

IoT for Product Design

From Smart Objects to Ecosystems

From Data to Better Products

Course Guide & Exercise Roadmap

Andrea Vitaletti — Sapienza University of Rome

Course Agreement & Ground Rules

What we agree on from day one:

- **Use of AI is perfectly fine, but it is not an excuse to do not understand**
- **Physical reality matters:** At the end, I want to see a physical product made of real, interconnected things.
- **No validation is unacceptable and dangerous:** If it isn't tested with real humans and real signals, it doesn't work.
- **Validation needs measures:** Provide concrete numbers, latency, error counts, and observations , **not** generic sentences.

1. Course Purpose & Vision

The course introduces product design students to physical computing and IoT using **ESP32** and **Arduino Cloud**, driven by two central goals:

1. **From Smart Objects to Ecosystems:**

Value emerges from the *relationships* between objects, users, and cloud services – not from isolated gadgets.

2. **From Data to Better Products:**

Sensor data collected in the field informs iterative redesign, maintenance, and user understanding (closing the loop).

- **Target Audience:** Designers with no prior coding experience.
- **Pedagogy:** Step-by-step progression; every technical step ends with a **Design Reflection** and an empirical **Validation**.

2. Prototyping Strategy: Three Workspaces

To minimize tool friction while respecting cloud quotas on the free tier, students use three complementary workspaces:

Tool	Role	Why Designers Use It
Wokwi	Browser simulator	Sketching behavior and wiring with zero risk and instant feedback
Arduino IDE 2	Desktop environment	Unlimited local uploads, physical testing, Serial Monitor/Plotter
Arduino Cloud	Connected IoT platform	Cloud variables, mobile control, shared dashboards, data storage

The Golden Rule: Try and debug cheaply in **Wokwi** and the **IDE** first; use **Arduino Cloud** only for the connected milestone.

3. The Running Scenario: "The Connected Studio"

All exercises contribute to a shared, living scenario: bringing a **co-working studio** to life.

- Each student group designs and builds **one studio object**:
 - *Focus Lamp, Focus Totem, Air Quality Monitor, Door / Presence Badge, Plant Pot, Noise Meter, Meeting Gauge, or Coffee Counter.*
- **Progressive Evolution**:
 1. **Phase A**: The object acts completely alone (**Smart Object**).
 2. **Phase B**: The object connects to the network and dashboards (**Connected Object**).
 3. **Phase C**: Objects cooperate directly through shared variables (**Ecosystem**).
 4. **Phase D**: Objects are deployed, gathering real telemetry to justify redesign (**Data-Driven Design**).

3.1 Studio Objects (1/2)

Object	Module(s)	States (Ex 3)	Writes	Reads
Focus Lamp	LDR or PIR	OFF → FOCUS → DO_NOT_DISTURB	lampMode	studioScene, co2
Focus Totem	NeoPixel ring	AVAILABLE → BUSY → BREAK_SUGGESTED	totemMode	studioScene, noiseLevel
Air Quality Monitor	DHT22, buzzer, <i>pot</i> = CO ₂	GOOD → STUFFY → OPEN_WINDOW	co2	roomOccupied
Door / Presence Badge	PIR, buzzer	EMPTY → SOMEONE_ARRIVED → OCCUPIED	someoneAtDoor, roomOccupied	studioScene

Every object uses the **Core Kit** + 1–2 modules, has **≥ 3 states**, and **writes** at least one variable that **another object reads**.

3.1 Studio Objects (2/2)

Object	Module(s)	States (Ex 3)	Writes	Reads
Plant Pot	DHT22 (+ soil probe) or servo	HAPPY → THIRSTY → CRITICAL	plantMode	studioScene
Noise Meter	Servo needle, <i>pot</i> = microphone	QUIET → LIVELY → TOO_LOUD	noiseLevel	studioScene
Meeting Gauge	Servo	FREE → IN_USE → ENDING_SOON	meetingMode	studioScene
Coffee Counter	NeoPixel ring or buzzer	IDLE → BREWING → BREAK_TIME	coffeeCount	roomOccupied, totemMode

Each object comes with a **design question**, e.g. *Noise Meter*: should a shared object judge people's behaviour?
Door Badge: where does a greeting end and surveillance begin?

