ESPE UNIVERSIDAD DE LAS FUERZAS ARMADAS INNOVACIÓN PARA LA EXCELENCIA	INFORME DE PRÁCTICA DE LABORATORIO

DEPARTAMENTO:	**Ciencias de la computación**	**CARRERA:**	**Ingeniería de Software**		
ASIGNATURA:	Pruebas de Software	**NIVEL:**	6to	**FECHA:**	05/30/2025
DOCENTE:	Luis Alberto Castillo Salinas	**PRÁCTICA N°:**	2	**CALIFICACIÓN:**	

Verificación y Validación

Jairo Smith Bonilla Hidalgo

RESUMEN

En el presente laboratorio se desarrolló una API de gestión de usuarios utilizando Node.js con el framework Express. Se implementaron las rutas necesarias para permitir operaciones básicas como la creación y recuperación de usuarios, simulando un almacenamiento en memoria mediante un arreglo local. Posteriormente, se construyeron pruebas automatizadas utilizando Jest y Supertest para validar el correcto funcionamiento de las rutas implementadas. Estas pruebas incluyeron tanto casos exitosos como errores controlados, verificando así el comportamiento del sistema en diferentes escenarios. Además, se realizó la verificación de cobertura de pruebas, logrando superar el 90% requerido mediante la incorporación de casos adicionales. Esta práctica permitió reforzar conocimientos en diseño de APIs REST, pruebas automatizadas y buenas prácticas de desarrollo, destacando la importancia de la validación de entradas y el manejo adecuado de errores.

Palabras Claves: API, Express, Pruebas Automatizadas

1. INTRODUCCIÓN:

La presente práctica de laboratorio tuvo como finalidad fortalecer las habilidades en el desarrollo y prueba de APIs REST utilizando tecnologías modernas de JavaScript, con énfasis en el entorno de Node.js y el framework Express. Durante la sesión se implementó un sistema básico de gestión de usuarios, integrando conceptos fundamentales como controladores, rutas, pruebas automatizadas, validación de datos y manejo de errores. Asimismo, se promovió la disciplina en la estructura del proyecto, respetando convenciones de organización de carpetas y archivos. Las actividades realizadas permitieron simular de manera controlada un flujo completo de operaciones (end-to-end), fomentando buenas prácticas de programación y el uso de herramientas como Jest, Supertest y ESLint para asegurar calidad y mantenimiento del código.

2. OBJETIVO(S):

2.1 Implementar una API REST básica para la gestión de usuarios utilizando Node.js y Express, estructurando el proyecto en carpetas organizadas.

2.2 Desarrollar pruebas automatizadas que validen el comportamiento correcto del sistema ante diferentes entradas.

2.3 Simular almacenamiento en memoria para representar una base de datos local.

2.4 Validar la cobertura de código con Jest y aplicar buenas prácticas mediante ESLint.

2.5 Comprender el flujo completo de desarrollo, prueba y verificación de una API funcional.

3. MARCO TEÓRICO:

Una API REST (Application Programming Interface - Representational State Transfer) es un conjunto de reglas que permiten la comunicación entre aplicaciones a través del protocolo HTTP. Este tipo de arquitectura se basa en recursos (como usuarios, productos, etc.) que pueden ser manipulados mediante operaciones básicas como GET, POST, PUT y DELETE.

Node.js es un entorno de ejecución de JavaScript orientado a eventos, ideal para desarrollar aplicaciones de red escalables. Junto con Express, un framework minimalista para Node.js, se facilita la creación de servidores web y APIs con una estructura limpia y modular.

Las pruebas automatizadas son fundamentales en el desarrollo de software moderno. Jest es una librería de pruebas desarrollada por Meta que permite crear y ejecutar pruebas unitarias o de integración de forma rápida. Supertest se usa en conjunto con Jest para simular peticiones HTTP a servidores Express, permitiendo validar las respuestas y el comportamiento de la API.

El uso de ESLint, una herramienta de análisis estático, permite mantener la calidad del código, asegurando el cumplimiento de convenciones de estilo y detectando errores comunes. Además, el concepto de almacenamiento en memoria permite simular una base de datos temporal mediante estructuras de datos locales (como arreglos), útil en fases tempranas del desarrollo y pruebas.

Estos elementos conforman una base sólida para el desarrollo profesional de software backend, fomentando la calidad, mantenibilidad y fiabilidad del código.

