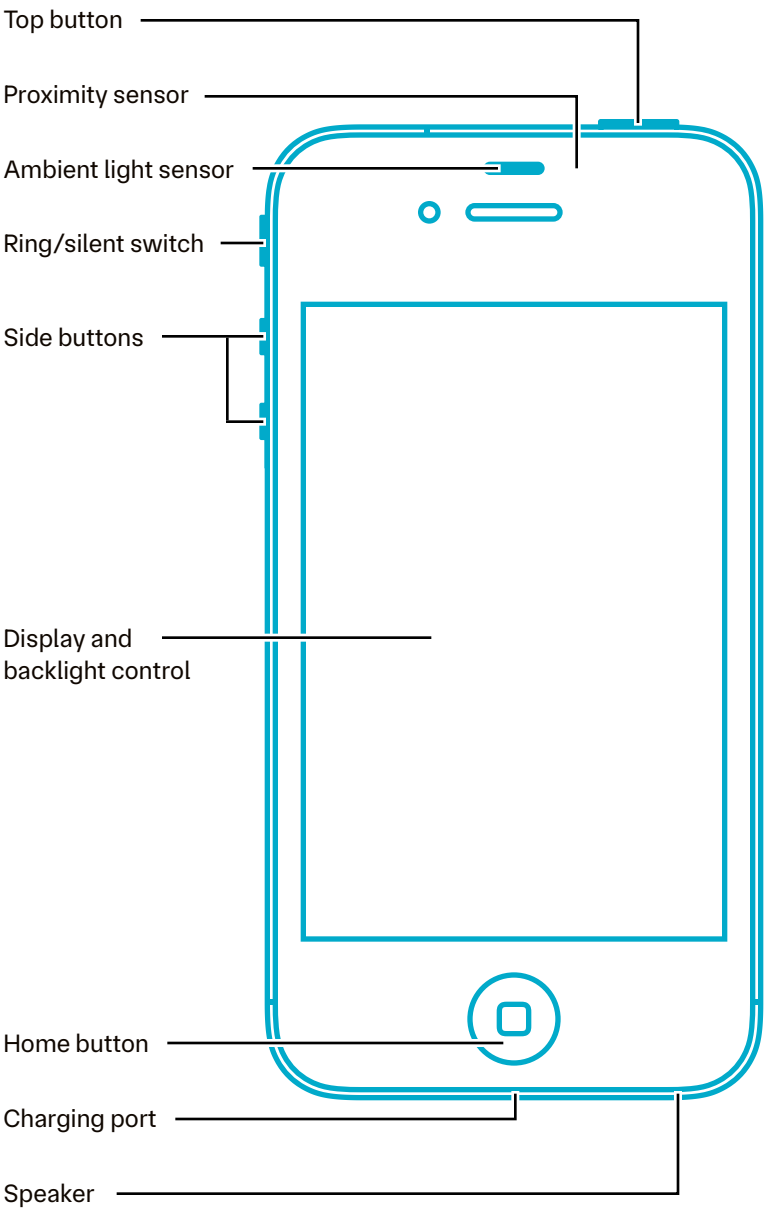
These adjustments happen automatically, managed by apps and processes that run on the command line.

Back to Contents



The black bezel, partially highlighted, blends in with the button panel's interface.

Components used by Smarthome



Hardware buttons

Panels have five built-in buttons that Smarthome uses. In iPhone terms, that's the sleep/wake (top) button, volume (side) buttons, ring/silent switch, and home button.

The sleep/wake button on the top-right of the panel works like it does on an iPhone: it puts the display to sleep and wakes it up again. The button panel app immediately returns when the display is woken up (no slide to unlock).

The volume buttons on the side set screen brightness, overriding readings from the ambient light sensor. Above those, the tiny ring/silent switch shows a service menu when toggled back and forth.

The home button brings up Smarthome's system-wide view, sort of like a home screen for Smarthome. All of these actions, even messing with the home button action, are possible because the hardware is jailbroken. When a hardware button is clicked, it's caught by the system and redirected for use by Smarthome. It does this without needing to modify the hardware itself.

Power outages

Panel hardware runs run off of AC charger power by default. When electric cuts out, the hardware switches over to battery. This event triggers Smarthome to exit the current app and open the no-power app. It can be accessed because it is a web app that lives on the hardware itself, always accessible as long as the panel is turned on.

The app is a simple clock with a fullscreen flashlight. It displays battery remaining by accessing the battery sensor directly. It also cranks up the brightness, overriding the automatic settings, when the display is acting as a flashlight.

Despite showing different buttons, the hardware works as normal. Screen brightness and color tints are still adjusted, just a little less eagerly to save battery. Hardware buttons are still captured for use by Smarthome, which the no-power app uses for emergency shortcuts to the flashlight.

While it's in this mode, the battery has enough capacity to last for a decently long while. In testing, fully-charged hardware running Smarthome's no-power mode lasts up to a day when the screen is fully on and bright. On standby, with the display dimmed, it lasts well over a week. Smarthome relies on iOS's built-in power management to handle all this, without the need for additional refinement.

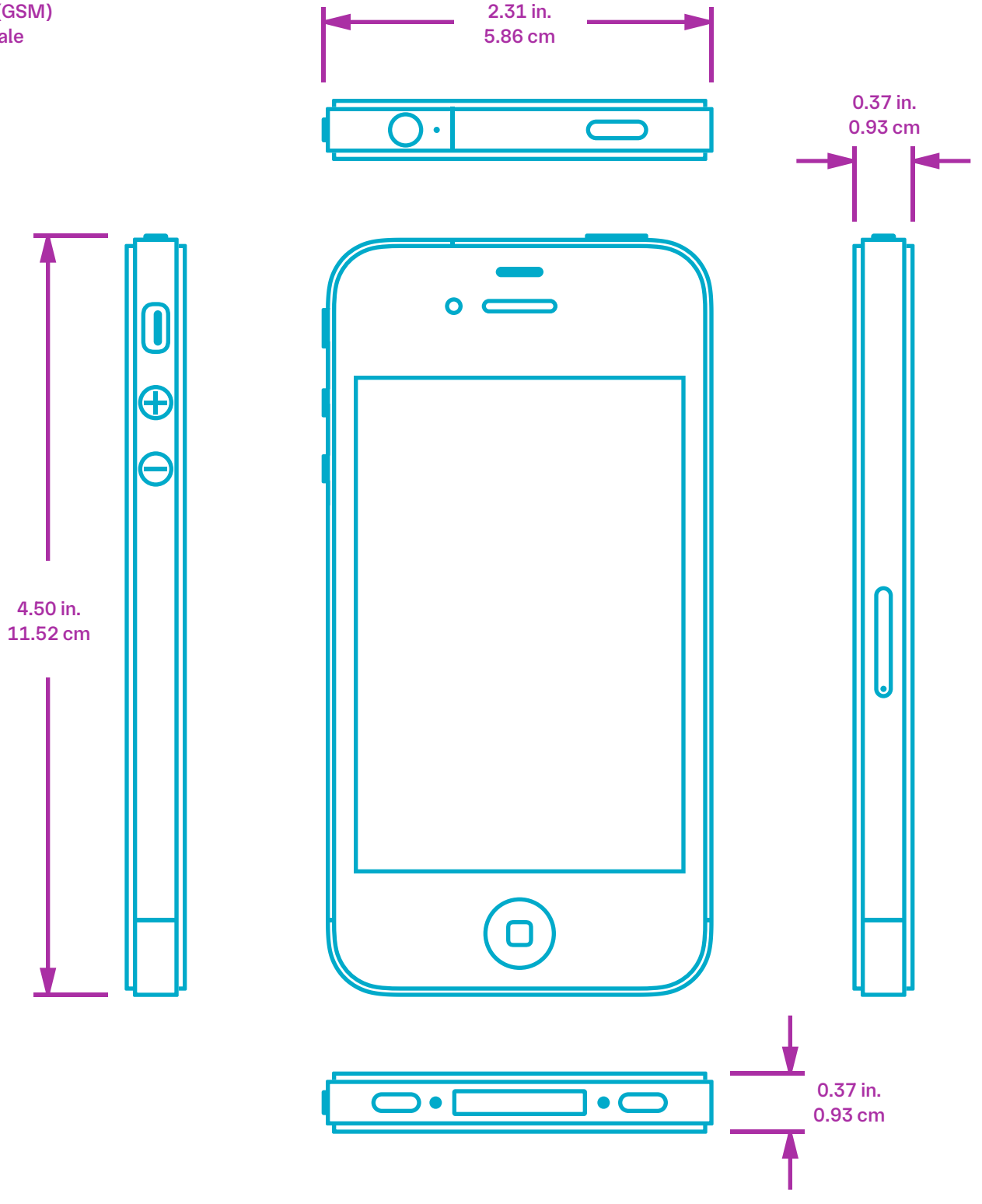
When power comes back, the hardware switches back to the charger and sends another system command to bring back the default button panel. Things connect automatically and the hardware chugs along.

