Pruebas Mock API con JSON Server – Informe Técnico

Estado general del módulo de pruebas:

La implementación de la Mock API mediante JSON Server cumple completamente con los requisitos establecidos para la simulación de datos del proyecto. Se confirmó la presencia de múltiples entidades relacionadas, datos realistas, variedad suficiente en los registros y configuración adecuada del servidor para emular endpoints consistentes con la arquitectura real del sistema.

- **Elementos verificados:**
  - Estructura del archivo db.json
  - Relaciones entre entidades
  - Realismo y variabilidad de los datos
  - Configuración de JSON Server
  - Scripts de ejecución incluidos en package.json
  - Ejecución de validaciones automáticas mediante un script de comprobación
  - Consulta manual y programática de endpoints

Resumen de Estado:

Requisito	Estado
Archivo db.json con dos o más entidades relacionadas	Cumple
Datos simulados realistas y variados	Cumple
Configuración funcional de JSON Server	Cumple
Scripts disponibles para iniciar el mock server	Cumple
Endpoints simulados correctamente y relaciones verificadas	Cumple
Validación programática mediante script	Cumple

1. Archivos principales verificados

Los archivos necesarios para ejecutar y validar el Mock Server se encuentran correctamente incluidos y configurados:

- **db.json:** Archivo principal de la Mock API que contiene datos simulados de múltiples entidades.
- **json-server-config.json:** Archivo de configuración para JSON Server (puerto, rutas estáticas, watch, entre otros).
- **package.json (raíz del proyecto):** Contiene el script de ejecución:

```
"mock:start": "json-server --watch db.json --config json-server-config.json --port 3001"
```

  e incluye json-server en las dependencias de desarrollo.
- **check-json-server.js** Script programático utilizado para levantar JSON Server y validar automáticamente los endpoints más relevantes.

2. Entidades presentes en db.json

Se verificó que el archivo contiene más de dos entidades relacionadas, cumpliendo con el requisito mínimo. Entre ellas destacan:

- users
- projects
- tasks
- projectMembers
- libraryItems
- notifications
- projectInvitations
- auditLogs

Estas entidades se encuentran relacionadas de manera coherente. Por ejemplo:

- projects ↔ tasks mediante projectId
- projects ↔ users mediante campos user o ownerId
- projectMembers funciona como entidad intermedia entre proyectos y usuarios
- libraryItems puede almacenar documentos o archivos asociados a proyectos

3. Calidad y variedad de los datos simulados

Los datos incluidos en db.json presentan atributos completos y realistas:

- Nombres, correos electrónicos y roles para usuarios
- Identificadores consistentes
- Proyectos con títulos, áreas temáticas, fechas ISO y descripciones coherentes
- Tareas con estados variados ("todo", "in_progress", "done"), prioridades diferenciadas y fechas válidas
- Elementos de biblioteca con URLs de documentos reales
- Notificaciones con timestamps y banderas de lectura
- Bitácoras (auditLogs) con información trazable

Asimismo, se evidencia variación adecuada entre los registros: múltiples proyectos, distintos usuarios y tareas distribuidas por estado y niveles de prioridad.

4. Configuración de JSON Server

La configuración se encuentra correctamente implementada.

Script en package.json

```
"mock:start": "json-server --watch db.json --config json-server-config.json --port 3001"
```

json-server-config.json

Contiene parámetros como:

- Puerto
- Directorio estático
- Rutas personalizadas (si aplica)

- Configuración de watch

Dependencias

Se confirmó la presencia de:

```
"json-server": "^0.x.x"
```

dentro de devDependencies.

5. Endpoints verificados

La configuraMediante pruebas manuales y mediante el script automatizado, se verificó el funcionamiento de los siguientes endpoints:

Consultas exitosas confirmadas

GET /projects: Retorna cinco proyectos en el conjunto de datos verificado.

GET /projects/1/tasks: Retorna correctamente las tareas cuya propiedad projectId coincide con la solicitada.

La consulta devolvió dos tareas identificadas programáticamente.

Otros endpoints accesibles:

- /users/1
- /tasks?projectId=1
- /projectMembers?projectId=1
- /libraryItems?projectId=1
- /notifications
- /projectInvitations
- /auditLogs

6. Verificación automática mediante script

El archivo check-json-server.js levanta JSON Server de forma programática y ejecuta solicitudes HTTP para validar la existencia y consistencia de los endpoints.

Resultados obtenidos:

/projects → OK, 5 elementos encontrados

/projects/1/tasks → OK, 2 elementos encontrados

La Mock API respondió de manera consistente con la estructura relacional contenida en db.json.

script que inicia programáticamente JSON Server y valida endpoints

/tools/check-json-server.js

```
const jsonServer = require('json-server');
const http = require('http');
const path = require('path');

async function startAndCheck() {
  const server = jsonServer.create();
  const router = jsonServer.router(path.resolve(__dirname, '..', 'db.json'));
  const middlewares = jsonServer.defaults();

  server.use(middlewares);
  server.use(router);
  const listener = server.listen(0, () => {
    const port = listener.address().port;
    console.log('json-server started on port', port);

    // Check /projects
    http.get({ hostname: '127.0.0.1', port, path: '/projects', timeout: 2000 }, (res) => {
      let data = '';
      res.on('data', (chunk) => (data += chunk));
      res.on('end', () => {
        try {
          const parsed = JSON.parse(data);
          console.log('/projects -> OK, found', Array.isArray(parsed) ? parsed.length : 'n/a',
'items');
        } catch (e) {
          console.error('Failed parsing /projects response');
        }

        // Check relation: /projects/1/tasks
        http.get({ hostname: '127.0.0.1', port, path: '/projects/1/tasks', timeout: 2000 }, (res2) =>
{
          let d2 = '';
          res2.on('data', (c) => (d2 += c));
          res2.on('end', () => {
            try {
              const parsed2 = JSON.parse(d2);
              console.log('/projects/1/tasks -> OK, found', Array.isArray(parsed2) ? parsed2.length
: 'n/a', 'items');
            } catch (e) {
              console.error('Failed parsing /projects/1/tasks response');
            }
            listener.close(() => process.exit(0));
          });
        }).on('error', (err) => {
          console.error('Error requesting /projects/1/tasks:', err.message);
          listener.close(() => process.exit(2));
        });
      });
    }).on('error', (err) => {
      console.error('Error requesting /projects:', err.message);
      listener.close(() => process.exit(2));
    });
  });
}
startAndCheck();
```