

DEPARTAMENTO:	Ciencias de la computación	CARRERA:	Software		
ASIGNATURA:	Pruebas de software	NIVEL:	6to	FECHA:	08/11/2025
DOCENTE:	Ing. LUIS ALBERTO CASTILLO SALINAS	PRÁCTICA N°:	2	CALIFICACIÓN:	

## Verificación y Validación

**Moisés Sebastián Benalcázar Farinango**

### RESUMEN

En la presente práctica se implementó una API REST para gestión de usuarios utilizando Node.js, Express, Jest + Supertest y ESLint, con el objetivo de aplicar técnicas de verificación y validación de software. Se estructuró el proyecto con carpetas src, routes, controllers y test, se configuró un servidor Express con endpoints GET y POST, y se simuló una base de datos en memoria mediante un arreglo. Se realizaron pruebas unitarias y end-to-end para validar el comportamiento de la API, logrando una cobertura superior al 90%. Además, se configuró ESLint con reglas personalizadas para garantizar buenas prácticas de codificación. Los resultados demuestran la importancia de integrar herramientas de testing y linting desde las primeras etapas del desarrollo para obtener código robusto, mantenible y libre de errores comunes.

**Palabras Claves:** Verificación, Validación, ESLint

### 1. INTRODUCCIÓN:

La práctica se centró en el uso disciplinado de herramientas modernas de desarrollo para garantizar la calidad del software mediante verificación (análisis estático del código) y validación (pruebas dinámicas). Se siguieron las indicaciones del docente, se documentó cada paso con capturas de pantalla y se utilizaron datos personales en las pruebas para evidenciar la autoría del trabajo, promoviendo buenas prácticas de programación y responsabilidad técnica en el entorno de laboratorio.

### 2. OBJETIVO(S):

- 2.1 Aplicar técnicas de verificación sobre el código.
- 2.2 Aplicar técnicas de validación sobre el código.
- 2.3 Generación de reglas para la verificación del código.
- 2.4 Creación de pruebas unitarias para la validación del código

### 3. MARCO TEÓRICO:

Node.js es un entorno de ejecución que permite correr JavaScript fuera del navegador, ideal para construir servidores backend. Express es un framework minimalista que simplifica la creación de APIs REST mediante enruteamiento y middlewares. Jest es un framework de pruebas unitarias con soporte para assertions, mocks y reportes de cobertura, mientras que Supertest permite realizar pruebas HTTP sobre aplicaciones Express sin necesidad de levantar un servidor real. ESLint es una herramienta de análisis estático que detecta errores, aplica reglas de estilo y fomenta buenas prácticas mediante configuraciones personalizables (eslint.config.js). Juntos, forman un flujo de desarrollo moderno enfocado en calidad y mantenibilidad.

### 4. DESCRIPCIÓN DEL PROCEDIMIENTO:

**PARTE 1:** Establecer la estructura del proyecto y configurar el ambiente de desarrollo

**Paso 1:** Estructura del proyecto.

a. Se debe crear una estructura básica para un proyecto JS teniendo en cuenta:

- Src
- Routes
- Controllers
- Test
- Archivos de configuración

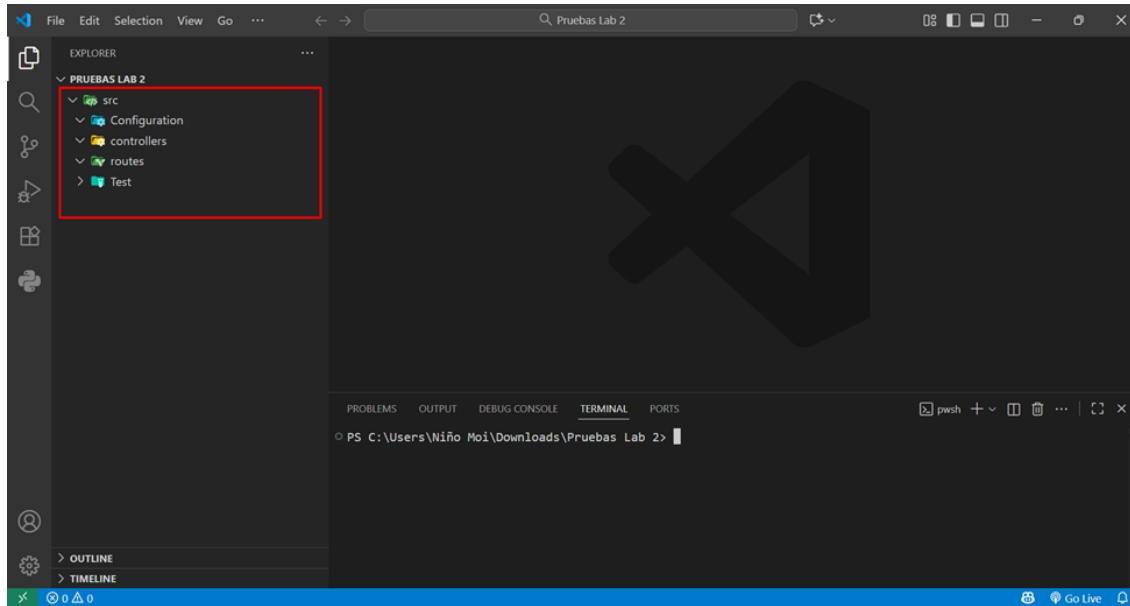
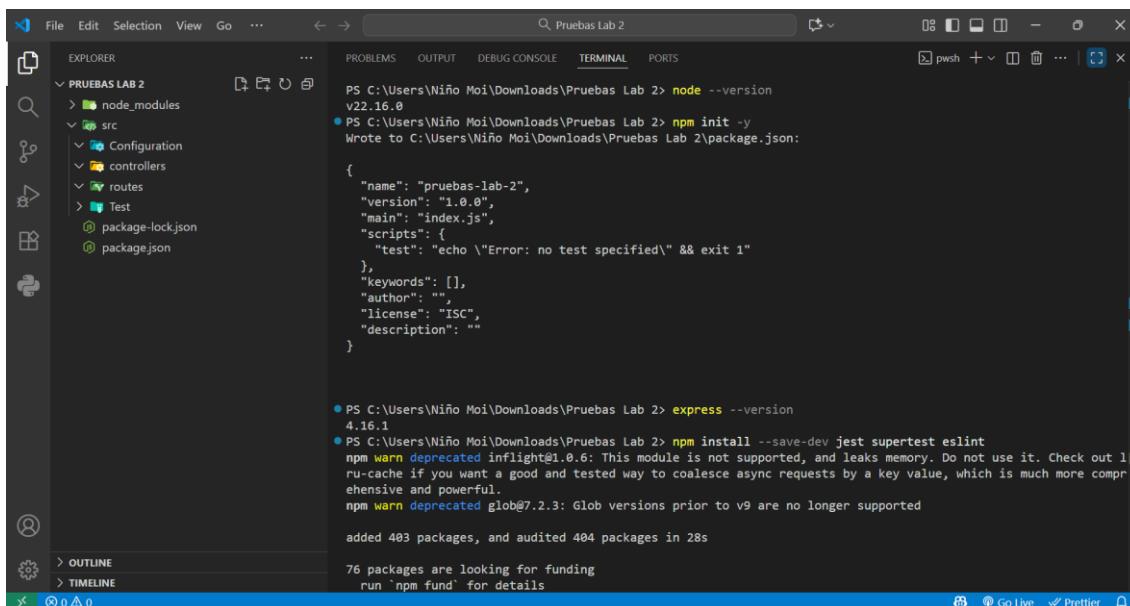


