Informe de Pruebas y Cobertura

Proyecto: backend

Fecha: 2025-10-06

Puntos a abordar:

1. **Cobertura superior al 70%**

   - Objetivo: Asegurar que las métricas críticas de cobertura del backend superen el 70% para statements y lines.
   - Acciones realizadas: se añadieron pruebas enfocadas y se ejecutaron ajustes de configuración de Jest (reporters, thresholds y testTimeout) para estabilizar ejecuciones y medir correctamente.
   - Evidencia (ejecución local): Statements 80.95%, Branches 71.17%, Functions 75.60%, Lines 84.59%.
   - Conclusión: La cobertura global del backend supera de forma holgada el umbral del 70% en statements y lines; branches y functions también quedan por encima del 70% tras las pruebas añadidas.

2. **Casos de prueba exhaustivos que cubren edge cases**

- Objetivo: Probar rutas, validaciones y flujos alternativos para reducir riesgo en producción.
- Ejemplos concretos de edge cases cubiertos:
  - Validaciones de entrada: llamadas a endpoints AI con body ausente o campos vacíos (p.ej. summarizeText sin text, handleChat sin message) devuelven 400 y mensajes claros.
  - Reintentos y errores transitorios: se probó la lógica de retry para llamadas a la API de generación (simulando RESOURCE_EXHAUSTED / 429) verificando que el código reintenta y finalmente responde correctamente cuando hay recuperación.
  - Fallbacks de parsing: se validó la función de extracción de texto extractText en distintas formas de respuesta (propiedad text, estructura candidates[].content.parts, ausencia total), garantizando retorno tolerante y sin excepciones.
  - Errores externos de terceros: se simuló error de SerpAPI (payload con error) y se comprobó que el controlador responde con 502 y mensaje indicativo; también se cubrieron respuestas sin resultados (lista vacía).
  - Manejo de tokens y auth: pruebas del middleware de autenticación cubren token ausente y token malformado (genera 401 y el flujo de error adecuado).

    - Error global: el errorMiddleware fue probado para asegurar que devuelve JSON con message y stack (en entornos no producción) y respeta el status code ya presente en res.

- Alcance: las pruebas incluyen tanto happy paths como condiciones de fallo, inputs inválidos, respuestas vacías y errores externos, proporcionando una cobertura práctica de los casos frontera más relevantes para estos controladores.

3. Mocking sofisticado de servicios complejos

- Objetivo: Aislar lógica de negocio de dependencias externas (Google GenAI, SerpAPI, DB en memoria) para pruebas deterministas y rápidas.
- Técnicas y herramientas usadas:
    - Mock programático de la librería @google/genai: se sustituyó la implementación concreta por un mock que permite definir por test el comportamiento de models.generateContent (resoluciones, rechazos con códigos como RESOURCE_EXHAUSTED) para probar retry, timeouts y parsing de respuestas.
    - Mock de HTTP externo con axios: se espiaron/resolvieron llamadas a axios.get para simular diferentes cuerpos de SerpAPI (errores, resultados con recursos PDF, resultados vacíos), evitando llamadas de red reales.
    - Instancias controladas de MongoDB en memoria: se usó mongodb-memory-server en tests de integración; se añadieron timeouts y teardown robusto (try/catch en disconnect/stop) para evitar condiciones de carrera y errores EPERM en CI/local.
    - Utilidad node-mocks-http: construcción de req/res falsos para invocar controladores de forma aislada y leer respuestas JSON sin necesidad de levantar el servidor HTTP completo.
    - Patrones de mocking usados: secuencias de comportamiento (mockRejectedValueOnce -> mockResolvedValueOnce) para simular throttling/retry; inyección de instancias mocked en el punto donde el módulo crea su cliente (GoogleGenAI) para que el código bajo prueba use el mock desde el inicio.
- Beneficio: las pruebas son deterministas, reproducibles en CI y permiten verificar ramas de error y recuperación sin depender de cuota/latencia/credenciales reales.

Resumen compacto final:

- Se alcanzó el objetivo de cobertura (>70%) con métricas de statements y lines significativamente por encima del umbral.
- Se añadieron pruebas que ejercitan casos frontera relevantes (validaciones, reintentos, parsing alternativo y errores de terceros).

- El uso de mocks avanzados para @google/genai, axios y la gestión controlada de MongoDB en memoria asegura pruebas rápidas y confiables en entornos locales y CI.

ANEXO:

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	80.95	71.17	75.6	84.59	
backend	72.41	66.66	0	80.76	
server.js	72.41	66.66	0	80.76	19-20,29,36-37
backend/config	75	100	100	75	
db.js	75	100	100	75	9-10
...nd/controllers	77.92	70.87	84.84	81.75	
aiController.js	80.53	68.37	87.5	80.41	...52,168,206-209
...Controller.js	87.87	77.41	100	87.87	...,92-98,105-106
...Controller.js	67.79	68.96	66.66	81.63	8-9,35-39,56,67
...Controller.js	62.5	68.75	66.66	68.75	13-14,23,28,43-58
...Controller.js	86.66	84.61	100	86.66	31-32,55-56
...end/middleware	100	80	100	100	
...Middleware.js	100	100	100	100	
...Middleware.js	100	50	100	100	5-20
backend/models	100	100	100	100	
...yItemModel.js	100	100	100	100	
projectModel.js	100	100	100	100	
taskModel.js	100	100	100	100	
userModel.js	100	100	100	100	
backend/routes	94.44	100	0	94.44	
aiRoutes.js	100	100	100	100	
libraryRoutes.js	85.71	100	0	85.71	6-7
projectRoutes.js	100	100	100	100	
userRoutes.js	100	100	100	100	

```
=============================== Coverage summary ===============================
Statements   : 80.95% ( 340/420 )
Branches     : 71.17% ( 158/222 )
Functions    : 75.6% ( 31/41 )
Lines        : 84.59% ( 324/383 )
================================================================================

Test Suites: 10 passed, 10 total
Tests:       44 passed, 44 total
Snapshots:   0 total
Time:        58.08 s, estimated 67 s
Ran all test suites.
```