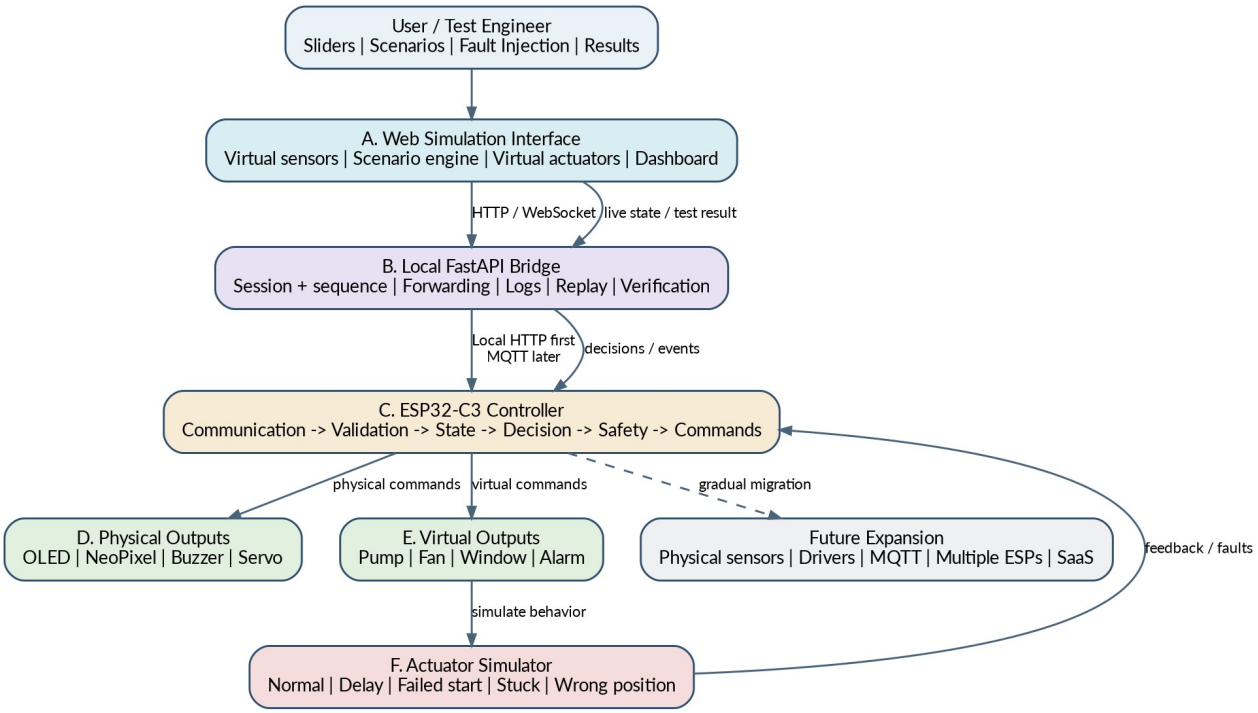
ESP32 VIRTUAL CONTROL LAB

Complete System Blueprint

Digital twin, hardware-in-the-loop control, safety, feedback, and automated testing



Version 1.0 | August 4, 2026

How to use this blueprint

This document shows the complete system, the responsibility of every branch, the contracts connecting branches, and the correct order for implementation. The project begins as a virtual test environment, becomes a hybrid system, and can later grow into a real smart-agriculture or embedded-control platform.

> **Core design rule**
>
> No interface, sensor, actuator, or communication method may bypass the central control loop. All inputs become canonical sensor state; all requested actions pass through the safety supervisor; all outputs produce observable events.