Ilustración 1: Estructura de carpetas y archivos para el laboratorio.

### Paso 2: Instalar las tecnologías necesarias para el ambiente de desarrollo y pruebas.

- Crear el archivo package.json ejecutando el siguiente comando: npm init
- Instalar express: npm install express
- Instalar las librerías necesarias: npm install --save-dev jest supertest eslint



```

PS C:\Users\Niño Moi\Downloads\Pruebas Lab 2> node --version
v22.16.0
PS C:\Users\Niño Moi\Downloads\Pruebas Lab 2> npm init -y
Wrote to C:\Users\Niño Moi\Downloads\Pruebas Lab 2\package.json:

{
  "name": "pruebas-lab-2",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \\"Error: no test specified\\\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": ""
}

PS C:\Users\Niño Moi\Downloads\Pruebas Lab 2> express --version
4.16.1
PS C:\Users\Niño Moi\Downloads\Pruebas Lab 2> npm install --save-dev jest supertest eslint
npm warn deprecated inflight@1.0.6: This module is not supported, and leaks memory. Do not use it. Check out lru-cache if you want a good and tested way to coalesce async requests by a key value, which is much more comprehensive and powerful.
npm warn deprecated glob@7.2.3: Glob versions prior to v9 are no longer supported
added 403 packages, and audited 404 packages in 28s
76 packages are looking for funding
  run `npm fund` for details
  
```

Ilustración 2: Se reviso y se instaló las herramientas necesarias para realizar el laboratorio como: express, jest, supertest y eslint.

## PARTE 2: Creación de API de Gestión de Usuarios

### Paso 1: Configurar el controlador.

- Crear el archivo controllers/user.controller.js.

- b. Simular de una base de datos en memoria (arreglo).
  - c. Función para devolver todos los usuarios almacenados.
  - d. Función para crear un nuevo usuario si se proveen nombre y correo válidos.
  - e. Generar una validación básica de entrada
  - f. Crear un objeto usuario y añadirlo al arreglo de usuarios
  - g. Responder con el usuario creado status 201
  - h. Exportar las funciones creadas con module.exports.

The screenshot shows the VS Code interface with the following details:

- File Explorer (Left):** Shows the project structure with files like `node_modules`, `src`, `controllers`, `user.controller.js`, `routes`, `user.routes.js`, `app.js`, `test`, `eslint.config.js`, `package-lock.json`, and `package.json`. The `user.controller.js` file is selected and highlighted with a red box.
- Code Editor (Center):** Displays the `user.controller.js` file content. The code implements a simulated database and handles user creation and retrieval. A red box highlights the first two functions: `getAllUsers` and `createUser`.
- Bottom Status Bar:** Shows the current file is `user.controller.js`, line 34, column 2. It also indicates 2 spaces, UTF-8 encoding, CRLF line endings, Java script as the language, and Prettier for code styling.

```
// Simulación de una base de datos en memoria
let users = [];

/**
 * Devuelve todos los usuarios almacenados
 */
function getAllUsers(req, res) {
  res.json(users)
}

/**
 * Crea un nuevo usuario si se proveen name y email válidos
 */
function createUser(req, res) {
  const { name, email } = req.body;

  // Validación básica de entrada
  if (!name || !email) {
    return res.status(400).json({ message: 'Name and email are required' })
  }

  // Creamos un objeto usuario
  const newUser = {
    id: Date.now(), // ID simulado
    name,
    email
  };
}
```

**Ilustración 3: configuración del controlador creando funciones para manejar usuarios, validando y almacenando en un arreglo simulado.**

#### **Paso 2:** Configurar las rutas del servidor.

- a. Crear el archivo routes/user.routes.js.
  - b. Importar los módulos necesarios: express y funciones del controlador.
  - c. Definir ruta GET para obtener todos los usuarios.
  - d. Definir ruta POST para crear un nuevo usuario.
  - e. Exportar el router con module.exports.

The screenshot shows a Visual Studio Code interface with the following details:

- File Explorer (Left):** Shows a project structure under "LAB2". The "routes" folder contains "user.routes.js", which is currently selected and highlighted with a red box.
- Code Editor (Center):** Displays the content of "user.routes.js".

```
1 const express = require('express');
2 const { getAllUsers, createUser } = require('../controllers/user.controller');
3
4 const router = express.Router();
5
6 // Ruta GET para obtener todos los usuarios
7 router.get('/', getAllUsers);
8
9 // Ruta POST para crear un nuevo usuario
10 router.post('/', createUser);
11
12 module.exports = router;
```
- Status Bar (Bottom):** Shows file paths like "user.routes.js", line numbers "Line 14", and other status indicators.

**Ilustración 4:** configuración las rutas del servidor, definir rutas GET y POST en un archivo de rutas para manejar usuarios.

