Pruebas Unitarias del Backend de Trivia

10 de diciembre de 2025

1. Introducción

Las pruebas unitarias constituyen la primera capa de aseguramiento de calidad del backend desarrollado para el sistema de trivia. Su objetivo es validar el correcto funcionamiento de la lógica de negocio del servidor, sin depender de la base de datos, HTTP ni Socket.io.

Para ello se utilizó el framework **Jest**, el cual permite crear suites de pruebas, aislar módulos mediante *mocks* y verificar el comportamiento esperado de cada componente.

Este documento describe las pruebas unitarias aplicadas a los servicios `AuthService` y `SessionService`, así como una prueba de integración sobre las rutas de autenticación.

2. Objetivos

Los objetivos principales de estas pruebas son:

- Validar la lógica interna de registro e inicio de sesión.
- Verificar la obtención de preguntas asociadas a una sesión.
- Simular modelos de base de datos sin necesidad de conectarse a MySQL.
- Detectar errores de forma temprana y mejorar la mantenibilidad del código.

3. Estructura del entorno de pruebas

Las pruebas se organizan dentro de la siguiente estructura de carpetas del proyecto:
En esta estructura se distinguen claramente las pruebas unitarias de servicios y las pruebas de integración:

Figura 1: Estructura de carpetas para las pruebas del backend.

```
backend/
  tests/
      services/
          AuthService.test.js
          SessionService.test.js
      integration/
          auth.integration.test.js
```

El comando de ejecución configurado en el archivo `package.json` es:

```
"scripts": {
  "test": "cross-env NODE_ENV=test jest --runInBand"
}
```

Gracias a la variable `NODE_ENV=test`, el servidor no intenta conectarse a la base de datos durante las pruebas, permitiendo una ejecución rápida y completamente aislada.

4. Pruebas Unitarias de `AuthService`

El módulo `AuthService` gestiona el registro e inicio de sesión de usuarios. Las pruebas se realizan mediante *mocks* que simulan:

- El modelo de Sequelize (`UserModel`)
- La librería de hash `bcrypt`
- La librería de tokens `jsonwebtoken`

4.1. Mock de dependencias

```
jest.mock('../../src/infrastructure/models/user.model', () => ({
  findOne: jest.fn(),
  create: jest.fn(),
}));

jest.mock('bcrypt');
jest.mock('jsonwebtoken');
```

Esto permite controlar cada posible respuesta del sistema sin acceder a infraestructura real.

4.2. Caso 1: Registro exitoso

Descripción: El correo no existe, se genera un hash simulado, se crea el usuario y el servicio devuelve los datos correctamente formateados.

```
test('register registra un usuario cuando el email no existe', async () => {
  UserModel.findOne.mockResolvedValue(null);
  bcrypt.hash.mockResolvedValue('hash-fake');
  UserModel.create.mockResolvedValue({
    user_id: 1,
    username: 'santi',
    email: 'test@example.com',
  });

  const result = await AuthService.register({
    username: 'santi',
    email: 'test@example.com',
    password: '123456',
  });

  expect(result).toEqual({
    id: 1,
    username: 'santi',
    email: 'test@example.com',
```

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

4.3. Caso 2: Login exitoso

Descripción: Existe un usuario, la contraseña coincide y se genera un token válido.

```
test('login devuelve token y usuario si las credenciales son válidas', async () => {
  UserModel.findOne.mockResolvedValue({
    user_id: 1,
    username: 'santi',
    email: 'test@example.com',
    password_hash: 'hash',
  });

  bcrypt.compare.mockResolvedValue(true);
  jwt.sign.mockReturnValue('fake-jwt-token');

  const result = await AuthService.login({
    username: 'santi',
    password: '123456',
  });

  expect(result.token).toBe('fake-jwt-token');
  expect(result.user.username).toBe('santi');
});
```

4.4. Evidencia gráfica de la ejecución

En la Figura 2 se observa el resultado de la ejecución de las pruebas unitarias sobre `AuthService`.

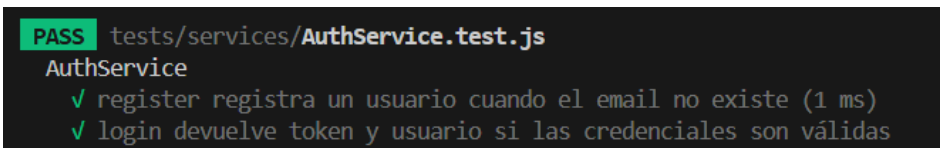

Figura 2: Resultados de las pruebas unitarias de `AuthService`.

5. Pruebas Unitarias de `SessionService`

El servicio `SessionService` maneja la recuperación de preguntas dentro de una sesión activa. Para estas pruebas se simula el modelo `SessionQuestionModel`.

