🧩 Mini Guía Técnica

Rol: Frontend – Lógica de campos

🎯 Objetivo del rol

Implementar y mantener la **lógica interna de los campos del formulario**, asegurando que cada tipo de campo:

- tenga propiedades coherentes según su tipo,
- pueda ser editado correctamente,
- se represente correctamente en el JSON final del pedido.

Este rol **NO** se encarga de diseño, backend ni arquitectura.

📌 Contexto del proyecto

El proyecto es un **constructor de órdenes** con drag & drop.

Actualmente:

- Los campos se arrastran desde una paleta al canvas.
- Cada campo tiene una estructura base:

```
{
  id: string,
  type: string,
  props: object
}
```

El drag & drop **ya funciona** y **no debe modificarse**.

🧠 Responsabilidades técnicas

1️⃣ Definir las props por tipo de campo

Cada tipo de campo debe tener **propiedades específicas**, no genéricas.

Text

```
props: {
  label: string,
  required: boolean,
  placeholder: string
}
```

Number

```
props: {
  label: string,
  required: boolean,
  min: number | null,
  max: number | null
}
```

Date

```
props: {
  label: string,
  required: boolean
}
```

Select (prioridad alta)

```
props: {
  label: string,
  required: boolean,
  options: [
    {
      value: string,
      label: string
    }
  ]
}
```

2️⃣ Implementar lógica real del campo select

Actualmente el campo select:

- no tiene opciones reales,
- se comporta como texto.

Debe:

- manejar un arreglo options
- permitir agregar nuevas opciones
- permitir eliminar opciones
- evitar que el campo quede sin opciones válidas

Reglas mínimas:

- options siempre debe existir
- debe existir al menos una opción
- cada opción debe tener value y label

3️⃣ Definir defaultProps por tipo de campo

Cuando un campo se arrastra al canvas, debe crearse con propiedades iniciales coherentes.

Ejemplo:

```
// Select
defaultProps: {
  label: 'Selector',
  required: false,
  options: [
    { value: 'option1', label: 'Opción 1' }
  ]
}
// Text
defaultProps: {
  label: 'Texto',
  required: false,
  placeholder: ''
}
```

⚠ No modificar el drag & drop, solo las propiedades iniciales.

4️⃣ Mantener los campos serializables a JSON

Todos los campos deben poder convertirse directamente en JSON **sin lógica de React**.

Ejemplo válido:

```
{
  "id": "uuid",
  "type": "select",
  "props": {
```

```
    "label": "Método de pago",
    "required": true,
    "options": [
      { "value": "cash", "label": "Efectivo" },
      { "value": "card", "label": "Tarjeta" }
    ]
  }
}
```

No deben existir:

- funciones
- estados
- referencias a componentes

5️⃣ Archivos que puede modificar

✔ Lógica de creación de campos
✔ Estructura de props
✔ Valores por defecto
✔ Lógica de actualización de propiedades

6️⃣ Archivos que NO debe modificar

✖ Drag & Drop (DndContext, SortableContext)
✖ Arquitectura general del proyecto
✖ Contrato JSON global
✖ Estilos / UI
✖ Backend

✅ Checklist de validación (autoevaluación)

Antes de entregar, debe cumplirse:

- Cada tipo de campo tiene props propias
- El campo select maneja opciones reales
- Ningún campo queda con props vacías
- Los campos se convierten directamente a JSON
- No se rompió el drag & drop
- No se agregó lógica de UI

🏗️ Frase guía para IA (copiar y pegar)

Estoy trabajando en la lógica de campos de un form builder en React. No debo tocar drag & drop ni UI. Mi objetivo es definir correctamente las props por tipo de campo (especialmente select con options), defaultProps y asegurar que los campos sean serializables a JSON.

🧠 Regla de oro

Si el backend puede leer el JSON sin conocer React, el trabajo está bien hecho.