**Paso 3:** Configurar la entrada principal.

- Crear el archivo app.js.
- Importar los módulos necesarios: express y el archivo donde se manejarán las rutas user.routes.js.
- Crea una instancia de la aplicación Express
- Usar un middleware para parsear JSON del cuerpo de las solicitudes.
- Establecer la ruta base para los usuarios.
- Usar un manejador de rutas no encontradas (404).
- Exportar la instancia app para poder usarla en tests o en un archivo de servidor separado.

```

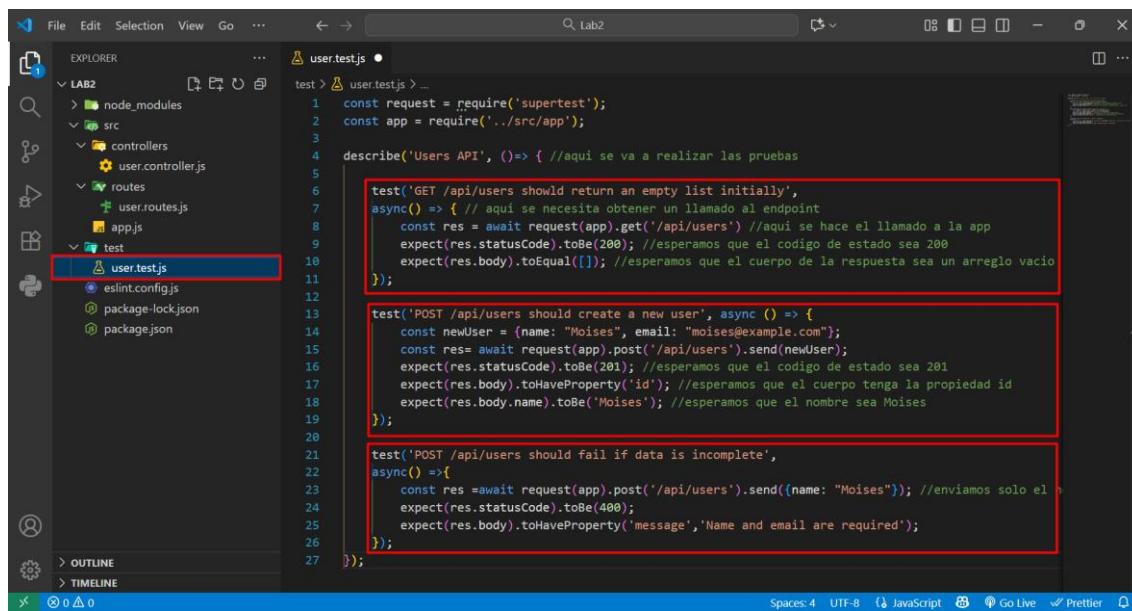
File Edit Selection View Go ...
user.controller.js app.js
src > app.js > ...
1 const express = require('express');
2 const userRoutes = require('./routes/user.routes');
3
4 const app = express(); // Crea una instancia de la aplicación Express
5
6 // Middleware para parsear JSON del cuerpo de las solicitudes
7 app.use(express.json());
8
9 // Ruta base para los usuarios
10 app.use('/api/users', userRoutes);
11
12 // Manejador de rutas no encontradas (404)
13 app.use((req, res) => {
14   res.status(404).json({ message: 'Route not found' });
15 });
16
17 // Exportamos app para poder usarla en tests o en un archivo de servidor separado
18 module.exports = app;
    
```

Ilustración 5: configuración la entrada principal, crear y configurar la aplicación Express con rutas y middleware para procesar solicitudes JSON.

**PARTE 3:** Verificación y validación

**Paso 1:** Validación: Pruebas con Jest y Supertest.

- Crear el archivo test/user.test.js.
- Importar el cliente HTTP son supertest para pruebas.
- Importar app Express.
- Crear prueba que GET devuelva lista vacía inicialmente.
- Crear prueba que POST cree un nuevo usuario correctamente
- Crear prueba que el endpoint rechace peticiones inválidas
- Ejecutar las pruebas con el comando: npm test



```

File Edit Selection View Go ... ⌂ ↻ 🔍 Lab2
EXPLORER LAB2 node_modules src controllers user.controller.js routes user.routes.js app.js test user.test.js eslint.config.js package-lock.json package.json
test > user.test.js ...
test('GET /api/users should return an empty list initially', async () => {
    const res = await request(app).get('/api/users') //aqui se hace el llamado a la app
    expect(res.statusCode).toBe(200); //esperamos que el codigo de estado sea 200
    expect(res.body).toEqual([]); //esperamos que el cuerpo de la respuesta sea un arreglo vacio
});

test('POST /api/users should create a new user', async () => {
    const newUser = {name: "Moises", email: "moises@example.com"};
    const res = await request(app).post('/api/users').send(newUser);
    expect(res.statusCode).toBe(201); //esperamos que el codigo de estado sea 201
    expect(res.body).toHaveProperty('id'); //esperamos que el cuerpo tenga la propiedad id
    expect(res.body.name).toBe('Moises'); //esperamos que el nombre sea Moises
});

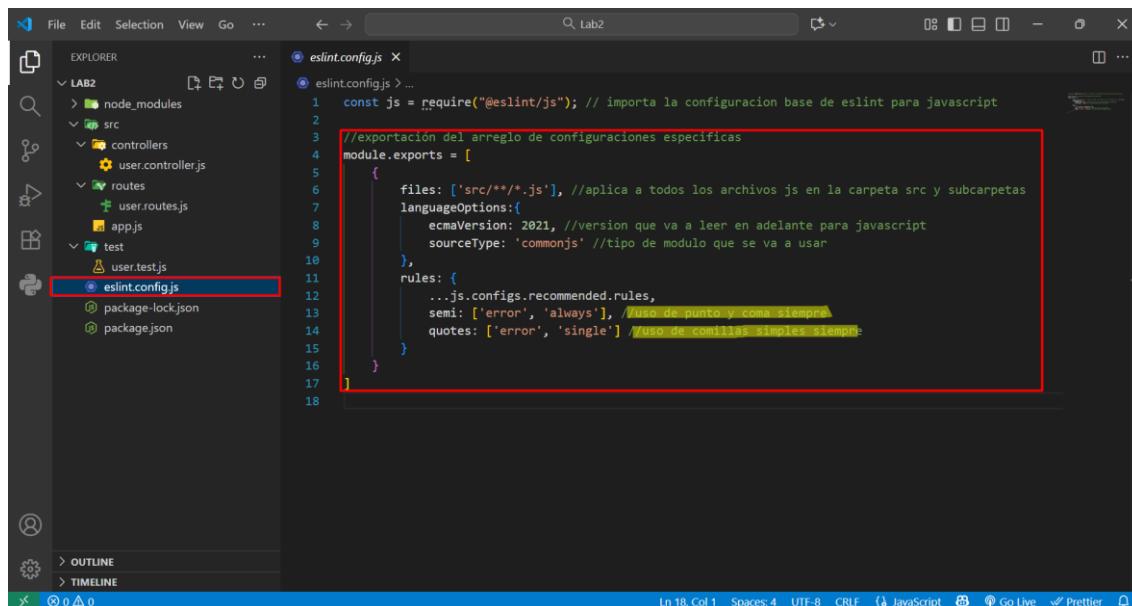
test('POST /api/users should fail if data is incomplete', async () => {
    const res = await request(app).post('/api/users').send({name: "Moises"}); //enviamos solo el nombre
    expect(res.statusCode).toBe(400);
    expect(res.body).toHaveProperty('message','Name and email are required');
});
}

```

Ilustración 6: Creación de pruebas con Jest y Supertest para verificar el comportamiento de los endpoints GET y POST y rechazar peticiones inválidas.