Additions

The hardware used in the system is reused. Smarthome can only afford to be a panel-per-room system because of their price — and since it moves with me from rental to rental, the hardware never becomes a sunk cost. Nearly all of the phones are purchased used.

Dimensions

iPhone 4 (GSM)
1:4.56 scale



When adding a new device, I look for listings that show a phone that's been well taken care of. The listing should have actual photos of the device (none of those "you might get something similar" ones). It should have one photo of the screen of the device showing its home screen (a tacit confirmation that the device is not placed under an activation lock).

I also double-check that the listing is truly an iPhone 4 and not a misidentified 4S. There are two ways to tell. On the back panel, the 4 has a bunch of regulatory text under the larger "iPhone" label (the 4S will not). When at the home screen, by default, the 4 has a starry-blue iOS 7 background and UI text set in Helvetica Neue (the 4S will likely have text set in the San Francisco font, on a blue-and-orange background, indicating iOS 9).

For spare batteries and cords, eBay is also good here. The local reuse store is likely to have an endless supply of 30-pin cords, too.

Upkeep

The hardware contains lithium-ion batteries, which need replacement every several years. When adding a panel, I check its battery cycles and health, and then I keep an eye on them during daily use. Only one battery has ever needed to be swapped out, my original iPhone in 2017 — to be fair, it survived college.

Since the screens pretty much show the same buttons day after day, image retention and burn-in are risks. To give their screens a break, the hardware automatically dims overnight. The button panel is still "on," though, accessible by clicking any hardware button.

Why iPhone?

I use an iPhone 4 as Smarthome's panel hardware because it continues to offer the most system capability in a fit-and-finished form factor. Starting ten years ago with experiments and continuing onto Smarthome, its ability to be repurposed, specifically display and interactive uses, is hard to match.

In those ten years, I have learned that the iPhone 4 is a pile of lucky breaks to support being converted from a phone into a Unix-like computer. For instance, iOS devices can only be upgraded to certain versions. For the iPhone 4, the only one it can upgrade to is one that can be opened up (jailbroken). The method is a once-and-done process (an untethered jailbreak). It works regardless of its original carrier (GSM versus CDMA is irrelevant).

Those facts together mean: any iPhone 4 today can be opened up, once, for permanent reuse. The reward is a high-resolution, multi-touch display packed with sensors and power management that can be freely put to use.

As the years go by, I keep tabs on other devices: newer iPhones, Android phones, non-phone panels and screens. Still, the iPhone 4 wins out because of its relative cost to all of the capabilities it has to offer. Other display technology, like OLED displays, offer brighter color, but a fixed image like the button panel apps would ruin them with burn-in. Buying pre-built panels would absolve me of some setup time, but their high cost would prohibit a per-home design and still wouldn't give Smarthome the full set of sensors it requires to work well. (I discuss more comparisons like these in Product's competitive analysis matrix.)

Jailbreak

Getting a panel ready for Smarthome takes about a day. All five in the system are the result of a one-time checklist: after I physically clean the hardware, reset it, and check its battery health, I jailbreak it.

Jailbreaking refers to the act of opening up a device that has been locked down by its manufacturer. For the iPhone 4 acting as the panel hardware, this removes restrictions like “only use apps from the App Store!” or “no touching that sensor!” By jailbreaking, the phone is restored to an opened state in the way that laptops and Raspberry Pis are. Since the process is a legal, once-and-done process, it’s an unfussy way to repurpose hardware in fully capable ways.

That setup checklist continues after the jalbreak is added. I change the appearance of the system to darker settings, I set up cause-and-effect actions to respond to outages and brightness changes, and I optimize background processes from the command line. All of these changes would otherwise be impossible on a typical, locked down iPhone.

Jail?!

Smarthome needs hardware that it can fully control. By default, iPhones do not meet that need: there's no third-party apps, no command line access, no file hosting. If you consider these features "locked away," then the jailbreak lets them run free. The capabilities that are usually off-limits are no longer restricted, which means you've got some nice, general purpose hardware to add to a system.

There's irony in Smarthome using such a severe-sounding process. "Jailbreak" sounds like I'm hacking my way into the phone to muck around with rickety apps — not the case. This process simply enables an unrestricted, Unix-like experience that would be impossible with unmodified iOS.

In any case, jailbreaking is legal in the United States through 1998's Digital Millenium Copyright Act. In 2010, it was clarified that the Act protects jailbreaking. The U.S. Copyright Office and the Fifth Circuit Court of Appeals independently clarified this, identifying it as a form of fair use with regard to copyright law.

Really, the one slight risk in all this is the possibility of a new iOS update for the iPhone, one that would patch up its jailbreak ability. This is very unlikely to happen: the iPhone 4's last update was ten years ago and counting.

One-time jailbreak

To add the jailbreak to the phone, I first plug the iPhone into a laptop. It uses a desktop application to send special instructions over USB. The process only take a couple of minutes with a few reboots and some tap-to-continues.

Once the jailbreak is added, the iPhone runs the same, but with one new app. It's Cydia, an "app store" that looks at different lists of community-made apps and system changes, which are called tweaks. The app (and the related command line inteface it enables) makes it possible to install these.

Each of these tweaks offers a slice of functionality to manage the hardware properly. One app, for instance, gives the hardware its custom backlight adjustments. Another tweak disables the lock screen so that waking the display jumps right back to the panel app.

Since this stuff changes hardware and system settings, only tested, trusted apps and tweaks are used in the system. In reality, this means I reach for the same list of apps and tweaks that have worked well for years — some were made for the very first iPhone in 2009!

Those apps and tweaks are Debian package files, either added manually or via Cydia. Once the files are on the hardware, they get installed, and I reboot. At this point, I'm ready to configure and calibrate the display.

Calibration

I calibrate the color temperature of the display and the brightening-and-dimming behavior of its backlight. It's applied across the entire operating system of the hardware, a feature of unrestricted system access.

The display's color temperature is adjusted with f.lux, using its GPS sensor. During setup, the f.lux app requests access to location information. The GPS reports coordinates, and f.lux reads it once to find sunrise and sunset.

At night, displays are set warmer than they are during the day. As sunrise and sunset change throughout the year, the timing changes, too. The f.lux app handles this in the background.

Backlight settings are similar, using a custom app on the hardware itself to configure its sliding brightness scale. The app handles smooth transitions and reactivity itself.

My part in all of this is to actually set and review those calibrated values. They need to fit the room at any brightness level, so it usually takes me a day. In the morning, I'll choose some values for daytime and nighttime. As the day progresses, I adjust them as needed.

Tweaks

With things calibrated, I move onto the many settings that make the hardware less phone-like and more panel-like. This does involve a couple of stock Settings toggles, but jailbreak tweaks make up the bulk of changes.

One collection of tweaks turns familiarities of iOS off, like Notification Center, the lock screen, and volume level overlays.

Tweaks also improve the experience of the web apps. They allow them to run in full screen (without the status bar) and add the ability to run web apps from the hardware's storage.

Others offer system management capabilities, handy during development and now useful for management over-the-air. This includes display sharing (VNC) and file sharing (SSH). I also disable some background processes (Darwin daemons) Smarthome doesn't need, like for Spotlight search and the iTunes Store.

There's little risk in the kind of tweaks I use because they hook into Cydia Substrate, a safe way to modify the actions of the operating system's code. With it, tweaks reference the code they will change, to "redirect" it to alternate actions. The original code isn't changed, it just sits untouched.

Activator

Activator is a tweak that carries out actions when a condition is met or a button is pressed. The actions are configurable through its app. Smarthome uses the tweak to respond to things that would normally be off-limits.

Activator can toggle sensors, lock the screen, and open apps. It makes automated hardware maintenance possible, like Smarthome's nightly dimming and weekly rebooting.

