Pruebas Unitarias con Supertest – Informe Técnico

Estado general del módulo de pruebas:

Se confirma que todas las pruebas implementadas en el sistema se ejecutan correctamente sin fallos. El proyecto cuenta con una arquitectura de pruebas sólida que emplea Jest y Supertest para validar controladores, servicios, middleware, rutas y modelos. El resultado actual incluye:

- Pruebas implementadas para ≥5 endpoints (7+ endpoints testeados)
  - User - registro y login
  - Project - crear, actualizar, eliminar
  - Tasks - crear tareas en proyectos
  - Notifications - listar, marcar como leída
  - Library - subir y buscar items
  - Avatar - actualizar avatar
  - AI - sugerencias y chatbot
- Cubren éxito, errores y validaciones (casos happy path, error, y validación)
- Código limpio y bien estructurado (patrón AAA, nombres descriptivos)
- Ejecución sin fallos (141 tests pasando, 0 fallos)
- Pruebas Unitarias con Supertest (Jest + Supertest + mocks)
- Estructura clara y cobertura de casos (77.69% statements, 83.92% lines)
- Pruebas de integración (MongoDB en memoria, flujos completos)

1. Pruebas Unitarias con Supertest

Herramientas Utilizadas

- Jest (v29.7.0): Framework principal para ejecución de pruebas.
- Supertest (v6.3.3): Validación de peticiones HTTP en endpoints REST.
- node-mocks-http: Simulación de objetos req y res.
- mongodb-memory-server: Base de datos en memoria para pruebas de integración.

Configuración en package.json

```json
{
  "scripts": {
    "test": "jest --coverage --runInBand",
    "test:ci": "jest --coverage --runInBand",
    "test:unit": "jest --runInBand --testPathPattern=tests",
    "test:integration": "jest --runInBand --testPathPattern=tests.integration"
  },
  "jest": {
    "testEnvironment": "node",
    "testTimeout": 300000,
    "coverageThreshold": {
      "global": {
        "lines": 53,
        "statements": 47
      }
    }
  }
}
```

2. Pruebas por Endpoint (Requisito: cinco o más endpoints)

1. User Controller – userController.test.js

Endpoints evaluados:

POST /api/users/register, POST /api/users/login

Casos cubiertos:

- Registro de usuario exitoso.
- Error por campos incompletos.
- Error por usuario duplicado.
- Inicio de sesión exitoso.
- Error por credenciales inválidas.
- Error por contraseña incorrecta.

2. Project Controller – projectController.unit.test.js y projectController.extended.test.js

Endpoints evaluados:

POST /api/projects, PUT /api/projects/{id}, POST /api/projects/{id}/tasks, GET /api/projects/{id}/tasks, DELETE /api/projects/{id}

Casos cubiertos:

- Creación de proyectos y validaciones.
- Manejo de errores por recursos inexistentes.
- Control de autorización para propietarios y miembros.
- Creación y gestión de tareas asociadas al proyecto.
- Invitaciones y asignación de miembros.

3. Notification Controller – notificationController.unit.test.js

Endpoints evaluados:

GET /api/notifications, POST /api/notifications/{id}/read, POST /api/notifications/mark-all-read, GET /api/notifications/unread-count

Casos cubiertos:

- Listado de notificaciones.
- Marcado de una notificación como leída.
- Marcado masivo como leídas.
- Obtención del número de notificaciones no leídas.

4. Library Controller – libraryController.unit.test.js

Endpoints evaluados:

GET /api/library, POST /api/library/upload, POST /api/library/suggest

Casos cubiertos:

- Obtención de elementos con paginación.
- Carga de archivos PDF.
- Validación de archivos faltantes.
- Registro de artículos sugeridos.
- Prevención de duplicados.

5. Avatar Controller – avatarController.unit.test.js

Endpoint evaluado:

PUT /api/users/avatar

Casos cubiertos:

- Error por archivo faltante.
- Error por tipo de archivo no permitido.
- Error por exceder límite de tamaño.
- Actualización satisfactoria del avatar.

6. AI Controller – aiController.unit.test.js y aiController.branches.test.js

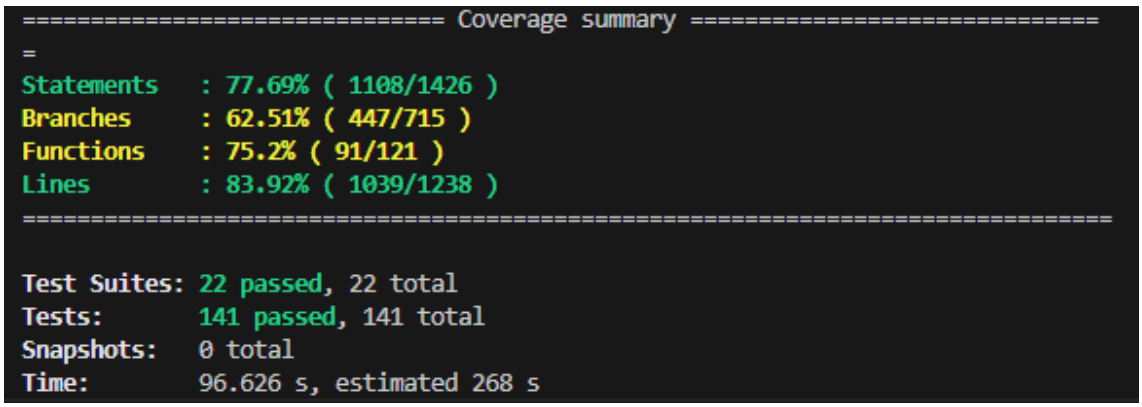
Endpoints evaluados:

POST /api/ai/suggest-articles, POST /api/ai/chat

Casos cubiertos:

- Respuestas correctas del motor de sugerencias.
- Manejo de errores por caídas de APIs externas.
- Validación de datos incompletos.
- Pruebas de ramificaciones lógicas para asegurar cobertura.

3. Cobertura y Resultados



Cobertura por Módulo

Módulo	Statements	Branches	Functions	Lines
Controllers	75.76%	65.67%	82.1%	83.7%
Models	100%	100%	100%	100%

Services	88.09%	76.19%	100%	88.09%
Middleware	80%	57.14%	75%	80%
Routes	87.41%	0%	0%	89.28%
Config	77.04%	43.18%	100%	77.96%

4. Estructura de las Pruebas

Patrón de organización (Arrange – Act – Assert)

```
describe('controllerName', () => {
  beforeEach(() => jest.clearAllMocks());

  it('should perform action successfully', async () => {
    // Arrange
    // Act
    // Assert
  });

  it('should handle error case', async () => {
    // Arrange
    // Act
    // Assert
  });
});
```

