Take-Home Challenge — Frontend

Goal

Build a **reusable, dataset-agnostic Filter Builder UI library** that allows users to construct **arbitrary nested conditions** (and / or groups) and serialize them into a JSON structure.
The library must be **schema-driven** (fields, types, operators provided via config) and support **sending filters to a server** via:

- **GET** — as a query string parameter.
- **POST** — as a JSON body.

Target JSON Format

```
{
  "and": [
    { "field": "age", "operator": "gt", "value": 30 },
    {
      "or": [
        { "field": "role", "operator": "eq", "value": "admin" },
        { "field": "isActive", "operator": "=", "value": true }
      ]
    }
  ]
}
```

Requirements

1. Configurable Schema

- Accept a **map of supported operators per type**:

```
{
  string: ['eq', 'neq', 'contains', 'starts_with', 'ends_with'],
  number: ['eq', 'neq', 'gt', 'lt', 'between'],
  boolean: ['eq', 'neq'],
  date: ['eq', 'neq', 'before', 'after', 'between']
}
```

---

2. Core Features

- Add/Edit/Remove conditions.
- Add/Edit/Remove nested and / or groups (unlimited depth).
- Type-aware value inputs (text, number, date picker, select, toggle).
- Validation rules based on operator:
    - between → exactly two values
    - in → array of values
    - is null / is not null → no value

---

3. Serialization & API Integration

- Convert UI state to JSON in the target format.
- Load JSON back into the UI for editing.
- **GET mode**: URL-safe query string
- **POST mode**: Send JSON in the request body.
- Allow the consumer to choose GET or POST via config.

---

4. Library Design

- Export as a reusable component (React) typescript library.
- Accept schema, operators, initial filter JSON, and API config as props/options.
- Emit filter JSON and/or query string via callback/event.
- No hard-coded dataset logic — must work with any schema.

---

5. UX & Accessibility

- Recursive UI for nested groups.
- Responsive Layout.
- Accessible (keyboard navigation, ARIA labels).

---

6. Testing

- Unit tests for:
    - Serialization/deserialization
    - Validation rules

    - GET/POST encoding
- Integration tests for:
    - Adding/removing/editing conditions and groups
    - API request generation

---

7. Deliverables

- Library source code.
- Example app showing usage with at least two different datasets (e.g., users, products).
- README with:
    - Installation & usage instructions
    - Configuration API
    - Architecture decisions