







Ejercicio: CRUD HTTP Manual con cURL y Herramientas Visuales








Descripción del ejercicio

Implementarás un sistema completo de gestión de estudiantes mediante peticiones HTTP, utilizando una API local con json-server. Documentarás todas las operaciones CRUD (Create, Read, Update, Delete) usando tres herramientas diferentes: cURL (línea de comandos), Thunder Client (interfaz gráfica) y REST Client (archivos .http).

Objetivos de aprendizaje

-  Comprender la anatomía completa de las peticiones HTTP
 -  Dominar los métodos HTTP: GET, POST, PUT, PATCH, DELETE
 -  Practicar con cURL desde la terminal
 -  Utilizar herramientas visuales (Thunder Client y REST Client)
 -  Gestionar variables de entorno con dotenv
 -  Aplicar buenas prácticas de Git y documentación
-

Requisitos previos

-  Node.js instalado (v18 o superior)
 -  Git instalado
 -  Visual Studio Code
 -  Extensiones de VS Code instaladas:
 -  Thunder Client
 -  REST Client
 -  Terminal (Git Bash en Windows, bash/zsh en Linux/Mac)
-

📁 Estructura del proyecto requerida

```

manual-http-[nombre-iniciales-apellidos]/
├── src/
│   ├── db/
│   │   └── db.json          # Base de datos json-server
│   └── crud-curl.js        # Script con funciones CRUD
├── scripts/
│   └── validate.sh         # Script de validación bash
├── images/                 # Capturas Thunder Client
├── peticiones-crud.http    # Peticiones REST Client
├── .env                    # Variables de entorno (NO versionar)
├── .env.example            # Template de variables
├── .gitignore
├── README.md               # Documentación completa
├── checklist.md            # Control de progreso
└── package.json
  
```

🎬 Parte 1: Configuración inicial del proyecto

1.1 🏗️ Inicialización del proyecto

- ☐ Crear carpeta del proyecto: `manual-http-[tu-nombre-iniciales-apellidos]`
- ☐ Inicializar proyecto Node.js con `npm init`
- ☐ Completar los datos del proyecto:
 - 🍌 Nombre: `manual-http-nombre-iniciales-apellidos`
 - 📦 Version: 1.0.0
 - 📝 Description: Descripción apropiada del proyecto
 - 👤 Author: Tu nombre




1.2 📦 Instalación de dependencias

- ☐ Instalar `json-server` como dependencia
- ☐ Instalar `dotenv` como dependencia

💡 Nota sobre los paquetes:

- **json-server:** Crea una API REST completa a partir de un archivo JSON
- **dotenv:** Permite cargar variables de entorno desde un archivo `.env`









1.3 Configuración de package.json

- ☐ Configurar el proyecto para usar **ESM (ES Modules)** en lugar de CommonJS
- ☐ Añadir los siguientes scripts en `package.json`:
 -  `server:up`: Debe levantar json-server en el puerto 4000 vigilando el archivo `db.json`
 -  `crud:curl`: Debe ejecutar el script `src/crud-curl.js`
 -  `validate`: Debe ejecutar el script de validación bash




1.4 Estructura de carpetas

- ☐ Crear todas las carpetas según la estructura requerida

1.5 Archivos de configuración

- ☐ Crear archivo `.env` con las siguientes variables:
 -  `PORT` (valor: 4000)
 -  `API_BASE_URL` (valor: `http://localhost`)
 -  `NODE_ENV` (valor: `development`)
- ☐ Crear archivo `.env.example` (template del anterior para versionarlo en Git)
- ☐ Crear archivo `.gitignore` que debe ignorar:
 -  `node_modules/`
 -  `.env`
 -  Archivos de logs
 -  Archivos del sistema operativo
 -  Carpetas de editores (opcional: `thunder-tests/`)

1.6 Base de datos json-server

- ☐ Crear archivo `src/db/db.json` con la estructura de base de datos proporcionada por el profesor
- ☐ La base de datos contiene:
 -  Colección `students` con 7 estudiantes
 -  Colección `courses` con 4 cursos
 -  Colección `enrollments` con 4 inscripciones