3.2 Simulating CO₂ and Noise with a Potentiometer

The kit has **no CO₂ sensor** (DHT22/BME280 measure only temperature & humidity) and **no microphone** → the Core Kit **potentiometer** plays the environment ("Wizard-of-Oz" sensor).

```
int readCo2()          { return map(analogRead(POT_PIN), 0, 4095, 400, 2000); } // ppm
int readNoiseLevel() { return map(analogRead(POT_PIN), 0, 4095, 0, 100); } // 0-100
```

- **The sensor is not the design:** the *behaviour* when the room gets stuffy or loud is.
- **Repeatable scenarios:** stage "the room fills up, then someone opens the window" in seconds → essential for validation.
- **Works in Wokwi,** which has no CO₂ sensor or microphone.
- **Invisible to the ecosystem:** other objects only see `co2` / `noiseLevel`.
- **Swap later:** a real SCD40 or MAX4466 only changes `readCo2()` / `readNoiseLevel()`.

4. The 2×2 Interaction Foundation

The foundational exercises are built on a symmetrical **2×2 matrix of physical computing primitives** requiring **zero external libraries**:

	Discrete / Binary (0 or 1)	Continuous / Modulated (0 ... Max)
Input (Sensing / Control)	Push Button (<code>digitalRead</code> , <code>INPUT_PULLUP</code>)	Potentiometer (<code>analogRead</code> , 0–4095)
Output (Actuation / Feedback)	LED On/Off (<code>digitalWrite</code> , HIGH/LOW)	LED Dimmer (<code>analogWrite</code> , PWM 0–255)

1:1 Bridge to Arduino Cloud Widgets:

- Physical Button $\longleftrightarrow$ Cloud Switch / Push Widget
- Physical LED $\longleftrightarrow$ Cloud Status Light Widget
- Physical Potentiometer $\longleftrightarrow$ Cloud Gauge / Percentage Widget
- Physical PWM Dimmer $\longleftrightarrow$ Cloud Slider / Dimmer Widget

5. Hardware Kit: Core vs. Project Modules

To prevent beginner frustration (I2C address conflicts, power brownouts, library mismatches), the kit is divided into two tiers:

5.1 Core Kit (Universal – Ex 0 to 2)

- ESP32 DevKit, breadboard, jumper wires, USB cable
- Standard LEDs + current-limiting resistors (220–330 Ω)
- Push buttons (wired straight to GND via internal pull-up)
- Rotary potentiometer (10 k Ω)

5.2 Specialized Project Modules (Introduced in Ex 3)

- **LDR / BH1750:** Ambient light sensing
- **DHT22 / BME280:** Temperature & humidity monitoring
- **PIR Sensor:** Human presence detection
- **RGB NeoPixel Ring:** Chromatic status signaling
- **Servo Motor & Buzzer:** Physical actuation & acoustic alerts

6. Course Structure Overview

Phase	Focus	Core Question	Tools
Phase A (Ex 0–3)	The Smart Object	What can an object sense, decide and do alone?	Wokwi, IDE
Phase B (Ex 4–6)	The Connected Object	What changes when an object is on the network?	Arduino Cloud
Phase C (Ex 7–10)	The Ecosystem	What emerges when objects cooperate?	Shared Cloud
Phase D (Ex 11–14)	Data-Driven Design	How can field data improve product and service?	Cloud, Sheets
Final Project (Ex 15)	Synthesis	Multi-object connected product-service ecosystem	All

Template for Every Exercise:

Goal → Concepts → Activities → **Design Reflection** → **Validation**

Phase A: The Smart Object

Exercises 0 – 3

Exploring local sensing, actuation, state logic, and the limits of isolation

Ex 0 — Hello Light: GPIO Output & Visual Feedback

- **Goal:** Familiarize with tools and control physical output through an LED.
- **Key Concepts:** Wokwi vs. physical board, Arduino IDE 2, `setup()` / `loop()`, `pinMode()`, `digitalWrite()`, `delay()`.
- **Activities:**
 - Run Blink in Wokwi, change rhythm, add a second alternating LED.
 - Upload sketch to physical ESP32 board.
 - Program distinct product "moods" (calm pulse, alarm, heartbeat).
- **Design Reflection:** Which blinking patterns are unambiguous to a user without instructions? What differs between simulated and real physical light?
- **Validation:** Test the three mood rhythms with a peer without advance explanation. Can the observer immediately identify which is "calm", "alarm", or "heartbeat"?

