Phase 0 — Setup (so you don't get stuck later)

1) Gather hardware + sanity-check it

You need

- ESP32 DevKit (DOIT / DevKit V1 style)
- SHT31 temp/humidity breakout (I²C)
- SSD1306 128×64 OLED (I²C, 4-pin)
- Breadboard + jumpers
- 2 pushbuttons

Learn: you're building an I²C "bus" (multiple devices share SDA/SCL).

---

2) Install the software (current versions / official docs)

1. **Install Visual Studio Code** (use the latest stable; VS Code updates monthly—currently 1.107 is listed). Visual Studio Code+1
2. In VS Code, install **PlatformIO IDE** extension (official PlatformIO instructions). docs.platformio.org+1
3. Install **Git** (latest Git for Windows is listed as 2.52.0 at time of this lookup). Git SCM+1
    - Optional (beginner-friendly): install **GitHub Desktop** too. GitHub+2GitHub Docs+2

Learn: PlatformIO bundles its own CLI tooling, so you generally don't need to install PlatformIO "Core" separately. docs.platformio.org

---

3) USB driver check (only if your ESP32 won't show up)

When you plug the ESP32 in, Windows should create a COM port.

- If the board uses **CP210x**, get the driver from Silicon Labs. Silicon Labs
- If it uses **CH340/CH341**, get the driver from WCH (official). WCH Microelectronics

Learn: lots of dev boards are identical *except* for the USB-to-serial chip.

Phase 1 — GitHub foundation (do this early)

4) Create your GitHub repo (first GitHub step)

On GitHub:

1. Create a new repo: `esp32-sensor-ui`
2. Check: **Add a README**
3. Choose a license (MIT is fine)
4. Add a `.gitignore` (PlatformIO)

Then clone it locally (either GitHub Desktop or `git clone`).

Learn: setting up version control early makes you look more "intern-ready" and saves you from losing work.

Phase 2 — Project creation + "bring-up" (make it work in small wins)

5) Create a PlatformIO project

In VS Code:

1. Open PlatformIO Home → **New Project**
2. Board: pick **DOIT ESP32 DEVKIT V1** (or closest match)

3. Framework: **Arduino**
4. Create project inside your cloned repo folder

Learn: Framework choice = your "SDK layer." Arduino is easiest to start; you'll go lower-level later.

PlatformIO's ESP32 platform uses Espressif's Arduino core; the current Arduino-ESP32 releases are tracked on Espressif's repo (ex: v3.3.5 is shown here). GitHub+1

---

6) Wire the hardware (keep it simple and safe)

I²C (shared bus)

- ESP32 **3V3** → SHT31 VIN, OLED VCC
- ESP32 **GND** → SHT31 GND, OLED GND
- ESP32 **GPIO21 (SDA)** → SHT31 SDA, OLED SDA
- ESP32 **GPIO22 (SCL)** → SHT31 SCL, OLED SCL

Buttons (use internal pull-ups)

- Button A: one leg to **GPIO32**, other leg to **GND**
- Button B: one leg to **GPIO33**, other leg to **GND**

Learn: internal pull-up = the pin reads HIGH normally, LOW when pressed.

(Tip: avoid GPIO0 / GPIO2 / GPIO15 for buttons early—they can affect boot on some boards.)

---

7) First firmware: I²C scanner (proves wiring)

Create a quick sketch that scans I²C and prints addresses to Serial.

- If you see **two devices**, you've won: the OLED and SHT31 are alive.

GitHub step: commit as
`milestone: i2c scanner working`

Learn: embedded debugging starts with proving each layer: power → bus → device → data.

---

8) OLED “Hello World”

Add an SSD1306 library and display text.

- Goal: show “HELLO” and maybe a counter.

GitHub step: commit as
`milestone: oled hello world`

Learn: you’re confirming the display pipeline before mixing in sensor logic.

---

9) Sensor read to Serial

Add SHT31 library and print temperature/humidity once per second.

GitHub step: commit as
`milestone: sht31 serial output`

Phase 3 — Combine + make it “real firmware”

10) Combine sensor + OLED (but avoid delay-based code)

Requirement: update readings every 1s **without** using long `delay()` calls.

- Use a timer approach (Arduino `millis()` pattern).

Learn: “non-blocking timing” is one of the first big embedded habits.

GitHub step: commit as
`milestone: oled shows live readings (non-blocking loop)`

---

11) Add button debouncing (don’t skip this)

Implement time-based debouncing:

- Track last raw state
- Track last change time
- Only accept a press when stable for ~30–50ms

Learn: real hardware is noisy; debouncing is a classic embedded problem.

GitHub step: commit as
`feature: debounced buttons`

---

12) Add a tiny UI state machine (screens)

Make screens:

1. **Live** (T/H now)
2. **Min/Max** (since boot)
3. **Settings** (toggle °C/°F, change refresh rate)

Structure idea:

- `enum Screen { LIVE, MINMAX, SETTINGS };`
- `renderScreen(screen);`
- Button A = next screen
- Button B = select/toggle

Learn: “state machine + render function” is how many embedded UIs are built.

GitHub step: commit as
`feature: 3-screen ui state machine`

Phase 4 — “Intern-ready” extras (small but impressive)

13) Serial monitor commands (super simple, but looks pro)

Implement a line-based command parser (type in serial monitor):

- `help`
- `units C` / `units F`
- `rate 500` (ms)
- `resetminmax`

Learn: command interfaces are common in firmware (debugging, manufacturing, field support).

GitHub step: commit as
`feature: serial command interface`

PlatformIO docs mention keeping platforms updated via PlatformIO Home → Platforms → Updates (useful when you want latest ESP32 platform/core). docs.platformio.org+1

---

14) Refactor into modules (this is where you level up)

Split files:

- `sensor.cpp/.h`
- `display.cpp/.h`
- `buttons.cpp/.h`
- `ui.cpp/.h`
- `main.cpp`

Learn: separation of concerns + clean interfaces = “professional firmware.”

GitHub step: commit as
`refactor: split into modules`

Phase 5 — Repo polish (what recruiters actually see)

15) README that sells the project (final GitHub steps)

Add to README:

- What it does (bullet list)
- Parts list
- Wiring table (pins)
- Build/flash steps (PlatformIO “Upload”)
- Screens description
- Short demo video/GIF (phone video is fine)
- “What I learned” section (I²C, debouncing, state machine, non-blocking timing)

GitHub step: commit as
`docs: complete README + demo media`

Optional: create a GitHub Release tag `v1.0.0` once it’s clean.