MERCOR

Airtable Multi-Table Form + JSON Automation

Goal: Design an Airtable-based data model and automation system that

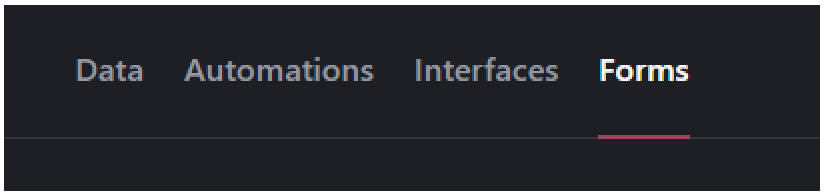
1. Collects contractor-application data through a structured, multi-table form flow
2. Local Python script that compresses the collected data into a single JSON object for storage and routing
3. Local Python script that decompresses the JSON back into the original, normalized tables when edits are needed
4. Auto-shortlists promising candidates based on defined, multi-factor rules
5. Uses an LLM endpoint to evaluate, enrich, and sanity-check each application

1. Airtable Schema Setup

Login to airtable/create account. Start by building from scratch. Understand how headers work with the ability to name then and assign data types. Walk through creating the Applicants table yourself. Once you understand that, copy and paste the remaining four tables with details into Omni, which is Airtable's AI, at once and it will create them for you. Confirm that the tables are structured correctly with the right names for key fields. Our applicants table is the base, with three linked tables (Personal details, work experience, and helper preferences), and the helper table (shortlisted leads). All child tables are linked back to **Applicants** by `Applicant ID`.

Table	Key Fields	Notes
Applicants (parent)	`Applicant ID` (primary), `Compressed JSON`, `Shortlist Status`, `LLM Summary`, `LLM Score`, `LLM Follow-Ups`	Stores one row per applicant and holds the compressed JSON + LLM outputs
Personal Details	`Full Name`, `Email`, `Location`, `LinkedIn`, *(linked to Applicant ID)*	One-to-one with the parent
Work Experience	`Company`, `Title`, `Start`, `End`, `Technologies`, *(linked to Applicant ID)*	One-to-many
Salary Preferences	`Preferred Rate`, `Minimum Rate`, `Currency`, `Availability (hrs/wk)`, *(linked to Applicant ID)*	One-to-one
Shortlisted Leads	`Applicant` (link to Applicants), `Compressed JSON`, `Score Reason`, `Created At`	Auto-populated when rules are met

2. User Input Flow



Airtable's native forms can't write to multiple tables simultaneously, so simulate the flow with **three forms** (one per child table) that each pre-fill or ask for the `Applicant ID`. Require applicants to submit all three forms.When you click on "New Form" you will be prompted to choose which table to create the form from. Do this for each of the three child forms, then click into them and press the "publish" button in the upper right corner for all forms.

Steps 3-4 can be done in a local Python file outside of Airtable. When you run the scripts you can just reflect the updates in Airtable using the API.

3. JSON Compression Automation

1. **Action:** Write a Python local script that gathers data from the three linked tables, builds a single JSON object, and writes it to `Compressed JSON`.
2. **Reading:** https://airtable.com/developers/web/api/introduction
3. **Setup**: Fill out the secrets.py script with your API keys and base key.

For this section, we will need your base ID and AirTable API key. You can view your access token under your profile section -> builder Hub -> personal access token. You can name it whatever you want, and under scope, select data.records:write. Under Access, select the name of your base (Applicant tracking or whatever you named it at the start of the project). Copy and store your access token in a secure location.

JSON
```json
{
  "personal": { "name": "Jane Doe", "location": "NYC" },
  "experience": [
    { "company": "Google", "title": "SWE" },
    { "company": "Meta",  "title": "Engineer" }
  ],
  "salary": { "rate": 100, "currency": "USD", "availability": 25
}
}
```

