Software Engineering Take-Home Exercise

Project: Household Accountability & Inspection System

Objective: Design and deploy a production-quality web application that enforces structured task accountability in a low-literacy household environment. The goal is to demonstrate clarity of system design, sound engineering judgment, and thoughtful tradeoffs.

1. Context & Problem Framing

A private household operates with domestic staff responsible for recurring duties, errands, maintenance, and inspection-based corrections. The current process relies on memory and verbal follow-up, resulting in forgotten tasks, repeated quality issues, and no objective performance tracking.

The household manager performs structured inspection walk-throughs and identifies deficiencies. There is no enforceable lifecycle, no required completion evidence, and no accountability model tying quality, consistency, and timeliness to performance.

The system must:

- Enforce explicit state transitions.

- Require evidence-based completion.

- Detect repeated inspection failures.

- Support recurring task templates.

- Maintain a structured scoring model prioritizing: Quality → Consistency → Speed → Volume.

- Operate effectively in a low-literacy, bilingual (English/Spanish) environment.

2. User Roles & Constraints

Household Mother (Primary Authority):

- Creates inspection tickets.

- Requires weekly operational reporting.

- Closes tickets.

- Needs trend visibility and repeat-issue detection.

Father (Co-Authority):

- Same closure permissions.

- Requires high-level oversight visibility.

Domestic Employees (Executors): handyman, housekeeper, cook, plumber, electrician, pool maintenance etc.

- Can create tickets.

- Can transition tickets to 'Needs Review' only.

- Low literacy; interface must favor simplicity, icons, and optional voice input.

- Incentivized via shared point system (can go negative; bonuses withholdable).

3. Functional Requirements

Ticket Lifecycle (explicit and enforced):

Open → In Progress → Needs Review → Closed

Recurring tasks may also be marked 'Skipped'.

Only parents may transition to Closed.

Evidence Rules:

- Photo required for completion.

- Inspection tickets require before + after photos.

- Reopened tickets incur additional penalties.

- Repeated inspection findings should be detectable within a 7-day window.

Severity Levels:

- Minor

- Needs Fix Today

- Immediate Interrupt — stop and fix now

Severity influences penalty weight and prioritization.

Recurring Tasks:

- Canonical checklist templates.

- Frequency support (daily, weekly, monthly, etc.).

- Skipped tasks affect consistency scoring.

Reporting (weekly, authority-only view):

- Open, Closed, Skipped, Reopened tickets.

- Points totals and penalties.

- Repeat issue detection.

- Notable patterns or trends.

4. Non-Goals

- Not a collaborative planning tool.

- Not a free-form chat system.

- Not a passive task list.

This is an enforcement and accountability system.

5. Architecture Expectations (Implicit Design Depth)

The system should demonstrate thoughtful handling of:

- Role-based access control.

- State transition validation.

- Data integrity and auditability.

- Recurring task generation logic.

- Time-based penalty computation.

- Repeat-issue detection.

- Separation of concerns between scoring logic and presentation.

- Clear domain modeling.

- Production-ready deployment considerations.

Tradeoffs should be documented. Simplicity is preferred over unnecessary complexity, but design decisions should be intentional.

6. Bonus (Optional but Valuable)

Provide a bilingual chat interface connected to the system's source of truth. The interface should respect user roles and allow:

- Create, read, update, delete tickets.

- Query weekly summaries.

- Retrieve performance metrics.

- Switch between English and Spanish.

- Generate concise operational summaries.

7. Deliverables

Required:

1. Deployed web application (public URL).

2. Dummy users for all personas.

3. Code repository (GitHub or Bitbucket).

4. Learning document outlining architecture decisions, tradeoffs, challenges, and reflections.

Optional:

5. Integrated bilingual chat interface interacting with live system data.

8. Evaluation Criteria

- Functional completeness.

- Rule enforcement rigor.

- Clarity of domain modeling.

- Code organization and readability.

- Production deployment quality.

- Documentation depth and self-awareness.

- Thoughtful handling of edge cases.

- Optional: Quality and integration of chat interface.