4. DESCRIPCIÓN DEL PROCEDIMIENTO:

Paso 1: Se inició la práctica creando una estructura organizada del proyecto. Esta estructura incluyó carpetas específicas para separar responsabilidades: src para la lógica principal de la aplicación, routes para definir las rutas de la API, controllers para manejar la lógica de negocio, test para las pruebas automatizadas, y archivos de configuración como package.json y eslint.config.js. Esta organización permitió mantener el código limpio, escalable y fácil de mantener.

Paso 2: Se procedió a configurar el entorno de desarrollo utilizando Node.js y NPM. Se ejecutó el comando npm init -y para generar el archivo package.json, y se instalaron las dependencias necesarias para el desarrollo y pruebas. Entre ellas, express como framework principal para construir la API, jest para realizar pruebas unitarias, supertest para pruebas de endpoints HTTP, y eslint para el análisis estático del código.

Paso 3: Se creó el archivo user.controller.js dentro de la carpeta controllers, donde se implementó la lógica para gestionar usuarios. Se utilizó un arreglo en memoria para simular una base de datos. Se desarrollaron dos funciones: una para listar los usuarios y otra para crear nuevos usuarios. Esta última incluye validaciones básicas para asegurar que se proporcionen el nombre y el correo antes de guardar el usuario, y ambas funciones fueron exportadas mediante module.exports.

Paso 4: En la carpeta routes se creó el archivo user.routes.js, donde se definieron las rutas de la API usando Express Router. Se configuraron dos rutas principales: GET /users para obtener todos los usuarios y POST /users para crear uno nuevo. Ambas rutas se conectaron con las funciones correspondientes del controlador, y finalmente se exportó el router para ser utilizado en el archivo principal de la aplicación.

Paso 5: El archivo principal app.js se ubicó en la carpeta src. Allí se importaron los módulos de Express y las rutas definidas. Se creó una instancia de la aplicación Express, se añadió el middleware express.json() para manejar el cuerpo de las solicitudes en formato JSON, y se enlazó el router a la ruta base /users. También se añadió un manejador para responder con un error 404 en caso de rutas no definidas, y se exportó la instancia app para facilitar su uso en pruebas.

Paso 6: En la carpeta test se creó el archivo user.test.js, donde se escribieron pruebas automatizadas con Jest y Supertest para validar el funcionamiento de la API. Se probó que GET /users devolviera una lista vacía inicialmente, que POST /users permitiera crear un usuario correctamente, y que se rechazaran las solicitudes incompletas. También se añadió una prueba para asegurar que se responda correctamente con un error 404 cuando se accede a una ruta no existente.

Paso 7: Se ejecutaron todas las pruebas mediante el comando npm test, y posteriormente se verificó la cobertura de las pruebas utilizando npx jest --coverage. Se analizaron los resultados y se añadieron pruebas adicionales hasta alcanzar una cobertura mayor al 90%, lo cual garantiza que la mayoría del código ha sido evaluado por las pruebas, aumentando así la fiabilidad del sistema.

Paso 8: Finalmente, se configuró ESLint para asegurar la calidad del código. Se creó el archivo eslint.config.js, donde se especificaron reglas de estilo recomendadas y se definieron los entornos y rutas de los archivos a analizar. Se ejecutó ESLint para detectar errores de sintaxis o estilo y se realizaron los ajustes necesarios en el código, asegurando que este sea consistente y conforme a las buenas prácticas de desarrollo.

5. ANÁLISIS DE RESULTADOS:

Para el análisis de resultados se registraron los datos obtenidos en las diferentes pruebas realizadas, procesando los resultados para evaluar el funcionamiento y la calidad del sistema. Se presentan a continuación tablas que resumen las pruebas ejecutadas antes y después de agregar casos adicionales

Tabla 1. Resultados de pruebas ejecutadas inicialmente

Prueba	Estado	Descripción
GET /users (lista vacía)	Pasó	La lista de usuarios inicialmente está vacía.
POST /users (crear usuario válido)	Pasó	Se crea correctamente un usuario con ID generado.
POST /users (datos incompletos)	Pasó	Fallo esperado si faltan datos obligatorios.

Tabla 2. Resultados después de pruebas adicionales

Prueba	Estado	Descripción
GET /users (lista con usuarios)	Pasó	La lista refleja los usuarios creados previamente.
POST /users (falta campo name)	Pasó	Se retorna error 400 con mensaje indicando campos obligatorios faltantes.
GET /noexiste (ruta inexistente)	Pasó	Se retorna error 404 con mensaje de "Ruta no encontrada".

