Task: Standardize and Extend JSON Dataset to Support Alternative Names

Objective

Modify the JSON dataset to support alternative names for jobs (e.g., “SWE” as an alternative for “Software Engineer”). The goal is to:

1. **Add a new field** (`alternative_names`) to jobs that have multiple names.
2. **Ensure backward compatibility** for jobs that don’t have alternative names.
3. **Clarify the scope**: Decide whether to apply this change to the entire dataset or only to entries with known alternative names.

Requirements

1. JSON Structure Update

Update the JSON structure to include an optional `alternative_names` field for jobs. Example:

```
{
  "jobs": [
    {
      "name": "Software Engineer",
      "alternative_names": ["SWE", "Software Dev", "Programmer"]
    },
    {
      "name": "Data Scientist"
      // No alternative_names field if none exist
    }
  ]
}
```

2. Scope of Changes

- **Option 1: Apply to All Jobs** Add an `alternative_names` field to **every job**, even if it’s an empty array (`[]`). This ensures consistency across the dataset.

```
{
  "name": "Example Job",
  "alternative_names": []
}
```

- **Option 2: Apply Only to Jobs with Alternative Names** Only add the `alternative_names` field to jobs that have known alternatives. This keeps the dataset lean but may require additional logic to handle missing fields.

Recommendation: Use **Option 1** for consistency and easier programmatic handling.

3. Implementation Steps

1. **Identify Alternative Names**
   - Manually or programmatically identify jobs with alternative names (e.g., “SWE” for “Software Engineer”).
   - Use industry standards, job postings, or team input to populate `alternative_names`.
2. **Update the JSON Dataset**
   - For each job, add the `alternative_names` field.
   - Example:

     ```
     {
       "name": "Software Engineer",
       "alternative_names": ["SWE", "Software Dev", "Programmer"]
     }
     ```

3. **Validation**
   - Ensure no duplicate `name` or `alternative_names` exist within the same domain.
   - Validate that the JSON remains syntactically correct.

4. Example Output

```
{
  "domains": [
    {
      "name": "Software Engineering",
      "jobs": [
        {
          "name": "Software Engineer",
          "alternative_names": ["SWE", "Software Dev", "Programmer"]
        },
        {
          "name": "Backend Developer",
          "alternative_names": ["Backend Engineer"]
        },
        {
          "name": "Frontend Developer",
          "alternative_names": []
        }
      ]
    }
  ]
}
```

5. Usage of the Updated Dataset

- **Search Functionality**: When searching for a job, check both `name` and `alternative_names` fields. Example:

```
def find_job(job_name, domain_data):
    for job in domain_data["jobs"]:
        if job_name == job["name"] or job_name in job.get("alternative_names", []):
            return job
    return None
```

- **Backward Compatibility**: Ensure existing code that only uses the `name` field continues to work.

6. Open Questions

- Should `alternative_names` be case-sensitive? (Recommendation: Convert all to lowercase for case-insensitive matching.)
- Should abbreviations (e.g., “SWE”) be expanded in the `name` field, or kept as-is in `alternative_names`?
- Should the dataset include regional or company-specific alternative names?