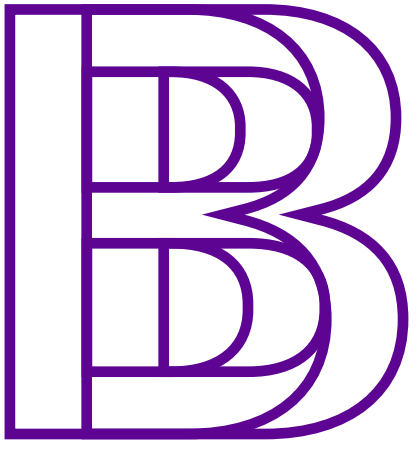
The no-power panel is a good example of what Activator can do. In the app, the no-power app is set to open when the hardware begins running on battery, and the regular panel app is set to open when power returns (the phone resumes charging). With that, Activator monitors these power events through its background process, and it automatically triggers those actions when it catches those changes. It happens seamlessly on the screen.

List of tweaks

On the opposite page is a list of some of Smarthome's tweaks. Piece by piece, they make up the overhauled phone experience. (Some tweaks in the list are installed but not used, instead reserved for future use.)

Tweak	Purpose	Developer
ActiSound	play interface sounds and clock chimes	Milo Darling
Activator	run custom actions from hardware buttons and sensors	Ryan Petrich
AirSpeaker	use panels to play music (AirPlay endpoint)	Karen / あけみ
BTC Mouse & Trackpad	connect to external Bluetooth buttons	Matthias Ringwald
Cmdivator	run shell commands from button apps	joedj
CustomBrightness	dim and brighten screen backlight	Harshad
f.lux	remove blue tint of displays	F.lux Software LLC
file:// for MobileSafari	access local web apps	Jay Freeman (saurik)
HideMe7	hide iOS components and decorations	CPDigitalDarkroom
Homer	prevent default home button action	John Matula
IOKit Tools	access battery and sensors directly (`ioreg`)	Jay Freeman (saurik)
KeyCommands	run actions from external keyboards	Ethan Vaughan
Maximization	hide status bar in button panel apps	rud0lf77
MouseSupport	add support for Bluetooth mice and scroll wheels	Matthias Ringwald
No Page Bounce	prevent rubber-banding in button apps	developersBliss
OpenSSH	manage files and processes over-the-air (`ssh`)	Jay Freeman (saurik)
SimulateTouch	trigger custom taps and gestures (`stouch`)	iolate
SkipLock	return to button apps when waking display	FilippoBiga
Veency	see and use panels over-the-air (VNC)	Jay Freeman (saurik)

Design



Buttons give Smarthome its interactive powers. The way they look and work is purposefully designed to suit the hardware they run on. Buttons, panels, clocks, and pickers do their best to offer the right amount of control on the wall.

Smarthome heeds best practices from user experience design, interior design, and ergonomics and human factors. While those buttons are important, they make up one part of the larger system to consider — each level of button, panel app, hardware, room, and apartment has constraints that Smarthome must consider.

Just like the hardware and hub, the button panel apps are designed to be useful for quite a while. That means the way they run is as important as the utility that they offer. Buttons within them are assembled to make the best use of the GPU. The panels make use of the hardware and hub, avoiding responsibilities of managing data and enabling an efficiently responsive experience.

Intent

If Smarthome does its job correctly, using it should be a boring experience. Harsh, but true: plastic light switches don't bring us much joy and wonder, either. Generally, if a design is really good, people won't think much about it. If it's a poor one, people will have no choice but to think about it, and hard.

Calm, reliable, understandable — "boring" is the glib way to refer to the system's ultimate goal, to design a system that controls a home in an ultra-reliable way. The joyful part of Smarthome is its ability to make device capabilities easier to access, not the way its buttons light up just so. The role of the design, from buttons to hardware, is to make that joy happen as quickly and easily as possible.

Pixels

Smarthome acts as the interface between me and my lights. I tell the system what to do by pressing buttons on the wall. Those humble buttons are forced to work in lots of scenarios, like dark rooms, direct sunlight, taps from odd angles, glances from a distance, drive-by taps in the hallway…

Things start at the display level, where elements are aligned to the pixel grid. Buttons and icons are positioned so they show up on full pixel values. This makes for a very sharp difference between "on" and "off" pixels. Those differences combine as the overall contrast of the display. To handle the previous plethora of scenarios, that contrast must be very high.

How does something look if it's positioned on half of a pixel, anyway? Displays handle this by anti-aliasing its position, or using averaged values to get close to the intended, partial value. For curves and text, this is good: that blocky effect is made much less noticeable.

For straight, vertical lines, like button or icon edges, it causes a fringe or blurred effect. The display is forced to average the whole length of it and show a blended color along the whole line. This results in lower contrast (less difference between the "on" and "off" pixels now). In dark spaces, that low contrast reduces overall legibility.

Text, button edges, and arrows are all aligned like this in the vertical direction. In very dark and very bright rooms where every scrap of contrast counts, text and controls are crisp. It also keeps screens legible at greater distances.

Buttons

Across the system, buttons look and operate one way, even as they control different devices and show different screens.

The kitchen panel, for instance, shows two types: wide and square. The square buttons show icons that represent both sides of the kitchen counter. The wider buttons, the main lights of the room, show icons for the (higher) kitchen ball and the (lower) dining table light.

The overall gist of them (design language) is the same. Both kinds contain one icon, drawn with thick lines. They show noticeably different states between "on" and "off." When you tap them, both light up very brightly.

On the bathroom's button panel, there are extra buttons that serve as dimmer controls and a temperature switch between soft warm and bright daylight light. These additional buttons are here to make full use of the single lamp found in the bathroom.

Tapping one of the full-size dimmer buttons dims and brightens the light; holding them does that more gradually. If the light is off, tapping or holding the "plus" button gradually fades in the light, starting at its lowest setting. Also when off, the temperature button turns on the lamp to its bright daylight light setting. (The main action of turning on and off a light is still most important, so it remains the widest button at the top.)

Those features are clearly presented on the bathroom panel, and they exist on every panel in the apartment in the form of tap-and-hold gestures. Dimming, hue selection, and extra tools are controlled as additional capabilities.

Going back to the dining table light, that wide button has three regions. Tapping and holding to the left of the icon dims its light; tapping and holding to the right of the icon brightens it, or fades it in from off. Tapping and holding the icon itself switches to the hue picker for that light, offering controls similar in function to the temperature button.

This ends up giving a button more like four quick capabilities for the day-to-day person using it. But they are just that: additional, not for getting in the way of ons and offs.

The similar look and interactivity of the buttons reinforces the fact that they function the same. Wide buttons all dim in the same way and with the same large tap areas, for example.

Clocks and tools

Keeping time front and center keeps me going. As a fella who manages his time blindness, having plenty of clocks is one way I'm able to successfully keep track of my tasks.

Clocks are an additional tool of Smarthome, to make more use of the well-tuned screens affixed to the walls. The bathroom and bedroom both offer a clock, fitting alongside the other buttons. It, too, is a button, bringing up a large type clock. As an additional tool, tapping and holding the clock button shows a few types to choose, like a 24-hour clock (and an Easter egg) when my sister visits. In the bedroom, the large type clock is tinted a deep red to serve radio clock realness.

There is also an additional tool found when pressing and holding a wide button's middle icon, the hue selector. Mentioned earlier, this shows Smarthome's system color presets that it can apply to the given light. It also offers a temperature switch and a random color selector. When I've got the right hue, there are full-size dimmer buttons to set its brightness.

In the foyer, the panel acts as a scene selector instead of individual control. The scenes, set in Home Assistant, control about a dozen bulbs with five different scenes. There is also a dedicated "off" button for heading out the door.

In the booth (a closet that operates as my recording booth), a clapperboard button opens up recording tools, with an A440 button for pitch and a marker button to announce the time. (There's also a metronome button, but it's one of those saved-for-later features that I discuss in the Product section.)

Continuous UX

A home has to be workable for it to be warm. Button panels are as thorough as they are so that they work clearly and are recoverable from accidental taps when they happen.

Understanding how to accommodate these errors, and even redesigning panels to avoid them, is all part of the user experience research and design process. For Smarthome, it's lightweight: it simply serves as a way of thinking so that I actively consider new ideas or reasoning behind accidental taps.