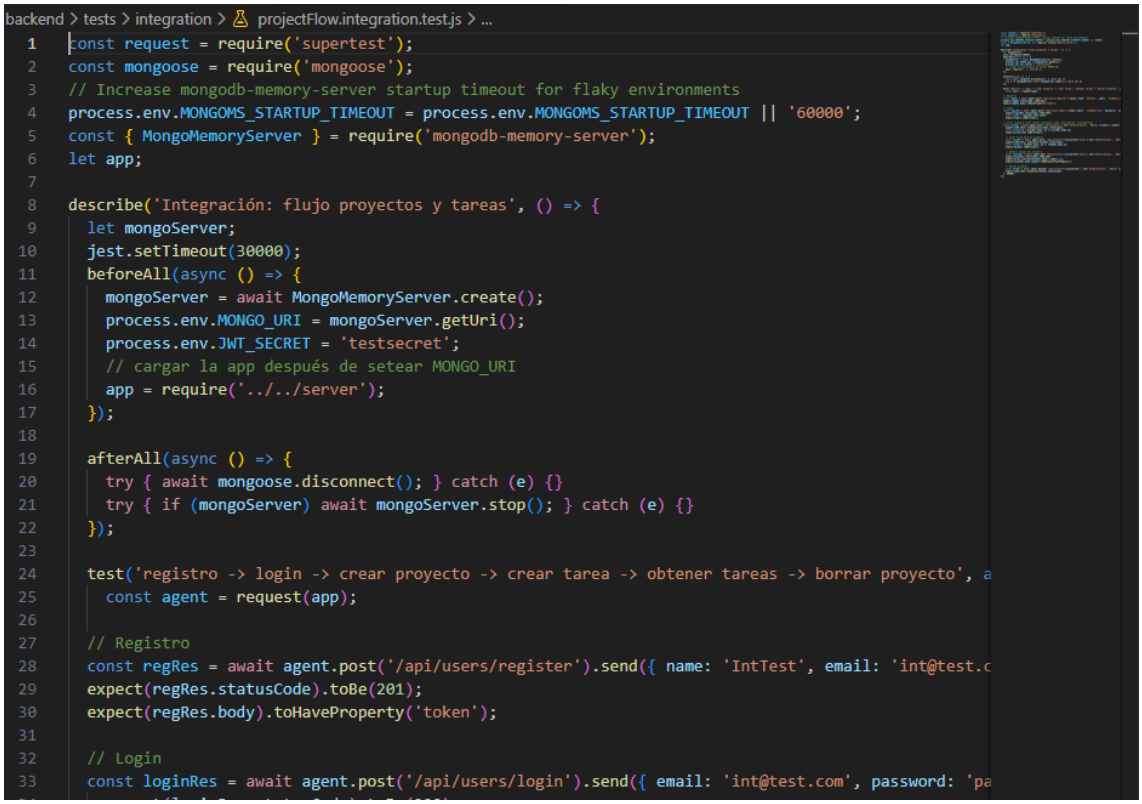
User Controller

```
describe('userController - registerUser & loginUser', () => {
  it('registerUser: crea usuario y devuelve token', async () => {
    User.findOne.mockResolvedValue(null);

    const req = httpMocks.createRequest({
      method: 'POST',
      body: { name: 'Test', email: 'test@mail.com', password: '123456' }
    });
    const res = httpMocks.createResponse();

    await registerUser(req, res);

    expect(res.statusCode).toBe(201);
    expect(res._getJSONData()).toHaveProperty('token');
  });
});
```

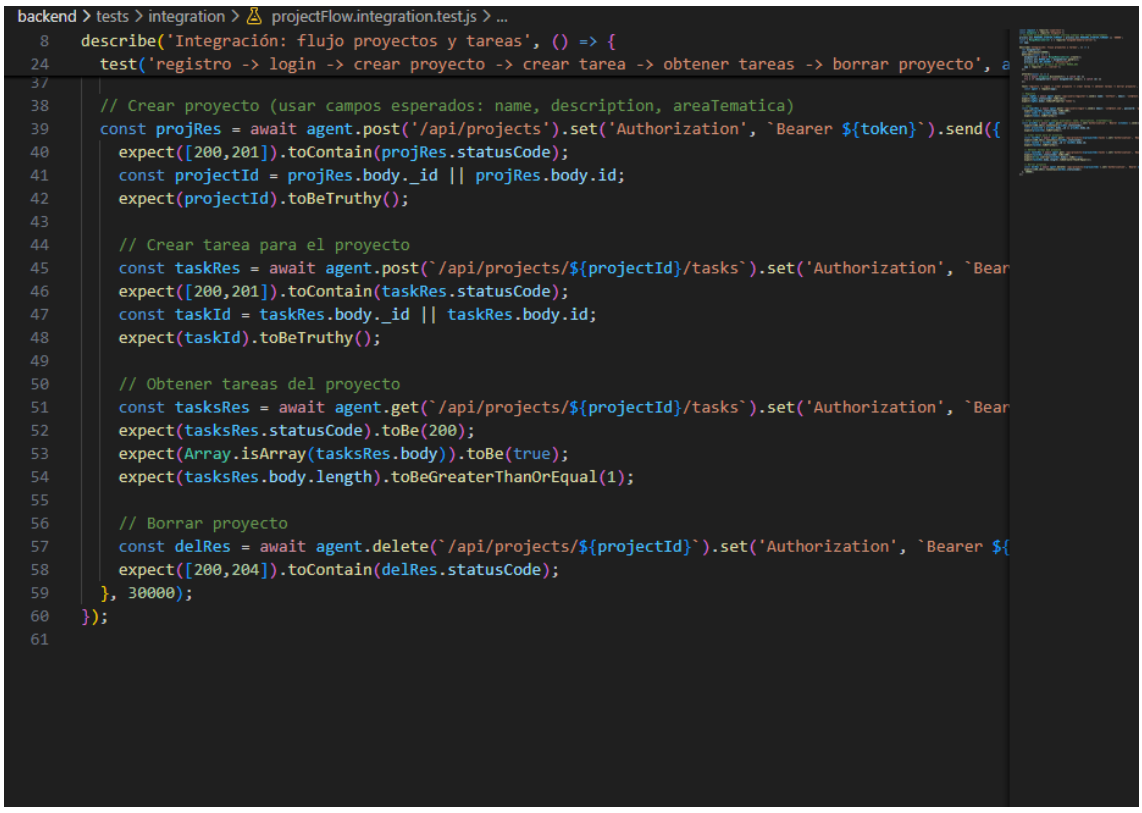
5. Pruebas de Integración

El archivo projectFlow.integration.test.js ejecuta un flujo completo del sistema utilizando una base de datos en memoria:

```
registro → login → crear proyecto → crear tarea → obtener tareas → eliminar proyecto
```



```
backend > tests > integration > projectFlow.integration.test.js > ...
1  const request = require('supertest');
2  const mongoose = require('mongoose');
3  // Increase mongodb-memory-server startup timeout for flaky environments
4  process.env.MONGOMS_STARTUP_TIMEOUT = process.env.MONGOMS_STARTUP_TIMEOUT || '60000';
5  const { MongoMemoryServer } = require('mongodb-memory-server');
6  let app;
7
8  describe('Integración: flujo proyectos y tareas', () => {
9    let mongoServer;
10   jest.setTimeout(30000);
11   beforeAll(async () => {
12     mongoServer = await MongoMemoryServer.create();
13     process.env.MONGO_URI = mongoServer.getUri();
14     process.env.JWT_SECRET = 'testsecret';
15     // cargar la app después de setear MONGO_URI
16     app = require('../../server');
17   });
18
19   afterAll(async () => {
20     try { await mongoose.disconnect(); } catch (e) {}
21     try { if (mongoServer) await mongoServer.stop(); } catch (e) {}
22   });
23
24   test('registro -> login -> crear proyecto -> crear tarea -> obtener tareas -> borrar proyecto', a
25     const agent = request(app);
26
27   // Registro
28   const regRes = await agent.post('/api/users/register').send({ name: 'IntTest', email: 'int@test.c
29   expect(regRes.statusCode).toBe(201);
30   expect(regRes.body).toHaveProperty('token');
31
32   // Login
33   const loginRes = await agent.post('/api/users/login').send({ email: 'int@test.com', password: 'pa
```