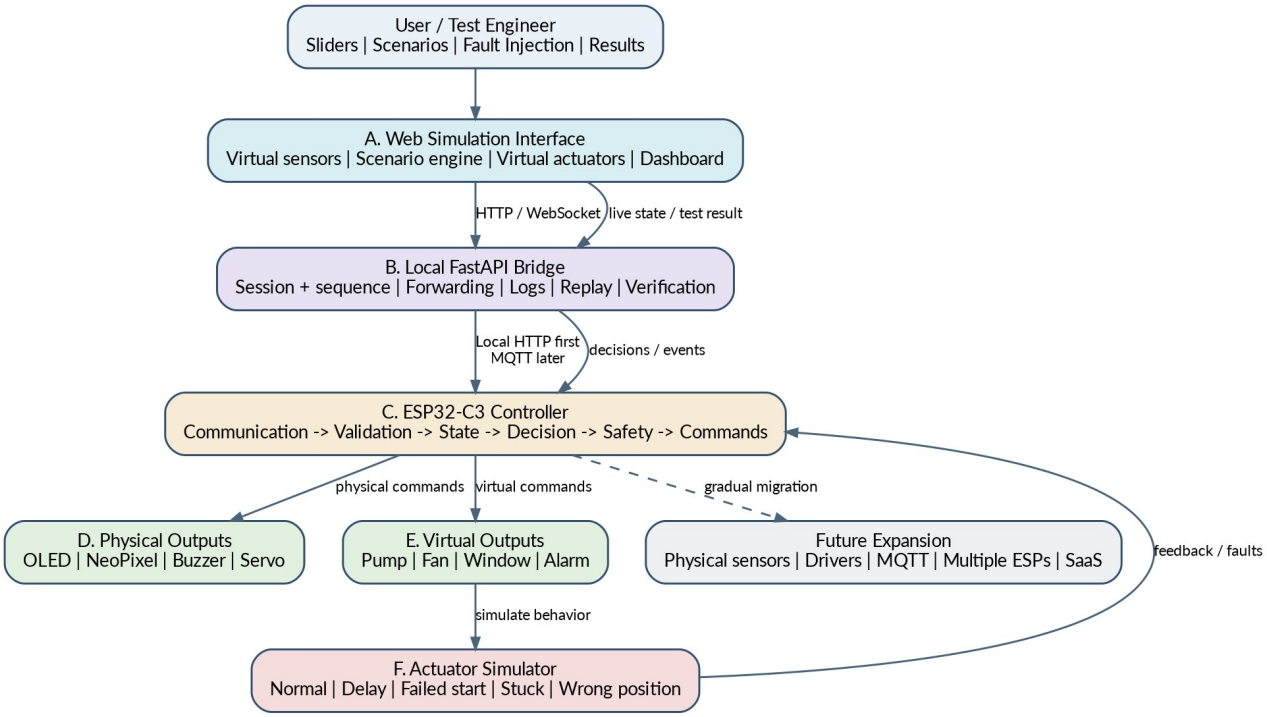
Contents

1. Mission and complete architecture

Build a digital-twin and hardware-in-the-loop control platform where a website generates virtual sensor conditions, a real ESP32-C3 makes the control decisions, and the results appear through both virtual and physical outputs.

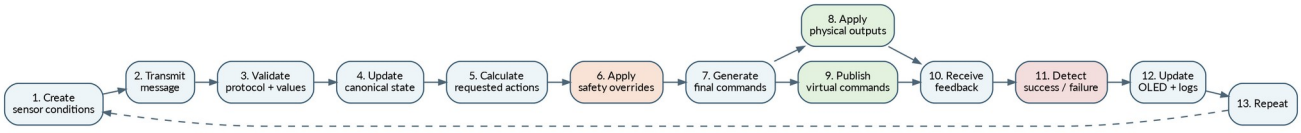
```
Virtual environment -> Virtual sensor data -> FastAPI bridge -> ESP32-C3 -> Validation -> State ->
Decision -> Safety -> Commands -> Physical + virtual outputs -> Feedback -> ESP32
```

Complete architecture



The ESP32 remains the control authority. The browser simulates the world, FastAPI coordinates communication and testing, and the actuator simulator returns feedback. Later, virtual devices can be replaced one by one with physical hardware.

2. Central control loop



1. Create sensor conditions manually or through a scenario.
2. Transmit a versioned sensor message.
3. Validate protocol, session, sequence, fields, types, and ranges.
4. Update the authoritative canonical sensor state.
5. Calculate requested actions using the stateful decision engine.
6. Apply safety priorities and overrides.
7. Generate final actuator commands.
8. Apply physical outputs and publish virtual commands.
9. Receive simulated or measured actuator feedback.
10. Detect success, delay, mismatch, or failure.
11. Update OLED, LED, buzzer, event history, and test result.
12. Repeat without blocking the runtime.

Architectural boundary

The decision engine requests actions. It does not directly touch GPIO, PWM, OLED, Wi-Fi, or HTTP. The safety supervisor converts requested actions into final commands, and the actuator manager applies them.

3. Project branches and interface contracts

The system is divided into branches so that each part can be developed and tested independently. Branches connect only through explicit data contracts.