Ex 1 – The Tactile Switch: GPIO Input & Interaction

- **Goal:** Read user intention via a push button and build a responsive control loop.
- **Key Concepts:** `digitalRead()`, internal pull-up (`INPUT_PULLUP`), Serial Monitor, conditional logic (`if/else`), momentary vs. latching/toggle state, switch bouncing.
- **Activities:**
 - Wire button to GND (no external resistor required). Inspect pin state.
 - Implement direct momentary feedback (push-to-light).
 - Implement state memory: toggle lamp state ON ↔ OFF per press.
 - Observe physical contact bouncing on hardware; add debouncing.
- **Design Reflection:** Momentary (doorbell) vs. toggle (table lamp) affordances. How critical is instantaneous visual feedback for user confidence?
- **Validation:** Confirm toggle reliability across 20 rapid presses with zero bounce glitches; validate that the physical tactile sensation matches user expectations.

Ex 2 – The Continuous Dial: Analog Read & PWM

- **Goal:** Capture continuous gestures and produce nuanced modulated output.
- **Key Concepts:** ADC (12-bit, 0–4095), potentiometer voltage divider, `analogRead()`, Serial Plotter, PWM (`analogWrite()`), linear scaling with `map()`.
- **Activities:**
 - Wire potentiometer, read ADC values, inspect smooth curves on Serial Plotter.
 - Dim an LED via PWM duty cycle modulation.
 - Map ADC input (0–4095) to PWM brightness (0–255).
 - Alternative modulation: map knob position to blink tempo (50 ms–1500 ms).
- **Design Reflection:** When is a continuous dial superior to a discrete button? How do friction, throw, and knob resistance communicate product quality?
- **Validation:** Check Serial Plotter for full-range ADC linearity without dead zones; validate with a user if they can dial in a target light level by physical feel alone.

Ex 3 – The Isolated Smart Object: State & Non-blocking

- **Goal:** Synthesize primitives into a standalone prototype with distinct modes.
- **Key Concepts:** Sense → Decide → Act loop, Finite-State Machines (`OFF`, `FOCUS`, `ALERT`), non-blocking timing with `millis()`, modular project hardware.
- **Activities:**
 - Eliminate `delay()`: replace with non-blocking timestamp arithmetic.
 - Optional. Integrate chosen Project Module (LDR, PIR, NeoPixel, Servo, or DHT22).
- **Design Reflection:** Compile the **Isolation Limitations List**: everything the object *cannot* do alone (know room occupancy, alert remotely, report wear).
- **Validation:** Verify zero input lag during continuous LED fading.

```
for (int b = 0; b <= 255; b++) {    // fade up
    analogWrite(LAMP_PIN, b);
    delay(8);                      // 256 × 8 ms ≈ 2 s of blindness
}
```

```
unsigned long lastToggleAt = 0;
const unsigned long INTERVAL_MS = 500;

void loop() {
    unsigned long now = millis();
    if (now - lastToggleAt >= INTERVAL_MS) {    // has the interval elapsed?
        lastToggleAt = now;
        ledOn = !ledOn;
        digitalWrite(LED_PIN, ledOn);
    }
    // ...everything else (button, pot) runs on every pass, thousands of times per second
}
```

Phase B: The Connected Object

Exercises 4 – 6

Putting the object online, binding to cloud variables, and designing remote services

The Mental Model of Arduino Cloud

“ A cloud variable is a variable that exists in two places at the same time – on the board and in the cloud – and Arduino Cloud keeps them in sync.”