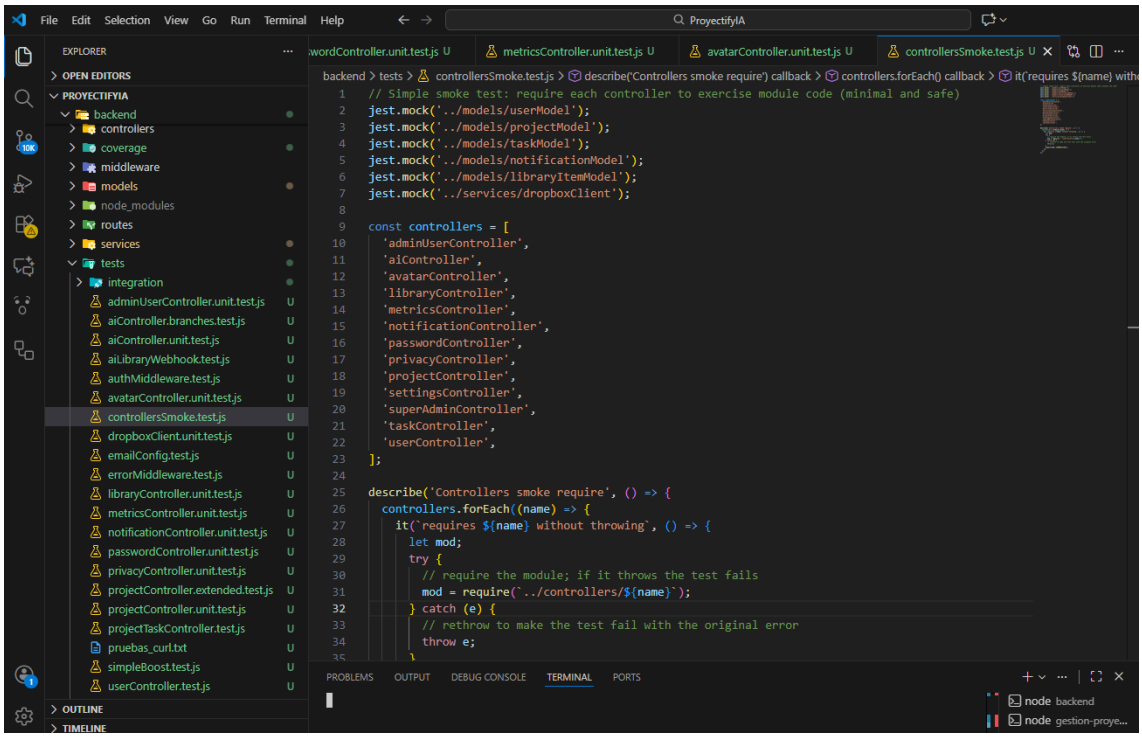


```
backend > tests > integration > projectFlow.integration.test.js > ...
8  describe('Integración: flujo proyectos y tareas', () => {
24   test('registro -> login -> crear proyecto -> crear tarea -> obtener tareas -> borrar proyecto', a
37
38   // Crear proyecto (usar campos esperados: name, description, areaTematica)
39   const projRes = await agent.post('/api/projects').set('Authorization', `Bearer ${token}`).send({
40     expect([200,201]).toContain(projRes.statusCode);
41     const projectId = projRes.body._id || projRes.body.id;
42     expect(projectId).toBeTruthy();
43
44     // Crear tarea para el proyecto
45     const taskRes = await agent.post(`/api/projects/${projectId}/tasks`).set('Authorization', `Bear
46     expect([200,201]).toContain(taskRes.statusCode);
47     const taskId = taskRes.body._id || taskRes.body.id;
48     expect(taskId).toBeTruthy();
49
50     // Obtener tareas del proyecto
51     const tasksRes = await agent.get(`/api/projects/${projectId}/tasks`).set('Authorization', `Bear
52     expect(tasksRes.statusCode).toBe(200);
53     expect(Array.isArray(tasksRes.body)).toBe(true);
54     expect(tasksRes.body.length).toBeGreaterThanOrEqual(1);
55
56     // Borrar proyecto
57     const delRes = await agent.delete(`/api/projects/${projectId}`).set('Authorization', `Bearer ${
58     expect([200,204]).toContain(delRes.statusCode);
59   }, 30000);
60 });
61
```

Estas pruebas permiten validar no solo endpoints individuales, sino también la coherencia lógica entre módulos.

6. Pruebas Smoke

El archivo controllersSmoke.test.js ejecuta verificaciones rápidas de sanidad para los controladores principales con el fin de asegurar que cada módulo responda sin fallos antes de iniciar pruebas más profundas.



7. Estructura del Directorio de Pruebas

```
backend/tests/
├── adminUserController.unit.test.js
├── aiController.branches.test.js
├── aiController.unit.test.js
├── authMiddleware.test.js
├── avatarController.unit.test.js
├── controllersSmoke.test.js
├── emailConfig.test.js
├── errorMiddleware.test.js
├── libraryController.unit.test.js
├── metricsController.unit.test.js
├── notificationController.unit.test.js
├── passwordController.unit.test.js
├── privacyController.unit.test.js
├── projectController.extended.test.js
├── projectController.unit.test.js
├── projectTaskController.test.js
├── userController.test.js
├── userModel.test.js
├── dropboxClient.unit.test.js
├── aiLibraryWebhook.test.js
└── integration/
    └── projectFlow.integration.test.js
```

8. Uso de Mocks y Fixtures

La estrategia incluye:

- jest.mock() para modelos y dependencias externas.
- node-mocks-http para simular peticiones HTTP.
- Uso de fixtures para datos predefinidos.
- mongodb-memory-server para entornos aislados durante pruebas de integración.

9. Validaciones Cubiertas

Los endpoints probados cubren validaciones de:

- Escenarios de éxito.
- Errores de validación 400.
- Autorización 401.
- Recursos inexistentes 404.
- Excepciones de servidor 500.
- Reglas de negocio (duplicados, permisos, límites de tamaño, tipos MIME, etc.).