### Paso 2: Verificación: ESLint.

- Crear el archivo eslint.config.js.
- Importar la configuración base de reglas recomendadas de ESLint para JavaScript.
- Exportar un arreglo de configuraciones específicas para ESLint.
- Configurar los archivos dentro de la carpeta src con extensión .js.
- Gestionar opciones del lenguaje para estos archivos.
- Crear reglas de ESLint que se aplicarán a estos archivos.



```

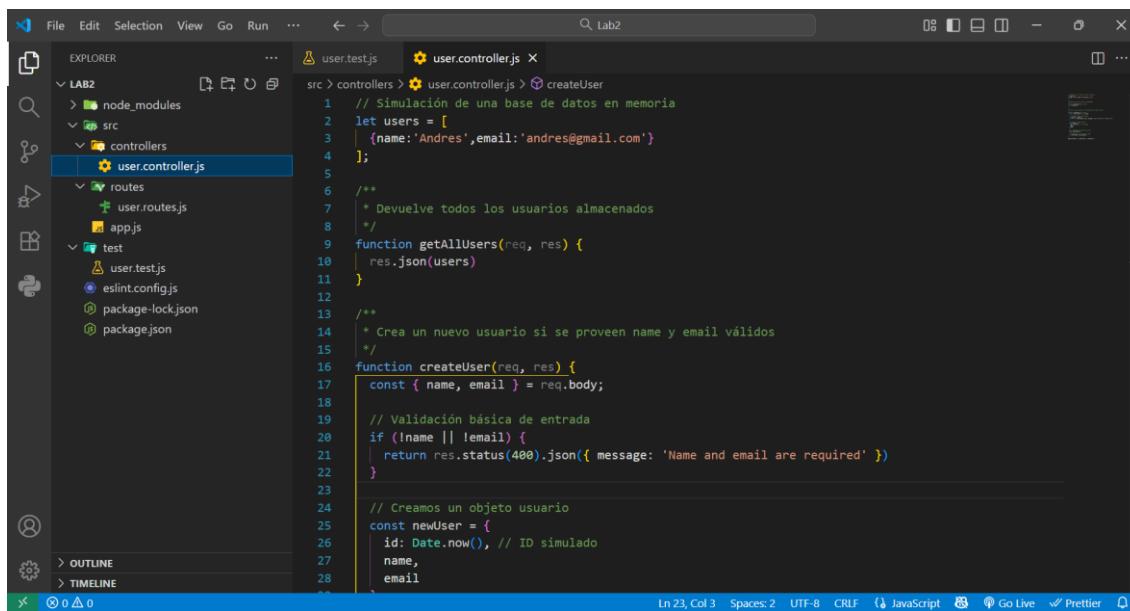
File Edit Selection View Go ... ⌂ ↻ 🔍 Lab2
EXPLORER LAB2 node_modules src controllers user.controller.js routes user.routes.js app.js test user.test.js eslint.config.js package-lock.json package.json
eslint.config.js ...
const js = require('@eslint/js'); // importa la configuración base de eslint para javascript
//exportación del arreglo de configuraciones específicas
module.exports = [
{
    files: ['src/**/*.js'], //aplica a todos los archivos js en la carpeta src y subcarpetas
    languageOptions: {
        esmaVersion: 2021, //versión que va a leer en adelante para javascript
        sourceType: 'commonjs' //tipo de módulo que se va a usar
    },
    rules: {
        ...js.configs.recommended.rules,
        semi: ['error', 'always'], //uso de punto y coma siempre
        quotes: ['error', 'single'] //uso de comillas simples siempre
    }
}
]

```

Ilustración 7: Configuración de ESLint creando un archivo de configuración, importando reglas base y definiendo reglas específicas para los archivos .js en la carpeta src.

## 5. PREGUNTAS/ACTIVIDADES:

- Actividad: Implementar una ruta GET /users que retorne una lista de usuarios en memoria (array local).



```

// Simulación de una base de datos en memoria
let users = [
  {name: 'Andres', email: 'andres@gmail.com'}
];

/**
 * Devuelve todos los usuarios almacenados
 */
function getAllUsers(req, res) {
  res.json(users);
}

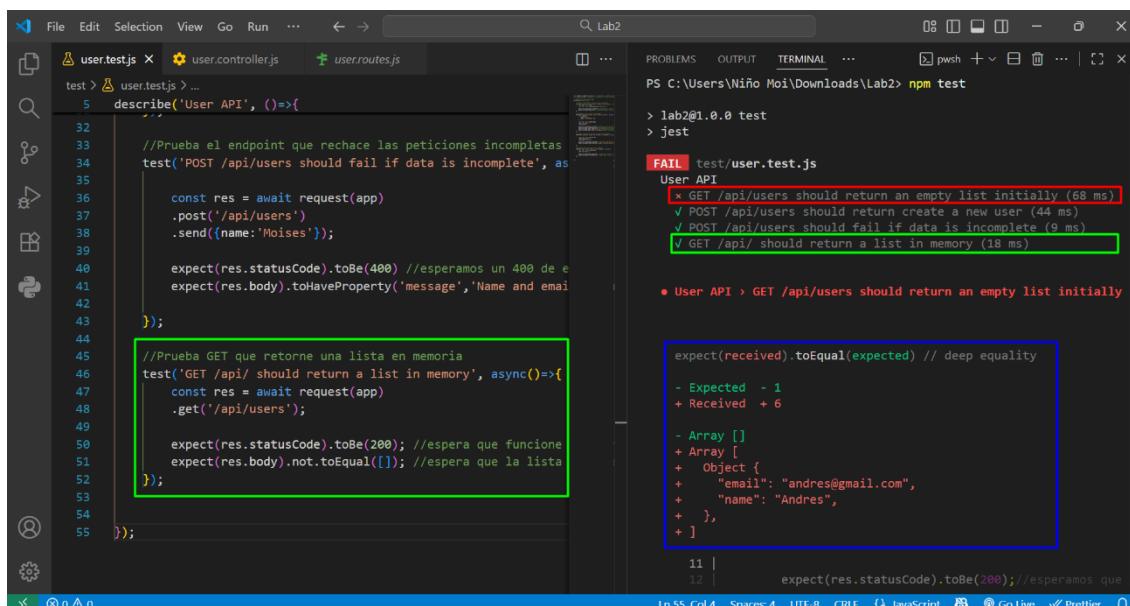
/**
 * Crea un nuevo usuario si se proveen name y email válidos
 */
function createUser(req, res) {
  const { name, email } = req.body;

  // Validación básica de entrada
  if (!name || !email) {
    return res.status(400).json({ message: 'Name and email are required' });
  }

  // Creamos un objeto usuario
  const newUser = {
    id: Date.now(), // ID simulado
    name,
    email
  };
}