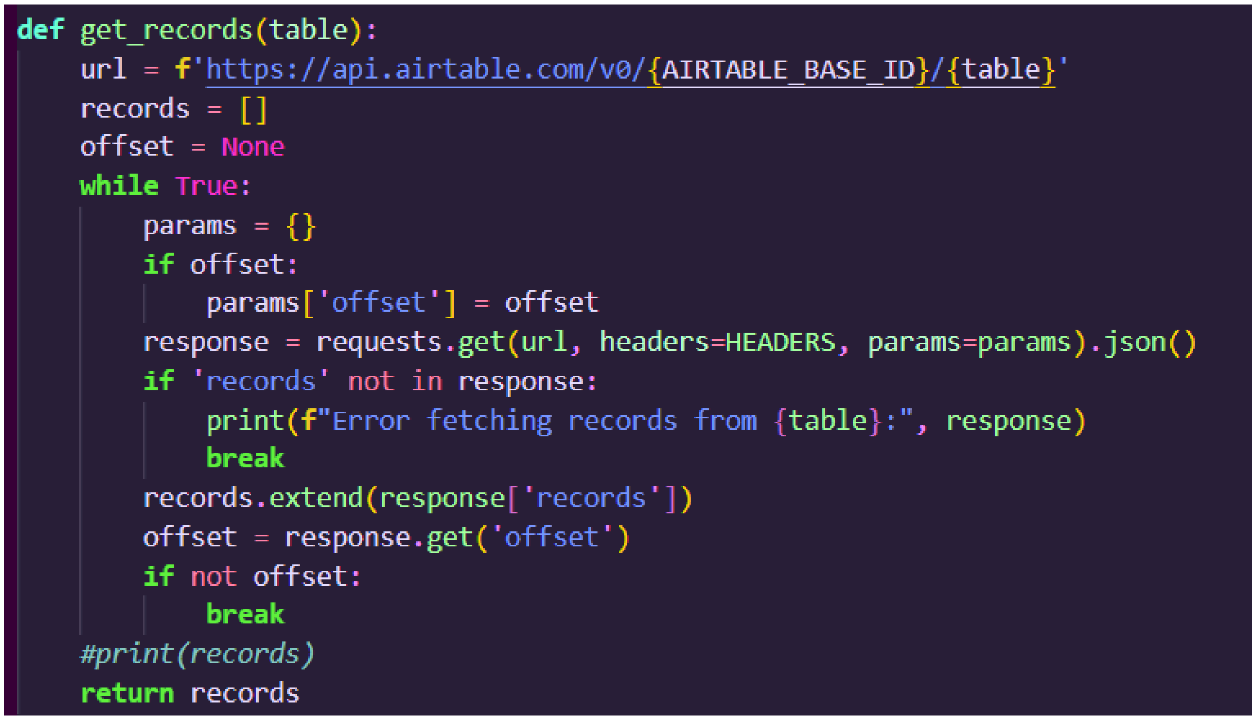
This script shows us the end compression goal. We accomplish this by first using a helper function that pulls every row from a table.

`personal_map` becomes a dictionary like
`{'rec123': {'name': 'Jane', 'location': 'NYC'}}`

`experience_map` becomes
`{'rec123': [{'company': 'Google', 'title': 'SWE'}, {...}]}`

`salary_map` becomes
`{'rec123': {'rate': 100, 'currency': 'USD', 'availability': 25}}`

These are all linked together by ApplicantID so our function build_json_for_applicant() updates the maps into a single compressed JSON entry. We finally update Airtable with our update_applicant_record() function.