Producer	Contract / output	Consumer
Website	Virtual sensor values and scenario events	FastAPI bridge
FastAPI	Versioned sensor message	ESP communication layer
Communication layer	Validated packet	Sensor adapter
Sensor adapter	Canonical sensor values	State manager
State manager	Current sensor/system/actuator state	Decision engine
Decision engine	Requested actions + reasons + rule IDs	Safety supervisor
Safety supervisor	Final commands + safety overrides	Actuator manager
Actuator manager	Physical driver calls	OLED, LED, buzzer, servo
Actuator manager	Virtual commands	Website actuator simulator
Actuator simulator	Simulated feedback + faults	ESP feedback handler
Every branch	Structured events	Logging and verification system

Branch overview

Branch	Purpose
1. Product definition	Defines mission, scope, sensors, actuators, qualities, and constraints.
2. Website simulation	Creates virtual conditions, scenarios, faults, and dashboards.
3. FastAPI bridge	Coordinates browser-to-ESP communication, sessions, logs, replay, and tests.
4. Protocol	Defines stable messages and validation rules.
5. ESP communication	Moves data without deciding behavior.
6. Sensor abstraction	Makes virtual and physical inputs equivalent after adaptation.
7. State management	Maintains authoritative sensor, system, and actuator state.
8. Decision engine	Calculates normal requested actions.
9. Safety supervisor	Has final authority and resolves conflicts.
10. Actuator abstraction	Separates requested, commanded, simulated, measured, and fault states.
11. Physical outputs	Shows local state through OLED, light, sound, and servo.
12. Observability	Records meaningful events and state changes.
13. Testing	Verifies logic, integration, failures, endurance, and replay.
14. Recording and replay	Reproduces experiments and compares firmware versions.
15. Physical expansion	Replaces virtual components gradually.
16. Platform expansion	Adds MQTT, multi-device support, database, authentication, and SaaS.

1 Product definition

Defines what the system must achieve before implementation begins.

Primary use case

A virtual greenhouse controller that can later become a real greenhouse or equipment-control system.

Initial sensors and actuators

Item	Range / state	Initial implementation
Temperature	-10 to 60 C	Virtual sensor
Soil moisture	0 to 100%	Virtual sensor
Water-tank level	0 to 100%	Virtual sensor
Rain	Yes / No	Virtual sensor
Greenhouse window	Closed / half / open	Physical servo + virtual display
Pump	On / Off	Virtual actuator
Ventilation fan	On / Off	Virtual actuator
Alarm	Normal / warning / critical	NeoPixel + buzzer + website
Information panel	Current state and reason	OLED

Required system qualities

- Local-first operation
- Explainable decisions
- Safe behavior after communication loss
- Modular virtual-to-physical replacement
- Independent ESP decision-making
- Replayable and testable behavior

Branch output

A written specification containing sensors, actuators, ranges, rules, limits, safe states, and acceptance tests.

2 Website simulation environment

Represents the world surrounding the ESP32.

Manual controls

- Sliders for continuous values
- Toggles for Boolean values
- Exact numeric entry
- Automatic-send interval
- Connection status and last response

Sensor behavior simulation

- Gradual change
- Noise
- Frozen value
- Disconnected sensor
- Impossible value
- Delayed transmission
- Random spikes

Scenario engine

```
0 s: normal -> 5 s: temperature rises -> 10 s: soil becomes dry -> 15 s: tank falls below safety limit
```

Virtual actuator display and feedback

- Show commanded state
- Simulate normal state
- Simulate delayed startup
- Simulate failed startup
- Simulate stuck on/off
- Return faults to ESP

Connection

Website -> FastAPI -> ESP for sensor input; ESP -> FastAPI -> Website for decisions, commands, events, and test results.

3 Local FastAPI bridge

Coordinates communication and test execution without taking control authority away from the ESP.

Initial responsibilities

- Accept website sensor data
- Create session IDs and sequence numbers
- Forward messages to the ESP
- Return ESP decisions
- Handle connection errors

- Create logs

Later responsibilities

- Run scenario timelines
- Record and replay sessions
- Compare expected and actual results
- Store history
- Support multiple devices
- Connect to MQTT

Initial API surface

```
POST /api/devices/{device_id}/sensors
GET  /api/devices/{device_id}/status
POST /api/devices/{device_id}/scenario
GET  /api/devices/{device_id}/events
```

Boundary

FastAPI may coordinate, store, and verify. The ESP32 remains responsible for decision and safety logic.

