Decision Matrix — When to Use Flow, LWC, or Apex

1) Quick rule of thumb

- **Start with Flow** for guided processes, record updates, and orchestration.
- **Add LWC** when the **UI** needs rich interaction, fast client-side logic, or dynamic layouts beyond standard components.
- **Add Apex** when you need **server-side transactions**, complex business rules, bulk processing, or secure integrations.

---

2) Quick Guide

	Flow (Declarative)	**LWC (Custom UI)**	**Apex (Server/Logic)**
Main Focus	Orchestrate steps; CRUD on records; user guidance	Rich, reactive UI; complex client logic; custom layouts	Complex business logic; transactions; bulk ops; integrations
UI complexity	Standard screens; simple conditional steps	High—dynamic grids, inline edits, autosuggest, virtualized lists	N/A (no UI)
Logic Complexity	Low–medium; branching, formulas, simple loops	Medium client logic that must feel instant	High; multi-object rules; cross-object constraints; recursion control
Data volume	Low–moderate; single record or small sets	Low–moderate on client with pagination	High; bulk DML, async (Queueable/Batch/Sched)
Transactions	Single-object, simple save	Client-side prep only	Multi-step, atomic commits, savepoints/rollbacks

Integrations	Simple callouts via invocable actions	Call UI services; avoid secrets in client	Secure callouts; signed auth; retries; error handling
Performance	Good for step-by-step	Great perceived speed with client rendering	Best for heavy compute/bulk; async patterns
Maintainability	Highest—admin-friendly; versioned	Good if componentized and documented	Good if layered (service classes), unit tested
Security/PHI	Field-level security respected	No secrets in client; guard with FLS	Enforce FLS/CRUD; platform encryption; shield logging
Accessibility	Very good with Screen Flows + SLDS	Must own ARIA/keyboard behavior	N/A (back end)
Release cadence	Frequent, safe iterations	Moderate; coordinate design reviews	Moderate; CI/CD + regression suite

---

3) Escalation triggers

Flow → LWC when:

- You need **dynamic, data-dense** layouts (nested tables, virtualized lists).
- Users need **real-time feedback** (debounced validation, live calculations) without server trips.
- You require **keyboard-first microinteractions** and ARIA beyond Screen Flow widgets.

Flow/LWC → Apex when:

- You must enforce **atomic business rules** across multiple objects in one transaction.
- You need **bulk-safe processing** (hundreds/thousands of records).
- You're doing **secure integrations**, async jobs, or complex query orchestration.

- Governor limits appear in Flow/LWC patterns that Apex can optimize.

---

4) Do / Don’t patterns

Flow — use when

- Guided intake, approvals, routing, SLA orchestration.
- Record create/update with **pre-submit validation** and **duplicate checks**.
- **Human-in-the-loop** steps (screens) with conditional branching.

Avoid / Don’t

- Large loops over big datasets (risk of limits).
- Multi-object atomic commits—use Apex.
- UI behavior that demands **millisecond** responsiveness—use LWC.

LWC — use when

- **High-density UI** (inline editing, grids, quick filters, client sorting).
- Client-side validation, typeahead, incremental search, optimistic UI.
- Composing multiple data sources into one responsive view.

Avoid / Don’t

- Storing secrets or PHI logic in client code.
- Reinventing standard forms that Screen Flow already solves.
- Heavy data joins on client—move to Apex services.

Apex — use when

- **Business invariants** must be enforced server-side.
- **Bulk** data transformations, batch/queueable jobs, scheduled tasks.
- **Resilient integrations** (retry, backoff, circuit breaker patterns).
- **Testable domain logic** shared by multiple entry points (Flow, LWC, triggers).

Avoid / Don’t

- UI rendering or simple CRUD you could do in Flow.
- Monolithic God classes—use service, domain, selector layers.

---

5) Flowchart (decision path)

1. **Is there a UI?**
    - **No** → Apex (service/batch/integration).
    - **Yes** → Go to 2.
2. **Is a standard guided form/list enough?**
    - **Yes** → Screen Flow.
    - **No** → Go to 3.
3. **Does the UI need fast, reactive behavior or complex layouts?**
    - **Yes** → LWC (call Apex as needed).
    - **No** → Screen Flow.
4. **Do you need atomic multi-object commits, bulk ops, or secure integrations?**

- **Yes** → Add Apex (invocable from Flow/LWC).
- **No** → Keep solution in Flow/LWC.

---

6) Governance & Quality Gates

- **Architecture review:** Confirm choice via matrix; record rationale in the user story.
- **Accessibility:**
  - Screen Flow: Use SLDS components; test keyboard + screen reader paths.
  - LWC: Provide labels, roles, aria-*; ensure focus management.
- **Security:** Enforce CRUD/FLS in Apex; no secrets in LWC; platform encryption for PHI.
- **Performance:** SOQL/DML limits respected; cache where safe; paginate.
- **Testing:**
  - Flow: Path coverage + automation tests (UTAM or equivalent).
  - LWC: Jest unit tests + a11y checks.
  - Apex: ≥85% critical-path coverage; mock callouts; negative tests.
- **Observability:** Emit telemetry (start/stop, errors, retries); log correlation IDs.
- **Documentation:** ADR (Architecture Decision Record) + “Runbook” for ops (timeouts, retries, known limits).

---

7) Reference Templates

Story Template — Flow first

- Goal: Guided intake for ___ role

- Data scope: <objects/fields> (≤ N records per op)
- Validation: <rules> (pre-submit)
- Routing: Omni-Channel by <skill/priority> on **Validated**
- Failure modes: <what happens>
- Telemetry: <events/metrics>

Story Template — LWC

- UX need: <interaction detail> (e.g., inline edit grid with filters)
- State mgmt: <client vs server>
- Data access: Apex controller(s)
- a11y: Keyboard flow, ARIA map
- Telemetry: UI events, latency goals

Story Template — Apex

- Domain rule(s): <invariants, cross-object constraints>
- Data volume: <expected size> (batch/queueable?)
- Integration: <endpoint, auth, retry policy>
- Limits strategy: <SOQL/DML patterns, pagination>
- Tests: <happy/edge/negative>; mocks

---

8) Example mappings from your program

- **Onboarding Wizard (mostly Flow)**

    - Screen Flow for steps, validation, duplicate checks
    - Invocable Apex for NPPES callout + doc completeness scoring
    - Omni-Channel SLA start on **Validated**
- **Emergency QA (Flow + LWC + Apex)**
    - Flow for severity-driven pathing and orchestration
    - LWC for “Docs Needed Now” panel (client validation, instant feedback)
    - Apex for secure file checks, status updates, and escalations

---

9) Anti-regret checks (run before you build)

- Can 80% of this be done in **Flow** with one small invocable Apex? Start there.
- Does the UI demand **rich, reactive interactions**? Add an **LWC** shell.
- Are you touching **multiple objects atomically** or doing **bulk/integration** work? Centralize in **Apex** services.
- Have you documented **why** lesser complexity won’t suffice? If not, you’re over-engineering.