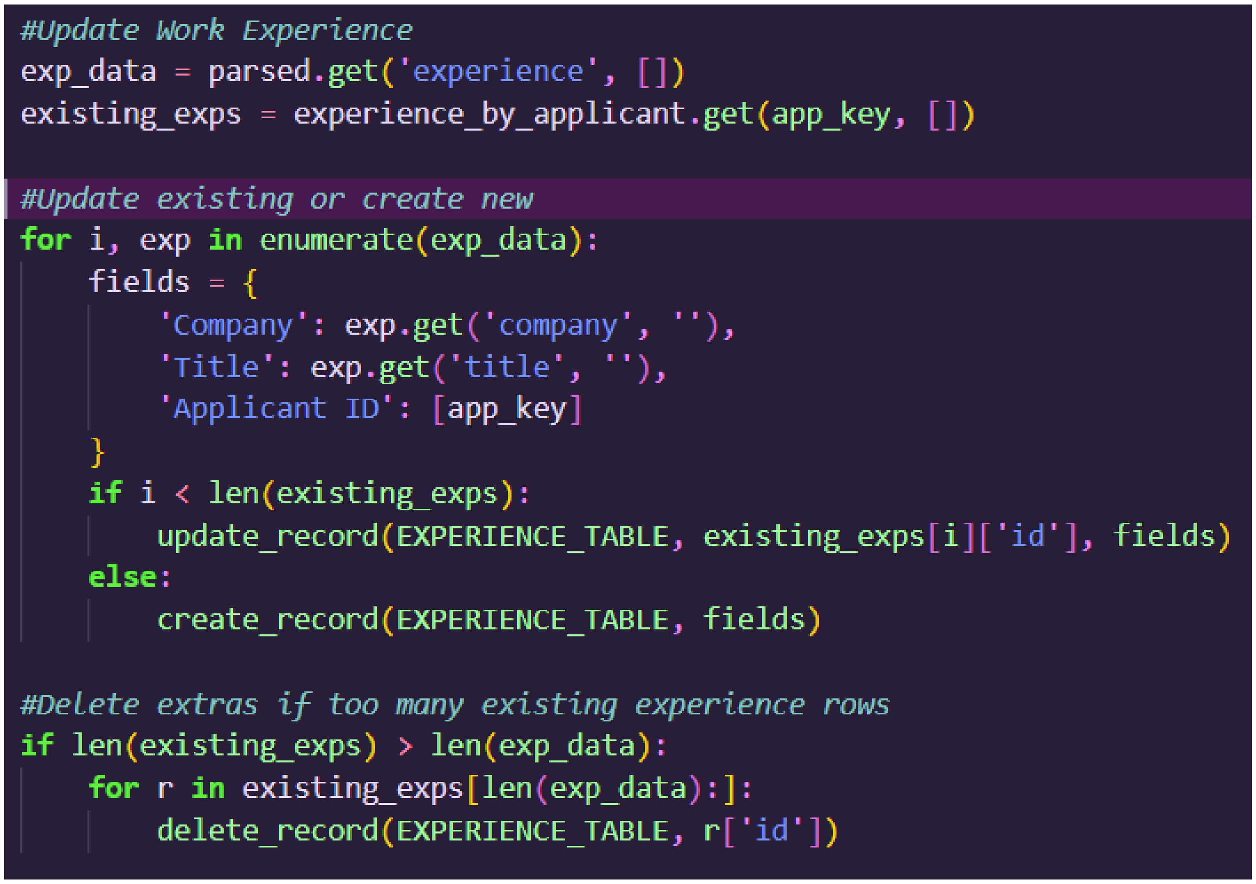


```python
def get_records(table):
    url = f'https://api.airtable.com/v0/{AIRTABLE_BASE_ID}/{table}'
    records = []
    offset = None
    while True:
        params = {}
        if offset:
            params['offset'] = offset
        response = requests.get(url, headers=HEADERS, params=params).json()
        if 'records' not in response:
            print(f"Error fetching records from {table}:", response)
            break
        records.extend(response['records'])
        offset = response.get('offset')
        if not offset:
            break
    #print(records)
    return records
```

4. JSON Decompression Automation

Write a separate Python local script that can:

1. Read `Compressed JSON`.
2. Upsert child-table records so they exactly reflect the JSON state.
3. Update look-ups/links as needed.



```python
#Update Work Experience
exp_data = parsed.get('experience', [])
existing_exps = experience_by_applicant.get(app_key, [])

#Update existing or create new
for i, exp in enumerate(exp_data):
    fields = {
        'Company': exp.get('company', ''),
        'Title': exp.get('title', ''),
        'Applicant ID': [app_key]
    }
    if i < len(existing_exps):
        update_record(EXPERIENCE_TABLE, existing_exps[i]['id'], fields)
    else:
        create_record(EXPERIENCE_TABLE, fields)

#Delete extras if too many existing experience rows
if len(existing_exps) > len(exp_data):
    for r in existing_exps[len(exp_data):]:
        delete_record(EXPERIENCE_TABLE, r['id'])
```