4 Communication protocol

Creates a stable agreement between browser, backend, and ESP.

Sensor message

```
{
  "protocol_version": 1,
  "device_id": "esp32-lab-01",
  "session_id": "web-73dfe2",
  "sequence": 18,
  "sent_at": "2026-08-04T03:00:00+09:00",
  "values": {
    "temperature_c": 38.0,
    "soil_moisture_percent": 22.0,
    "water_level_percent": 70.0,
    "rain": false
  }
}
```

ESP response

```
{
  "accepted": true,
  "boot_id": "boot-2319",
  "sequence": 18,
  "mode": "automatic",
  "commands": {"pump": true, "fan": true, "window_angle": 170},
  "triggered_rules": ["IRRIGATION-001", "TEMPERATURE-002"],
  "reasons": ["Soil moisture is below 30%", "Temperature is above 35 C"]
}
```

Protocol rules

- New browser start creates a new session_id
- ESP restart creates a new boot_id
- Sequence increases inside each session
- ESP monotonic time controls safety timeout
- Unknown, missing, oversized, duplicated, or impossible values are rejected
- Protocol version is included in important messages

5 ESP communication layer

Moves information reliably but does not decide actuator behavior.

- Connect and reconnect to Wi-Fi
- Accept local HTTP requests
- Limit request size

- Parse JSON
- Return structured errors
- Report communication state
- Never block safety and output tasks

Communication states

```
OFFLINE -> CONNECTING -> ONLINE -> DATA_ACTIVE -> DATA_STALE -> RECONNECTING
```

Branch output

Either a validated internal packet or a precise rejection error. No rule evaluation occurs here.

6 Sensor abstraction

Makes network and GPIO sources equal only after they are converted into the same internal model.

```
Virtual website -> Sensor adapter -> Canonical value
Physical GPIO/ADC/I2C -> Sensor adapter -> Canonical value
```

Canonical sensor record

```
{
  "name": "temperature",
  "value": 38.0,
  "unit": "celsius",
  "source": "virtual",
  "quality": "good",
  "received_at_ms": 456789,
  "valid": true
}
```

Source modes

- VIRTUAL
- PHYSICAL
- HYBRID
- DISABLED

Important distinction

Network data does not become an electrical GPIO signal. Both network values and physical readings are adapted into the same canonical sensor state.

7 State management

Maintains one authoritative state used by decision, safety, display, and reporting.

State domains

State	Contains
Sensor state	Current canonical sensor values, validity, source, quality, and age.
System state	Mode, communication state, data quality, alarm level, boot ID, and recovery status.
Actuator state	Requested, commanded, simulated, measured, and fault states.

System state transitions

```
BOOT -> CONNECTING -> READY -> AUTOMATIC -> WARNING / SAFE / FAULT -> RECOVERY -> AUTOMATIC
```

Rule

State changes must follow defined transitions. Avoid disconnected Boolean flags that can create impossible combinations.

8 Decision engine

Calculates what should happen under valid normal operating conditions.

Temperature rules

Condition	Requested fan	Requested window
Temperature <= 28 C	Off	Closed
28 C < temperature <= 35 C	On	Half open
Temperature > 35 C	On	Fully open

Irrigation rules

- Moisture below 30%, tank at least 15%, and no rain -> request pump ON
- Moisture above 40% -> request pump OFF
- Between 30% and 40% -> keep previous pump state

Decision output

- Requested actuator values
- Triggered rule IDs
- Human-readable reasons
- Decision ID
- No direct hardware calls

Statefulness

Hysteresis requires previous actuator state. Decision tests must include sequences, not only isolated values.

9 Safety supervisor

Has final authority over every actuator command.

Priority order

1. Emergency
2. Safety
3. Equipment protection
4. Automatic operation
5. Manual request
6. Optimization

Conflict example

```
Decision engine: dry soil -> pump ON
Safety supervisor: tank below 15% -> pump OFF
Final command: pump OFF
```

Safe-state matrix

Condition	Pump	Fan	Window	Alarm
Startup	Off	Off	Closed	Startup indication
Valid operation	Decision	Decision	Decision	Normal
Low tank	Off	Decision	Decision	Warning
Stale data	Off	Configured safe state	Safe angle	Warning
Controller fault	Off	Configured state	Safe angle	Critical
Emergency stop	Off	Off	Safe angle	Critical

Recovery

```
Failure -> Safe state -> Several consecutive valid messages -> Stable communication confirmed ->
Clear fault -> Resume automatic operation
```

10 Actuator abstraction

Separates intent, command, simulation, measurement, and fault evidence.