```

Ilustración 8: Creamos una lista en memoria de manera local para manejar la prueba solicitada.



```

describe('User API', ()=>{
  //Prueba el endpoint que rechace las peticiones incompletas
  test('POST /api/users should fail if data is incomplete', async()=>{
    const res = await request(app)
      .post('/api/users')
      .send({name:'Moises'});

    expect(res.statusCode).toBe(400) //esperamos un 400 de error
    expect(res.body).toHaveProperty('message','Name and email are required');
  });

  //Prueba GET que retorne una lista en memoria
  test('GET /api/users should return a list in memory', async()=>{
    const res = await request(app)
      .get('/api/users');

    expect(res.statusCode).toBe(200); //espera que funcione
    expect(res.body).not.toEqual([]); //espera que la lista no sea vacía
  });
});

```

PS C:\Users\Niño Moi\Downloads\Lab2> npm test

FAIL test/user.test.js

User API

- x GET /api/users should return an empty list initially (68 ms)
- ✓ POST /api/users should return create a new user (44 ms)
- ✓ POST /api/users should fail if data is incomplete (9 ms)
- ✓ GET /api/ should return a list in memory (18 ms)

User API > GET /api/users should return an empty list initially

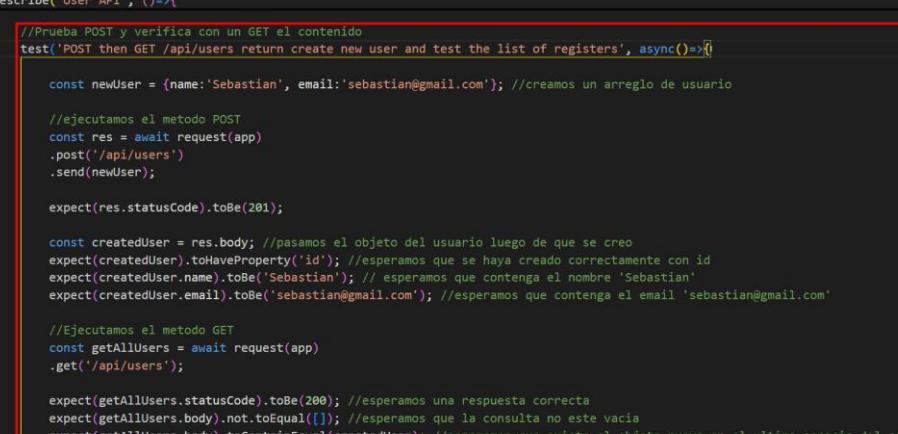
```

expect(received).toEqual(expected) // deep equality
- Expected - 1
+ Received + 6
- Array []
+ Array [
+ Object {
+   "email": "andres@gmail.com",
+   "name": "Andres",
+ },
+ ]

```

Ilustración 9: Tenemos error al ejecutar la primera prueba ya que espera una lista vacía en el rectángulo rojo debido a que ya existe una lista local la cual se observa en el rectángulo azul, la prueba que se realizó y se aprobó en los rectángulos verdes.

b. Prueba: Crear usuarios con POST /users y luego verificar el contenido de GET /users.



```
test('POST then GET /api/users return create new user and test the list of registers', async()=>{
    const newUser = {name:'Sebastian', email:'sebastian@gmail.com'}; //Creamos un arreglo de usuario

    //ejecutamos el metodo POST
    const res = await request(app)
        .post('/api/users')
        .send(newUser);

    expect(res.statusCode).toBe(201);

    const createdUser = res.body; //pasamos el objeto del usuario luego de que se creo
    expect(createdUser).toHaveProperty('id'); //esperamos que se haya creado correctamente con id
    expect(createdUser.name).toBe('Sebastian'); //esperamos que contenga el nombre 'Sebastian'
    expect(createdUser.email).toBe('sebastian@gmail.com'); //esperamos que contenga el email 'sebastian@gmail.com'

    //Ejecutamos el metodo GET
    const getAllUsers = await request(app)
        .get('/api/users');

    expect(getAllUsers.statusCode).toBe(200); //esperamos una respuesta correcta
    expect(getAllUsers.body).not.toEqual([]); //esperamos que la consulta no este vacia
    expect(getAllUsers.body).toContainEqual(createdUser); //esperamos que exista el objeto nuevo en el ultimo espacio del arreglo
});
```

**Ilustración 10:** Creamos el nuevo usuario y verificamos si consta como ultimo ingreso del arreglo, para esto almacenamos en una variable al usuario creado y comprobamos con el ultimo elemento de la lista con 'toContainEqual'.

