1-Day Founding Solutions Engineer Project (No AI)

Project: Workflow Agent Implementation Simulator + Airtable CRM Integration

Goal

Build a small but realistic “customer implementation” platform that simulates deploying an AI agent for a business **without using AI**.

Instead of an LLM, the system uses a **deterministic workflow engine** (rules + step execution) to: - interpret incoming customer requests - validate business rules - create bookings - create/update CRM records (Airtable) - handle edge cases - provide logs + observability

This directly demonstrates the skills required for a **Founding Solutions Engineer** role: customer onboarding, integrations, workflow configuration, debugging, and internal tooling.

Why This Project Fits the Job

What the job wants

- Design and deploy agents tailored to customer workflows
- Configure flows, integrations, and automation (bookings, payments, CRM)
- Debug edge cases and reliability issues
- Build internal tooling/templates/playbooks
- Track implementation health and performance

What this project proves

Even without AI, this project demonstrates: - **Customer discovery → configuration → go-live workflow** - Integration to a real external system (**Airtable CRM**) - Deterministic flow logic (similar to agent tool execution) - Debuggable, observable execution pipeline - Template-driven onboarding (reusable workflow)

This is essentially “the scaffolding of an agent platform.”

Final Deliverable

A deployed web app (Vercel) with:

- **Admin Setup** (business hours + services)

- **Inbox Simulator** (simulate messages from web/email/sms)
- **Workflow Engine** (rule-based execution pipeline)
- **Airtable Integration** (create/update Leads + Bookings)
- **Logs Dashboard** (debugging timeline of decisions)

Core Demo Scenario (Interview Flow)

Demo Script

1. Open `/admin` and show business setup.
2. Open `/inbox` and send a message:
3. “Hi, I’m Marco. Can I book a haircut tomorrow at 3pm?”
4. Show the system response.
5. Open Airtable:
6. new Lead record created
7. new Booking record created
8. Open `/logs`:
9. show intent detection
10. show datetime parsing
11. show Airtable API call result

Optional edge case demo: - “Tomorrow at 10pm” → outside business hours - “I’m angry, this is ridiculous” → escalation rule triggered

---

Scope Constraints (1-Day Build)

Explicitly in scope

- Deterministic workflow engine
- Booking + lead creation in Airtable
- Business config stored locally (DB or JSON)
- Logs stored locally
- UI pages for config/testing/logs

Explicitly out of scope

- OAuth integrations (Google Calendar)
- payments processing (Stripe)
- real messaging provider (Twilio)
- multi-tenant auth
- production security hardening

Recommended Tech Stack

Frontend + Backend

- **Next.js (TypeScript)** (fullstack)

Database

Pick one: - **SQLite + Prisma** (recommended) - or simple JSON storage (fast but less credible)

External Integration

- **Airtable REST API**

Deployment

- Vercel

Airtable Integration Design

Why Airtable

- API key auth (fast)
- visible output (looks like real CRM)
- highly demo-friendly
- minimal setup friction

Airtable Base

Base name: `Customer Agent Demo`

Table 1: Leads

Fields: - `Name` (Single line text) - `Channel` (Single select: web/email/sms) - `Intent` (Single select: booking/cancel/reschedule/info/unknown) - `LastMessage` (Long text) - `CreatedAt` (Date) - `UpdatedAt` (Date)

Table 2: Bookings

Fields: - `Name` (Single line text) - `Service` (Single line text) - `DateTime` (Date) - `Status` (Single select: pending/confirmed/cancelled) - `SourceMessage` (Long text) - `Channel` (Single select)

Airtable Credentials Needed

- Airtable Personal Access Token
- Base ID

Stored in `.env.local`:

```
AIRTABLE_TOKEN=...
AIRTABLE_BASE_ID=...
```

Application Pages (UI)

1) `/admin` — Customer Onboarding Config

Purpose: mimic implementation onboarding.

Features: - Business name input - Timezone input (optional) - Opening hours editor (Mon–Sun open/close) - Services list editor: - service name - duration - Save config

Data stored in DB: - Business - OpeningHours - Services

---

2) `/inbox` — Message Simulator

Purpose: mimic real customer messages.

UI elements: - Dropdown: Channel (web/email/sms) - Text area: incoming message - Button: “Process Message”

Output panel: - Detected intent - Extracted datetime (if any) - Extracted service (if any) - Final response message

Also: - Pre-built “Test Message Buttons” (edge case pack)

---

3) `/logs` — Debug Timeline Dashboard

Purpose: show SE debugging skills.

Features: - List of workflow runs (newest first) - Click run → show detailed log events

Log event format: - timestamp - event type - severity - message - metadata JSON

Example events: - `message_received` - `intent_detected` - `datetime_parsed` - `service_matched` - `validation_failed_outside_hours` - `airtable_lead_created` - `airtable_booking_created`

System Architecture

High-Level Flow

1. Incoming message submitted (Inbox)
2. Create `Message` record
3. Execute workflow engine
4. Store `WorkflowRun`
5. Store `LogEvents`
6. Perform side effects:
7. create/update lead in Airtable
8. create booking in Airtable (if booking flow)
9. Return response to UI

Workflow Engine (Core Logic)

Philosophy

We simulate an “agent runtime” without AI.

The workflow engine: - executes a sequence of steps - each step can read/write a shared context - steps emit log events - steps can stop execution if user input is missing

This mirrors how real agent platforms work (tools + orchestration).

Workflow Context Object

A shared state object passed through steps.

