# Informe de Configuración Jest y CI
# Fecha: 2025-10-06
# Proyecto

**Puntos a abordar:**

1. Configuración optimizada con scripts personalizados
    - Scripts añadidos/ajustados en package.json:
        - "test": jest --coverage --runInBand
        - "test:ci": jest --coverage --runInBand
        - "test:unit": jest --runInBand --testPathPattern=tests
        - "test:integration": jest --runInBand --testPathPattern=tests.integration
    - Razonamiento: test:ci es el entry point claro para CI (sin watch, con coverage); --runInBand evita problemas de concurrencia con mongodb-memory-server en runners ligeros.
    - Ajustes operativos: se añadió testTimeout (300000 ms) para reducir flakes relacionados con arranques lentos de servicios en memoria.
2. **Reportes de cobertura detallados y configuración de thresholds**
    - Reporters configurados en Jest (backend): json, lcov, text, text-summary — genera lcov para herramientas externas y resúmenes legibles en CI/local.
    - Thresholds globales añadidos (coverageThreshold.global): mínimo garantizado para Lines y Statements = 70%. Esto hace que la ejecución falle si las métricas críticas bajan del umbral.
    - Resultado local tras ejecución: Statements 80.95%, Branches 71.17%, Functions 75.6%, Lines 84.59% (por encima del umbral para statements/lines).
    - Práctica recomendada aplicada: mantener reports lcov para subir a servicios de reporte (Codecov/Coveralls) o para análisis en CI.
3. **Integración con herramientas CI/CD**
    - Workflow GitHub Actions añadido (ci.yml) con dos jobs independientes:
        - backend-tests: checkout, setup-node (Node 18), npm ci en backend, npm run test:ci, subir carpeta coverage como artifact.
        - frontend-tests: checkout, setup-node (Node 18), npm ci en gestion-proyectos-frontend, npm run test:ci, subir coverage.
    - Comportamiento: el workflow corre en push y pull_request sobre la rama principal; falla si test:ci falla o si los thresholds de Jest no se cumplen.
    - Ventajas: ejecución reproducible en CI, artefactos de coverage disponibles para descarga, y un punto único (test:ci) para invocar en pipelines.

**Notas operativas y recomendaciones del equipo**

- Mantener coverageThreshold en global para statements/lines y, si se desea, extender per-path para módulos críticos.
- Publicar coverage/lcov.info a Codecov/Coveralls desde el workflow para historial y badges automáticos.
- Evitar subir node_modules al repo (ya se corrigió localmente); usar .gitignore para limpiar el repositorio.
- Verificar en GitHub Actions la primera ejecución del workflow (artifacts y logs) y ajustar si algún runner requiere variables de entorno (API keys, timeouts adicionales).

**ANEXOS:**

ci.yml:

```
name: CI

on:
 push:
  branches: [ main, master ]
 pull_request:
  branches: [ main, master ]

jobs:
 backend-tests:
  name: Backend tests
  runs-on: ubuntu-latest
  steps:
   - uses: actions/checkout@v4
   - name: Use Node.js 18
     uses: actions/setup-node@v4
     with:
       node-version: 18
   - name: Install backend dependencies
     working-directory: backend
     run: npm ci
   - name: Run backend tests (CI)
     working-directory: backend
     run: npm run test:ci
   - name: Upload backend coverage
     uses: actions/upload-artifact@v4
     with:
       name: backend-coverage
       path: backend/coverage

 frontend-tests:
  name: Frontend tests
  runs-on: ubuntu-latest
```

```
    steps:
      - uses: actions/checkout@v4
      - name: Use Node.js 18
        uses: actions/setup-node@v4
        with:
          node-version: 18
      - name: Install frontend dependencies
        working-directory: gestion-proyectos-frontend
        run: npm ci
      - name: Run frontend tests (CI)
        working-directory: gestion-proyectos-frontend
        run: npm run test:ci
      - name: Upload frontend coverage
        uses: actions/upload-artifact@v4
        with:
          name: frontend-coverage
          path: gestion-proyectos-frontend/coverage
```