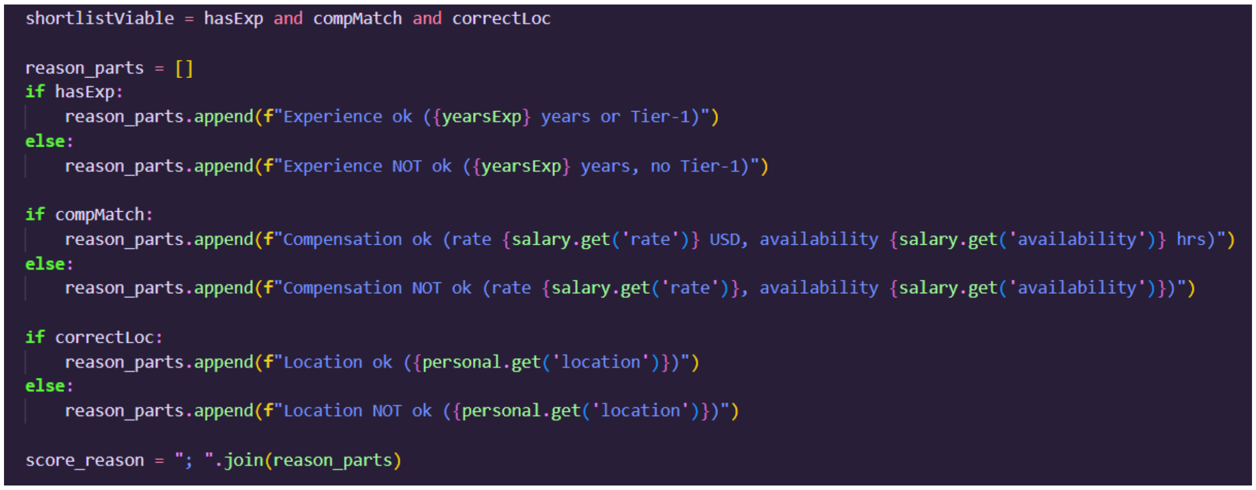
5. Lead Shortlist Automation

After compression, evaluate rules:

Criterion	Rule
Experience	≥ 4 years total **OR** worked at a Tier-1 company (Google, Meta, OpenAI, etc.)

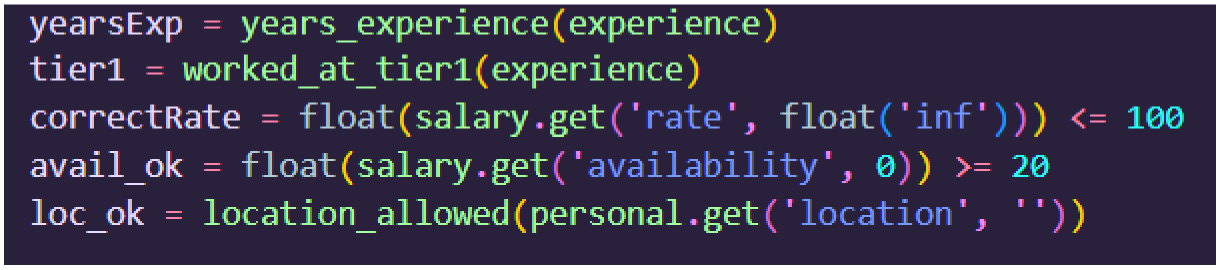
Compensation	`Preferred Rate` ≤ $100 USD/hour **AND** `Availability` ≥ 20 hrs/week
Location	In `US`, `Canada`, `UK`, `Germany`, or `India`

If all criteria are met, create a **Shortlisted Leads** record and copy `Compressed JSON`. Populate `Score Reason` with a human-readable explanation.



```
shortlistViable = hasExp and compMatch and correctLoc

reason_parts = []
if hasExp:
    reason_parts.append(f"Experience ok ({yearsExp} years or Tier-1)")
else:
    reason_parts.append(f"Experience NOT ok ({yearsExp} years, no Tier-1)")

if compMatch:
    reason_parts.append(f"Compensation ok (rate {salary.get('rate')} USD, availability {salary.get('availability')} hrs)")
else:
    reason_parts.append(f"Compensation NOT ok (rate {salary.get('rate')}, availability {salary.get('availability')})")

if correctLoc:
    reason_parts.append(f"Location ok ({personal.get('location')})")
else:
    reason_parts.append(f"Location NOT ok ({personal.get('location')})")

score_reason = "; ".join(reason_parts)
```

This script can easily be extended or changed based on modifying the thresholds for years of experience, the list of tier1 companies, the expected rate, availability, and location. This might look like changing the rate from 100 down to 80, or increasing the required availability from 20 to 40. Additionally, based on the addition of other columns such as having a referral, you could choose to shortlist an applicant.



```
yearsExp = years_experience(experience)
tier1 = worked_at_tier1(experience)
correctRate = float(salary.get('rate', float('inf'))) <= 100
avail_ok = float(salary.get('availability', 0)) >= 20
loc_ok = location_allowed(personal.get('location', ''))
```

6. LLM Evaluation & Enrichment

6.1 Purpose

Exercise a modern LLM (e.g., OpenAI, Anthropic, Gemini) to automate qualitative review and sanity checks. We choose to secure the API key in a “secrets” env variable, and limit the amount of tokens and attempts for security.