Parte 2: Script CRUD con funciones JavaScript

2.1 Archivo `src/crud-curl.js`







- ☐ Importar y configurar `dotenv` al inicio del archivo
- ☐ Cargar las variables de entorno `PORT` y `API_BASE_URL`
- ☐ Construir la `BASE_URL` completa usando las variables

2.2 Funciones CRUD requeridas

Debes implementar las siguientes funciones. Cada función debe:

-  Tener un comentario JSDoc explicando su propósito
-  **Recibir parámetros cuando sea necesario** (IDs, datos a enviar, etc.)
-  Imprimir por consola el comando cURL correspondiente
-  El comando debe estar correctamente formateado

Funciones a implementar:

- ☐  `createStudent(studentData)` - Recibe objeto con datos del estudiante, imprime comando CREATE
- ☐  `readAllStudents()` - Imprime comando para leer todos los estudiantes (sin parámetros)
- ☐  `readStudentById(id)` - Recibe el ID del estudiante, imprime comando para leerlo
- ☐  `updateStudent(id, studentData)` - Recibe ID y datos completos, imprime comando PUT
- ☐  `patchStudent(id, partialData)` - Recibe ID y datos parciales, imprime comando PATCH
- ☐  `deleteStudent(id)` - Recibe ID del estudiante, imprime comando DELETE

2.3 Ejecución del script

- ☐ Al final del archivo, ejecutar todas las funciones en orden pasando los parámetros apropiados
- ☐ Añadir mensajes informativos al inicio y final
- ☐ El output debe ser claro y estructurado











Parte 3: Documentación CRUD con cURL

3.1 README.md - Sección CRUD





Debes documentar en tu README cada operación CRUD con el siguiente formato:

Para cada operación (CREATE, READ ALL, READ BY ID, UPDATE, PATCH, DELETE):

- ☐  Título descriptivo de la operación

- ☐  Descripción de qué hace
- ☐  Comando cURL completo y funcional
- ☐  Explicación detallada de cada parte del comando:
 -  Qué hace cada flag (-i, -X, -H, -d)
 -  Por qué se usa ese método HTTP
 -  Qué headers se envían y por qué
- ☐  Respuesta HTTP real obtenida (headers + body)
- ☐  Explicación del código de estado HTTP recibido







3.2 Pruebas reales

- ☐  Levantar el servidor json-server
- ☐  Ejecutar cada comando cURL generado por tu script
- ☐  Capturar las respuestas reales
- ☐  Documentar las respuestas en el README





Parte 4: Thunder Client

4.1 Peticiones

Crear las siguientes peticiones en Thunder Client:

- ☐  CREATE Student (POST)
- ☐  GET All Students (GET)
- ☐  GET Student by ID (GET)
- ☐  UPDATE Student (PUT)
- ☐  PATCH Student (PATCH)
- ☐  DELETE Student (DELETE)

4.2 Capturas de pantalla





- ☐ Realizar captura de pantalla de cada petición
- ☐ Cada captura debe mostrar:
 -  Request completo (método, URL, headers, body si aplica)
 -  Response completo (status, headers, body)
- ☐  Guardar capturas en carpeta `images/` con nombres descriptivos
- ☐  Incluir las capturas en el README

4.3 Documentación









- ☐ Explicar en el README cómo usar Thunder Client
- ☐ Incluir las capturas con descripción de cada operación

Parte 5: REST Client


5.1 Archivo peticiones-crud.http

- ☐ Crear archivo en la raíz del proyecto
- ☐  Definir variables al inicio (@baseUrl, @port, @apiUrl)
- ☐  Implementar todas las operaciones CRUD usando la sintaxis de REST Client
- ☐  Separar cada petición con ###
- ☐  Añadir comentarios descriptivos

5.2 Peticiones requeridas

- ☐  CREATE - Crear estudiante
- ☐  READ - Todos los estudiantes
- ☐  READ - Estudiante por ID
- ☐  READ - Filtrar estudiantes activos
- ☐  READ - Filtrar por nivel
- ☐  UPDATE - Actualizar estudiante completo (PUT)
- ☐  PATCH - Actualizar campo específico
- ☐  DELETE - Eliminar estudiante