**aiController.branches.test.js**

```
jest.mock('@google/genai', () => ({
  GoogleGenAI: jest.fn(),
}));

const httpMocks = require('node-mocks-http');
const axios = require('axios');
const { GoogleGenAI } = require('@google/genai');

// Provide a default mock implementation so controller's module-level `ai` is usable
const defaultInst = { models: { generateContent: jest.fn() } };
GoogleGenAI.mockImplementation(() => defaultInst);
// Now require controller after mocking GoogleGenAI
const { summarizeText, handleChat, suggestArticles } =
require('../controllers/aiController');

describe('aiController branches and extraction', () => {
  let inst;
  beforeAll(() => {
    // ensure SerpAPI key present for suggestArticles branches
    process.env.SERPAPI_API_KEY = process.env.SERPAPI_API_KEY || 'testkey';
  });

  beforeEach(() => {
    // ensure fresh mock per test and update controller.ai reference via
mockImplementation
    inst = { models: { generateContent: jest.fn() } };
    GoogleGenAI.mockImplementation(() => inst);
```

```
  // ensure the controller's module-level `ai` (defaultInst) uses this test's mock
  if (defaultInst && defaultInst.models) {
   defaultInst.models.generateContent = inst.models.generateContent;
  }
 });

 test('summarizeText returns 400 when text missing', async () => {
  const req = httpMocks.createRequest({ method: 'POST', url: '/api/ai/summarize',
body: {} });
  const res = httpMocks.createResponse();
  await summarizeText(req, res);
  expect(res.statusCode).toBe(400);
  const data = res._getJSONData();
  expect(data).toHaveProperty('message', 'text requerido');
 });

 test('summarizeText retries on RESOURCE_EXHAUSTED then succeeds and uses text
property', async () => {
  // first call rejects with RESOURCE_EXHAUSTED, second resolves with text
  inst.models.generateContent
   .mockRejectedValueOnce({ code: 'RESOURCE_EXHAUSTED', message: 'quota' })
   .mockResolvedValueOnce({ text: 'Resumen final' });

  const req = httpMocks.createRequest({ method: 'POST', url: '/api/ai/summarize',
body: { text: 'Hola mundo' } });
  const res = httpMocks.createResponse();
  await summarizeText(req, res);
  expect(res.statusCode).toBe(200);
  const data = res._getJSONData();
  expect(data).toHaveProperty('summary', 'Resumen final');
  expect(data).toHaveProperty('model');
 });

 test('handleChat returns 400 when message missing', async () => {
  const req = httpMocks.createRequest({ method: 'POST', url: '/api/ai/chat', body: {} });
  const res = httpMocks.createResponse();
  await handleChat(req, res);
  expect(res.statusCode).toBe(400);
  expect(res._getJSONData()).toHaveProperty('message', 'message requerido');
 });

 test('handleChat extracts text from candidates.parts fallback', async () => {
  inst.models.generateContent.mockResolvedValueOnce({ candidates: [{ content: {
parts: [{ text: 'parte1' }, { text: 'parte2' }] } }] });

  const req = httpMocks.createRequest({ method: 'POST', url: '/api/ai/chat', body: {
message: 'hola' } });
  const res = httpMocks.createResponse();
  await handleChat(req, res);
  expect(res.statusCode).toBe(200);
  const data = res._getJSONData();
  expect(data).toHaveProperty('text');
```

```
  expect(data.text).toContain('parte1');
 });

 test('suggestArticles handles SerpAPI error and returns 502', async () => {
  jest.spyOn(axios, 'get').mockResolvedValueOnce({ data: { error: 'quota' } });
  const req = httpMocks.createRequest({ method: 'POST', url: '/api/ai/suggest', body: {
query: 'fake' } });
  const res = httpMocks.createResponse();
  await suggestArticles(req, res);
  expect(res.statusCode).toBe(502);
  const data = res._getJSONData();
  expect(data).toHaveProperty('message');
  axios.get.mockRestore();
 });

 test('suggestArticles maps results and pdfUrl correctly', async () => {
  const fakeItem = {
   title: 'T',
   link: '',
   snippet: 'Sni',
   publication_info: { authors: [{ name: 'A' }], summary: '(2020) example' },
   resources: [{ file_format: 'PDF', link: 'http://file.pdf' }],
   result_id: 'r1',
  };
  jest.spyOn(axios, 'get').mockResolvedValueOnce({ data: { organic_results: [fakeItem]
} });
  const req = httpMocks.createRequest({ method: 'POST', url: '/api/ai/suggest', body: {
query: 'q' } });
  const res = httpMocks.createResponse();
  await suggestArticles(req, res);
  expect(res.statusCode).toBe(200);
  const data = res._getJSONData();
  expect(Array.isArray(data.results)).toBe(true);
  expect(data.results[0]).toHaveProperty('pdfUrl', 'http://file.pdf');
  axios.get.mockRestore();
 });
});
```