✒️ Mini Guía Técnica

Rol: Backend – API

🎯 Objetivo del rol

Diseñar e implementar la **API REST** que:

- reciba órdenes creadas desde el frontend,
- persista su información,
- devuelva órdenes almacenadas,
- valide el contrato JSON.

El backend **NO** construye formularios, **solo los recibe y gestiona**.

📌 Contexto del proyecto

El frontend construye una **orden** usando drag & drop y genera un JSON con esta estructura general:

```
{
  "order": {
    "id": "uuid",
    "title": "Orden de Pedido",
    "createdAt": "ISO_DATE",
    "fields": [
      {
        "id": "uuid",
        "type": "text | number | date | select",
        "order": 1,
        "props": { }
      }
    ]
  }
}
```

El backend **no debe inferir ni modificar campos**.

□ Responsabilidades técnicas

1️⃣ Implementar endpoints REST

Endpoints mínimos requeridos:

Crear orden

POST /api/orders

Recibe:

```
{
  "title": "string",
  "fields": [ ... ]
}
```

Devuelve:

```
{
  "id": "uuid",
  "createdAt": "ISO_DATE"
}
```

Obtener todas las órdenes

GET /api/orders

Obtener una orden por ID

GET /api/orders/:id

(Opcional futuro)

PUT /api/orders/:id

DELETE /api/orders/:id

2️⃣ Validar contrato JSON (obligatorio)

El backend debe validar:

- fields es un array
- cada campo tiene:
    - id
    - type
    - order

    - props
- type es uno de:
    - text
    - number
    - date
    - select

⚠ El backend **no debe intentar corregir** datos inválidos.
Si el JSON es inválido → **error 400**.

3️⃣ Persistencia transparente

El backend:

- guarda el JSON **tal cual**
- no reordena campos
- no transforma props
- no agrega lógica extra

Ejemplo válido:

```
props: {
  "label": "Método de pago",
  "required": true,
  "options": [...]
}
```

4️⃣ Separación de capas

La API debe respetar:

- Routes
- Controllers
- Services
- Models

Ejemplo:

/routes/orders.routes.js

/controllers/orders.controller.js

/services/orders.service.js

/models/order.model.js

⚠ No lógica de negocio en routes.

5️⃣ Manejo de errores claro

Ejemplos:

Caso	Código
JSON inválido	400
Orden no existe	404
Error interno	500

🗂️ Archivos que puede modificar

✔ Routes
✔ Controllers
✔ Services
✔ Validaciones
✔ Configuración de Express

🚫 Archivos que NO debe modificar

✖ Frontend
✖ Contrato JSON
✖ Estructura de campos
✖ Lógica de drag & drop
✖ UI

🧪 Checklist de validación

- API acepta órdenes válidas
- Rechaza órdenes inválidas
- No modifica el JSON recibido
- Devuelve respuestas claras
- Separación de capas correcta
- No depende del frontend

🧠🏭 Frase guía para IA (copiar y pegar)

Estoy implementando una API REST en Node + Express para recibir y almacenar órdenes construidas en un frontend. No debo modificar ni inferir campos. Mi trabajo es validar el contrato JSON, persistirlo y exponer endpoints REST claros.

Regla de oro

Si mañana el frontend cambia de framework y la API sigue funcionando, el backend está bien diseñado.