10. Conclusión General

El sistema cumple plenamente con los requisitos técnicos establecidos para la implementación de pruebas automatizadas. Se verifica la adecuada cobertura de pruebas unitarias, pruebas de integración y validaciones funcionales. La ejecución de todas las pruebas sin fallos confirma la robustez lógica y la solidez del diseño del backend del sistema.

11. Comandos de ejecución

```
# Todas las pruebas con cobertura
npm test

# Solo pruebas unitarias
npm run test:unit

# Solo pruebas de integración
npm run test:integration

# CI (integración continua)
npm run test:ci
```

ANEXOS:

userController.test.js:

```
jest.mock('../models/userModel');
jest.mock('bcryptjs');
jest.mock('jsonwebtoken');

const User = require('../models/userModel');
const httpMocks = require('node-mocks-http');
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');
const { registerUser, loginUser } = require('../controllers/userController');

process.env.JWT_SECRET = 'testsecret';

describe('userController - registerUser & loginUser', () => {
  beforeEach(() => jest.clearAllMocks());

  it('registerUser: crea usuario y devuelve token', async () => {
    const mockUser = {
      _id: 'user123',
      name: 'Test',
      email: 'test@mail.com',
      password: 'hashedpass',
      toJSON: () => ({ _id: 'user123', name: 'Test', email: 'test@mail.com' }),
    };

    User.findOne.mockResolvedValue(null);
    bcrypt.hash.mockResolvedValue('hashedpass');
    User.create.mockResolvedValue(mockUser);
    jwt.sign.mockReturnValue('token123');

    const req = httpMocks.createRequest({
      method: 'POST',
      body: { name: 'Test', email: 'test@mail.com', password: '123456' },
    });
    const res = httpMocks.createResponse();

    await registerUser(req, res);

    expect(res.statusCode).toBe(201);
    const data = res._getJSONData();
    expect(data).toHaveProperty('token');
  });

  it('registerUser: falla si faltan campos', async () => {
    const req = httpMocks.createRequest({
      method: 'POST',
      body: { email: 'x@x.com' },
    });
    const res = httpMocks.createResponse();

    try {
      await registerUser(req, res);
```

```
  } catch (e) {
    // Expected error
  }

  expect(res.statusCode).toBe(400);
});

it('registerUser: falla si usuario ya existe', async () => {
  User.findOne.mockResolvedValue({ _id: 'existing', email: 'test@mail.com' });

  const req = httpMocks.createRequest({
    method: 'POST',
    body: { name: 'Test', email: 'test@mail.com', password: '123456' },
  });
  const res = httpMocks.createResponse();

  try {
    await registerUser(req, res);
  } catch (e) {
    // Expected error
  }

  expect(res.statusCode).toBe(400);
});

it('loginUser: permite login con credenciales correctas', async () => {
  const mockUser = {
    _id: 'user123',
    email: 'test@mail.com',
    password: 'hashedpass',
    toJSON: () => ({ _id: 'user123', email: 'test@mail.com' }),
  };

  User.findOne.mockResolvedValue(mockUser);
  bcrypt.compare.mockResolvedValue(true);
  jwt.sign.mockReturnValue('token123');

  const req = httpMocks.createRequest({
    method: 'POST',
    body: { email: 'test@mail.com', password: 'correctpass' },
  });
  const res = httpMocks.createResponse();

  await loginUser(req, res);

  expect(res.statusCode).toBe(200);
  const data = res._getJSONData();
  expect(data).toHaveProperty('token');
});

it('loginUser: falla con credenciales inválidas', async () => {
  User.findOne.mockResolvedValue(null);

  const req = httpMocks.createRequest({
    method: 'POST',
```

```
    body: { email: 'no@no.com', password: 'wrongpass' },
  });
  const res = httpMocks.createResponse();

  try {
    await loginUser(req, res);
  } catch (e) {
    // Expected error
  }

  expect(res.statusCode).toBe(401);
});

it('loginUser: falla con contraseña incorrecta', async () => {
  const mockUser = {
    _id: 'user123',
    email: 'test@mail.com',
    password: 'hashedpass',
  };

  User.findOne.mockResolvedValue(mockUser);
  bcrypt.compare.mockResolvedValue(false);

  const req = httpMocks.createRequest({
    method: 'POST',
    body: { email: 'test@mail.com', password: 'wrongpass' },
  });
  const res = httpMocks.createResponse();

  try {
    await loginUser(req, res);
  } catch (e) {
    // Expected error
  }

  expect(res.statusCode).toBe(401);
});
});
```