Required vocabulary

- requested_state - wanted by normal rules
- commanded_state - final value after safety

- simulated_state - result produced by the virtual actuator
- measured_state - physical feedback from a real sensor
- fault_state - known failure or mismatch

Example

```
Requested pump: ON
Commanded pump: OFF
Simulated pump: OFF
Measured pump: unavailable
Fault: LOW_WATER_BLOCK
```

> **Truthfulness rule**
>
> Without physical feedback, the ESP knows only what it commanded. Never label a command as the actual physical state.

11 Physical output layer

Provides immediate local visibility and demonstration behavior.

Output	Purpose	Recommended behavior
OLED	Local information	Sensor summary, commands, reason, connection, warning
NeoPixel	System status	Green normal, blue automatic, yellow warning, red critical, purple manual
Buzzer	State-change alarm	Short confirmation, warning pattern, critical pattern, connect/disconnect tones
Servo	Window representation	10 degrees closed, 90 half open, 170 fully open

- Warnings interrupt normal OLED page rotation
- Buzzer sounds only on state changes
- Servo uses an appropriate external power supply
- All physical drivers are called only by the actuator manager

12 Event logging and observability

Makes every important decision and failure explainable.

```
SENSOR_RECEIVED
SENSOR_REJECTED
STATE_UPDATED
RULE_TRIGGERED
SAFETY_OVERRIDE
COMMAND_CHANGED
FEEDBACK_RECEIVED
FAULT_DETECTED
SAFE_MODE_ENTERED
RECOVERY_COMPLETED
```

Event path

```
ESP serial log -> ESP event response -> FastAPI event storage -> Website timeline
```

> **Noise control**
>
> Log meaningful transitions, not the same state on every loop iteration.

13 Testing system

Connects to every branch from the beginning, not only at the end.

Test level	Purpose	Example
Unit	Pure logic	40 C -> window fully open
Boundary	Exact thresholds	28.0, 28.1, 35.0, 35.1
Sequence	Stateful hysteresis	50 -> 20 -> 35 -> 45% moisture
Integration	Connected components	FastAPI -> ESP -> servo/OLED -> response
Failure	Safe behavior	Bad JSON, timeout, Wi-Fi loss, actuator failure
Endurance	Long-run stability	Thousands of updates, memory and reconnection checks

Verification rule

Every requirement should have a matching test ID, expected result, and completion gate.

14 Recording and replay

Reproduces complete experiments and detects regressions.

- Record protocol and rule versions
- Record input timing and values
- Record decisions and safety overrides
- Record commands, feedback, faults, and test results
- Replay identical sequences against changed firmware
- Compare old and new behavior

```
Recorded scenario -> FastAPI replay engine -> ESP -> New result -> Comparison report
```

15 Future physical expansion

Replaces virtual parts gradually without changing the core architecture.

Virtual component	Physical replacement
Virtual temperature	Real temperature sensor
Virtual soil moisture	Real moisture sensor
Virtual tank level	Float, ultrasonic, pressure, or capacitive sensor
Virtual rain	Real rain sensor
Virtual pump	Driver + low-voltage pump
Simulated feedback	Current, flow, position, or limit feedback

Permanent feature

Virtual mode remains valuable after real hardware is added because it supports fault injection, regression testing, and demonstrations.

16 Future platform expansion

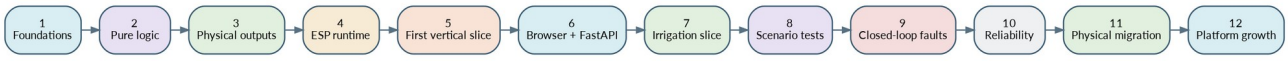
Grows the local lab into a reusable product only after the core is stable.

```
One ESP -> Multiple ESP devices -> MQTT broker -> Database -> Authentication -> Remote dashboard -> SaaS platform
```

Possible products

- Smart greenhouse controller
- Equipment-testing platform
- Embedded software training system
- Digital-twin laboratory
- Automated firmware verification service
- Small-farm monitoring and control platform

5. Correct development order



Execution strategy

Build one thin end-to-end vertical slice first: virtual temperature -> ESP decision -> servo -> OLED -> response. Stabilize it before adding irrigation, scenarios, faults, MQTT, or real equipment.