Tabla 3. Cobertura de pruebas (Antes y Después)

Métrica	Antes (%)	Después (%)
Declaraciones (Stmts)	96	100
Ramas (Branches)	100	100
Funciones (Funcs)	66.66	100
Líneas (Lines)	96	100

Al ejecutar las pruebas iniciales, se confirmó el correcto funcionamiento básico de la API: creación y listado de usuarios, así como manejo de datos incompletos. Sin embargo, la cobertura parcial en funciones evidenció que algunos casos no estaban contemplados en las pruebas. La incorporación de pruebas adicionales para listar usuarios después de su creación, validar errores por campos faltantes y verificar respuestas ante rutas inexistentes permitió alcanzar cobertura completa del código.

LINTER (ESLint)

Para asegurar la calidad del código y el cumplimiento de buenas prácticas, se ejecutó el análisis con ESLint. A continuación, se resumen los errores y advertencias encontrados, organizados en tablas por tipo y archivo afectado, y se analiza su impacto.

Tabla 4. Resumen general de resultados de ESLint

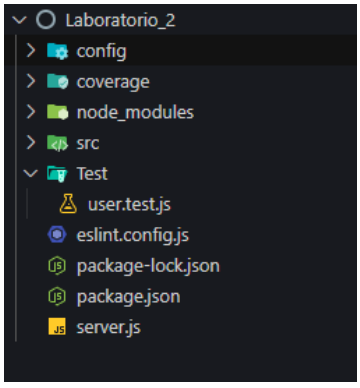
Elemento Analizado	Tipo de Problema	Descripción	Cantidad	Archivos Afectados	Solución
Errores no-undef	Error	Uso de require o module sin entorno Node	6	app.js, user.routes.js, user.controller.js	Agregar env: { node: true } en .eslintrc
Errores quotes	Error	Comillas dobles en lugar de simples	5	user.routes.js, user.controller.js	Usar comillas simples o aplicar --fix
Advertencia eslint-disable no usada	Advertencia	Directiva innecesaria, no hay reglas desactivadas	3	Archivos en /coverage/lcov-report/	Eliminar o ignorar archivos de cobertura
Problemas corregibles con --fix	Corrección rápida	Problemas de estilo (como comillas)	8	Principalmente en archivos fuente (src/)	Ejecutar npx eslint . --fix
Archivos externos generados	Ruido en ESLint	Archivos /coverage/ reportan advertencias	3	block-navigation.js, prettify.js, sorter.js	Excluir carpeta coverage/ en .eslintignore

El análisis con ESLint detectó errores principalmente por falta de configuración del entorno Node.js, errores de estilo en comillas, y advertencias por directivas innecesarias en archivos generados de cobertura. También se identificaron problemas corregibles automáticamente y archivos que deberían excluirse del análisis. La mayoría de los errores pueden solucionarse ajustando la configuración y aplicando correcciones automáticas.

6. GRÁFICOS O FOTOGRAFÍAS:

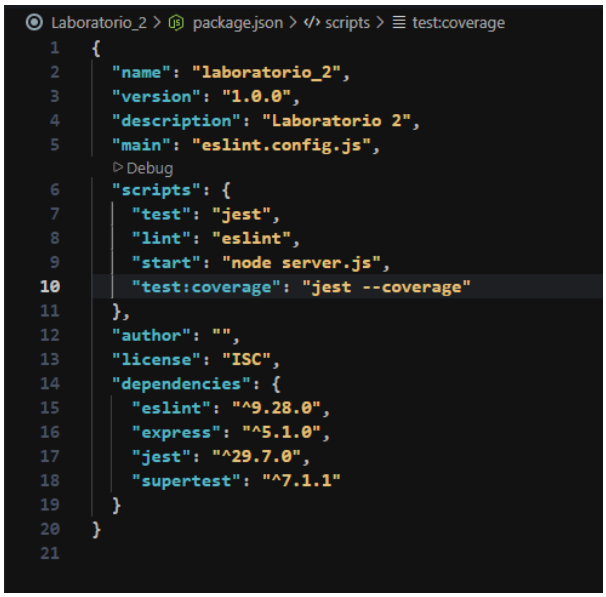
Estructura de carpetas

Se muestra la organización de archivos y carpetas en el proyecto, lo que facilita la navegación y mantenimiento del código.