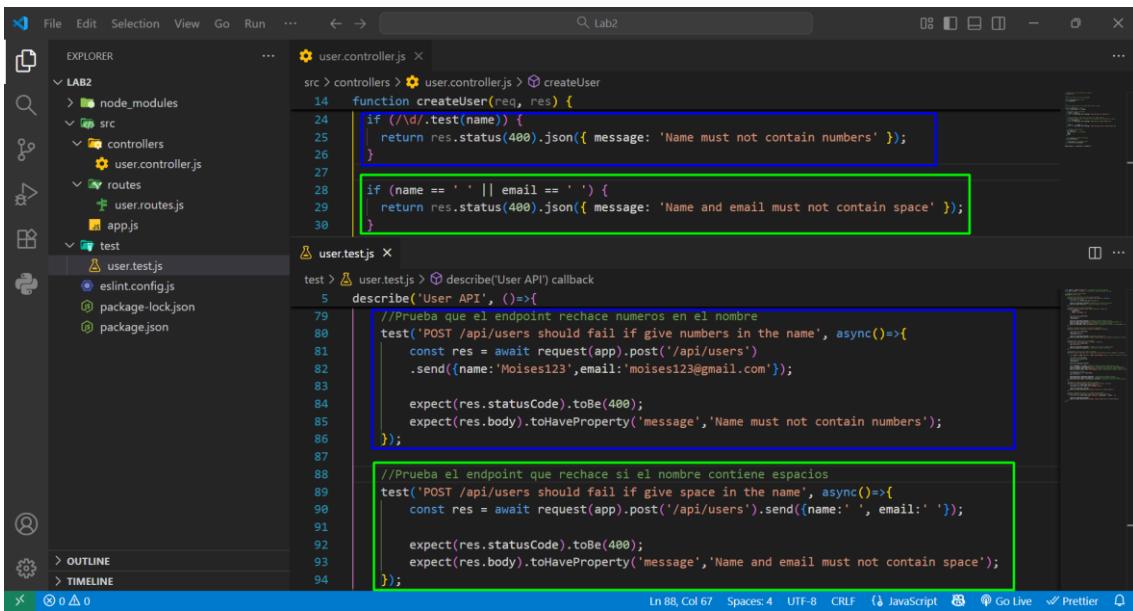
The screenshot shows a VS Code interface with the following details:

- File Explorer (Left):** Shows a project structure under 'LAB2' with files like 'node\_modules', 'src', 'controllers', 'routes', 'app.js', and 'test' (containing 'user.test.js', 'eslint.config.js', 'package-lock.json', and 'package.json').
- Editor (Top Center):** Displays 'user.test.js' code, which includes a Jest test for a 'User API' endpoint.
- Terminal (Bottom):** Shows the command 'PS C:\Users\Niño Moi\Downloads\Lab2> npm test' being run, followed by the test results:
  - PASS test/user.test.js
  - User API
    - ✓ GET /api/users should return an empty list initially (35 ms)
    - ✓ POST /api/users should return create a new user (18 ms)
    - ✓ POST /api/users should fail if data is incomplete (5 ms)
    - ✓ GET /api/ should return a list in memory (5 ms)
    - ✓ POST then GET /api/users return create new user and test the list of registers (10 ms)

Test Suites: 1 passed, 1 total  
Tests: 5 passed, 5 total  
Snapshots: 0 total  
Time: 0.609 s, estimated 1 s  
Ran all test suites.

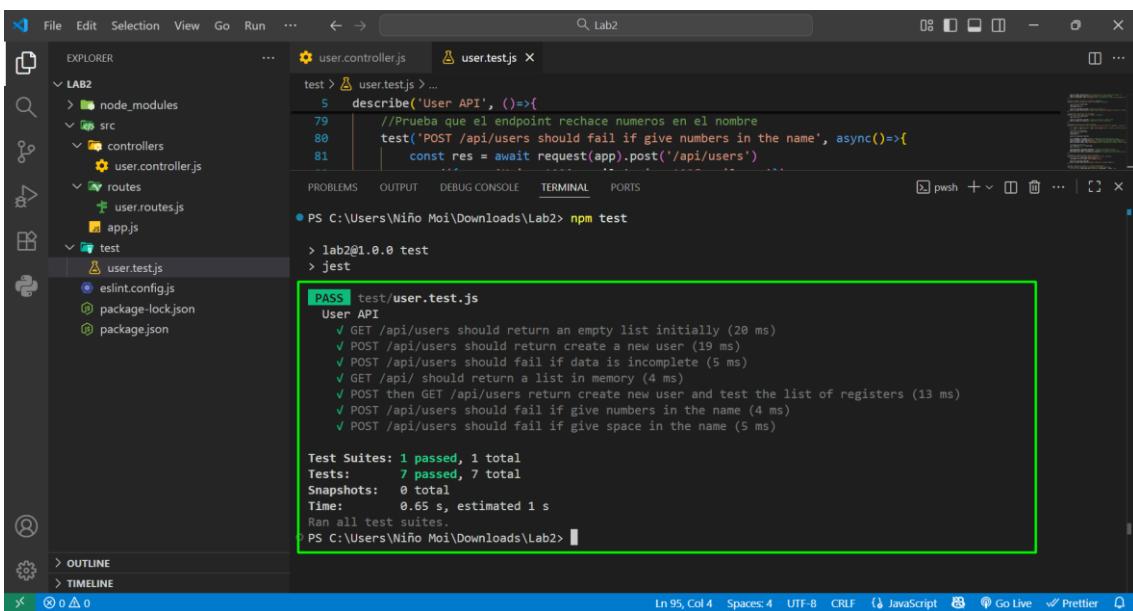
**Ilustración 11:** Verificamos que funcione correctamente la comprobación de crear y obtener el usuario.

c. Objetivo: Pruebas de flujo completo (end-to-end simulado).



The screenshot shows the Visual Studio Code interface. The left sidebar displays a file tree with a project structure: LAB2, node\_modules, src, controllers, routes, app.js, and test. Inside the test folder are user.test.js, eslint.config.js, package-lock.json, and package.json. The main area has two tabs open: 'user.controller.js' and 'user.test.js'. The code in 'user.controller.js' contains logic for creating a user, specifically checking if the name or email contains numbers or spaces. The code in 'user.test.js' contains Jest test cases for these validation rules. The terminal at the bottom shows the command 'npm test' being run.

Ilustración 12: Se crearon dos pruebas adicionales que validan si el nombre no contenga números y que el nombre o correo no acepte espacios en blanco.