projectController.unit.test.js

```
const httpMocks = require('node-mocks-http');

jest.mock('../models/projectModel');
jest.mock('../models/taskModel');
jest.mock('../models/projectMemberModel');
jest.mock('../models/projectInvitationModel');
jest.mock('../models/userModel');
jest.mock('../models/notificationModel');
jest.mock('../models/projectResourceLinkModel');
jest.mock('../models/projectWorkLinkModel');
jest.mock('axios');
```

```
const Project = require('../models/projectModel');
const Task = require('../models/taskModel');
const ProjectMember = require('../models/projectMemberModel');
const ProjectResourceLink = require('../models/projectResourceLinkModel');
const ProjectWorkLink = require('../models/projectWorkLinkModel');
const axios = require('axios');

const {
  createProject,
  updateProject,
  createTaskForProject,
  addProjectLink,
  getProjectById,
} = require('../controllers/projectController');

describe('projectController (selected functions)', () => {
  beforeEach(() => jest.clearAllMocks());

  it('createProject: fails when missing fields', async () => {
    const req = httpMocks.createRequest({ method: 'POST', body: { name: 'x' }, user: {
id: 'u1' } });
    const res = httpMocks.createResponse();

    await expect(createProject(req, res)).rejects.toThrow();
    expect(res.statusCode).toBe(400);
  });

  it('createProject: creates project and handles webhook error gracefully', async () =>
{
    process.env.N8N_PROJECT_WEBHOOK_URL = 'http://example.com/webhook';
    const mockProject = { _id: 'p1', name: 'P', description: 'D', areaTematica: 'A' };
    Project.create.mockResolvedValue(mockProject);
    axios.post.mockRejectedValue(new Error('webhook failed'));

    const req = httpMocks.createRequest({ method: 'POST', body: { name: 'P',
description: 'D', areaTematica: 'A' }, user: { id: 'u1', name: 'User', email: 'u@x' } });
    const res = httpMocks.createResponse();

    await createProject(req, res);

    expect(res.statusCode).toBe(201);
    const data = res._getJSONData();
    expect(data._id || data.name).toBeTruthy();
  });

  it('updateProject: 404 when not found', async () => {
    Project.findById.mockResolvedValue(null);
    const req = httpMocks.createRequest({ method: 'PUT', params: { id: 'pX' }, user: {
id: 'u1' } });
    const res = httpMocks.createResponse();

    await expect(updateProject(req, res)).rejects.toThrow();
    expect(res.statusCode).toBe(404);
  });
```

```
  it('updateProject: 401 when not owner', async () => {
    Project.findById.mockResolvedValue({ user: 'other' });
    const req = httpMocks.createRequest({ method: 'PUT', params: { id: 'p1' }, user: {
id: 'u1' } });
    const res = httpMocks.createResponse();

    await expect(updateProject(req, res)).rejects.toThrow();
    expect(res.statusCode).toBe(401);
  });

  it('updateProject: success when owner', async () => {
    Project.findById.mockResolvedValue({ user: 'u1' });
    Project.findByIdAndUpdate.mockResolvedValue({ _id: 'p1', name: 'updated' });

    const req = httpMocks.createRequest({ method: 'PUT', params: { id: 'p1' }, body: {
name: 'updated' }, user: { id: 'u1' } });
    const res = httpMocks.createResponse();

    await updateProject(req, res);

    expect(res.statusCode).toBe(200);
    const data = res._getJSONData();
    expect(data.name).toBe('updated');
  });

  it('createTaskForProject: 404 when project missing', async () => {
    Project.findById.mockResolvedValue(null);
    const req = httpMocks.createRequest({ method: 'POST', params: { id: 'p1' }, user: {
id: 'u1' } });
    const res = httpMocks.createResponse();

    await expect(createTaskForProject(req, res)).rejects.toThrow();
    expect(res.statusCode).toBe(404);
  });

  it('createTaskForProject: 404 when not owner nor member', async () => {
    Project.findById.mockResolvedValue({ _id: 'p1', user: 'owner' });
    ProjectMember.exists.mockResolvedValue(false);
    const req = httpMocks.createRequest({ method: 'POST', params: { id: 'p1' }, user: {
_id: 'u1', id: 'u1' }, body: { title: 't' } });
    const res = httpMocks.createResponse();

    await expect(createTaskForProject(req, res)).rejects.toThrow();
    expect(res.statusCode).toBe(404);
  });

  it('createTaskForProject: success with dueInDays and webhook error', async () => {
    process.env.N8N_TASK_WEBHOOK_URL = 'http://example.com/task-webhook';
    const project = { _id: 'p1', user: 'u1', name: 'proj' };
    Project.findById.mockResolvedValue(project);
    ProjectMember.exists.mockResolvedValue(false);
    Task.create.mockResolvedValue({ _id: 't1', title: 't1' });
    axios.post.mockRejectedValue(new Error('webhook fail'));
```

```
    Task.findById.mockImplementation(() => ({ populate:
jest.fn().mockResolvedValue({ _id: 't1', title: 't1' }) }));

    const req = httpMocks.createRequest({ method: 'POST', params: { id: 'p1' }, user: {
id: 'u1', _id: 'u1', email: 'u@x', name: 'n' }, body: { title: 'Title', dueInDays: '3' } });
    const res = httpMocks.createResponse();

    await createTaskForProject(req, res);
    expect(res.statusCode).toBe(201);
  });

  it('addProjectLink: validates url and creates link', async () => {
    const project = { _id: 'p1', user: 'u1' };
    Project.findById.mockResolvedValue(project);
    ProjectMember.exists.mockResolvedValue(true);
    ProjectResourceLink.create.mockResolvedValue({ _id: 'l1', name: 'n', url:
'https://x', createdAt: new Date() });

    const req = httpMocks.createRequest({ method: 'POST', params: { id: 'p1' }, user: {
_id: 'u1' }, body: { name: 'Link', url: 'https://example.com' } });
    const res = httpMocks.createResponse();

    await addProjectLink(req, res);
    expect(res.statusCode).toBe(201);
    const data = res._getJSONData();
    expect(data.url).toContain('https://');
  });

  it('getProjectById: not found or not authorized', async () => {
    Project.findById.mockImplementation(() => ({ lean:
jest.fn().mockResolvedValue(null) }));
    const req = httpMocks.createRequest({ method: 'GET', params: { id: 'pX' }, user: {
_id: 'u1' } });
    const res = httpMocks.createResponse();
    await expect(getProjectById(req, res)).rejects.toThrow();
    expect(res.statusCode).toBe(404);

    // not owner nor member
    Project.findById.mockImplementation(() => ({ lean: jest.fn().mockResolvedValue({
_id: 'p1', user: 'other' }) }));
    ProjectMember.exists.mockResolvedValue(false);
    const req2 = httpMocks.createRequest({ method: 'GET', params: { id: 'p1' }, user: {
_id: 'u1' } });
    const res2 = httpMocks.createResponse();
    await expect(getProjectById(req2, res2)).rejects.toThrow();
    expect(res2.statusCode).toBe(404);
  });
});
```