- **Device:** The physical ESP32 registered with Device ID & Secret Key.
- **Thing:** The digital twin containing cloud variables and network credentials.
- **Cloud Variables:** Defined by name, type, update policy, and permission (*Read only* for telemetry; *Read & Write* for remote control).
- **Auto-generated Skeleton:** Handles Wi-Fi connection and sync in `ArduinoCloud.update()`.

Ex 4 — Going Online: Device, Thing & First Dashboard

- **Goal:** Connect the physical prototype to the cloud and stream live sensor data.
- **Key Concepts:** Provisioning third-party devices, cloud variables (*Read only*), `thingProperties.h`, `ArduinoCloud.update()`, live dashboard widgets.
- **Activities:**
 - Register ESP32 in Arduino Cloud; configure Thing and Wi-Fi credentials.
 - Add sensor variables (e.g. `light`, `temperature`, `status`).
 - Merge Exercise 3 logic into the cloud sketch and upload.
 - Build a live monitoring dashboard with value, gauge, and status widgets.
- **Design Reflection:** Your data is now visible to the internet. What are the privacy implications? What should stay local at the edge vs. sent to the cloud?
- **Validation:** Measure latency between physical sensor changes and dashboard updates; conduct a peer privacy audit on data visible on screen.

Ex 5 – Remote Control: Mobile App & Consistency

- **Goal:** Enable remote actuation while keeping physical and digital controls consistent.
- **Key Concepts:** *Read & Write* variables, `onXxxChange()` callbacks, control widgets (switches, sliders, color pickers), **Arduino IoT Remote** mobile app.
- **Activities:**
 - Add controllable variables (e.g. `brightness`, `mode`, `lampColor`).
 - Implement callbacks to actuate LEDs/servos when cloud values change.
 - Synchronize physical inputs: pressing the hardware button updates the cloud.
 - Optimize the dashboard layout for mobile smartphones.
- **Design Reflection:** The object has two simultaneous interfaces (physical & digital). Who "wins" when both are operated at once? How do you prevent confusion?
- **Validation:** Concurrently actuate the physical knob and cloud slider; validate that both converge without feedback loops or desynchronization.

Ex 6 – The Object as a Service: Policies & Resilience

- **Goal:** Package the prototype into a reliable, secured, and well-designed service.
- **Key Concepts:** Update policies (*on change* with delta threshold vs. *periodic*), specialized variable types, connection callbacks, security of secret keys.
- **Activities:**
 - Tune update policies to minimize energy and cloud traffic.
 - Implement offline signaling on the board (e.g. slow pulse while connecting).
 - Redesign dashboards specifically tailored to distinct user personas.
 - Compile the Cloud sketch locally in Arduino IDE 2 to bypass daily quotas.
- **Design Reflection:** Which part is the product: the physical object or the digital service? Sketch the end-to-end service blueprint.
- **Validation:** Test dashboard usability with a target persona (under 30s comprehension); disconnect Wi-Fi to validate graceful offline behavior.

Phase C: The Ecosystem

Exercises 7 – 10

Loose coupling, variable synchronization, orchestration, and emergent behavior

The Shared Variable Contract

For objects built by different teams to interoperate, they must share a common schema:

Variable	Type	Permission	Purpose	Producer
studioScene	int (0–3)	Read & Write	Global room mode (Normal, Focus, Pres., Closed)	Hub, Scene Switch
someoneAtDoor	bool	Read only	Person arrived at studio entrance	Door Badge
roomOccupied	bool	Read only	Human presence in room	Presence Totem
co2 / noiseLevel	int	Read only	Environmental air/sound quality	Studio Monitors
<obj>Heartbeat	int	Read only	Incremental liveness counter (1/min)	Every Object

Design Insight: Designing the variable contract is as fundamental as designing the physical industrial casing!