The function of the temperature button changed early on as I built Smarthome. Originally, the button toggled between color and daylight light when the bulb was on, and do nothing when off. The panel is right next to the medicine cabinet, so my original intent was to avoid accidental taps when opening it. In reality, I found myself tapping it wanting bright light. I was trying to switch away from its last setting before off, which I remembered was darker color. Giving those mistaps some UX thought, that button now supports that function and acts as a shortcut to bright daylight light.

Even for the honest mistake, like flipping the wrong switch, it's a simple oop-tap-tap. The hub supports my quick correction with fast responses to button taps (around 10 to 50 ms).

Beyond the buttons, the UX process influenced how much time I gave to smoothing out screen animations and button transitions: quite a bit. Originally, the apps were just jumpy enough to make me pause. Optimizing it was a tech task (keep layers on GPU, optimize SVG drawing, cleverly use jailbreak features) that improved the overall experience.

Blending

Smarthome has sufficient contrast and large tap areas according to the WCAG's guidelines. Those guidelines dictate how web content should behave and be structured.

But Smarthome isn't really a webpage, and some guidelines, like keyboard access, are irrelevant to the system. At the same time, new guidelines come into play to support the physical design needs of the system, such as the system's ergonomics and human factors.

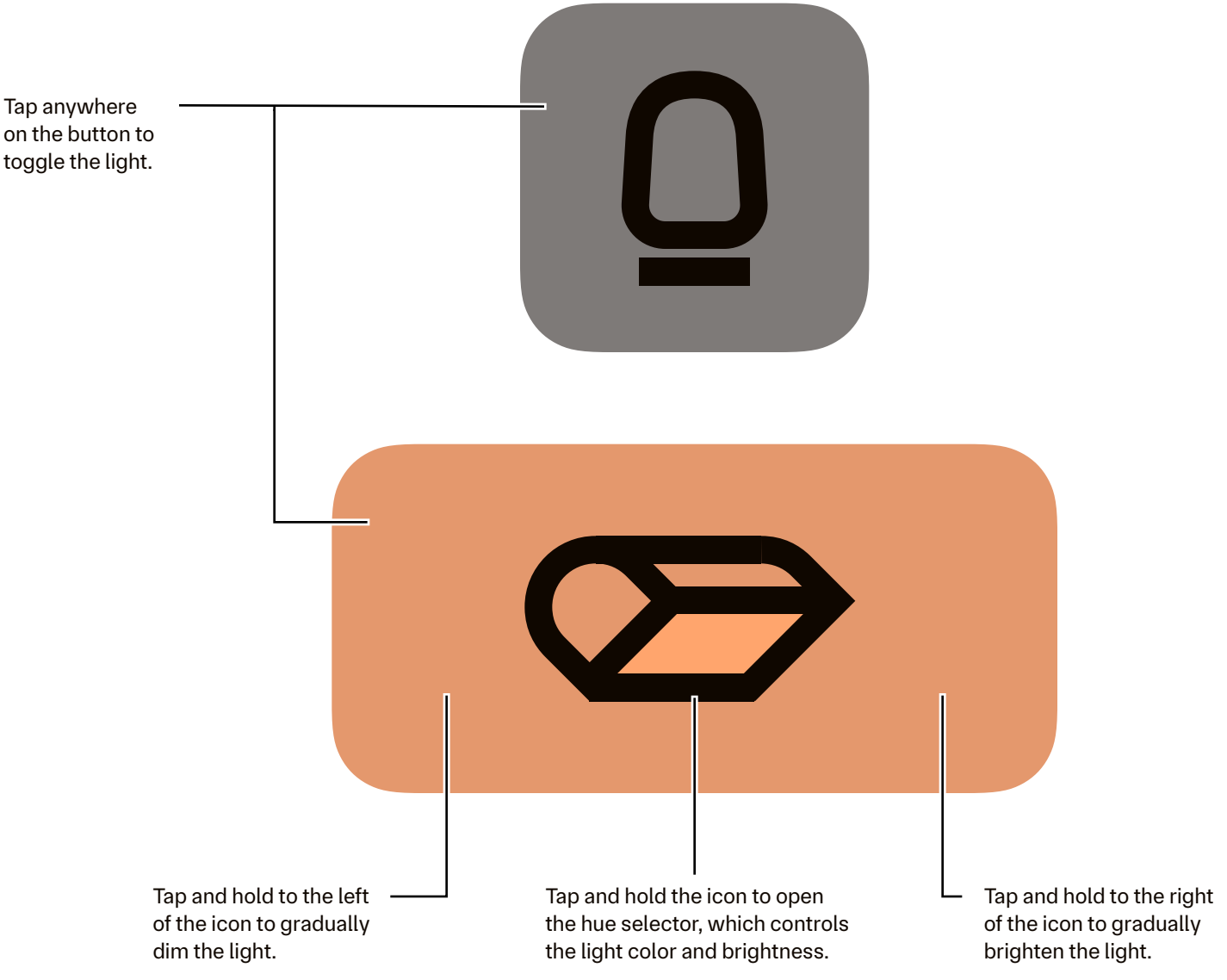
For example, the bathroom's panel has purple buttons so that they appear brighter with the room's purple paint. Beyond aesthetics, the purple light from the buttons is reflected more than other colors, which helps legibility in low light. They're shaded a full-toned purple (not dark, not light) to keep buttons bright enough for use at night — enough so I can aim a tap. As the web guidelines are written for general use, and these buttons are a hardware-specific specialty, their color contrast can be tuned lower without adversely affecting their utility.

Physically, panels are affixed at the same height from the ground as other wall switches in the apartment. The exception to this is the bedroom panel, which is at a lower height so that it can be workable from bed.

I admit, there's a lot of refinement on screen brightness, button contrast, the speed they operate, the placement of them all — this tight fit is the reason I'm not "operating" Smarthome but truly using it. It sees my elbow quite a bit, and I use the large print clock way more than I thought I would. The panels blend into the apartment; they're fast and reliable enough to blend into my routine.

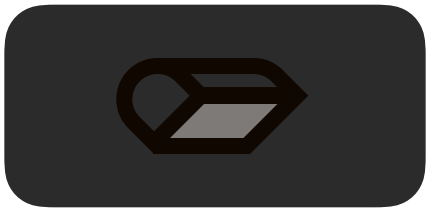
Behavior

Buttons provide the same statuses and feedback across the system.

