Exercício – Testes de CRUD com Jest (ESM) e JSON-Server

Objetivo

Escrever uma suíte de testes **Jest** (usando **ECMAScript Modules**) para validar **todas** as operações do CRUD e **PATCH** da API simulada pelo **json-server**. Os testes devem iniciar e encerrar os recursos necessários **dentro da própria suíte** (ex.: subir/derrubar o `json-server` via `beforeAll` / `afterAll`), sem passos manuais.

Requisitos do Projeto

- **Node.js 18+** (usa `fetch` nativo)
- Projeto configurado como **ESM** (`"type": "module"` no `package.json`)
- **Jest** 29+

1) Script personalizado para Jest (ESM)

No `package.json` , crie um script **customizado** que use a flag experimental recomendada no *Guide* do Jest para ESM:

```
{
  "type": "module",
  "scripts": {
    "test:esm": "node --experimental-vm-modules node_modules/jest/bin/jes
  },
  "devDependencies": {
    "jest": "^29.7.0"
  }
}
```

> Use `npm run test:esm` para executar os testes.

2) Arquivo de configuração do Jest

Crie `jest.config.js` (como ESM) com `testEnvironment: "node"` e um *timeout* suficiente para subir/derrubar serviços:

```
// jest.config.js  (ESM - Node 18+)
export default {
  testEnvironment: "node",
  testMatch: ["**/tests/**/*.test.js"],
  transform: {}, // sem Babel/ts-jest
  verbose: true,
  testTimeout: 30000, // tempo extra para start/stop do json-server
};
```

Tarefa

A estrutura mínima do projeto deve ser a seguinte:

```
project-root/
 ├─ db.json
 ├─ package.json
 ├─ jest.config.js
 └─ tests/
     └─ equipamentos.test.js
```

Crie `tests/equipamentos.test.js` (sem usar libs externas de fetch; utilize o `fetch` nativo do Node 18+). Sua suíte deve:

1. **Preparação/Finalização**
    - `beforeAll` : iniciar os recursos necessários aos testes (ex.: start programático do `json-server` apontando para o `db.json` fornecido pela disciplina).
    - `afterAll` : encerrar todos os recursos iniciados no `beforeAll` (não deixar processos pendurados).
2. **Cobertura de Rotas**
    - `GET /equipamentos`
    - `GET /equipamentos/:id`
    - `POST /equipamentos`
    - `PUT /equipamentos/:id`
    - `PATCH /equipamentos/:id`
    - `DELETE /equipamentos/:id`

3. **Validações**
    - Utilizar **o máximo possível de matchers** do Jest (`toBe` , `toEqual` , `toContain` , `toHaveProperty` , `toBeDefined` , `toMatchObject` , `toBeGreaterThan` , `toMatch` , etc.).
    - Validar **status codes**, **estrutura** e **conteúdo** das respostas.
    - Garantir **idempotência/isolamento**: limpar artefatos criados durante os testes (e.g., remover registros criados).
4. **Casos Negativos (Erros)**
    - Buscar, atualizar e deletar **ID inexistente** (verificar `404`).
    - Enviar payloads **inválidos** (verificar `400` /comportamento do `json-server`).
    - Tentar `PATCH` com campo não permitido (observar resposta).
5. **Boas Práticas**
    - Evitar dependência de **ordem** entre testes (cada teste deve preparar seu cenário).
    - Usar `--runInBand` (já presente no script) para evitar concorrência ao manipular o mesmo "banco" (`db.json`).

Critérios de Avaliação

- Execução via **script customizado** `npm run test:esm` (com `--experimental-vm-modules`).
- Configuração correta do **Jest** em ESM (`jest.config.js` com `testEnvironment: "node"`).
- Cobertura completa de **CRUD + PATCH**.
- Uso amplo e correto de **matchers** do Jest.
- Implementação de **casos negativos** e validações de **status code**.
- **Start/stop programático** dos recursos em `beforeAll` / `afterAll` , sem passos manuais.
