UNIVERSIDAD NACIONAL DE COLOMBIA

Taller Testing

Presentado por:
Cristian Felipe Moreno Gómez
✉crmorenogo@unal.edu.co
Diego Alejandro Rojas Reina
✉ drojasre@unal.edu.co
Edinson Sánchez Fuentes
✉ edsanchezf@unal.edu.co
Sebastián Olarte Ramírez
✉ molartera@unal.edu.co

Facultad de Ingeniería
Departamento de Sistemas e Industrial
Ingeniería de Software 1 (2016701) - Grupo 2
2024 – 2

Contenido

1. Introducción

Aun en la actualidad, muchas personas consideran que ensamblar una PC sencilla es una tarea complicada y fuera de su alcance. Sin embargo, esta actividad puede ser una puerta de entrada al fascinante mundo de los computadores de escritorio, ofreciendo la oportunidad de aprender conceptos interesantes sin necesidad de tener conocimientos avanzados en computación.

Frecuentemente, quienes dudan en intentar expresan preocupaciones como el temor de adquirir partes incompatibles entre sí. Por ello, se plantea una solución práctica y educativa: un ensamblador de PC sencillo, diseñado tanto para facilitar el proceso como para fomentar el aprendizaje, eliminando barreras y haciendo que esta experiencia sea accesible para todos.

La aplicación web tiene como objetivo permitir a los usuarios ensamblar computadoras de escritorio de manera sencilla y eficiente. El sitio será accesible para todos los usuarios, pero requerirá autenticación para interactuar con funciones avanzadas como el ensamblaje y la gestión personalizada de configuraciones. También se priorizará el diseño atractivo, el rendimiento óptimo y la facilidad de uso.

Para garantizar que la aplicación funcione de manera estable y confiable, se han implementado pruebas automatizadas que validan el correcto comportamiento de sus componentes clave. Estas pruebas no solo ayudan a detectar posibles errores antes de que afecten a los usuarios, sino que también aseguran que las funcionalidades, como la compatibilidad de hardware y la autenticación, operen sin inconvenientes. De esta manera, se contribuye a una experiencia más fluida y segura, reforzando la confianza en la plataforma.

2. Testing

2.1. Test 1

Nombre	Tipo de prueba	Descripción	Herramienta / Framework
Maicol Sebastian Olarte Ramirez	Prueba unitaria	Se prueba la función createUserHandler, verificando que cree un usuario correctamente y que valide la ausencia de campos requeridos.	Jest, Supertest

Código test:

```js
import request from 'supertest';
import express from 'express';
import { createUserHandler } from '../src/controllers/userController.js';
import { PrismaClient } from '@prisma/client';
import { describe, it, expect, beforeAll, afterAll, afterEach } from '@jest/globals';

const prisma = new PrismaClient();

// Configurar la app express para pruebas
const app = express();
app.use(express.json());
app.post('/api/users', createUserHandler);

describe('User Controller', () => {
  beforeAll(async () => {
    await prisma.$connect();
  });

  afterAll(async () => {
    await prisma.$disconnect();
  });

  afterEach(async () => {
    // Limpia la base de datos después de cada prueba
    await prisma.usuario.deleteMany();
  });

  it('Debería crear un nuevo usuario', async () => {
    const newUser = {
      nombre: 'Test User',
      correo: 'test@example.com',
      contrasena: 'testpassword',
      rol: 'usuario'
    };

    const response = await request(app)
      .post('/api/users')
      .send(newUser);

    expect(response.status).toBe(201);
    expect(response.body.nombre).toBe(newUser.nombre);
    expect(response.body.correo).toBe(newUser.correo);
    expect(response.body.rol).toBe(newUser.rol);

    // Verificar que la contraseña esté encriptada
    const userInDb = await prisma.usuario.findUnique({ where: { correo: newUser.correo } });
    expect(userInDb).toBeTruthy();
    expect(userInDb.contrasena).not.toBe(newUser.contrasena);
  });

  it('Debería devolver un error si faltan campos requeridos', async () => {
    const incompleteUser = {
      nombre: 'Incomplete User',
      // Falta el campo correo y contrasena
    };

    const response = await request(app)
      .post('/api/users')
      .send(incompleteUser);

    expect(response.status).toBe(400); // Cambié a 400 Bad Request
    expect(response.body.error).toBe('Campos requeridos faltantes');
  });
});