projectController.extended.test.js

```
const httpMocks = require('node-mocks-http');

jest.mock('../models/projectModel');
```

```
jest.mock('../models/projectMemberModel');
jest.mock('../models/projectInvitationModel');
jest.mock('../models/userModel');
jest.mock('../models/notificationModel');
jest.mock('../models/projectResourceLinkModel');
jest.mock('../models/projectWorkLinkModel');

const Project = require('../models/projectModel');
const ProjectMember = require('../models/projectMemberModel');
const ProjectInvitation = require('../models/projectInvitationModel');
const User = require('../models/userModel');
const Notification = require('../models/notificationModel');
const ProjectWorkLink = require('../models/projectWorkLinkModel');

const {
  inviteToProject,
  listMyInvitations,
  acceptInvitation,
  declineInvitation,
  listProjectMembers,
  removeProjectMember,
  getProjectStats,
  listProjectInvitations,
  cancelProjectInvitation,
  getProjectWorkLink,
  setProjectWorkLink,
} = require('../controllers/projectController');

// Helpers to mock mongoose chainable query methods like .populate().sort().lean()
function mockFindResolve(fn, value) {
  fn.mockImplementation(() => {
    const chain = {
      populate: function () { return chain; },
      sort: function () { return chain; },
      lean: function () { return Promise.resolve(value); },
    };
    return chain;
  });
}

function mockFindOneResolve(fn, value) {
  fn.mockImplementation(() => ({
    lean: () => Promise.resolve(value),
  }));
}

describe('projectController extended', () => {
  beforeEach(() => jest.clearAllMocks());

  it('inviteToProject: handles missing project and various failures', async () => {
    Project.findById.mockResolvedValue(null);
    const req = httpMocks.createRequest({ method: 'POST', params: { id: 'p1' }, user: {
_id: 'u1', id: 'u1', name: 'N', email: 'n@x' }, body: { email: 'a@b.com' } });
    const res = httpMocks.createResponse();
    await expect(inviteToProject(req, res)).rejects.toThrow();
```

```
    expect(res.statusCode).toBe(404);

    // project exists but not owner
    Project.findById.mockResolvedValue({ _id: 'p1', user: 'other' });
    const req2 = httpMocks.createRequest({ method: 'POST', params: { id: 'p1' }, user:
{ _id: 'u1', id: 'u1' }, body: { email: 'a@b.com' } });
    const res2 = httpMocks.createResponse();
    await expect(inviteToProject(req2, res2)).rejects.toThrow();
    expect(res2.statusCode).toBe(403);
  });

  it('inviteToProject: success path', async () => {
    const project = { _id: 'p1', user: 'u1', name: 'Proj' };
    Project.findById.mockResolvedValue(project);
    User.findOne.mockResolvedValue({ _id: 'invitee', email: 'a@b.com' });
    ProjectMember.exists.mockResolvedValue(false);
    mockFindOneResolve(ProjectInvitation.findOne, null);
    ProjectInvitation.create.mockResolvedValue({ _id: 'inv1' });
    Notification.create.mockResolvedValue({});

    const req = httpMocks.createRequest({ method: 'POST', params: { id: 'p1' }, user: {
_id: 'u1', id: 'u1', name: 'Owner', email: 'owner@x' }, body: { email: 'a@b.com' } });
    const res = httpMocks.createResponse();

    await inviteToProject(req, res);
    expect(res.statusCode).toBe(201);
    const data = res._getJSONData();
    expect(data.ok).toBe(true);
  });

  it('listMyInvitations returns list', async () => {
    mockFindResolve(ProjectInvitation.find, [{ _id: 'inv1' }]);
    const req = httpMocks.createRequest({ method: 'GET', user: { _id: 'u1', email:
'u@x' } });
    const res = httpMocks.createResponse();
    await listMyInvitations(req, res);
    expect(res.statusCode).toBe(200);
    expect(Array.isArray(res._getJSONData())).toBe(true);
  });

  it('acceptInvitation and declineInvitation handle not found and success', async () =>
{
    ProjectInvitation.findById.mockResolvedValue(null);
    const req = httpMocks.createRequest({ method: 'POST', params: { id: 'i1' }, user: {
_id: 'u1', email: 'u@x' } });
    const res = httpMocks.createResponse();
    await expect(acceptInvitation(req, res)).rejects.toThrow();
    expect(res.statusCode).toBe(404);

    // success accept
    const inv = { _id: 'i2', status: 'pending', inviteeUser: 'u1', project: 'p1', save:
jest.fn().mockResolvedValue(true) };
    ProjectInvitation.findById.mockResolvedValue(inv);
    ProjectMember.updateOne.mockResolvedValue({});
    Notification.create.mockResolvedValue({});
```

```
    const req2 = httpMocks.createRequest({ method: 'POST', params: { id: 'i2' }, user:
{ _id: 'u1', email: 'u@x', name: 'Me' } });
    const res2 = httpMocks.createResponse();
    await acceptInvitation(req2, res2);
    expect(res2.statusCode).toBe(200);
    expect(res2._getJSONData().ok).toBe(true);

    // decline not found
    ProjectInvitation.findById.mockResolvedValue(null);
    const req3 = httpMocks.createRequest({ method: 'POST', params: { id: 'no' }, user:
{ _id: 'u1', email: 'u@x' } });
    const res3 = httpMocks.createResponse();
    await expect(declineInvitation(req3, res3)).rejects.toThrow();
    expect(res3.statusCode).toBe(404);

    // decline success
    const inv2 = { _id: 'i3', status: 'pending', inviteeUser: 'u1', save:
jest.fn().mockResolvedValue(true), inviter: 'owner' };
    ProjectInvitation.findById.mockResolvedValue(inv2);
    Notification.create.mockResolvedValue({});
    const req4 = httpMocks.createRequest({ method: 'POST', params: { id: 'i3' }, user:
{ _id: 'u1', email: 'u@x' } });
    const res4 = httpMocks.createResponse();
    await declineInvitation(req4, res4);
    expect(res4.statusCode).toBe(200);
  });

  it('listProjectMembers returns owner + members', async () => {
    const project = { _id: 'p1', user: 'ownerId' };
    Project.findById.mockResolvedValue(project);
    ProjectMember.exists.mockResolvedValue(true);
    mockFindResolve(ProjectMember.find, [{ _id: 'm1', user: 'mem1' }]);
    User.findById.mockImplementation(() => ({ select: () => ({ lean: () =>
Promise.resolve({ _id: 'ownerId', name: 'Owner', email: 'owner@x', avatarUrl: '' }) })
}));

    const req = httpMocks.createRequest({ method: 'GET', params: { id: 'p1' }, user: {
_id: 'ownerId' } });
    const res = httpMocks.createResponse();
    await listProjectMembers(req, res);
    expect(res.statusCode).toBe(200);
    const data = res._getJSONData();
    expect(Array.isArray(data)).toBe(true);
  });

  it('removeProjectMember prevents removing owner and allows removing member',
async () => {
    const project = { _id: 'p1', user: 'ownerId', name: 'Proj' };
    Project.findById.mockResolvedValue(project);

    // attempt to remove owner
    const req = httpMocks.createRequest({ method: 'DELETE', params: { id: 'p1',
userId: 'ownerId' }, user: { _id: 'ownerId' } });
    const res = httpMocks.createResponse();
    await expect(removeProjectMember(req, res)).rejects.toThrow();
```

```
    expect(res.statusCode).toBe(400);

    // success remove
    const req2 = httpMocks.createRequest({ method: 'DELETE', params: { id: 'p1',
userId: 'mem1' }, user: { _id: 'ownerId' } });
    const res2 = httpMocks.createResponse();
    ProjectMember.deleteOne.mockResolvedValue({});
    Notification.create.mockResolvedValue({});
    await removeProjectMember(req2, res2);
    expect(res2.statusCode).toBe(204);
  });

  it('getProjectStats returns members and pending when owner', async () => {
    const project = { _id: 'p1', user: 'ownerId' };
    Project.findById.mockResolvedValue(project);
    ProjectMember.exists.mockResolvedValue(true);
    ProjectMember.countDocuments.mockResolvedValue(2);
    ProjectInvitation.countDocuments.mockResolvedValue(1);

    const req = httpMocks.createRequest({ method: 'GET', params: { id: 'p1' }, user: {
_id: 'ownerId' } });
    const res = httpMocks.createResponse();
    await getProjectStats(req, res);
    expect(res.statusCode).toBe(200);
    const data = res._getJSONData();
    expect(data.members).toBe(3);
  });

  it('listProjectInvitations and cancelProjectInvitation flows', async () => {
    const project = { _id: 'p1', user: 'ownerId' };
    Project.findById.mockResolvedValue(project);
    mockFindResolve(ProjectInvitation.find, [{ _id: 'inv1' }]);

    const req = httpMocks.createRequest({ method: 'GET', params: { id: 'p1' }, user: {
_id: 'ownerId' } });
    const res = httpMocks.createResponse();
    await listProjectInvitations(req, res);
    expect(res.statusCode).toBe(200);

    // cancel not found
    ProjectInvitation.findById.mockResolvedValue(null);
    const req2 = httpMocks.createRequest({ method: 'DELETE', params: { id: 'no' },
user: { _id: 'ownerId' } });
    const res2 = httpMocks.createResponse();
    await expect(cancelProjectInvitation(req2, res2)).rejects.toThrow();
    expect(res2.statusCode).toBe(404);

    // cancel success
    const inv = { _id: 'i1', status: 'pending', project: 'p1', save:
jest.fn().mockResolvedValue(true), inviteeUser: 'mem1' };
    ProjectInvitation.findById.mockResolvedValue(inv);
    Project.findById.mockImplementation(() => ({ select: () => Promise.resolve({ _id:
'p1', user: 'ownerId', name: 'Proj' }) }));
    Notification.create.mockResolvedValue({});
```

```
    const req3 = httpMocks.createRequest({ method: 'DELETE', params: { id: 'i1' },
user: { _id: 'ownerId' } });
    const res3 = httpMocks.createResponse();
    await cancelProjectInvitation(req3, res3);
    expect(res3.statusCode).toBe(204);
  });

  it('getProjectWorkLink and setProjectWorkLink validate and return', async () => {
    const project = { _id: 'p1', user: 'ownerId' };
    Project.findById.mockResolvedValue(project);

    // get empty
    const req = httpMocks.createRequest({ method: 'GET', params: { id: 'p1' }, user: {
_id: 'ownerId' } });
    const res = httpMocks.createResponse();
    mockFindOneResolve(ProjectWorkLink.findOne, null);
    await getProjectWorkLink(req, res);
    expect(res.statusCode).toBe(200);
    expect(res._getJSONData().url).toBe('');

    // set invalid url
    const req2 = httpMocks.createRequest({ method: 'PUT', params: { id: 'p1' }, user: {
_id: 'ownerId' }, body: { url: 'ftp://x' } });
    const res2 = httpMocks.createResponse();
    await expect(setProjectWorkLink(req2, res2)).rejects.toThrow();
    expect(res2.statusCode).toBe(400);

    // set valid
    const req3 = httpMocks.createRequest({ method: 'PUT', params: { id: 'p1' }, user: {
_id: 'ownerId' }, body: { url: 'https://ok.com' } });
    const res3 = httpMocks.createResponse();
    ProjectWorkLink.findOneAndUpdate.mockImplementation(() => ({ lean: () =>
Promise.resolve({ url: 'https://ok.com' }) }));
    await setProjectWorkLink(req3, res3);
    expect(res3.statusCode).toBe(200);
    expect(res3._getJSONData().url).toBe('https://ok.com');
  });
});
```