Example:

```
export type WorkflowContext = {
  businessId: string;
  channel: "web" | "email" | "sms";
  rawMessage: string;
```

```
  intent?: "booking" | "cancel" | "reschedule" | "info" | "unknown";
  customerName?: string;
  requestedDateTime?: Date;
  requestedService?: string;

  validationErrors: string[];
  actionsTaken: string[];

  responseMessage?: string;
};
```

Workflow Template (Stored JSON)

Example:

```
{
  "name": "Booking Assistant v1",
  "trigger": "incoming_message",
  "steps": [
    "detect_intent",
    "extract_name",
    "extract_datetime",
    "extract_service",
    "validate_required_fields",
    "validate_opening_hours",
    "create_airtable_lead",
    "create_airtable_booking",
    "build_response"
  ]
}
```

In v1, this can be hardcoded in code, but stored as JSON in DB for realism.

Step-by-Step Workflow Specification

Step 1: detect_intent

Logic: keyword matching.

Rules: - booking: book, appointment, reserve, schedule - cancel: cancel, can’t make it - reschedule: change, move, reschedule - info: open, hours, price, where

Outputs: - `context.intent`

Logs: - `intent_detected`

Step 2: extract_name

Logic: - detect patterns like: - “I’m Marco” - “My name is Marco”

Outputs: - `context.customerName`

Logs: - `name_extracted`

Fallback: - if missing, proceed anyway (CRM lead can be anonymous)

Step 3: extract_datetime

Logic: - use `chrono-node` to parse natural language: - tomorrow at 3pm - monday 10:30 - today at 14

Outputs: - `context.requestedDateTime`

Logs: - `datetime_parsed`

Fallback: - if missing, add validation error

Step 4: extract_service

Logic: - match against business services list. - case insensitive string match.

Outputs: - `context.requestedService`

Logs: - `service_matched`

Fallback: - if missing, add validation error

Step 5: validate_required_fields

Rules: - If intent == booking: - service required - datetime required

If missing: - stop execution and return response: - ask for missing info

Example response:

> “Sure — what service would you like (Haircut / Beard Trim)?”

Logs: - `validation_failed_missing_fields`

Step 6: validate_opening_hours

Rules: - check requestedDateTime day-of-week - verify within open/close time

If outside hours: - stop execution - respond with hours + suggestion

Example:

> “We’re closed at that time. We’re open Mon–Fri 09:00–18:00. Would you like 10:00 instead?”

Logs: - `validation_failed_outside_hours`

Step 7: create_airtable_lead

Rules: - create a Lead record if none exists - or update existing lead (optional in v1)

Data: - Name - Channel - Intent - LastMessage

Logs: - `airtable_lead_created` / `airtable_lead_failed`

Step 8: create_airtable_booking

Rules: - only if intent == booking

Data: - Name - Service - DateTime - Status = confirmed - SourceMessage

Logs: - `airtable_booking_created` / `airtable_booking_failed`

Step 9: build_response

Rules: - construct final user-facing response

Examples: - booking success:

“Perfect — you’re booked for a Haircut tomorrow at 15:00.”

• cancel flow:

“Okay — I can help cancel your booking. What time was it scheduled for?”

Logs: - `response_generated`

Edge Case Pack (Prebuilt Test Inputs)

Missing datetime

"Can I book a haircut?" Expected: asks for date/time.

Missing service

"Tomorrow at 3pm please" Expected: asks for service.

Outside hours

"Tomorrow at 22:00" Expected: suggest alternative.

Angry customer escalation

"This is ridiculous. You never answer." Expected: escalation response.

Cancellation

"Cancel my appointment tomorrow" Expected: ask which booking / mark cancelled.

Escalation Rule (Non-AI but Realistic)

If message contains: - angry - ridiculous - complaint - terrible - never answer

Then: - set `intent = info` - respond with human handoff message - log `handoff_triggered`

Observability & Metrics (Minimal but Strong)

Metrics to compute (optional)

- total runs
- runs with errors
- booking success rate
- missing-field rate

Even a simple summary panel on `/logs` is enough.

Database Schema (Recommended: Prisma)

Tables: - Business - OpeningHours - Service - Message - WorkflowRun - LogEvent

Airtable is the external “CRM”.

Implementation Plan (1-Day Timeline)

Hour 1: Setup

- Next.js + TS
- Prisma + SQLite
- basic page routing

Hour 2: Admin Config

- Business + Services + Hours

Hour 3: Inbox Simulator UI

- message submit
- show response

Hour 4–5: Workflow Engine

- step executor
- intent detection
- datetime parsing
- validation

Hour 6: Airtable Integration

- create lead
- create booking

Hour 7: Logs + DB

- store workflow run
- store events
- logs UI

Hour 8: Polish

- edge case buttons
- clean UI
- deploy to Vercel

Definition of Done

Must Have

- Admin setup works
- Inbox simulation works
- Workflow engine processes booking request end-to-end
- Airtable receives booking + lead
- Logs show step execution

Nice to Have

- cancellation flow
- conflict detection
- metrics summary

Interview Value Mapping

"Own end-to-end implementations"

→ Admin onboarding + deployed workflow

"Configure conversation flows"

→ Workflow template definition + deterministic steps

"Debug real-world edge cases"

→ logs + validation + edge case pack

"Bridge customers and engineering"

→ config UI mimics discovery → implementation translation

"Build internal tooling"

→ logs dashboard + reusable workflow template

Final Notes (Positioning)

This project is not trying to be a chatbot.

It is a **workflow execution + integration + debugging platform**, which is exactly what a founding solutions engineer builds and maintains.

AI can always be layered later, but this system proves the core competency: **shipping reliable customer automation.**