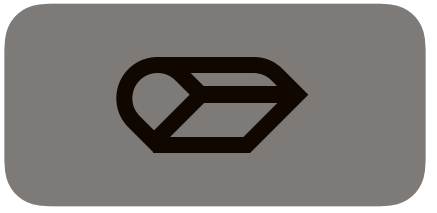


States

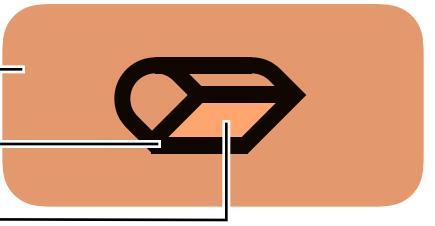
unavailable



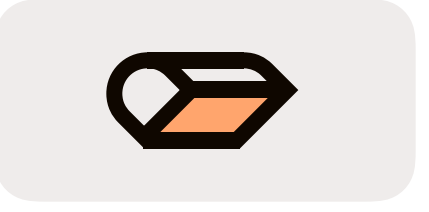
off

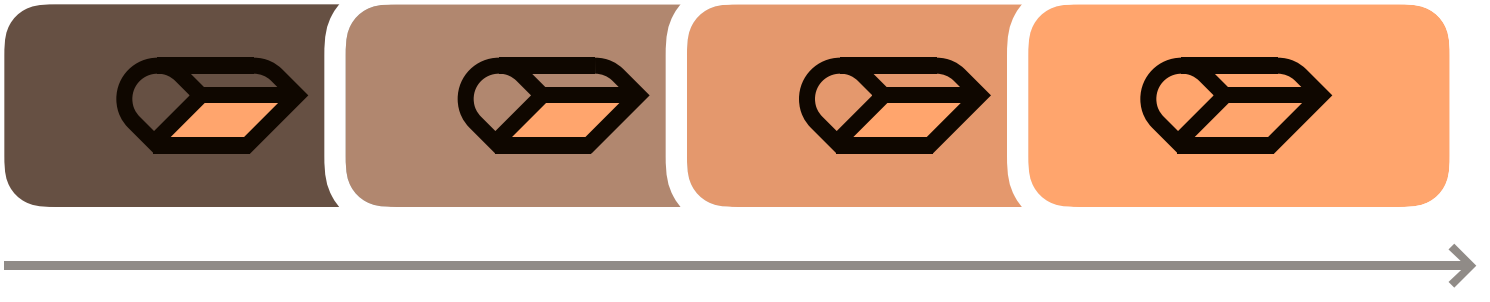


on



active

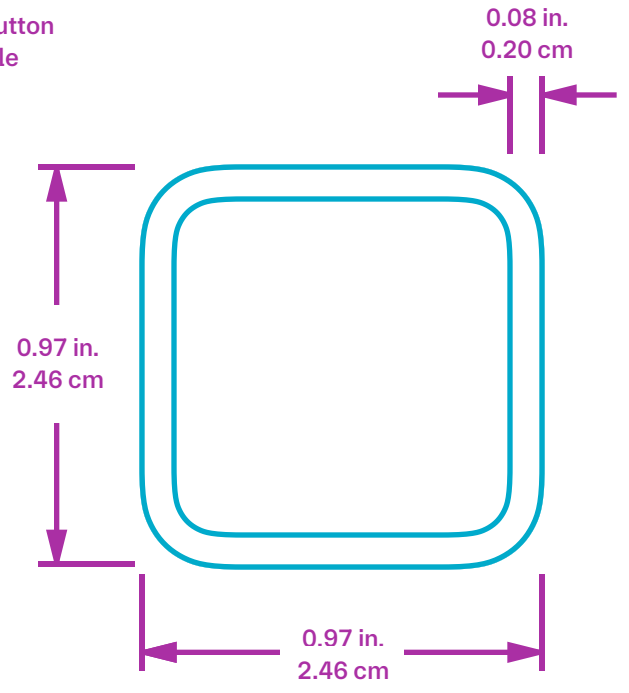




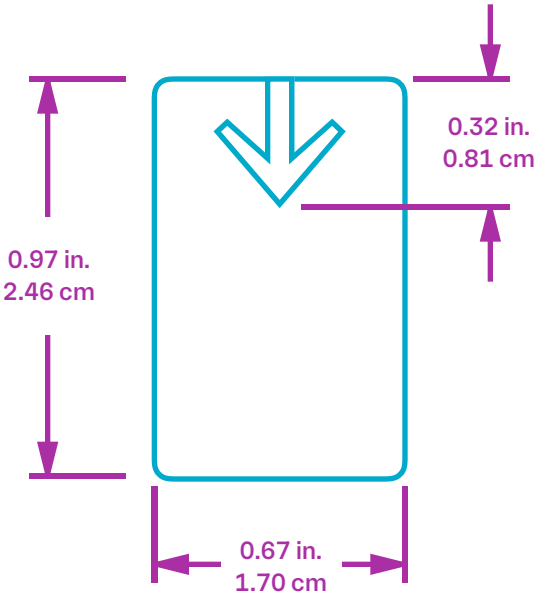
As brightness increases, the button panel brightens from dark gray to the selected hue. The hue indicator remains steady at all brightnesses.

Dimensions

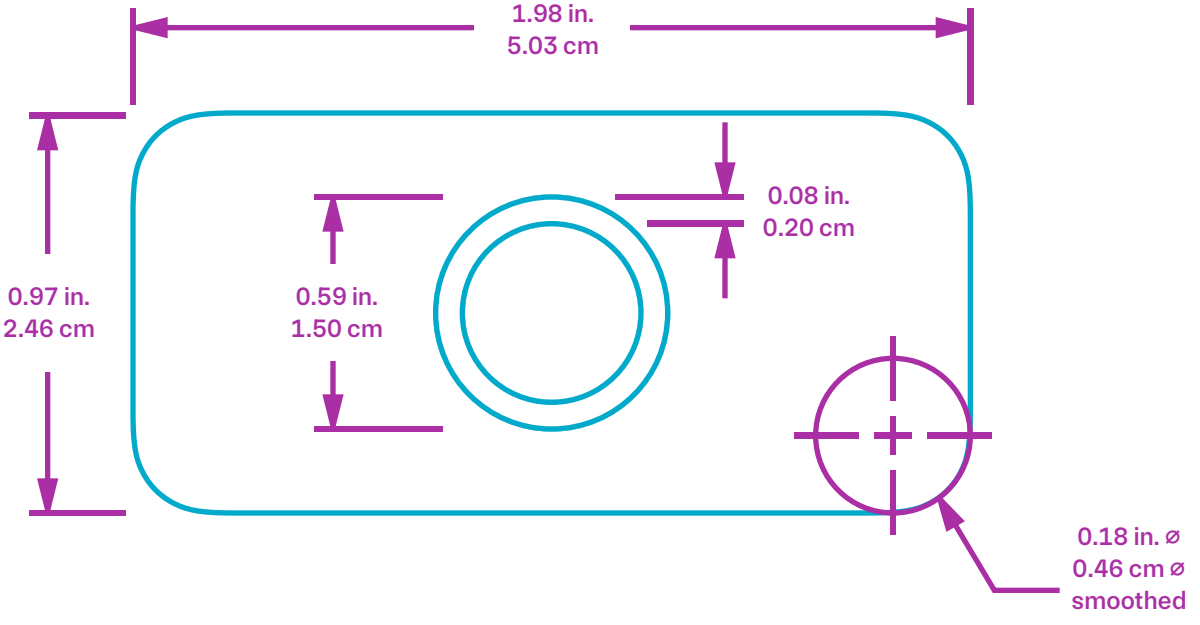
Square button
2.1:1 scale



Selector button
2.1:1 scale

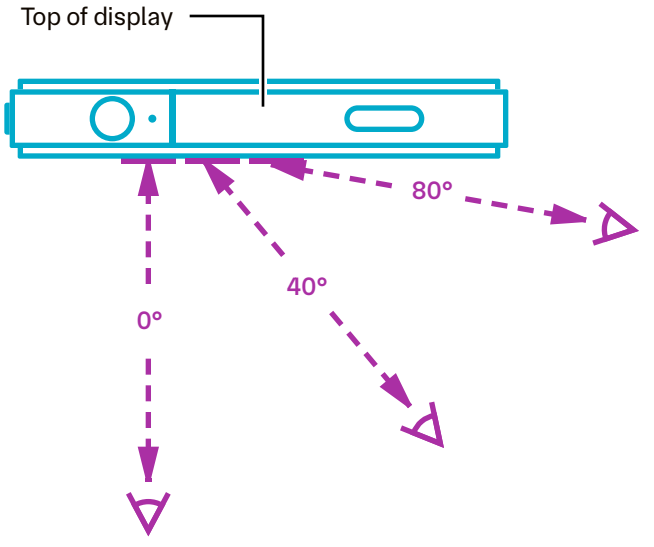


Wide button
2.1:1 scale



Legibility

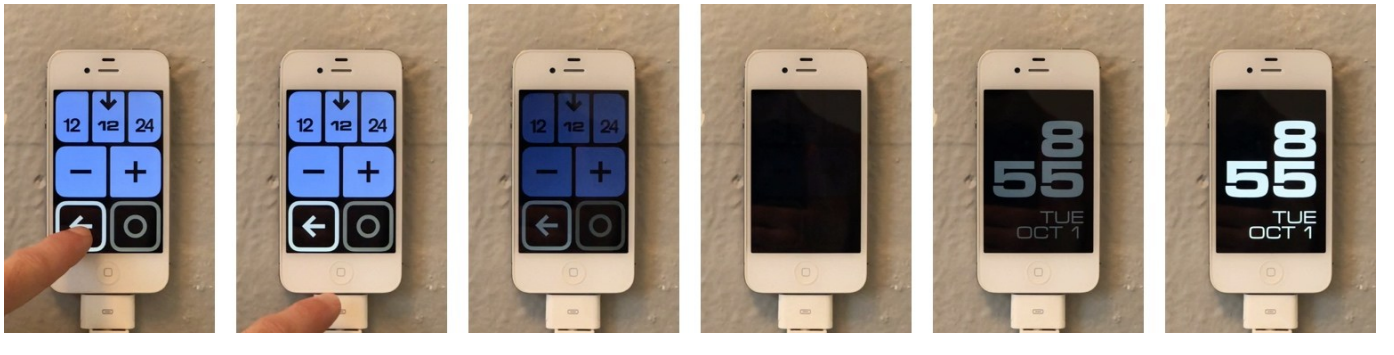
I measured these distances on a sunny day with screen brightness at its maximum setting, at three angles. In reality, these are more measurements of my own visual acuity; a full, proper test design would include participants, random values, and real-world tests — like a road sign's legibility might be.



