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 mínimo un Layout, landing, navbar y footer.  
• Incluya validaciones robustas en backend para el envío 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 última semana a pedido.

# 4) Interfaz Web (Views)

• Tecnologías: Node.js, Express y EJS (o el motor de plantillas que prefieras).  
• Layout base: Navbar (Dashboard, Agregar País, Acerca de) y Footer (autor, repo del proyecto).  
• Dashboard (tabla): mostrar por registro como mínimo: name o name.official, capital, borders, area, population, gini (avanzado), timezones, creador.  
• Fila de totales (al pie de la tabla avanzado): suma de population, suma de area, promedio de gini.  
• Funciones CRUD: Agregar País, Editar País (ambos 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.  
• gini: 0–100.  
  
Manejo de errores:  
• No permitir guardar datos inválidos.  
• Mostrar mensajes legibles y amistosos.

# 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 cómo se mueve la información dentro del servidor.  
 • Qué hacen las funciones y comandos escritos en el código.

# 7) Criterios de evaluación

• Funcionalidad: cumplimiento de especificaciones y validaciones.  
• Calidad del código: organización, nombres claros, modularidad, comentarios útiles.  
• Demostración: entender tu código, sus funciones y casos límites.

# 8) Pasos sugeridos de desarrollo

1. Configuración inicial: npm init -y, instalar dependencias (express, ejs, mongoose, axios, express-validator, dotenv, method-override, morgan).  
2. Consumo de API: crear 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.  
5. Backend (rutas + controladores): listar, agregar, editar, eliminar, manejar errores.  
6. Validaciones: implementar reglas y mensajes.  
7. Frontend (vistas): layout + dashboard + formularios, agregar fila de totales, todo lo avanzado.

# 9) Consejos y buenas prácticas

• Normalizá gini: elegir último año disponible o promedio.  
• Fallback de nombres: si no existe name.spa.official, usar name.official.  
• Ergonomía de formularios: mantener valores al haber errores.  
• Manejo de null/undefined: validar arrays vacíos.  
• Evitar duplicados: considerar upsert al hacer seed.

# 10) Desafío avanzado (opcional, suma extra)

Elige al menos dos:  
1. Búsqueda y filtrado en el dashboard.  
2. Paginación del listado.  
3. Cacheo del consumo de API.  
4. Normalización de Gini con selector y gráfico.  
5. Cálculo de promedio Gini.  
6. Exportación a CSV.