Interview Task

Preface

We work extensively with **dynamic forms**. Each internal organization that uses our application can create its own **custom forms** to store different types of information. For example, one could imagine modules for creating **Products**, **Events**, or **Projects**.

> ***Note:*** *This is not exactly what our application does, but the concepts could easily be represented as a similar system for managing custom forms bound to specific entities.*

However, we don't allow users to create entirely arbitrary forms. For instance, in the **Event** module, we provide a **template form** available to all users, which includes fields such as **Event Name**, **Venue Location**, **Country**, and **Organizer**.

If a suborganization needs to capture additional details beyond the predefined fields in the template form, a user with the appropriate permissions can **add new attributes** for their organization or **remove unnecessary ones**. These customized fields will then appear only within their organization's forms whenever a new **Event** is created.

To help visualize this concept, you can search Google for **"Form Builder"**. You'll find many examples of UI/UX designs where users can compose their own custom forms. In our case, these forms are always **bound to a specific entity**.

For example, if users want to review all the **Event** records they've created, they should be able to open an overview page with a **paginated table** displaying all records - each populated with its corresponding dynamic form values (such as *Event Name*, *Venue Location*, etc.).

Task

Your task is to implement a **minimal reproduction** of our system using **NestJS** as the backend framework, **Angular** as the client framework, and **Postgre SQL** as the database. You may use any query builder or ORM of your choice.

You should use **TypeScript**.

You do **not** need to implement multi-tenancy or any authentication/authorization layer.

You may use SAP's Angular Components to quickly scaffold the UI. Don't focus too much on visual design - the main goal is to demonstrate your approach to implementing the **logic for rendering dynamic forms** and your understanding of **HTML/CSS layouts**.

The goal is to design a way to **store dynamic form definitions in the database** and **bind field values** to the specific records they belong to.

For example, imagine we've defined a field called **Event Sponsor** in our **Event** form. When a new **Event** record is stored, making a request to the backend at `GET /api/events/[id]`
should return all field values associated with that record, including the **Event Sponsor** value.

Example response (You don't need to follow this structure exactly - use whichever approach best fits your architecture):

```
{
  "id": 42,
  "eventName": "Global Tech Conference",
  "venueLocation": "San Francisco",
  "country": "USA",
  "governmentSecurityRating": "High",
  "eventSponsor": "TechWorld Inc."
}
```

Your application should include:

- **A page for creating a form**
- **A page for filling in the form** - forms should be single-page, not multi-step
- **A page for displaying the submitted data in a table** - filtering is not required; pagination is optional

The SAP component library provides suitable UI components for all of these, which you are free to use.

As mentioned earlier, the full application must support multiple forms bound to multiple entities. However, for this task, a **single form type bound to a single entity** is sufficient. You can use **Event** as the example entity whose records you'll manage - or choose any other entity of your preference. Nevertheless, the design of the application should allow easy extension to more form types.

Goal

This is a **take-home task**, and we understand candidates have jobs and families, so we're not expecting a production-grade, multi-month solution. The goal is to demonstrate a **practical design** that meets the requirements - not an elaborate system that would take weeks to build.

We're not optimizing for high performance here; small latency differences are acceptable. What matters most is **data correctness** - losing information is far worse than a slightly slower operation.

After you submit the task, we'll ask you to demo it and walk us through your code. Be prepared to explain major decisions such as **project structure**, how you **represented and stored dynamic form definitions and values in PostgreSQL**, and any **notable tradeoffs** you made.