5.3 Pruebas











- ☐ Probar cada petición desde VS Code
- ☐  Verificar que todas funcionan correctamente

Parte 6: Script de validación



6.1 Archivo scripts/validate.sh

Debes crear un script bash que valide:

- ☐  Existencia de package.json
- ☐  Existencia de src/db/db.json
- ☐  Existencia de .gitignore
- ☐  Existencia de .env.example
- ☐  Existencia de README.md
- ☐  Existencia de checklist.md
- ☐  Existencia de peticiones-crud.http
- ☐  Existencia de carpeta src/
- ☐  Existencia de src/crud-curl.js

- ☐  Existencia de carpeta images/
- ☐  Existencia de carpeta scripts/
- ☐  Verificar en package.json:
 -  Que tiene "type": "module"
 -  Que dotenv está instalado
 -  Que json-server está instalado
 -  Que existe script server:up
 -  Que existe script crud:curl
- ☐  Verificar existencia de al menos 6 capturas de Thunder Client en images/
- ☒  Mostrar mensaje final indicando si pasó o falló la validación









6.2 Configuración

- ☐  Dar permisos de ejecución al script
- ☒  Verificar que funciona desde terminal

Parte 7: Checklist de progreso

7.1 Archivo checklist.md



Debes crear un archivo que sirva como control de progreso. Debe incluir checkboxes para:

- ☐  Todas las tareas de configuración inicial
- ☐  Todos los scripts requeridos
- ☐  Todas las funciones del crud-curl.js
- ☐  Todas las operaciones documentadas
- ☐  Todas las peticiones de Thunder Client
- ☐  Todas las peticiones de REST Client
- ☒  Validación completa
- ☐  Todas las tareas de Git





El checklist debe estar organizado por fases o hitos.

Parte 8: Git y GitHub

8.1 Repositorio

- ☐ Crear repositorio en GitHub
- ☐  Nombre: `manual-http-[tu-nombre]`
- ☐  Añadir al profesor como colaborador




8.2 Configuración local

- ☐  Inicializar Git en tu proyecto local
- ☐  Conectar con el repositorio remoto
- ☐  Crear rama principal `main`
- ☐  Subir código inicial











8.3 Rama de desarrollo

- ☐  Crear rama `m1/http-request-response`
- ☐  Cambiar a esa rama para trabajar



8.4 Commits

- ☐  Realizar commits incrementales por cada fase del checklist
- ☐  Los mensajes de commit deben ser descriptivos
- ☐  Usar convención: `feat:`, `docs:`, `fix:`, etc.

8.5 Pull Request

- ☐  Subir la rama al repositorio remoto
- ☐  Crear Pull Request desde `m1/http-request-response` hacia `main`
- ☐  Título del PR: "Entrega M1/http-request-response"
- ☐  Descripción detallada del PR que incluya:
 -  Resumen de lo implementado
 -  División del trabajo (si es en pareja)
 -  Dificultades encontradas y soluciones
 -  Características completadas
- ☐  Asignar al profesor como reviewer
- ☐  Enviar el PR











8.6 Después de la aprobación

- ☐  Una vez el profesor apruebe y haga merge
- ☐  Actualizar tu rama `main` local

- ☐  Crear tag **M1/http-request-response**
- ☐  Subir el tag al repositorio remoto