Ex 7 – Objects Talking to Objects: Variable Sync

- **Goal:** Enable direct multi-device cooperation without centralized code.
- **Key Concepts:** Variable synchronisation between Things, contract adherence, loose coupling (producer does not know consumer), events vs. commands.
- **Activities:**
 - Pair up with another group's Thing (e.g. Door Badge triggers Desk Lamp greeting).
 - Link variables using *Sync with other Things*. Write reactive callbacks.
 - Swap partners: connect to a different group's object without editing firmware.
- **Design Reflection:** What made connecting to another group's object easy or painful? How does a contract mirror physical connector standards?
- **Validation:** Validate interoperability by swapping partner objects; verify that data binding works seamlessly without code recompilation.

Ex 8 – Who is There? Heartbeats & Resilience

- **Goal:** Make the ecosystem resilient to missing, disconnected, or unpowered devices.
- **Key Concepts:** Device online/offline events, heartbeat counters, fallback strategies, **graceful degradation**, studio map overview dashboard.
- **Activities:**
 - Publish `<object>Heartbeat` and capability strings every minute.
 - Monitor partner heartbeats; switch to local fallback if silent for 3 minutes.
 - Unplug devices at random to test network fault tolerance.
 - Build a shared **Studio Map Dashboard** with live floor plan status.
- **Design Reflection:** In IoT, failures are normal daily events. How should an object communicate to users that its partner is missing without appearing "broken"?
- **Validation:** Unplug partner objects and verify that local fallback takes over within the timeout; check that the studio map dashboard clearly reflects liveness.

Ex 9 – The Orchestrator: Scenes, Hub & Services

- **Goal:** Introduce central coordination and external data feeds.
- **Key Concepts:** Coordinated scenes (`studioScene`), Hub Thing (context rules), time-based schedulers, cloud triggers, webhooks & weather services.
- **Activities:**
 - Build a studio scene selector (e.g. "Presentation": dim lamps, silence totems).
 - Create hub rules combining inputs (e.g. Empty room + window open → alert).
 - Add scheduled automation (e.g. evening studio shutdown).
 - Ingest external data: sync with a Weather Thing (rain forecast → umbrella cue).
- **Design Reflection:** Edge vs. Cloud: which rules belong in the device (fast, private, offline) vs. in the cloud (global view, multi-source context)?
- **Validation:** Trigger multi-device scenes and inspect synchronization; disconnect external internet to observe which local interactions survive.

Ex 10 – Ecosystem Design Sprint: Emergence & Conflicts

- **Goal:** Conceive, model, and resolve system-wide ecosystem interactions.
- **Key Concepts:** System mapping, emergent properties, contradictory rule conflicts, user override hierarchies, space tagging in Arduino Cloud.
- **Activities:**
 - Map all objects, variables, spatial tags, and actors in the room.
 - Identify and implement an **emergent feature** (e.g. unified comfort index).
 - Simulate and resolve a **conflict** (e.g. auto-off rule vs. reading in a corner).
 - Tag all Things by zone/role (`area:desk` , `role:sensor`).
- **Design Reflection:** Compare current system capabilities with the limitation list from Exercise 3. What new risks (lock-in, complexity) were created?
- **Validation:** Stage opposing rules simultaneously to validate override priorities; test whether the emergent comfort index aligns with actual human perception.

Phase D: Data-Driven Design

Exercises 11 – 14

Gathering field telemetry, understanding human use, predictive maintenance, and OTA evolution

Ex 11 – Remembering: Storing & Visualising Data

- **Goal:** Transition from ephemeral live streams to historical design telemetry.
- **Key Concepts:** Cloud time-series, telemetry (continuous) vs. events (discrete actions), event counters, **instrumentation plans**, CSV export.
- **Activities:**
 - Draft an instrumentation plan: connect design questions to specific metrics.
 - Instrument the sketch with event variables (e.g. `dimmerUses`, `lastMode`).
 - Build a dedicated historical "Usage Dashboard" with time-window charts.
 - Download raw CSV logs; create initial pivot tables in a spreadsheet.
- **Design Reflection:** Data only answers questions you anticipated. What would you like to know that cannot be sensed? What data must be minimized (GDPR)?
- **Validation:** Compare 24-hour logs against known physical actions to confirm no events were lost; audit metrics against data minimization standards.