Stage 1 - Foundations

1. Confirm the exact ESP32-C3 board.
2. Record the MicroPython version.
3. Create the complete pin map.

4. Verify OLED, NeoPixel, buzzer, and servo wiring.
5. Define the first use case.
6. Define safe states.
7. Define the communication protocol.
8. Define acceptance tests.

Stage 2 - Pure logic

9. Create canonical sensor structures.
10. Create system and actuator state structures.
11. Implement the stateful decision engine on the computer.
12. Implement the safety supervisor.
13. Test normal conditions.
14. Test exact boundaries.
15. Test conflicting rules.
16. Test state sequences.

Stage 3 - Local physical outputs

17. Test OLED independently.
18. Test NeoPixel independently.
19. Test buzzer independently.
20. Test servo independently.
21. Test all outputs together.
22. Connect hardcoded decisions to outputs.
23. Confirm servo power does not reset the ESP.

Stage 4 - ESP runtime

24. Build the asynchronous runtime.
25. Add shared state management.
26. Add the event system.
27. Add stale-data detection.
28. Add recovery logic.
29. Add the HTTP server.
30. Add request-size and validation limits.

Stage 5 - First vertical slice

31. Implement only virtual temperature.
32. Send temperature with curl.
33. Validate the message.
34. Calculate the window command.
35. Apply the servo command.
36. Display the reason on OLED.
37. Return the decision as JSON.
38. Test repeated messages.
39. Test invalid values.
40. Test timeout and recovery.

Stage 6 - Browser and FastAPI

41. Create the local FastAPI bridge.
42. Forward temperature to the ESP.
43. Create the website connection panel.
44. Create one temperature slider.
45. Display the ESP response.

46. Display the virtual window.
47. Add an event log.
48. Run hundreds of updates.

Stage 7 - Irrigation slice

49. Add soil moisture.
50. Add tank level.
51. Add rain.
52. Add pump hysteresis.
53. Add low-tank protection.
54. Add rain protection.
55. Add virtual pump and fan.
56. Connect LED and buzzer warnings.
57. Add corresponding OLED pages.

Stage 8 - Scenario testing

58. Add normal, hot, dry-soil, low-tank, rain, invalid-data, and communication-loss scenarios.
59. Define expected commands, mode, alarm, and response time.
60. Add automatic PASS/FAIL comparison.

Stage 9 - Closed-loop simulation

61. Add simulated actuator feedback.
62. Add startup delays.
63. Add failed startup.
64. Add stuck-on and stuck-off faults.
65. Add incorrect virtual servo position.
66. Make the ESP respond to feedback faults.

Stage 10 - Reliability

67. Add recording and replay.
68. Add rule versioning.
69. Run long-duration tests.
70. Add a watchdog only after runtime stability.
71. Document known limits.