```

Resultado:

```
 PASS  test/createUserController.test.js
  User Controller
    √ Debería crear un nuevo usuario (152 ms)
    √ Debería devolver un error si faltan campos requeridos (6 ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        1.227 s, estimated 2 s
Ran all test suites matching /createUserController.test.js/i.
```

2.2. Test 2

Nombre	Tipo de prueba	Descripción	Herramienta / Framework
Cristian Felipe Moreno Gomez	Prueba unitaria	Se prueba la función de autenticación en el controlador de usuarios, verificando que permita el acceso con credenciales correctas y que valide la ausencia de campos obligatorios.	Jest, Supertest

Código test:

```
import request from 'supertest';
import express from 'express';
import { authenticateUserHandler } from '../src/controllers/userController.js';
import { PrismaClient } from '@prisma/client';
import bcrypt from 'bcrypt'; // Importa bcrypt
import { describe, it, expect, beforeAll, afterAll, afterEach } from '@jest/globals';

const prisma = new PrismaClient();

// Configurar la app express para pruebas
const app = express();
app.use(express.json());
app.post('/api/login', authenticateUserHandler);

describe('Authentication Controller', () => {
  beforeAll(async () => {
    await prisma.$connect();
  });

  afterAll(async () => {
    await prisma.$disconnect();
  });

  afterEach(async () => {
    // Limpia la base de datos después de cada prueba
    await prisma.usuario.deleteMany();
  });

  it('Debería autenticar un usuario con credenciales correctas', async () => {
    // Crear un usuario de prueba en la base de datos
    const hashedPassword = await bcrypt.hash('testpassword', 10);
    const testUser = {
      nombre: 'Test User',
      correo: 'test@example.com',
      contrasena: hashedPassword,
      rol: 'usuario'
    };
    await prisma.usuario.create({ data: testUser });

    // Enviar una solicitud de inicio de sesión con las credenciales correctas
    const response = await request(app)
      .post('/api/login')
      .send({ correo: 'test@example.com', contrasena: 'testpassword' });

    expect(response.status).toBe(200);
    expect(response.body.correo).toBe(testUser.correo);
    expect(response.body.rol).toBe(testUser.rol);
  });

  it('Debería devolver un error si las credenciales son incorrectas', async () => {
    // Crear un usuario de prueba en la base de datos
    const hashedPassword = await bcrypt.hash('testpassword', 10);
    const testUser = {
      nombre: 'Test User',
      correo: 'test@example.com',
      contrasena: hashedPassword,
      rol: 'usuario'
    };
    await prisma.usuario.create({ data: testUser });

    // Enviar una solicitud de inicio de sesión con una contraseña incorrecta
    const response = await request(app)
      .post('/api/login')
      .send({ correo: 'test@example.com', contrasena: 'wrongpassword' });

    expect(response.status).toBe(401);
    expect(response.body.error).toBe('Authentication failed');
  });

  it('Debería devolver un error si falta el campo correo', async () => {
    // Enviar una solicitud de inicio de sesión sin el campo correo
    const response = await request(app)
      .post('/api/login')
      .send({ contrasena: 'testpassword' });

    expect(response.status).toBe(400);
    expect(response.body.error).toBe('Campos requeridos faltantes');
  });

  it('Debería devolver un error si falta el campo contrasena', async () => {
    // Enviar una solicitud de inicio de sesión sin el campo contrasena
    const response = await request(app)
      .post('/api/login')
      .send({ correo: 'test@example.com' });

    expect(response.status).toBe(400);
    expect(response.body.error).toBe('Campos requeridos faltantes');
  });
});

```

Resultado:

```
PASS test/loginUserController.test.js
  Authentication Controller
    √ Debería autenticar un usuario con credenciales correctas (200 ms)

    √ Debería devolver un error si las credenciales son incorrectas (164 ms)

    √ Debería devolver un error si falta el campo correo (3 ms)

    √ Debería devolver un error si falta el campo contrasena (3 ms)


Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        1.39 s, estimated 4 s
Ran all test suites matching /loginUserController.test.js/i.
```

2.3. Test 3

Nombre	Tipo de prueba	Descripción	Herramienta / Framework
Edinson Sanchez Fuentes	Prueba unitaria	Se prueba el controlador getMotherBoards, verificando que retorne correctamente una lista de motherboards desde la base de datos.	Jest, Supertest