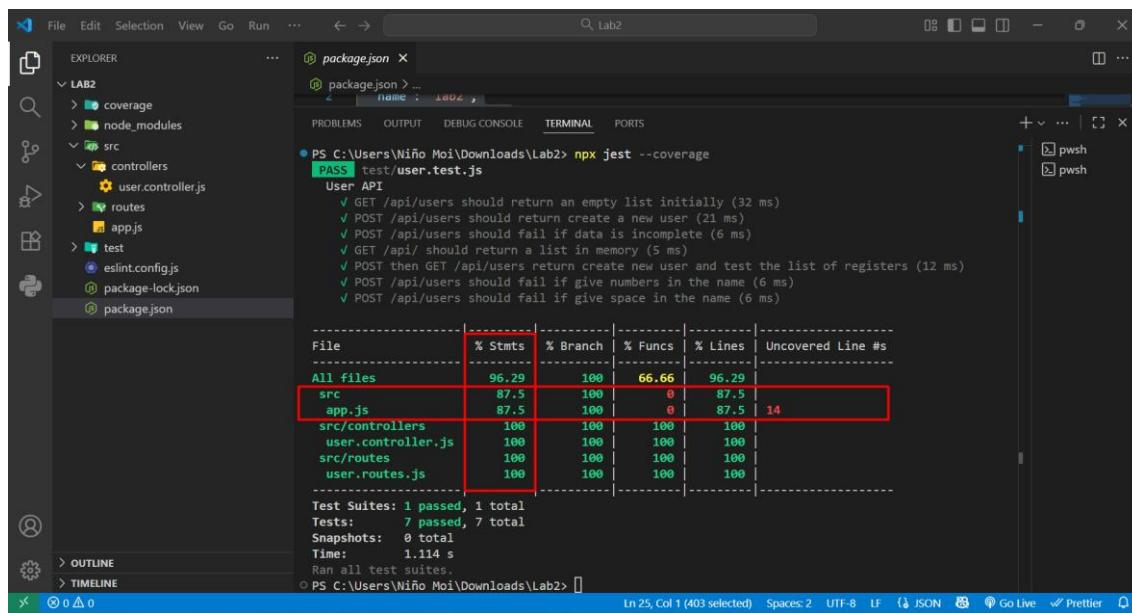


The screenshot shows the Visual Studio Code interface. The left sidebar displays a file tree with a project structure: LAB2, node\_modules, src, controllers, routes, app.js, and test. Inside the test folder are user.test.js, eslint.config.js, package-lock.json, and package.json. The main area has two tabs open: 'user.controller.js' and 'user.test.js'. The terminal at the bottom shows the command 'npm test' being run, followed by the output of the Jest test suite. The output indicates 7 passed tests and 0 total snapshots.

Ilustración 13: Se comprobó que las pruebas sean 100% válidas para continuar con el laboratorio.

## 2. Verificar cobertura de pruebas con Jest

a. Actividad: Ejecutar jest --coverage, analizar el reporte y agregar pruebas hasta lograr >90% de cobertura.

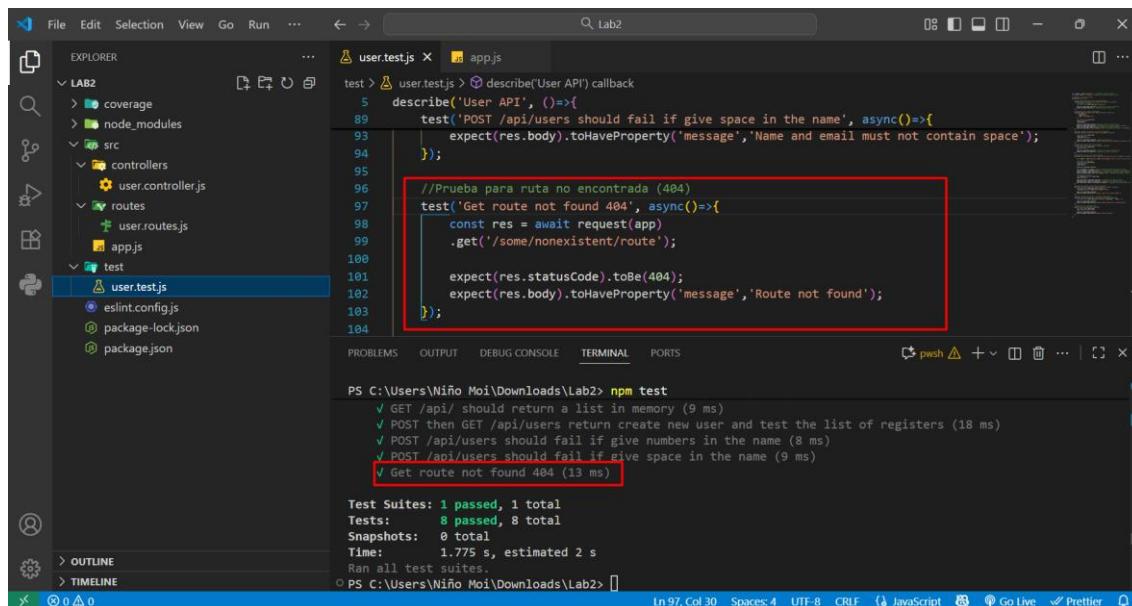


File	%Stmts	%Branch	%Funcs	%Lines	Uncovered Line #s
All files	96.29	100	66.66	96.29	
src	87.5	100	0	87.5	
app.js	87.5	100	0	87.5	14
src/controllers	100	100	100	100	
user.controller.js	100	100	100	100	
src/routes	100	100	100	100	
user.routes.js	100	100	100	100	

Test Suites: 1 passed, 1 total  
Tests: 7 passed, 7 total  
Snapshots: 0 total  
Time: 1.114 s  
Ran all test suites.

Ilustración 14: Ejecución de 'npx jest --coverage' validar el porcentaje de cobertura que se realizo mediante las pruebas, demostrando que no se ha tomado en cuenta una api de 'app.js' en la linea 14.