6.2 Technical Requirements

Aspect	Requirement
Trigger	After `Compressed JSON` is written **OR** updated
Auth	Read API key from an Airtable **Secret** or env variable (do **not** hard-code)
Prompt	Feed the full JSON and ask the LLM to: • Summarize the applicant in ≤ 75 words • Assign a quality score from 1-10 • Flag any missing / contradictory fields • Suggest up to three follow-up questions
Outputs	Write to `LLM Summary`, `LLM Score`, `LLM Follow-Ups` fields on **Applicants**
Validation	If the API call fails, log the error and retry up to 3× with exponential backoff
Budget Guardrails	Cap tokens per call and skip repeat calls unless input JSON has changed

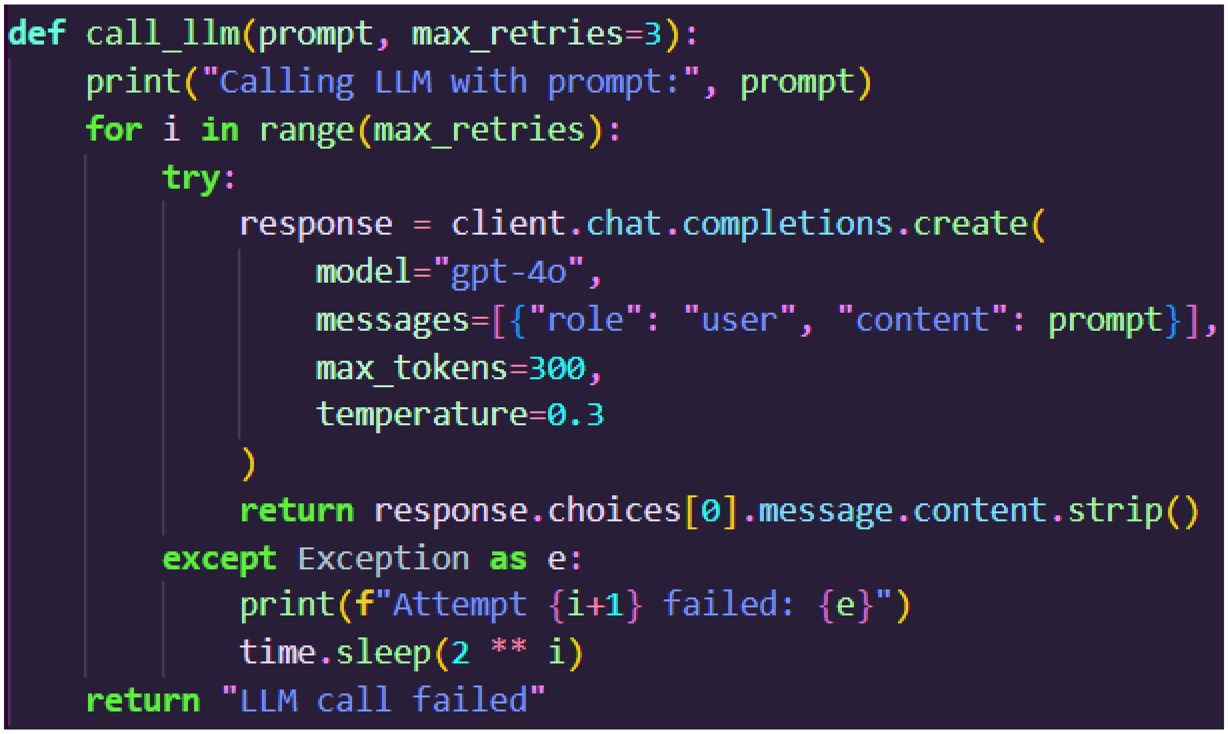
6.3 Sample Prompt (pseudo-code)

```None
You are a recruiting analyst. Given this JSON applicant profile,
do four things:
1. Provide a concise 75-word summary.
2. Rate overall candidate quality from 1-10 (higher is better).
3. List any data gaps or inconsistencies you notice.
4. Suggest up to three follow-up questions to clarify gaps.

Return exactly:
Summary: <text>
Score: <integer>
Issues: <comma-separated list or 'None'>
```

```
Follow-Ups: <bullet list>
```

6.4 Expected Results

Field	Example Value
`LLM Summary`	*“Full-stack SWE with 5 yrs experience at Google and Meta…”*
`LLM Score`	`8`
`LLM Follow-Ups`	• “Can you confirm availability after next month?”• “Have you led any production ML launches?”



```python
def call_llm(prompt, max_retries=3):
    print("Calling LLM with prompt:", prompt)
    for i in range(max_retries):
        try:
            response = client.chat.completions.create(
                model="gpt-4o",
                messages=[{"role": "user", "content": prompt}],
                max_tokens=300,
                temperature=0.3
            )
            return response.choices[0].message.content.strip()
        except Exception as e:
            print(f"Attempt {i+1} failed: {e}")
            time.sleep(2 ** i)
    return "LLM call failed"
```

◦