Type of content	at 0°	40°	80°
shapes	34 feet	16′	12′
large type clock	15′	14′	10′
small clock button	8′	7′	4′

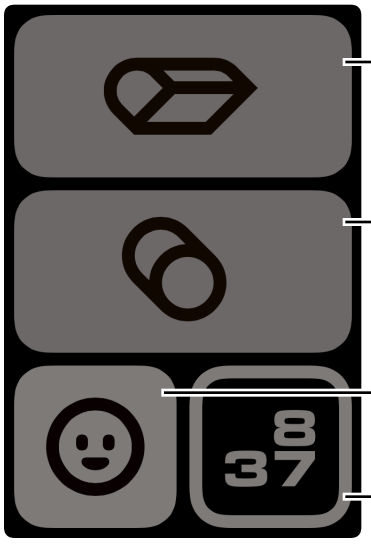
Transitions

A quick fade happens when switching between screens. These slow-motion stills come from early speed tests.



Panels

Bedroom, lights off

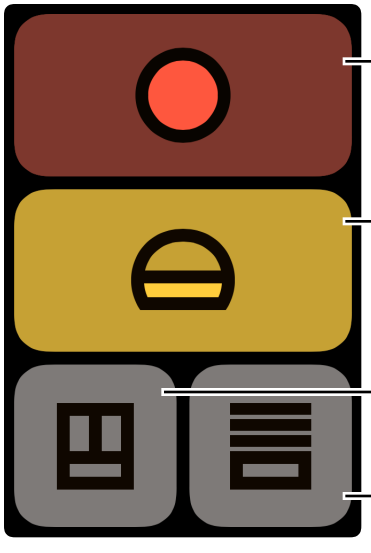


- desk light — off
- bedside light — off
- neon — off
- large-type clock

Bedroom, lights on

- desk light — medium yellow
- bedside light — medium purple
- neon — on
- large-type clock

Kitchen and dining table

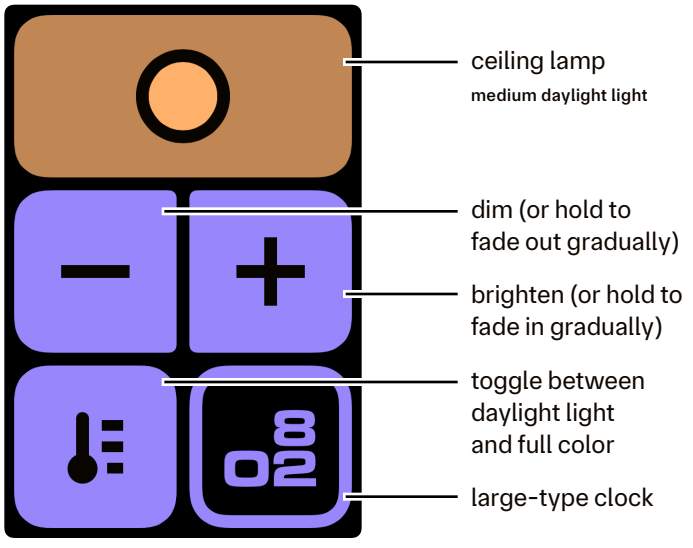


- kitchen ball — medium red
- dining table light — medium yellow
- light under cupboards — off
- countertop light — off

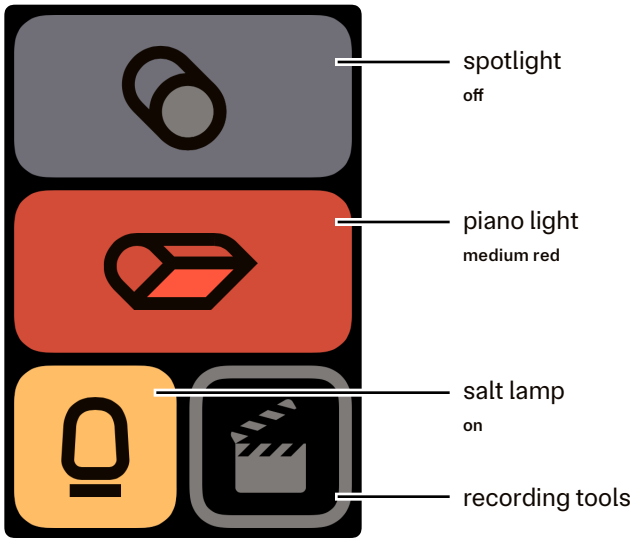
Foyer and living room

- sunny scene
- dusk scene
- piano scene
- party scene
- low red scene
- living room and foyer lights off

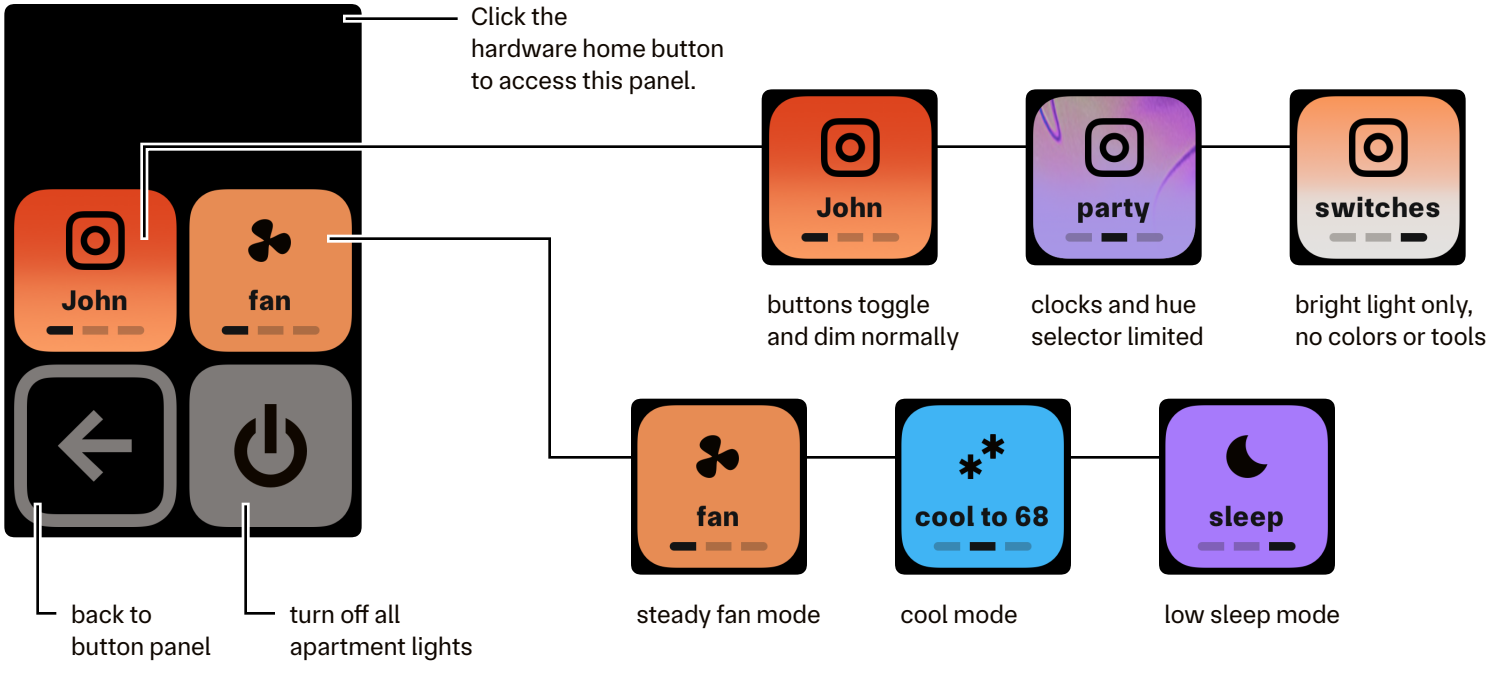
Bathroom



Recording booth



Whole-home controls



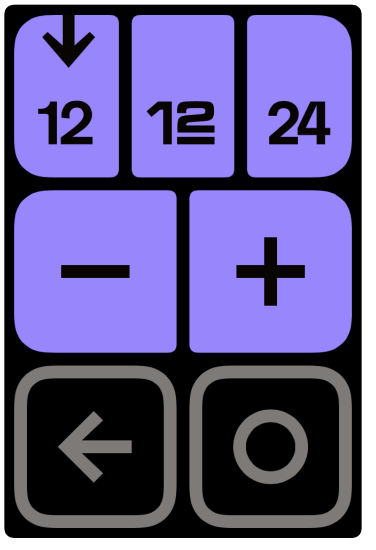
Clocks

Large-type clock

Default clock that shows
when tapping a clock button

Large-type clock, bedroom

Same layout, but tinted
very dark red for low light



Clock selector

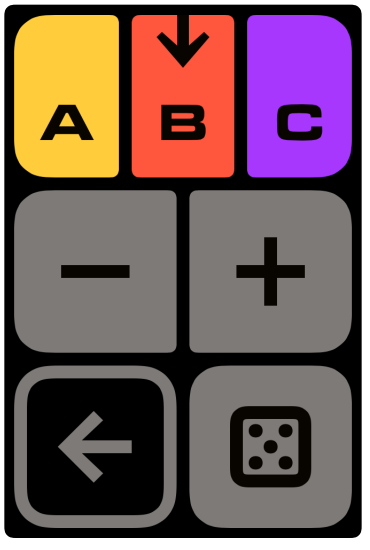
Appears when tapping
and holding a clock button.