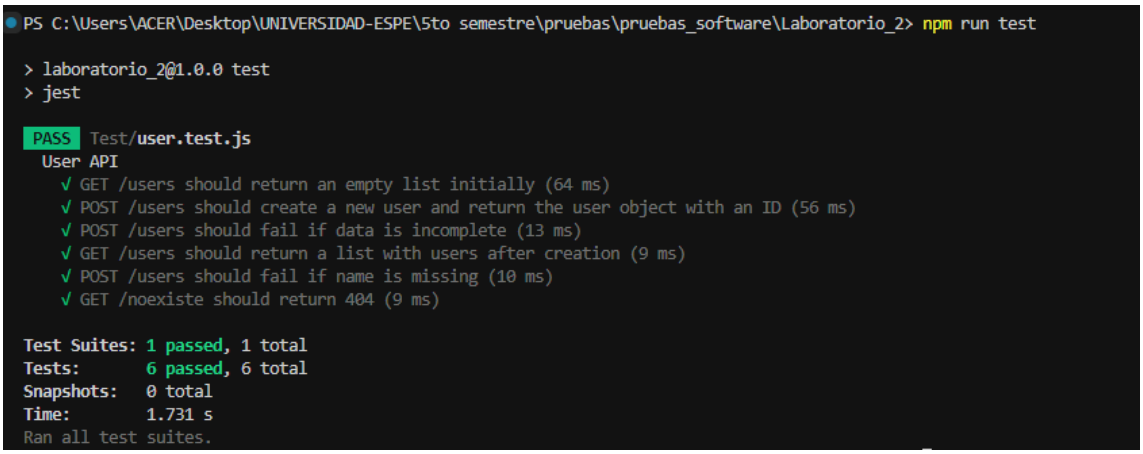
Dependencias instaladas

Captura del archivo package.json con las librerías y herramientas utilizadas para las pruebas y análisis estático.



Resultado de la ejecución de pruebas con npm run test

Se evidencia que todas las pruebas unitarias y funcionales se ejecutaron exitosamente, garantizando la funcionalidad del sistema.



Resultado de la ejecución de pruebas con cobertura usando npm run test:coverage

Reporte que muestra un alto porcentaje de cobertura de código, indicando la calidad y alcance de las pruebas realizadas.

```
PS C:\Users\ACER\Desktop\UNIVERSIDAD-ESPE\5to semestre\pruebas\pruebas_software\Laboratorio_2> npm run test:coverage

> laboratorio_2@1.0.0 test:coverage
> jest --coverage

 PASS  Test/user.test.js
  User API
    √ GET /users should return an empty list initially (43 ms)
    √ POST /users should create a new user and return the user object with an ID (47 ms)
    √ POST /users should fail if data is incomplete (11 ms)
    √ GET /users should return a list with users after creation (9 ms)
    √ POST /users should fail if name is missing (10 ms)
    √ GET /noexiste should return 404 (16 ms)

----------------------|---------|----------|---------|---------|-------------------
File                  | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
----------------------|---------|----------|---------|---------|-------------------
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:       6 passed, 6 total
Snapshots:   0 total
Time:        1.626 s, estimated 2 s
Ran all test suites.
```

Ejecución del comando npm run lint para análisis estático con ESLint

Se visualizan los errores y advertencias detectados en el código antes de aplicar correcciones.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL                    powershell - Laboratorio_2

C:\Users\ACER\Desktop\UNIVERSIDAD-ESPE\5to semestre\pruebas\pruebas_software\Laboratorio_2\coverage\lcov-report\block-navigation.js
  1:1  warning  Unused eslint-disable directive (no problems were reported)

C:\Users\ACER\Desktop\UNIVERSIDAD-ESPE\5to semestre\pruebas\pruebas_software\Laboratorio_2\coverage\lcov-report\prettify.js
  1:1  warning  Unused eslint-disable directive (no problems were reported)

C:\Users\ACER\Desktop\UNIVERSIDAD-ESPE\5to semestre\pruebas\pruebas_software\Laboratorio_2\coverage\lcov-report\sorter.js
  1:1  warning  Unused eslint-disable directive (no problems were reported)

C:\Users\ACER\Desktop\UNIVERSIDAD-ESPE\5to semestre\pruebas\pruebas_software\Laboratorio_2\src\app.js
   1:17  error  'require' is not defined  no-undef
   2:20  error  'require' is not defined  no-undef
  19:1   error  'module' is not defined   no-undef

