PRÁCTICO: Desarrollo con Node.js + Express + MongoDB

Tema: Gestión de países hispanohablantes (datos desde API externa)
Duración sugerida: 1–2 semanas
Dificultad: Intermedia (con extras avanzados)

---

1) Objetivo

Construir una aplicación web con **Node.js**, **Express** y **MongoDB** que:

- Consuma datos desde una API externa.
- Procese y filtre los países **que tengan español** como idioma.
- Guarde los datos procesados en **MongoDB** (estructura limpia y consistente).
- Muestre un **dashboard web** con tabla, totales y operaciones **CRUD** (agregar, editar, eliminar).
- Las vistas deben tener como minimo un Layout, landing, navbar y footer.
- Incluya **validaciones** robustas en backend para el envio de formularios.

---

2) Fuente de datos (API)

- **Endpoint:** `https://restcountries.com/v3.1/region/america`
- **Tarea:**
  - Obtener todos los países de América.
  - **Filtrar**: conservar **solo** países que incluyan el idioma español (clave `"spa"` en `languages`).
  - **Limpiar** cada país **eliminando** estas propiedades:
    - `translations`, `tld`, `cca2`, `ccn3`, `cca3`, `cioc`, `idd`, `altSpellings`, `car`, `coatOfArms`, `postalCode`, `demonyms`, `etc`
  - **Agregar** propiedad nueva:
    - `"creador": "TU_NOMBRE_REAL"`

---

3) Almacenamiento en MongoDB

- **Base de datos:** MongoDB (local o Atlas).
- **Requisito:** Persistir **solo** las propiedades necesarias tras el filtrado/limpieza + `creador`
- **Ejemplo de schema:** Se habilita la ultima semana a pedido

---

4) Interfaz Web (Views)

- **Tecnologías:** Node.js, Express y **EJS** (o el motor de plantillas que prefieras).
- **Layout base:**
  - **Navbar** (enlaces: Dashboard, Agregar País, Acerca de).
  - **Footer** (autor, repo del proyecto).
- **Dashboard (tabla):** mostrar por registro como minimo:
  - `nombre o nombre oficial` (o fallback a `name.official` si no existe en español)
  - `capital`
  - `borders`
  - `area`
  - `population`
  - **(Avanzado)** `gini`
  - `timezones`
  - `creador`
- **Fila de totales (al pie de la tabla AVANZADO):**
  - **Suma** de `population`
  - **Suma** de `area`
  - **Promedio** de `gini` (solo de los que tengan valor)

Funciones adicionales (CRUD):

- **Agregar País:** formulario con validaciones.
- **Editar País:** formulario con validaciones.

- **Eliminar País:** confirmación y borrado.

---

5) Validaciones (obligatorias)

Se aplican en:

- Esquema Mongoose (nivel de modelo).
- **express-validator** o middleware propio (nivel de rutas).
- Mensajes claros en la vista (nivel de UI).

Reglas:

- `name.official` : **3–90** caracteres.
- `capital` : cada elemento **3–90** caracteres.
- `borders` : cada código **3 letras mayúsculas**
- `area` : **número positivo**.
- `population` : **entero positivo**.
- **(Avanzado)** `gini` : **0–100**.

Manejo de errores:

- No permitir guardar datos inválidos.
- Mostrar mensajes legibles y amistosos (resaltar campos con error y conservar lo ya escrito).

---

6) Entregables

1. **Código fuente**
    - Proyecto completo, limpio y ejecutable. En github.
    - **Instrucciones** para instalar y correr.
2. **Documentación (README)**
    - Objetivos.
    - Tecnologías usadas.
    - Pasos de ejecución.

- Consideraciones especiales (variables de entorno, etc.).

3. **Demostración**
   - Explicación de validaciones y comportamiento ante datos inválidos.
   - Conocimiento de como se mueve la informacion adentro de tu servidor.
   - Que hacen las funciones y comandos escritos en tu codigo.

---

7) Criterios de evaluación

- **Funcionalidad:** cumplimiento de especificaciones y validaciones.
- **Calidad del código:** organización, nombres claros, modularidad, comentarios útiles.
- **Demostracion:** Entender tu codigo, sus funciones y sus casos limites.

---

8) Pasos sugeridos de desarrollo

1. **Configuración inicial**
   - `npm init -y`
   - Instalar dependencias: `express`, `ejs`, `mongoose`, `axios`, `express-validator`, `dotenv`, `method-override` (para PUT/DELETE), `morgan` (log), etc.
2. **Consumo de API**
   - Crear un servicio que: **obtenga → filtre por `languages` → elimine campos → agregue `creador`**.
3. **Modelo de datos**
   - Definir esquema Mongoose **solo con propiedades necesarias** y validaciones.
4. **Persistencia**
   - Guardar en MongoDB. Verificar integridad (tipos y valores).
5. **Backend (rutas + controladores)**
   - Listar, agregar, editar, eliminar. Manejar errores.
6. **Validaciones**
   - Implementar reglas y mensajes. Prevenir guardado inválido.
7. **Frontend (vistas)**

- Layout + dashboard + formularios.
- Agregar fila de totales (population, area, promedio gini).
- Todo lo de nivel Avanzado.

---

9) Consejos y buenas prácticas

- **Normalizá** `gini`: la API suele traer un objeto por año (`{ "2016": 31.9 }`). Elegí el último año disponible o calculá un promedio simple y guardá un número.
- **Fallback de nombres:** si no existe `name.spa.official`, usá `name.official`.
- **Ergonomía de formularios:** mantené los valores ingresados cuando haya errores; mostrálos con claridad.
- **Manejo de null/undefined:** validá arrays vacíos (`capital`, `borders`, `timezones`) antes de iterar.
- **Evitar duplicados:** al hacer seed, considerá un upsert por nombre oficial o por combinación única.

---

10) Desafío avanzado (opcional, suma extra)

Elige **al menos dos**:

1. **Búsqueda y filtrado en el dashboard** (por nombre, capital, rango de población, región).
2. **Paginación** del listado (servidor o cliente).
3. **Cacheo** del consumo de API (evitá llamadas repetidas; invalida cada X horas).
4. **Normalización de Gini** con selector de año y gráfico simple (por ejemplo, barras con `area/population`).
5. **Calculo de promedio Gini** se tiene que tener en cuenta que hay paises que no tienen evaluado el valor.
6. **Exportación a CSV** del listado filtrado actual.