Wide clock with date

24-hour clock with date

24-hour clock (Easter egg)

Additional tools



Hue selector

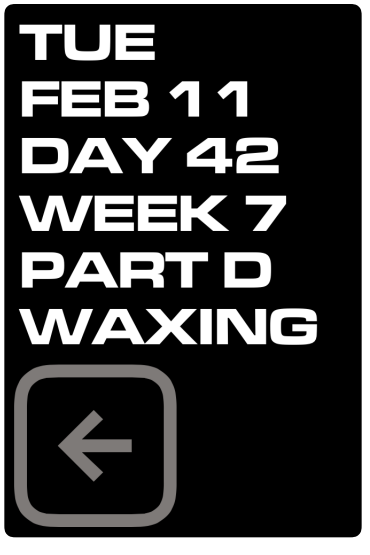
Sets the color and brightness of a light, or set a random hue

Air conditioner

Sets the fan and thermostat, shows mode and temperature

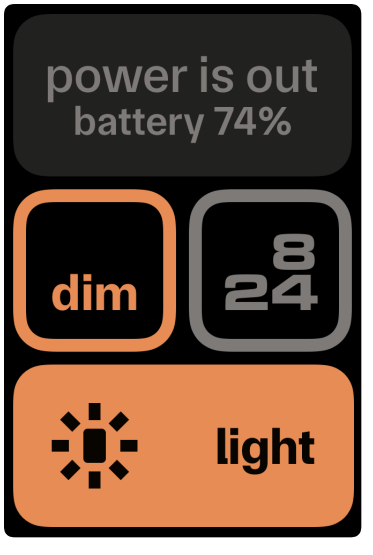
Recording tools

Clock, markers, and tempo (shaded darker for the booth)



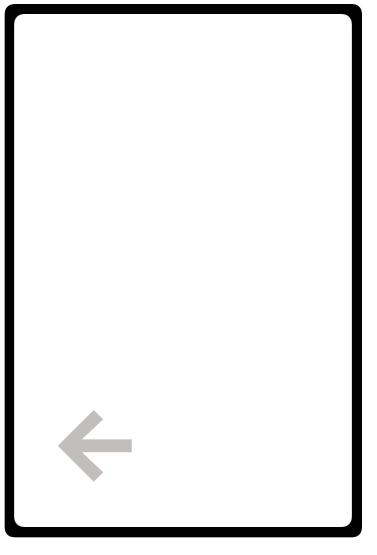
Wall calendar

Non-clock time keeping



No-power panel

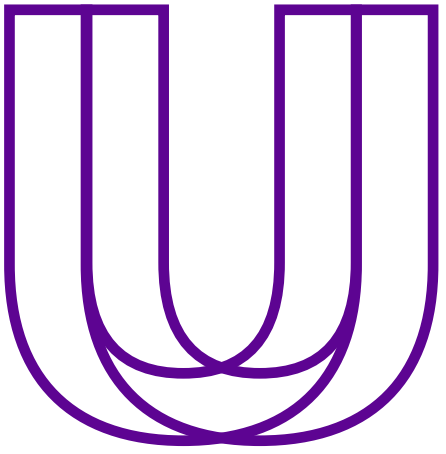
Clock and light for safety



Light, no-power panel

Backlight-maxed screen light

Product



Underneath every product lies a riddle: build something thoughtfully without overthinking it. Give a product too little thought, and it runs the risk of becoming unhelpful or breaking quickly. But give it too much thought, and an idea runs the risk of being a nice project plan and nothing else.

Smarthome answers that riddle with a continuous, iterative process to plan, build, and test work. Used informally since the start of the project, it's how I try new ideas and debug and refine existing ones. It's a continuous cycle: when I test work, I also plan out further work on it. Or, I find a good stopping point, consider how to add it to the roadmap, and I move on.

That roadmap holds ideas for Smarthome that I can work on — not now, but later. It contains experimental ideas and feature refinements alike. It's ordered by priority, estimating each idea's cost, time, and benefit. When I'm ready to load up my work queue, I take the most promising ideas and break them down into tasks that fit the plan-build-test cycle.

Stories

Smarthome brings up many questions about usability and utility, all with little jobs of varying importance and difficulty. I think of Smarthome as a product, so that I stay focused on priorities.

What makes something a priority? One way to answer that is through user stories. They're goals that take the form of single sentences, and they keep the person using a product in focus. They help each iteration of the plan-build-test cycle, whether that's at the outset of a new project or in the middle of it.

Smarthome has an overarching one: "as a renter, I want to use my smart lights like they were any other light in the place so I can put my phone away." This broad statement breaks down into manageable chunks of work. As a matter of fact, the goals at the start of this book were from this story.

Stories are handy in a personal project and downright indispensable in a large, distributed team. They are necessary for engineering and design teams to stay aligned — not just to work, but a broader goal. They also serve as one indicator of a task's priority, determining what makes most sense to work on next.

Comparisons

Comparisons, or competitive analyses, also help set a project's priority. They reveal what products do well, and what mine should avoid. These might include some binary comparisons ("I want mine easier than theirs!"), but the true value of them lies in discerning smaller differences and strategic trade-offs.

During Smarthome's development, I took a look at other systems for sale, each from a different product category: a remote, a panel, a voice assistant, and a couple of switches. In a big matrix, I then looked at what each did well and what things Smarthome could try to improve. There were many, and an excerpt of that matrix is on the following pages. I'll touch on two that ended up affecting the product.

One opportunity came in the size of controls, specifically that they needed to be huge. Brilliant, a smart home touch panel, has large plastic sliders on one side of its panel, and they act as dimmer controls. However, its touch buttons are puny. The simpler TP-Link switch, by comparison, had a much larger button, and videos showed people using this in a quicker fashion than the precise taps that Brilliant's product required.

Another was the remote control form factor, which ended up being partially included. In my initial thinking, I really, really wanted to include a remote version of Smarthome in its first version — note the lack of a user story here. Remotes such as Control4's Neeo and the discontinued Logitech Harmony were intriguing because reviews of them commended them on its physical feel. Their interfaces were busier than their switch counterparts — and thus an opportunity for Smarthome to improve it. From the competitive analysis alone, a Smarthome remote sounded like a promising option.

Of these two examples, only the button size directly affected Smarthome (Smarthome's button size is closer to TP-Link's). Why weren't both ideas implemented?

Though the comparison indicated that the remote could be a possible feature of the system, user stories said otherwise. Where stories that supported buttons were easy ("buttons that are easy to tap"), the most interesting ones for the remote focused on its whole-home control, not the remote itself. In this case, the competitive analysis had an indirect effect, motivating Smarthome's home view when the physical home button is clicked.

Roadmap

Still, there are a couple of ideas that would likely benefit from a remote control form factor, like A/C control or controlling speakers from a wall panel. However, making a version of Smarthome work well in a different form factor isn't quick, and makes for a lot of work to support only two quick ideas.

That's why those things currently live on Smarthome's roadmap. It is the list of things that I'd like to add to Smarthome at a later point. The list is sorted by priority, taking into account a task's feasibility, its urgency, my time, and similar factors. The list also has estimated timeframes on it, or rough guesses as to when I can get to something.