Ex 12 – How is it Really Used? Field Deployment

- **Goal:** Deploy in real contexts to extract qualitative & quantitative design insights.
- **Key Concepts:** In-the-wild deployment, usage analytics, identifying "struggle" patterns, comparing intended vs. actual user behavior.
- **Activities:**
 - Deploy objects in the studio, department, or home for **at least 1 week**.
 - Analyze CSV logs for non-used features, struggle patterns, or flawed defaults.
 - Conduct short follow-up user interviews to contextualize the numbers.
 - Formulate 3 actionable redesign proposals supported by data.
- **Design Reflection:** Where did data surprise you? Where would data alone have misled you without interviewing real users?
- **Validation:** Cross-validate identified anomalies (e.g. repeated button presses) with qualitative interview quotes to ensure true user intent is understood.

Ex 13 – Keeping it Alive: Health & Maintenance

- **Goal:** Instrument products for diagnostics, reliability, and maintenance.
- **Key Concepts:** Health metrics (uptime, Wi-Fi RSSI, heap memory, battery voltage), wear counters (cycles, on-hours), drift detection, **predictive maintenance**.
- **Activities:**
 - Add persistent wear counters saved across reboots.
 - Configure multi-tier alarms (critical, major, minor, warning).
 - Build a "Fleet Health" dashboard for the space manager/technician.
 - Implement automated push/email triggers for diagnostic alerts.
- **Design Reflection:** How does maintenance telemetry alter physical industrial design (battery access, antenna placement, modular replacement parts)?
- **Validation:** Simulate hardware faults (voltage drop, signal shielding, frozen sensor); confirm alerts fire at correct severity without notification fatigue.

Ex 14 – Closing the Loop: Settings & OTA Updates

- **Goal:** Update and improve deployed products remotely based on data insights.
- **Key Concepts:** Cloud configuration variables (zero-recompile parameter tuning), A/B testing in hardware, Over-The-Air (OTA) firmware deployment.
- **Activities:**
 - Expose operational parameters as cloud sliders (e.g. `autoOffDelay`, `sensitivity`).
 - Deploy an A/B test across studio objects to evaluate two feedback variants.
 - Measure variant performance on an A/B analytics dashboard.
 - Deploy winning firmware over-the-air across the fleet.
- **Design Reflection:** A product is no longer static when sold. What does continuous evolution mean for brand trust, user agency, and ethics?
- **Validation:** Alter remote settings and confirm instant live adaptation; push an OTA update and verify smooth installation with zero manual intervention.

Final Synthesis

Exercise 15 & Cross-Cutting Themes

Ex 15 – The Final Capstone Project

- **Brief:** Design a complete **Product-Service Ecosystem** addressing a chosen context (workplace, healthcare, retail, domestic, or cultural spaces):
 1. ≥ 3 **Interconnected Objects:** Sharing a documented variable contract.
 2. **Emergent Value:** Delivering functionality impossible for a single device.
 3. **Instrumentation & Field Study:** Quantitative telemetry + user interviews.
 4. **Data-Driven Iteration:** Demonstrating at least one validated redesign (OTA / settings) justified by data.
- **Validation Requirements:**
 - Live ecosystem demonstration under real environmental conditions.
 - Interoperability test verifying error-free contract compliance.
 - Empirical proof that field data justified the redesign iteration.
 - Critical evaluation of privacy, ethical constraints, and offline degradation.

Cross-Cutting Design Themes

Throughout all phases, keep these strategic questions at the center of the design critique:

- **Privacy & Ethics:** Transparency to bystanders; data minimization by default.
- **Security:** Guarding secret keys, Wi-Fi credentials, and dashboard access.
- **Sustainability:** Power optimization (sleep modes), longevity, and circular repair.
- **Platform Dependency:** Avoiding catastrophic lock-in; defining an exit plan.
- **Interoperability:** From the studio contract to emerging standards (Matter, Home Assistant).
- **Edge vs. Cloud:** Preserving core product utility when connectivity drops.

Design with Data. Build in Ecosystems.

Ready to prototype in Wokwi, IDE 2, and Arduino Cloud.

[Read Full Course Document \(intro.md\)](#)

Andrea Vitaletti — *IoT for Product Design*