Rúbrica de Evaluación





Criterios y Pesos

#	Criterio	Peso	 Nivel 1	 Nivel 2	 Nivel 3	Nota
1	 Configuración (10%)	1.0	Archivos faltantes, dependencias mal instaladas	Errores menores, falta .env.example	Configuración perfecta, todo funciona	<input type="checkbox"/> __/1.0
2	 Script CRUD (20%)	2.0	Funciones sin parámetros, comandos incorrectos	Funciones con parámetros, errores sintácticos	Comandos cURL perfectos y funcionales	<input type="checkbox"/> __/2.0
3	 Doc. README (20%)	2.0	Incompleta, sin respuestas HTTP reales	Completa pero explicaciones superficiales	Detallada, con headers y respuestas reales	<input type="checkbox"/> __/2.0
4	 Thunder Client (20%)	2.0	< 4 peticiones, capturas incompletas	6 peticiones, sin colección organizada	Colección completa, capturas profesionales	<input type="checkbox"/> __/2.0
5	 REST Client (15%)	1.5	< 4 peticiones, sintaxis incorrecta	CRUD básico, sin filtros/variables	CRUD + filtros, variables configuradas	<input type="checkbox"/> __/1.5
6	 Git workflow (10%)	1.0	< 3 commits, mensajes genéricos	Commits incrementales, PR básico	Mensajes descriptivos, PR profesional	<input type="checkbox"/> __/1.0
7	 Validación (5%)	0.5	Script no ejecuta o incompleto	Ejecuta, valida < 10 elementos	Validación completa, sin errores	<input type="checkbox"/> __/0.5

Cálculo de la nota final

 **Nota Final = Σ (Nivel obtenido \times Peso del criterio)**

Escala de niveles:

-  **Sin realizar** (no hay cruz en ningún nivel) = 0 puntos
-  **Nivel 1** = 33% del peso del criterio
-  **Nivel 2** = 66% del peso del criterio
-  **Nivel 3** = 100% del peso del criterio

 **Nota máxima:** 10.0 puntos

 **Requisito mínimo para aprobar:** 5.0 puntos + Script de validación debe pasar sin errores



Ejemplo de uso de la rúbrica

Si un alumno obtiene:

- 1 Criterio 1 (Configuración): ● Nivel 3 → $1.0 \times 100\% = 1.0$ puntos
- 2 Criterio 2 (Script): ● Nivel 2 → $2.0 \times 66\% = 1.32$ puntos
- 3 Criterio 3 (Documentación): ● Nivel 3 → $2.0 \times 100\% = 2.0$ puntos
- 4 Criterio 4 (Thunder Client): ● Nivel 2 → $2.0 \times 66\% = 1.32$ puntos
- 5 Criterio 5 (REST Client): ● Nivel 3 → $1.5 \times 100\% = 1.5$ puntos
- 6 Criterio 6 (Git): ● Nivel 2 → $1.0 \times 66\% = 0.66$ puntos
- 7 Criterio 7 (Validación): ● Nivel 3 → $0.5 \times 100\% = 0.5$ puntos



Nota Final = 8.3 / 10



Recursos de consulta

- 📖 Módulo 1 - Fundamentos HTTP y Herramientas
- 📖 Módulo 2 - Herramientas Visuales para APIs
- 🔗 Documentación oficial de json-server
- 🔗 Documentación oficial de cURL
- 🔗 Documentación de Thunder Client
- 🔗 Documentación de REST Client



Entrega



Método: Pull Request en GitHub



Fecha límite: [Definir por el profesor]



Importante:

- ✅ El PR debe estar creado antes de la fecha límite
- ✅ El script de validación debe pasar sin errores
- ❌ El .env NO debe estar en el repositorio



Trabajo en pareja (opcional)

Si trabajas en pareja:

- 🤝 Ambos deben aparecer como autores en los commits
- 📋 Debe haber división clara de responsabilidades

- 🧠 Ambos deben entender todo el código
 - 📝 La descripción del PR debe detallar quién hizo qué
-

? Preguntas frecuentes

? **¿Las funciones de crud-curl.js deben ejecutar los comandos o solo imprimirlos?**

Solo imprimirlos. El alumno los copiará y ejecutará manualmente.

? **¿Qué significa que las funciones reciban parámetros?**

Por ejemplo, `readStudentById(id)` debe poder recibir cualquier ID y generar el comando correspondiente.

? **¿Qué puerto usa json-server?**

El especificado en tu `.env` (por defecto 4000)

? **¿Cómo sé si mi script de validación funciona?**

Ejecútalo y debe mostrar todos los checks en verde

? **¿Debo usar Git Bash en Windows?**

Sí, para ejecutar el script `.sh`

? **¿Puedo modificar el `db.json`?**

Puedes añadir estudiantes, pero no eliminar los 7 existentes