For teams, this can be separated into a planned "roadmap" and a "parking lot" for ideas not yet scheduled. Its estimates are more precise, using time estimates or agile sprint points. Mine is simpler to suit a one-person project.

Smarthome gets any product-style roadmap at all because I want to juggle the time I spend on it with non-personal-project things. Having a list also makes the likelihood of a good, sudden shower thought a little greater. A portion of this list is on the opposite page.

Costs

Here's a user story: as John, I want to include low-cost components in the system so that I can feasibly use Smarthome in every room.

The iPhone 4 in its jailbroken state supports this very well. Besides the one that surivived college, I bought the other panels from sellers on eBay for around $25 per panel. (A nice little touch panel for the price of an Olive Garden dinner?!)

Those panels benefit from the free, open-source capabilities and extensibility offered by Home Assistant. The Raspberry Pi that things run on has chugged along for a while, serving other projects and now Smarthome. Back when I bought it, it cost $50.

That's $75 for the hub and one panel, or a rough total of $200 for the hub, five panels, one dev panel, and assorted extras like chargers.

The design and setup of Smarthome cost me time, not money. The research and foundational part of this is complete (adding display status to the hub, optimizing web apps, finalizing the set of hardware tweaks). The time spent choosing which buttons to show, putting them on a panel, and launching the app is equatable to the time spent installing and configuring other smart systems.

Finally, there are the devices that Smarthome controls. Averaging around $8 each, the collection has grown to over 20 devices over the last five years. Add in the air conditioner, and the cost of devices in my apartment adds up to about $340.

Overall, Smarthome adds button panels in my apartment and controls over 20 devices across it for under $600.

Roadmap item	Remarks	Timeframe
display current weather	I'd benefit from seeing the temperature, "feels like" temp, and forecast when I'm at the front door, about to head out.	spring 2025
smooth values between different kinds of LEDs in a single bulb	I'd like to create a dimming system to smoothly brighten from dark RGB colors to super-bright white colors, giving me a wider range of perceived brightnesses.	summer 2025
add display status support for room and home scenes	I'd like to automatically handle lights that I forgot to power on before choosing a scene, using its intended state from the display status mechanism. (Right now, I have to go flick a bulb's power on and reapply the scene.)	summer 2025
add remote control for air conditioner and music	For system-wide light control, I'm likely to be at a panel. For non-light actions, like air conditioner and music control, I'd find it useful to set these away from the wall.	summer 2025
refactor separate gesture libraries into one unified system	The libraries that enable instant app taps and long taps are separate libraries, and the hacks that allow both to coexist are finicky. Simplifying this will make gesture support easier to include.	fall 2025
create a build process for exporting SVG art to button panel apps	Though the panels are easy enough to assemble, it's a manual markup job right now. I'd like to eliminate some of the clean-up reptition with some improved `npm` jobs.	fall 2025
add timer to no-power mode	I have a gas stove, so I could use a timer to help me cook when the power is out.	2026
add overlay system to panel apps	Panels use a page-based system for controls. Overlays would be useful for emergency messages and quick controls, especially once gesture support is improved.	2026
add screensavers	Panels are static, and image retention does occur. I'd like to fold in the intent of the burn-in tool to the panel apps themselves, so that they can exercise pixels without needing to be interrupted to control lights.	2026
show feedback for physical controls	Once overlays are supported, I'd like to show quick displays (not unlike the volume overlay that appears when adjusting your iPhone).	—
improve power and process management	The hardware is able to detect when some system process has become unresponsive, and I'd like to figure out what conditions should trigger an automated reset.	—

Competitive analysis

This matrix is an excerpt of my research into other systems. It's a mix of publicly posted specs and measurements based off of them. I've added Smarthome's current capabilities as a reference.

Criteria	**Smarthome** hub and one panel	**Big-brand hub** Philips Hue bridge and dimmer	**Wall display panel** Brilliant, 2-slider panel (2018)
form factor of controls	touch screen on wall	smartphone app, wall remote-switch hybrid	touch screen on wall with 2 grooves
name and style of hub or protocol	open, Home Assistant	proprietary, Hue (Zigbee)	not a hub, integratable
normal operation during internet outage	✓ yes	✓ yes	✓ yes
temporarily installable (no hardwiring needed)	✓ yes	× switch is hardwired	× hardwired
has ability to set a bulb's color	✓ yes	✓ yes, in app	✓ yes, in app
tactile controls	× only for menus	✓ yes, plastic switch	✓ yes, 2 grooves
guest use (without logins or instructions)	✓ yes	✓ only from switch	✓ yes
motion detection in controller itself	× only short range	× no	✓ yes, 8 – 15 feet
LCD pixel size (width × height @ pixel density)	640 × 960 @ 326 ppi	—	720 × 1280 @ 259 ppi
LCD physical size (width × height / diagonal)	2.0″ × 2.9″ / ↘ 3.5″	—	2.8″ × 4.9″ / ↘ 5.6″
appearance in a fully off-and-dark room	buttons turn light gray, screen fully dims	nightlight shines	no change?
number of actions from main view	6 (or 12 with gestures)	4 (or 6 with gestures?)	8
size of smallest button	main: 1.0″ W × 1.0″ H add'l: 0.7″ W × 1.0″ H	1.4″ W × 0.7″ H	button: 0.6″ W × 0.9″ H; groove: 0.5″ W × 3.1″ H
tap reaction time	10 – 50 ms	5 – 20 ms	20 – 1500 ms
taps to gradually dim or brighten lights	1 tap or 1 tap-and-hold, depending on button	1 press-and-hold	1 "move" (finger slides along grooved slider)
rough base cost of system	$75	$85	$450
rough cost per additional panel or switch	$25 per	$20 per	$250 – $450 per
rough cost per additional full-color lightbulb	$5 – $10 per	$30 – $60 per	$10 – $15 per

Smart plastic wall switch TP-Link KS225 (2024)	**Bulb and app only** Philips WiZ bulb, 60 W	**Remote control** AVA Home Remote (2022)	**Remarks**
paddle switch on wall	smartphone app	remote control with Android touch screen	These systems purposefully come from different categories and cost levels.
open, Matter	proprietary, WiZ	not a hub, integratable	
✓ yes	× no	× no	
× hardwired	✓ yes	✓ yes	Brilliant sells one kit of a non-hardwired type.
✓ yes, in app	✓ yes, in app	✓ yes, in app	Ability to choose a color for one single bulb.
✓ yes, plastic switch	× no	✓ yes	
✓ yes	× no	✓ yes, shortcut buttons	
× no	× no	× no	Alas, no Smarthomesense for now.
—	—	480 × 1170 @ 280 ppi	
—	—	1.7″ × 4.2″ / ↘ 4.5″	
nightlight shines	in app, buttons display an off switch	display shuts off	An opportunity for Smarthome here: work well at night.
1	depends on smartphone	around 8 to 12	Light buttons only, no pairing or reset buttons.
on/off: 1.2″ W × 2.0″ H; dim: 0.6″ W × 0.3″ H	0.5″ W × 0.6″ H, but depends on smartphone	1.7″ W × 0.4″ H	Excludes reset, pairing, or setup buttons. “Add’l” refers to buttons found in additional tools.
5 – 20 ms	5 – 20 ms	depends on device	Same as previous row.
1 press-and-hold	2 to 3 taps, plus taps to get to the phone app	depends on device and shortcut settings	Includes the number of taps it takes, through menus or other screens, plus the action itself.
$22	$11	$1,300 + install contract	Except for the remote, costs exclude installation.
$22 per	—	$1,300 per	
$5 – $10 per	$11 per	varies by installed apps	Bulbs that are directly integratable with the hub.

Smarthome comes from
a long history of web app experiments.

I made my first light-control experiment in 2014.
I had never made wall controls up to that point,
just wall weather displays you looked at.
Those first few buttons were tiiiiny.

Still, the app was a snappy one.
When you tapped a button, it'd send a radio signal out
and you'd hear a whole bunch of lights click on.
When it worked, it was a treat. (When.)

It always seemed to act up before bed,
and it always, always was those kitchen ones
with the switch up above the cupboards.
I spent more than a few nights waving around
tinfoil and the Raspberry Pi, trying in vain
to get the signal to reach that plug.

All that, and my helpless geeky self would
still be thinking, "it's kinda neat they're stuck on."

Thank you to past me for those experiments,
and to my friends and family for putting up with them.
You make up the most important part of Smarthome.

For Anthony


johnmatu.la