**Laravel Backend Developer Practical Assessment**

**Project Overview: Job Board with Advanced Filtering**

Create a Laravel application that manages job listings with complex filtering capabilities similar to [Airtable](https://airtable.com/). The application should handle different job types with varying attributes using Entity-Attribute-Value (EAV) design patterns alongside traditional relational database models.

**Requirements**

1. Core Job Model and Standard Relationships

* Create a Job model with the following standard fields:
  + id (primary key)
  + title (string)
  + description (text)
  + company\_name (string)
  + salary\_min (decimal)
  + salary\_max (decimal)
  + is\_remote (boolean)
  + job\_type (enum: 'full-time', 'part-time', 'contract', 'freelance')
  + status (enum: 'draft', 'published', 'archived')
  + published\_at (timestamp)
  + Standard timestamps (created\_at, updated\_at)

**2. Many-to-Many Relationships**

Implement the following many-to-many relationships with the Job model:

* **Languages**: Programming languages required for the job
  + Create a Language model with id and name fields
  + Implement appropriate pivot table and relationship methods
* **Locations**: Possible locations for the job
  + Create a Location model with id, city, state, country fields
  + Implement appropriate pivot table and relationship methods
* **Categories**: Job categories/departments
  + Create a Category model with id and name fields
  + Implement appropriate pivot table and relationship methods

**3. Entity-Attribute-Value (EAV) Implementation**

Implement an EAV system to allow for dynamic attributes based on job types:

* Create the following tables:
  + attributes (id, name, type, options)
  + job\_attribute\_values (id, job\_id, attribute\_id, value)
* The type field in attributes should support at least:
  + 'text' (free text input)
  + 'number' (numeric values)
  + 'boolean' (true/false)
  + 'date' (date values)
  + 'select' (selection from predefined options)
* For 'select' type, store possible options as JSON in the options field
* Create appropriate models and relationships to manage this EAV structure

**4. Advanced Filtering API**

Create a RESTful API endpoint that allows for complex filtering of jobs:

GET /api/jobs

The API should accept query parameters for filtering with the following capabilities:

1. **Basic Filtering by Field Type**:
   * Text/String fields (title, description, company\_name, etc.)
     + Equality: =, !=
     + Contains: LIKE
   * Numeric fields (salary\_min, salary\_max, etc.)
     + Equality: =, !=
     + Comparison: >, <, >=, <=
   * Boolean fields (is\_remote, etc.)
     + Equality: =, !=
   * Enum fields (job\_type, status, etc.)
     + Equality: =, !=
     + Multiple values: IN
   * Date fields (published\_at, created\_at, etc.)
     + Equality: =, !=
     + Comparison: >, <, >=, <=
2. **Relationship Filtering**:
   * Filter by languages (e.g., jobs requiring PHP AND JavaScript)
   * Filter by locations (e.g., jobs in New York OR San Francisco)
   * Filter by categories
   * Operations supported:
     + Equality: = (exact match)
     + Has any of: HAS\_ANY (job has any of the specified values)
     + Is any of: IS\_ANY (relationship matches any of the values)
     + Existence: EXISTS (relationship exists)
3. **EAV Filtering by Attribute Type**:
   * Text attributes
     + Equality: =, !=
     + Contains: LIKE
   * Number attributes
     + Equality: =, !=
     + Comparison: >, <, >=, <=
   * Boolean attributes
     + Equality: =, !=
   * Select attributes
     + Equality: =, !=
     + Multiple values: IN
4. **Logical Operators**:
   * Support for AND/OR logical operators.
   * Support for grouping conditions.
5. **Query Parameter Format**:
   * Design a clean, expressive query parameter format that supports all these operations
   * Document this format clearly in your README

Example of a complex filter:

/api/jobs?filter=(job\_type=full-time AND (languages HAS\_ANY (PHP,JavaScript))) AND (locations IS\_ANY (New York,Remote)) AND attribute:years\_experience>=3

**5. Filter Builder**

Create a service class that parses the filter parameters and builds the appropriate Eloquent query:

* Implement a JobFilterService that parses the filter parameters
* Use query scopes, where clauses, and joins appropriately
* Support filtering by EAV attributes
* Handle AND/OR logic and grouping

**7. Documentation**

* Document the API endpoints
* Explain how the filtering syntax works
* Include examples of complex queries

**Evaluation Criteria**

Your solution will be evaluated based on:

1. **Code Quality**:
   * Clean, maintainable code
   * Proper use of Laravel conventions
   * Appropriate use of design patterns
2. **Database Design**:
   * Efficient schema design
   * Appropriate use of migrations
   * Indexing strategy for optimized filtering
3. **Query Efficiency**:
   * Efficient query building
   * Minimizing N+1 problems
   * Handling large datasets
4. **Filter Implementation**:
   * Completeness of filter capabilities
   * Handling edge cases and errors
   * Extensibility of the filter system
5. **Documentation**:
   * Clear API documentation
   * Well-documented code

**Submission Requirements**

1. Create a GitHub repository with your solution
2. Include migrations, seeders, and sample data
3. Provide a README with setup instructions and API documentation
4. Include a Postman collection or similar for testing the API
5. Add notes on any assumptions, design decisions, or trade-offs you made