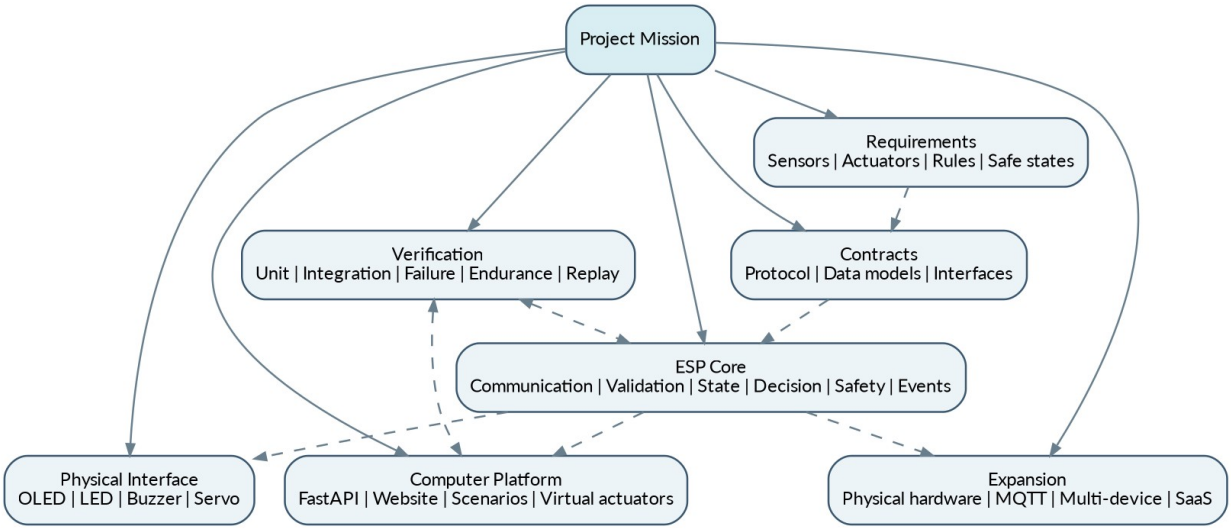
Stage 11 - Physical migration

72. Add a real temperature sensor.
73. Add per-sensor source selection.
74. Add real soil, tank, and rain sensors.
75. Add a properly driven pump.
76. Add physical feedback where possible.

Stage 12 - Platform growth

77. Add MQTT.
78. Add multiple ESP devices.
79. Add persistent storage.
80. Add authentication.
81. Add remote access.
82. Package the system as a reusable control and testing platform.

6. Dependency tree



Critical path

```
Requirements -> Contracts -> Decision + safety logic -> Physical drivers -> ESP communication ->
First vertical slice -> Scenario verification -> Reliability -> Expansion
```

Features outside the early critical path

- User accounts
- Cloud hosting
- Database
- MQTT
- Multiple ESP devices
- Advanced animations
- AI recommendations
- Real pump and agricultural hardware

> **Scope control**
> Do not allow non-critical platform features to delay the first proven control loop.

7. First complete project version

The first fully connected release should demonstrate the following sequence from beginning to end:

1. User opens the local website.
2. FastAPI confirms that the ESP is online.
3. User selects a preset or changes sensor controls.
4. FastAPI sends a versioned message.
5. ESP validates session, sequence, and values.
6. ESP updates canonical sensor state.
7. Decision engine requests actions.
8. Safety supervisor approves or overrides them.
9. Servo, OLED, LED, and buzzer update.
10. ESP returns commands, rules, and explanations.
11. Website animates the virtual pump, fan, and window.
12. Virtual actuators return simulated feedback.
13. ESP detects whether commands succeeded.
14. Dashboard and OLED show the final state.
15. Test engine reports PASS or FAIL.

What this proves

This version proves simulation, embedded control, safety, physical output, virtual output, feedback, observability, and automated verification in one connected system.

8. Completion gates and future growth

Gate A - First vertical slice

- Temperature message accepted and validated
- Correct window command at all boundaries
- Servo moves without resetting the ESP
- OLED shows value and reason
- Website receives the response
- Timeout enters defined safe state
- Recovery requires stable valid messages

Gate B - Greenhouse MVP

- Temperature, soil, tank, and rain supported
- Pump hysteresis works
- Low tank and rain override irrigation
- OLED, LED, buzzer, and servo are coherent
- Preset scenarios pass automatically
- Commanded and simulated states are separate

Gate C - Test platform

- Actuator failures can be injected
- Feedback faults change ESP behavior
- Sessions can be recorded and replayed
- Regression results compare firmware or rule versions
- Endurance run completes without unrecovered failure

Gate D - Physical migration

- At least one physical sensor can replace its virtual counterpart
- Per-sensor source selection works
- Decision and safety code does not require rewriting
- Virtual mode remains available for testing

Gate E - Platform expansion

- Multiple devices justify MQTT
- Persistent history justifies a database
- Remote users justify authentication
- A repeatable customer workflow justifies SaaS packaging

Final principle

Grow the system only when the current layer is observable, testable, and stable. The value of the project comes from connected correctness, not from the number of features.

Appendix - Core engineering terms

Term	Meaning in this project
Digital twin	A virtual representation of sensor conditions, actuator behavior, and faults.
Hardware-in-the-loop	Real ESP firmware controls a system whose environment or actuators may be simulated.
Canonical state	The single normalized internal representation used by all logic.
Decision engine	Calculates desired actions under normal valid conditions.
Safety supervisor	Overrides desired actions when risk or failure conditions require it.
Hysteresis	Different start and stop thresholds that prevent rapid switching.
Fault injection	Deliberately creating bad data, delays, disconnects, or actuator failures.
Regression test	Replaying known inputs to verify that new firmware still behaves correctly.
Vertical slice	A small but complete path across interface, communication, logic, output, and feedback.