C:\Users\ACER\Desktop\UNIVERSIDAD-ESPE\5to semestre\pruebas\pruebas_software\Laboratorio_2\src\controllers\user.controller.js
  17:22  error  Strings must use singlequote  quotes
  31:1   error  'module' is not defined       no-undef

C:\Users\ACER\Desktop\UNIVERSIDAD-ESPE\5to semestre\pruebas\pruebas_software\Laboratorio_2\src\routes\user.routes.js
   1:17  error  'require' is not defined      no-undef
   1:25  error  Strings must use singlequote  quotes
   8:37  error  'require' is not defined      no-undef
   8:45  error  Strings must use singlequote  quotes
  11:12  error  Strings must use singlequote  quotes
  14:13  error  Strings must use singlequote  quotes
  17:1   error  'module' is not defined       no-undef

✖15 problems (12 errors, 3 warnings)
```

Resultado tras ejecutar npx eslint . –fix

Muestra la reducción significativa de errores y advertencias después de aplicar las correcciones automáticas con ESLint.

```
PS C:\Users\ACER\Desktop\UNIVERSIDAD-ESPE\5to semestre\pruebas\pruebas_software\Laboratorio_2> npm run lint


> laboratorio_2@1.0.0 lint
> eslint


C:\Users\ACER\Desktop\UNIVERSIDAD-ESPE\5to semestre\pruebas\pruebas_software\Laboratorio_2\src\app.js
   1:17  error  'require' is not defined  no-undef
   2:20  error  'require' is not defined  no-undef
  19:1   error  'module' is not defined   no-undef

C:\Users\ACER\Desktop\UNIVERSIDAD-ESPE\5to semestre\pruebas\pruebas_software\Laboratorio_2\src\controllers\user.controller.js
  31:1  error  'module' is not defined  no-undef

C:\Users\ACER\Desktop\UNIVERSIDAD-ESPE\5to semestre\pruebas\pruebas_software\Laboratorio_2\src\routes\user.routes.js
   1:17  error  'require' is not defined  no-undef
   8:37  error  'require' is not defined  no-undef
  17:1   error  'module' is not defined   no-undef

✖ 7 problems (7 errors, 0 warnings)
```

7. DISCUSIÓN:

Este laboratorio permitió aplicar conceptos clave de la asignatura de pruebas de software, como pruebas automatizadas, cobertura de código y análisis estático. Se logró un 100% de cobertura con Jest, lo cual valida el correcto funcionamiento de las rutas y controladores implementados. Además, el uso de ESLint ayudó a identificar errores comunes y mejorar la calidad del código siguiendo buenas prácticas.

La comparación entre teoría y práctica demuestra que las herramientas automatizadas son esenciales para asegurar la fiabilidad y mantenibilidad del software, alineándose con los principios de verificación y validación vistos en clase.

8. CONCLUSIONES:

Durante el desarrollo del laboratorio se cumplieron todos los objetivos propuestos. Se implementó una API REST básica utilizando Node.js y Express, con una estructura organizada en carpetas, lo que facilitó la mantenibilidad del proyecto. Se desarrollaron pruebas automatizadas con Jest, permitiendo verificar el correcto funcionamiento de las rutas y el flujo de datos ante diferentes entradas, incluyendo casos válidos e inválidos.

Se simuló correctamente el almacenamiento en memoria, cumpliendo con la funcionalidad esperada de una base de datos local temporal. La ejecución del comando jest --coverage permitió alcanzar una cobertura del 100%, lo que refleja un adecuado nivel de validación del sistema.

Finalmente, se aplicaron buenas prácticas de desarrollo utilizando ESLint, lo que permitió identificar errores comunes y mejorar la calidad del código. En conjunto, este laboratorio permitió comprender de forma práctica el ciclo completo de desarrollo, prueba y verificación de una API funcional.

9. BIBLIOGRAFÍA:

Documentation. (s/f). Eslint.org. Recuperado el 31 de mayo de 2025, de https://eslint.org/docs/latest/

Getting started. (s/f). Jestjs.Io. Recuperado el 31 de mayo de 2025, de https://jestjs.io/docs/getting-started

Index. (s/f). Nodejs.org. Recuperado el 31 de mayo de 2025, de https://nodejs.org/docs/latest/api/

10. Anexos

https://github.com/JairoBonilla2004/Pruebas-de-software/tree/main