notificationController.unit.test.js

```
const httpMocks = require('node-mocks-http');

jest.mock('../models/notificationModel');
const Notification = require('../models/notificationModel');

const { listMyNotifications, markRead, markAllRead, unreadCount } =
require('../controllers/notificationController');

describe('notificationController', () => {
  beforeEach(() => jest.clearAllMocks());
```

```
  it('listMyNotifications returns items', async () => {
    Notification.find.mockImplementation(() => ({
      sort: () => ({
        limit: () => Promise.resolve([{ _id: 'n1', text: 'hello' }]),
      }),
    }));

    const req = httpMocks.createRequest({ method: 'GET', user: { _id: 'u1' } });
    const res = httpMocks.createResponse();

    await listMyNotifications(req, res);
    expect(res.statusCode).toBe(200);
    const data = res._getJSONData();
    expect(Array.isArray(data)).toBe(true);
    expect(data[0].text).toBe('hello');
  });

  it('markRead returns 404 when not found and success when found', async () => {
    Notification.findOne.mockResolvedValue(null);
    const req = httpMocks.createRequest({ method: 'POST', params: { id: 'no' }, user: {
_id: 'u1' } });
    const res = httpMocks.createResponse();
    await expect(markRead(req, res)).rejects.toThrow();
    expect(res.statusCode).toBe(404);

    const doc = { _id: 'n1', read: false, save: jest.fn().mockResolvedValue(true) };
    Notification.findOne.mockResolvedValue(doc);
    const req2 = httpMocks.createRequest({ method: 'POST', params: { id: 'n1' }, user:
{ _id: 'u1' } });
    const res2 = httpMocks.createResponse();
    await markRead(req2, res2);
    expect(res2.statusCode).toBe(200);
    expect(res2._getJSONData().ok).toBe(true);
    expect(doc.save).toHaveBeenCalled();
  });

  it('markAllRead calls updateMany and returns ok', async () => {
    Notification.updateMany.mockResolvedValue({ nModified: 2 });
    const req = httpMocks.createRequest({ method: 'POST', user: { _id: 'u1' } });
    const res = httpMocks.createResponse();
    await markAllRead(req, res);
    expect(res.statusCode).toBe(200);
    expect(res._getJSONData().ok).toBe(true);
    expect(Notification.updateMany).toHaveBeenCalledWith({ user: 'u1', read: false },
{ $set: { read: true } });
  });

  it('unreadCount returns the count', async () => {
    Notification.countDocuments.mockResolvedValue(5);
    const req = httpMocks.createRequest({ method: 'GET', user: { _id: 'u1' } });
    const res = httpMocks.createResponse();
    await unreadCount(req, res);
    expect(res.statusCode).toBe(200);
    expect(res._getJSONData().count).toBe(5);
  });
```

```
});
```

