Informe de Pruebas Unitarias y de Integración – Backend

Fecha: 2025-10-11
Proyecto: Sistema de Análisis de Texto con IA

Resumen Ejecutivo

Este documento presenta el análisis completo de las suites de pruebas implementadas para el backend del sistema. Se han desarrollado pruebas unitarias y de integración que cubren los módulos críticos de autenticación, gestión de textos, análisis con IA y middleware de seguridad.

1. **Configuración de Entorno de Pruebas**
   Estrategia de Base de Datos
   - **Base de datos de prueba:** MongoDB en localhost:27017 con bases dedicadas (testdb, testdb-user, testdb-text)
   - **Limpieza de datos:** Eliminación de colecciones después de cada prueba (afterEach)
   - **Gestión de conexiones:** Conexión/desconexión controlada en beforeAll/afterAll

   Scripts de Ejecución

```
{
  "test": "jest --coverage --runInBand",
  "test:ci": "jest --coverage --runInBand",
  "test:unit": "jest --runInBand --testPathPattern=tests",
  "test:integration": "jest --runInBand --testPathPattern=tests.integration"
}
```

2. **Análisis por Módulo de Pruebas**
   a. **Módulo de Autenticación (auth.test.js)**
      Cobertura: Registro y login de usuarios
      Casos probados:
      - Registro exitoso de nuevo usuario (201 Created)
      - Prevención de duplicados por correo (409 Conflict)
      - Login exitoso con credenciales válidas (200 OK + token)
      - Rechazo de login con contraseña incorrecta (401 Unauthorized)

      Patrones implementados:
      - Base de datos limpia entre pruebas

- Usuarios de prueba autocontenidos
- Verificación de respuestas HTTP y estructura de datos

b. **Middleware de Autenticación (authMiddleware.test.js)**
   Cobertura: Validación de tokens JWT
   Casos probados:
   - Rechazo de peticiones sin token (400 Bad Request)
   - Rechazo de tokens inválidos/malformados (400 Bad Request)
   - Aprobación de tokens válidos (llamada a next())
   - Adjuntado de datos de usuario a req.user

   Técnicas avanzadas:

   - Mock de objetos request y response de Express
   - Verificación de llamadas a funciones next()

c. **Gestión de Textos (text.test.js)**
   Cobertura: Guardado y upload de textos
   Endpoints probados:
   - **POST /api/save-text**
     - Guardado exitoso con texto válido (201 Created)
     - Rechazo de texto vacío (400 Bad Request)
   - **POST /api/upload**
     - Procesamiento exitoso de archivos .txt (201 Created)
     - Rechazo de peticiones sin archivo (400 Bad Request)
     - Rechazo de tipos de archivo no .txt (400 Bad Request)

d. **Análisis con IA (ai.test.js)**
   Cobertura: Generación de preguntas y detección de falacias
   Mocking estratégico:
   - Funciones de IA mockeadas para evitar llamadas reales
   - Respuestas simuladas controladas para cada caso

   Casos probados:

   - **GET /api/analyze/:textId/questions**
     - Generación exitosa de preguntas (200 OK)
     - Manejo de texto no encontrado (404 Not Found)
   - **GET /api/analyze/:textId/fallacies**
     - Detección exitosa de falacias (200 OK)
     - Manejo de texto no encontrado (404 Not Found)

e. **Gestión de Usuarios (getUserById.test.js)**
   Cobertura: Consulta de usuarios por ID
   Casos de borde cubiertos:
   - Usuario existente (200 OK + datos sin contraseña)

- Usuario no existente (404 Not Found)
- ID con formato inválido (400 Bad Request)
- Error de servidor (500 Internal Server Error)

f. **Conexión a Base de Datos (db.test.js)**
Cobertura: Conexión y manejo de errores de MongoDB
Técnicas de testing:
- Mock completo del módulo mongoose
- Spy en console.log y console.error
- Simulación de respuestas exitosas y fallidas

3. **Métricas de Calidad**
Cobertura de Código (Estimada)
- Statements: >80%
- Branches: >70%
- Functions: >75%
- Lines: >80%

Thresholds Configurados

```
coverageThreshold: {
  global: {
    statements: 70,
    branches: 70,
    functions: 70,
    lines: 70
  }
}
```

4. **Integración CI/CD**
Workflow GitHub Actions
Estructura de jobs:
- backend-tests: Ejecución en Node.js 18 con npm run test:ci
- frontend-tests: Suite paralela para frontend

Triggers: Push y PR sobre ramas main/master

Artifacts: Carpeta coverage disponible para descarga

5. **Recomendaciones y Mejoras Futuras**

   Inmediatas (High Priority)

   - Configurar thresholds específicos por módulo para componentes críticos
   - Implementar reportes LCOV para integración con Codecov/Coveralls
   - Añadir pruebas de estrés para endpoints de IA con alto volumen

   Mediano Plazo (Medium Priority)

   - Mock de servicios externos más robusto (Google GenAI, SerpAPI)
   - Pruebas de rendimiento para operaciones con textos largos
   - Validación de esquemas de respuesta con JSON Schema

   Largo Plazo (Low Priority)

   - Pruebas de seguridad (inyección, XSS, etc.)
   - Pruebas de carga para endpoints públicos
   - Monitorización continua de cobertura en CI

6. **Conclusiones**

   El sistema de pruebas demuestra una cobertura robusta de los componentes críticos del backend. La arquitectura de pruebas sigue mejores prácticas con:

   - Aislamiento completo entre pruebas
   - Mocking estratégico de dependencias externas
   - Verificación de estados HTTP y estructuras de datos
   - Manejo adecuado de casos de error y bordes
   - Integración fluida con pipeline CI/CD

   La suite actual proporciona una base sólida para desarrollo continuo, permitiendo refactorizaciones con confianza y manteniendo la calidad del código.