Código test:

```
1  import request from 'supertest';
2  import express from 'express';
3  import { getMotherBoards } from '../src/controllers/ensambleController.js';
4  import { PrismaClient } from '@prisma/client';
5  import { describe, it, expect, beforeAll, afterAll, afterEach, jest } from '@jest/globals';
6
7  const prisma = new PrismaClient();
8
9  // Configurar la app express para pruebas
10 const app = express();
11 app.get('/api/motherboards', getMotherBoards);
12
13 describe('getMotherBoards', () => {
14   beforeAll(async () => {
15     await prisma.$connect();
16   });
17
18   afterAll(async () => {
19     await prisma.$disconnect();
20   });
21
22   afterEach(async () => {
23     jest.clearAllMocks(); // Limpiar todos los mocks después de cada prueba
24   });
25
26   it('Debería obtener una lista de motherboards', async () => {
27     const response = await request(app).get('/api/motherboards').send();
28     expect(response.status).toBe(200);
29     expect(response.body.length).toBeGreaterThan(0);
30     expect(response.body[0]).toHaveProperty('nombre');
31   });
32 });
33
```

Resultado:

```
$ npm test getMotherBoards.test.js

> backend@1.0.0 test
> jest getMotherBoards.test.js

 PASS  test/getMotherBoards.test.js
  getMotherBoards
    √ Debería obtener una lista de motherboards (65 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        0.945 s, estimated 1 s
Ran all test suites matching /getMotherBoards.test.js/i.
```

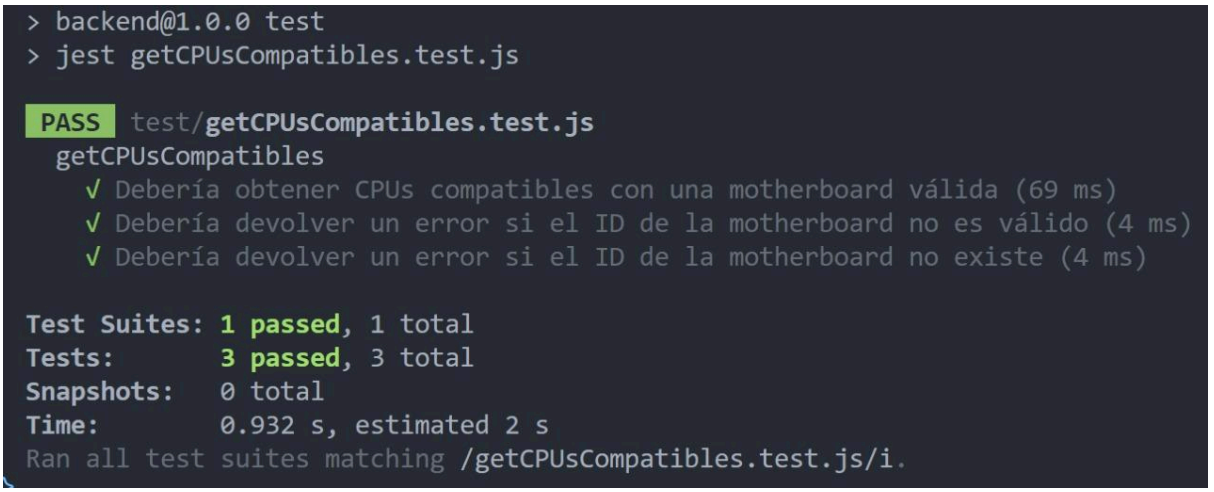
2.4. Test 4

Nombre	Tipo de prueba	Descripción	Herramienta / Framework
Diego Alejandro Rojas Reina	Prueba unitaria	Se prueba el controlador getCPUsCompatibles, que recibe un ID de motherboard y devuelve una lista de CPUs compatibles. Se validan respuestas correctas y errores cuando el ID no es válido o la motherboard no existe.	Jest, Supertest