libraryController.unit.test.js

```
const httpMocks = require('node-mocks-http');
jest.mock('dropbox', () => ({
  Dropbox: function () {
    return {
      filesUpload: jest.fn(async () => ({ result: { path_lower: '/proyectifyia/file.pdf' } })),
      sharingCreateSharedLinkWithSettings: jest.fn(async () => ({ result: { url:
'https://dropbox.com/s/link?dl=0' } })),
    };
  }
}));

jest.mock('../models/libraryItemModel');
// Mockeamos también Notification para evitar validaciones de mongoose en unit
tests
jest.mock('../models/notificationModel', () => ({ create:
jest.fn().mockResolvedValue({}) }));
const LibraryItem = require('../models/libraryItemModel');
const controller = require('../controllers/libraryController');

describe('libraryController unit tests', () => {
  beforeEach(() => {
    jest.clearAllMocks();
    process.env.DROPBOX_ACCESS_TOKEN = 'fake-token';
  });

  test('getLibraryItems - success with search', async () => {
  const fakeItems = [{ title: 'A' }, { title: 'B' }];
  // Mock que soporte la cadena .sort().skip().limit().select().lean().maxTimeMS()
  LibraryItem.find.mockImplementation(() => ({
    sort: () => ({
      skip: () => ({
        limit: () => ({
          select: () => ({
            lean: () => ({
              maxTimeMS: () => Promise.resolve(fakeItems),
            }),
          }),
        }),
      }),
    }),
  }));
    LibraryItem.countDocuments.mockImplementation(() => ({ maxTimeMS: () =>
Promise.resolve(2) }));

    const req = httpMocks.createRequest({ method: 'GET', user: { id: 'u1' }, query: {
search: 'A' } });
    const res = httpMocks.createResponse();
    await controller.getLibraryItems(req, res);
    expect(res.statusCode).toBe(200);
    const data = res._getJSONData();
```

```
    // El controlador devuelve { items, total, page, limit }
    expect(Array.isArray(data.items)).toBe(true);
  });

  test('uploadLibraryItem - pdf success', async () => {
    const created = { _id: 'li1', title: 'file' };
    LibraryItem.create.mockResolvedValue(created);

    const req = httpMocks.createRequest({
      method: 'POST',
      user: { id: 'u1' },
      body: { summary: 's', tags: 't1,t2', itemType: 'pdf' },
      file: { originalname: 'doc.pdf', buffer: Buffer.from('PDF') }
    });
    const res = httpMocks.createResponse();
    await controller.uploadLibraryItem(req, res);
    expect(res.statusCode).toBe(201);
    const data = res._getJSONData();
    expect(data).toHaveProperty('_id');
  });

  test('uploadLibraryItem - no file -> throw', async () => {
    const req = httpMocks.createRequest({ method: 'POST', user: { id: 'u1' }, body: {
itemType: 'pdf' } });
    const res = httpMocks.createResponse();
    await expect(controller.uploadLibraryItem(req, res)).rejects.toThrow('No se ha
subido ningún archivo PDF.');
  });

  test('saveSuggestedArticle - success and duplicate', async () => {
    LibraryItem.findOne.mockResolvedValue(null);
    LibraryItem.create.mockResolvedValue({ _id: 'li2', title: 'X' });
    const req = httpMocks.createRequest({ method: 'POST', user: { id: 'u1' }, body: {
title: 'T', link: 'L', summary: 'S', resultId: 'r1' } });
    const res = httpMocks.createResponse();
    await controller.saveSuggestedArticle(req, res);
    expect(res.statusCode).toBe(201);

    // Duplicate
    LibraryItem.findOne.mockResolvedValue({ _id: 'li2' });
    const req2 = httpMocks.createRequest({ method: 'POST', user: { id: 'u1' }, body: {
title: 'T', link: 'L', summary: 'S', resultId: 'r1' } });
    const res2 = httpMocks.createResponse();
    await expect(controller.saveSuggestedArticle(req2, res2)).rejects.toThrow('Este
artículo ya está en tu biblioteca.');
  });

  test('deleteLibraryItem - not found and unauthorized and success', async () => {
    LibraryItem.findById.mockResolvedValue(null);
    const req = httpMocks.createRequest({ method: 'DELETE', user: { id: 'u1' },
params: { id: 'x' } });
    const res = httpMocks.createResponse();
    await expect(controller.deleteLibraryItem(req, res)).rejects.toThrow('Elemento no
encontrado');
```

```
    // Unauthorized
    LibraryItem.findById.mockResolvedValue({ _id: 'i1', user: 'other', deleteOne:
jest.fn() });
    const req2 = httpMocks.createRequest({ method: 'DELETE', user: { id: 'u1' },
params: { id: 'i1' } });
    const res2 = httpMocks.createResponse();
    await expect(controller.deleteLibraryItem(req2, res2)).rejects.toThrow('Usuario no
autorizado');

    // Success
    const delMock = jest.fn().mockResolvedValue(true);
    LibraryItem.findById.mockResolvedValue({ _id: 'i2', user: 'u1', deleteOne: delMock
});
    const req3 = httpMocks.createRequest({ method: 'DELETE', user: { id: 'u1' },
params: { id: 'i2' } });
    const res3 = httpMocks.createResponse();
    await controller.deleteLibraryItem(req3, res3);
    expect(res3.statusCode).toBe(200);
  });
});
```

avatarController.unit.test.js

```
const httpMocks = require('node-mocks-http');
const fs = require('fs');

jest.mock('../models/userModel');
const User = require('../models/userModel');

const { updateAvatar } = require('../controllers/avatarController');

describe('avatarController - updateAvatar', () => {
  beforeEach(() => {
    jest.clearAllMocks();
  });

  it('returns 400 if no file provided', async () => {
    const req = httpMocks.createRequest({ method: 'PUT', user: { _id: 'u1' } });
    const res = httpMocks.createResponse();

    await expect(updateAvatar(req, res)).rejects.toThrow();
    expect(res.statusCode).toBe(400);
  });

  it('returns 400 if file not image', async () => {
    const req = httpMocks.createRequest({
      method: 'PUT',
      user: { _id: 'u1' },
      file: { mimetype: 'text/plain', size: 100 }
    });
    const res = httpMocks.createResponse();

    await expect(updateAvatar(req, res)).rejects.toThrow();
    expect(res.statusCode).toBe(400);
```

```
  });

  it('returns 400 if file too large', async () => {
    const req = httpMocks.createRequest({
      method: 'PUT',
      user: { _id: 'u1' },
      file: { mimetype: 'image/png', size: 3 * 1024 * 1024 }
    });
    const res = httpMocks.createResponse();

    await expect(updateAvatar(req, res)).rejects.toThrow();
    expect(res.statusCode).toBe(400);
  });

  it('writes file and updates user avatarUrl on success', async () => {
    // Mock fs.promises
    jest.spyOn(fs.promises, 'mkdir').mockResolvedValue();
    jest.spyOn(fs.promises, 'writeFile').mockResolvedValue();

    User.updateOne.mockResolvedValue({ acknowledged: true });

    const req = httpMocks.createRequest({
      method: 'PUT',
      protocol: 'http',
      get: () => 'localhost:3000',
      user: { _id: 'u1' },
      file: { mimetype: 'image/jpeg', size: 1024, buffer: Buffer.from('abc') }
    });
    const res = httpMocks.createResponse();

    await updateAvatar(req, res);

    expect(res.statusCode).toBe(200);
    const data = res._getJSONData();
    expect(data).toHaveProperty('avatarUrl');

    fs.promises.mkdir.mockRestore();
    fs.promises.writeFile.mockRestore();
  });
});
```

aiController.unit.test.js

```
const httpMocks = require('node-mocks-http');
const aiController = require('../controllers/aiController');

jest.mock('@google/genai', () => ({
  GoogleGenAI: function () {
    return {
      models: {
        generateContent: jest.fn(async () => ({ text: 'Resumen de prueba' }))
      }
    };
  }
```

```
}));

const axios = require('axios');
jest.mock('axios');

describe('aiController unit tests', () => {
  beforeEach(() => jest.clearAllMocks());

  beforeAll(() => {
    process.env.SERPAPI_API_KEY = 'fake-key';
  });

  test('summarizeText - éxito', async () => {
    const req = httpMocks.createRequest({ method: 'POST', body: { text: 'Esto es un
texto largo' } });
    const res = httpMocks.createResponse();
    await aiController.summarizeText(req, res);
    expect(res.statusCode).toBe(200);
    const data = res._getJSONData();
    expect(data).toHaveProperty('summary');
  });

  test('summarizeText - body vacío -> error (400)', async () => {
    const req = httpMocks.createRequest({ method: 'POST', body: {} });
    const res = httpMocks.createResponse();
    await aiController.summarizeText(req, res);
    expect(res.statusCode).toBe(400);
    const data = res._getJSONData();
    expect(data).toHaveProperty('message');
  });

  test('suggestArticles - SerpAPI éxito', async () => {
    axios.get.mockResolvedValue({ data: { organic_results: [{ title: 'Art 1', link: 'http://a'
}] } });
    const req = httpMocks.createRequest({ method: 'POST', body: { query: 'test' } });
    const res = httpMocks.createResponse();
    await aiController.suggestArticles(req, res);
    expect(res.statusCode).toBe(200);
    const data = res._getJSONData();
    expect(Array.isArray(data.results)).toBe(true);
  });
});
```

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();
  });
});
```