b. Objetivo: Comprender el impacto de la cobertura y escribir pruebas adicionales.



```

describe('User API', ()=>{
  test('POST /api/users should fail if give space in the name', async()=>{
    expect(res.body).toHaveProperty('message','Name and email must not contain space');
  });

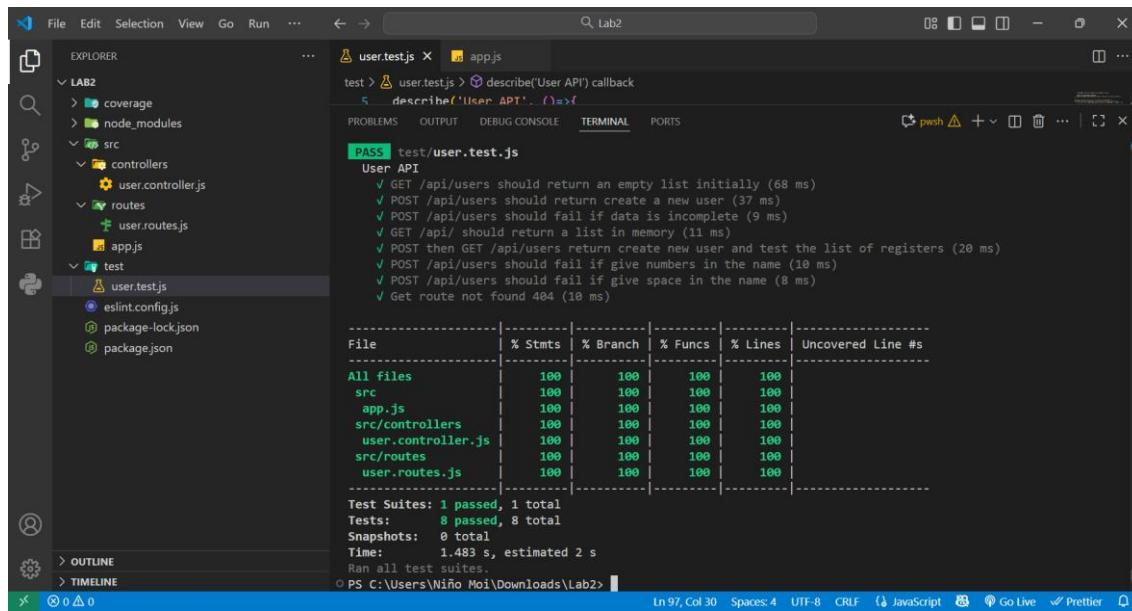
  //Prueba para ruta no encontrada (404)
  test('Get route not found 404', async()=>{
    const res = await request(app)
      .get('/some/nonexistent/route');

    expect(res.statusCode).toBe(404);
    expect(res.body).toHaveProperty('message','Route not found');
  });
});

Test Suites: 1 passed, 1 total
Tests: 8 passed, 8 total
Snapshots: 0 total
Time: 1.775 s, estimated 2 s
Ran all test suites.

```

Ilustración 15: Se agrego una nueva prueba que lleve el error de ruta no encontrada y verificando que es una prueba exitosa, para continuar con la evaluación de la cobertura.



```

PASS  test/user.test.js
  User API
    ✓ GET /api/users should return an empty list initially (68 ms)
    ✓ POST /api/users should return create a new user (37 ms)
    ✓ POST /api/users should fail if data is incomplete (9 ms)
    ✓ GET /api/ should return a list in memory (11 ms)
    ✓ POST then GET /api/users return create new user and test the list of registers (20 ms)
    ✓ POST /api/users should fail if give numbers in the name (10 ms)
    ✓ POST /api/users should fail if give space in the name (8 ms)
    ✓ Get route not found 404 (10 ms)

File           | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #
---            | ---     | ---      | ---     | ---     | ---
All files      | 100    | 100     | 100    | 100    | 
src            | 100    | 100     | 100    | 100    | 
app.js         | 100    | 100     | 100    | 100    | 
src/controllers | 100    | 100     | 100    | 100    | 
user.controller.js | 100    | 100     | 100    | 100    | 
src/routes     | 100    | 100     | 100    | 100    | 
user.routes.js | 100    | 100     | 100    | 100    | 

Test Suites: 1 passed, 1 total
Tests:       8 passed, 8 total
Snapshots:   0 total
Time:        1.483 s, estimated 2 s
Ran all test suites.
PS C:\Users\Niño Moi\Downloads\Lab2>

```

Ilustración 16: Se logró obtener un 100% de cobertura demostrando que se cumple con la cobertura en todo el código.

## 6. CONCLUSIONES:

Las técnicas aplicadas para la verificación y validación del código fueron muy interesantes debido a que es más rápido revisar el código de esa manera, y sin necesidad de estar repitiendo tantas veces para ver si esta ejecutándose correcta o simplemente algo falla en el código.

Además, las reglas que se coloca para la verificación van de acuerdo con la API que se desea probar puesto que se puede probar la API completa o simplemente una parte de esta.

## 7. RECOMENDACIONES:

Desde mi criterio, recomendaría que se practique con sistemas ya levantados para probar que tan difícil es aplicar las pruebas sobre código realista en el que se suelen omitir los comentarios de los métodos o realizar métodos anidados.

Al iniciar el proceso de pruebas y luego al tener un avance significativo realizar una prueba de cobertura podría ayudar a identificar la API que nos falta probar, logrando evitar pérdidas de tiempo primero a que nos indica cuantas APIs tenemos que realizar las API's y segundo que nos permite conocer si ya terminamos las pruebas cumpliendo con una cobertura >90% del código.

Creo que sería muy efectivo buscar ejemplos de pruebas unitarias aplicadas a Node.js ya que nos podríamos hacer la idea de las posibles pruebas que se pueden realizar y ver las que son recomendables para cierto tipo de API's, lo que permitirá realizar pruebas efectivas y necesarias.

## 8. BIBLIOGRAFÍA:

Express.js. (s.f.). *Express - Fast, unopinionated, minimalist web framework for Node.js*. Recuperado el 7 de noviembre de 2025, de <https://expressjs.com/>

Jest Team. (2025). *Jest · Delightful JavaScript Testing*. Recuperado el 7 de noviembre de 2025, de <https://jestjs.io/>

Supertest. (s.f.). *HTTP assertions made easy via superagent*. GitHub. Recuperado el 7 de noviembre de 2025, de <https://github.com/ladjs/supertest>

ESLint. (2025). *ESLint - Pluggable JavaScript Linter*. Recuperado el 7 de noviembre de 2025, de <https://eslint.org/>

Node.js Foundation. (2025). *Node.js — Run JavaScript Everywhere*. Recuperado el 7 de noviembre de 2025, de <https://nodejs.org/>

Castillo, L. (2024). *Guía para las Prácticas de Laboratorio - Práctica N° 2: Verificación y Validación.* Código: VDC-GUI-2024-V2-015. Departamento de Ciencias de la Computación, ESPE Sede Santo Domingo.

**9. Anexos:**

Enlace al repositorio en github:

<https://github.com/Sebas8173/UniversityRepository/tree/main/Pruebas%20de%20software/U1/Lab%202/Lab%202%20-%20practica>