Código test:

```
/* eslint-disable no-undef */
import request from 'supertest';
import express from 'express';
import { getCPUsCompatibles } from '../src/controllers/ensambleController.js';
import { PrismaClient } from '@prisma/client';
import { describe, it, expect, beforeAll, afterAll, afterEach } from '@jest/globals';

const prisma = new PrismaClient();

// Configurar la app express para pruebas
const app = express();
app.use(express.json());
app.post('/api/cpus-compatibles', getCPUsCompatibles);

describe('getCPUsCompatibles', () => {
  beforeAll(async () => {
    await prisma.$connect();
  });

  afterAll(async () => {
    await prisma.$disconnect();
  });

  afterEach(async () => {
    jest.clearAllMocks(); // Limpiar todos los mocks después de cada prueba
  });

  it('Debería obtener CPUs compatibles con una motherboard válida', async () => {
    const motherboardId = 10122; // Usar un ID válido existente en tu base de datos

    const response = await request(app)
      .post('/api/cpus-compatibles')
      .send({ motherboardId });

    expect(response.status).toBe(200);
    expect(response.body.cpusCompatibles.length).toBeGreaterThan(0); // Suponemos que hay CPUs compatibles
  });

  it('Debería devolver un error si el ID de la motherboard no es válido', async () => {
    const response = await request(app)
      .post('/api/cpus-compatibles')
      .send({ motherboardId: 'invalid-id' });

    expect(response.status).toBe(400);
    expect(response.body.message).toBe('❌ ID de motherboard no válido');
  });

  it('Debería devolver un error si el ID de la motherboard no existe', async () => {
    const motherboardId = 99999; // Usar un ID que no exista en tu base de datos

    const response = await request(app)
      .post('/api/cpus-compatibles')
      .send({ motherboardId });

    expect(response.status).toBe(404);
    expect(response.body.message).toBe('❌ Motherboard no encontrada');
  });
});

```

Resultado:


> backend@1.0.0 test
> jest getCPUsCompatibles.test.js
PASS test/getCPUsCompatibles.test.js
getCPUsCompatibles
√ Debería obtener CPUs compatibles con una motherboard válida (69 ms)
√ Debería devolver un error si el ID de la motherboard no es válido (4 ms)
√ Debería devolver un error si el ID de la motherboard no existe (4 ms)
Test Suites: 1 passed, 1 total
Tests: 3 passed, 3 total
Snapshots: 0 total
Time: 0.932 s, estimated 2 s
Ran all test suites matching /getCPUsCompatibles.test.js/i.

3. Lecciones y dificultades

3.1. Lecciones

- Aprendimos que las pruebas unitarias ayudan a garantizar que los controladores funcionan correctamente antes de integrarlos con el resto del sistema.
- Detectamos errores en la validación de datos y respuestas del backend antes de que llegaran a producción.
- Jest y otras herramientas como Supertest y Prisma están diseñadas para trabajar con CommonJS por defecto, lo que causó conflictos con nuestro proyecto basado en ES Modules. Para solucionar esto, fue necesario configurar Babel y modificar la sintaxis de importación y exportación para que Jest pudiera ejecutar correctamente los tests.
- Prisma se debe conectar y desconectar correctamente para evitar errores de conexión.
- Es importante limpiar los datos de prueba después de cada ejecución para que los tests sean independientes entre sí y no dependan del estado anterior de la base de datos.
- Se comprobó la importancia de manejar correctamente los errores en el backend, devolviendo mensajes claros y códigos HTTP apropiados.
- Se verificó que el backend rechace datos inválidos y maneje adecuadamente casos en los que la información solicitada no existe.

3.2. Dificultades

- Como Jest y otras herramientas vienen por defecto configuradas para CommonJS, fue necesario configurar Babel para poder ejecutar los tests con ES Modules. Esto requirió agregar un archivo de configuración (`babel.config.json`) y modificar los imports y exports en los archivos de prueba.
- Hubo problemas al establecer y cerrar la conexión con la base de datos antes y después de cada prueba. Se solucionó asegurando que `prisma.$connect()` y `prisma.$disconnect()` se ejecutarán correctamente en los métodos `beforeAll` y `afterAll`.
- Algunas pruebas fallaban porque dependían de que hubiera registros previos en la base de datos. Se resolvió asegurando que los tests crearán y limpiarán sus propios datos antes y después de ejecutarse.
- Se identificaron fallos en la validación de los datos de entrada, por ejemplo, cuando se enviaba un `motherboardId` incorrecto o vacío.
- Se mejoró la validación en el backend para devolver mensajes de error más claros y evitar respuestas ambiguas.

4. Referencias

[1] *ChatGPT*. (2025). Chatgpt.com. https://chatgpt.com/