5.1. Mock del modelo

```
jest.mock('../../src/infrastructure/models/sessionQuestion.model', () => ({
  findOne: jest.fn(),
}));
```

5.2. Caso 1: No hay más preguntas

Descripción: Cuando no se encuentra una pregunta en el orden solicitado, el servicio declara la sesión como finalizada.

```
test('devuelve finished=true si no hay pregunta', async () => {
  SessionQuestionModel.findOne.mockResolvedValue(null);

  const result = await SessionService.getNextQuestion(1, 0);

  expect(result).toEqual({ finished: true });
});
```

5.3. Caso 2: Existe una pregunta

Descripción: Cuando sí se encuentra la pregunta, se devuelve su contenido.

```
test('devuelve la pregunta cuando existe', async () => {
  SessionQuestionModel.findOne.mockResolvedValue({
    payload: { question: '2+2?', options: ['3', '4'], correct: '4' },
  });

  const result = await SessionService.getNextQuestion(1, 0);

  expect(result.finished).toBe(false);
  expect(result.question.question).toBe('2+2?');
});
```

5.4. Evidencia gráfica de la ejecución

La Figura 3 muestra la salida de Jest para las pruebas unitarias de `SessionService`.

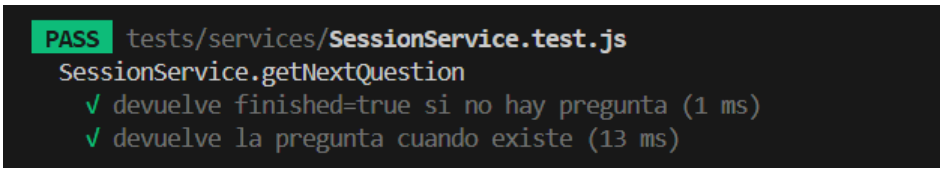

Figura 3: Resultados de las pruebas unitarias de `SessionService`.

6. Prueba de Integración de Rutas de Autenticación

Además de las pruebas unitarias, se implementó una prueba de integración sobre las rutas `/auth/register` y `/auth/login`. Para esto se creó una aplicación de Express de prueba en el archivo `auth.integration.test.js`, montando las rutas reales de autenticación y utilizando `supertest` para realizar las peticiones HTTP.

La evidencia de la ejecución de esta prueba se presenta en la Figura 4.

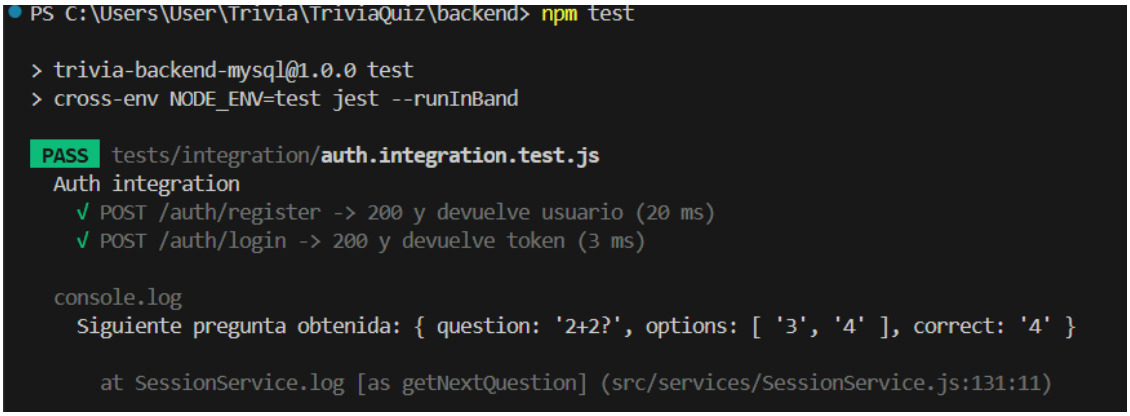

Figura 4: Resultados de la prueba de integración de las rutas de autenticación.

7. Resumen global de la ejecución de pruebas

En la Figura 5 se muestra el resumen general de Jest, donde se evidencia que las tres suites de pruebas (`AuthService`, `SessionService` y la prueba de integración de autenticación) se ejecutan correctamente, con un total de seis pruebas aprobadas.

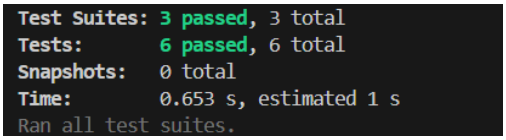

Figura 5: Resumen global de las suites de pruebas ejecutadas con Jest.

8. Conclusiones

Las pruebas unitarias implementadas permiten asegurar el correcto funcionamiento de la lógica de negocio del backend. Entre los beneficios observados:

- Se validan los servicios más críticos del sistema: autenticación y gestión de preguntas de sesión.
- Se aíslan correctamente dependencias externas gracias al uso de *mocks*.
- Se evitan conexiones innecesarias a la base de datos durante las pruebas.
- Se facilita la detección temprana de errores y regresiones.

La prueba de integración complementa este trabajo verificando que las rutas HTTP de autenticación se comporten correctamente de extremo a extremo. En conjunto, estas pruebas constituyen una base sólida para la calidad del backend del sistema de trivia.