Universidad de las Fuerzas Armadas - ESPE

Nombre: Abner Arboleda

NRC: 27854

Fecha: 13/11/2025

EXAMEN PRÁCTICO (7 puntos)

Tema: Verificación y Validación en API REST – Node.js + Express + ESLint + Jest/Supertest
Entrega: Proyecto + PDF de evidencias (separados)
Nombre del proyecto:
vvexamen-NombreApellido
Prefijo de endpoints:
/api/nombreapellido/users

```
const express = require('express');
const userRoutes = require('./routes/user.routes');

const app = express(); // Crea una instancia de la aplicación Express

// Middleware para parsear JSON del cuerpo de las solicitudes
app.use(express.json());

app.use('/api/abnerarboleda/users', userRoutes);

// Manejador de rutas no encontradas (404)
app.use((req, res) => {
  res.status(404).json({
    message: 'Route not found',
    path: req.originalUrl,
  });
});

module.exports = app;
```

Indicaciones generales

- Tiempo: 1 hora
- El código, nombres, endpoints y capturas **deben contener el nombre del estudiante**.
- Si el nombre no aparece → **0 puntos en ese numeral**.
- Deben entregar:
    - Proyecto comprimido (VVExamen-NombreApellido.zip)
    - PDF con evidencias (ApellidoNombre_Examen.pdf)
- Sin carpeta *node_modules*
- Pruebas obligatorias con Jest y Supertest

Tareas obligatorias del examen (7 puntos)

1. Endpoints básicos (2.0 pts)

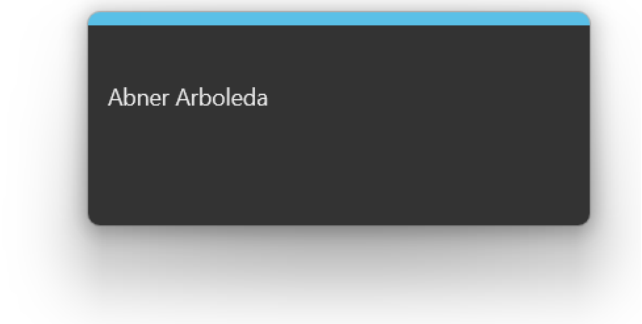
Crear los 4 endpoints siguientes:

Acción	Ruta	Descripción
GET	/api/nombreapellido/users	Devuelve todos los usuarios
POST	/api/nombreapellido/users	Crea usuario {name, email}
GET	/api/nombreapellido/users/:id	Devuelve usuario por ID
DELETE	/api/nombreapellido/users/:id	Elimina usuario por ID

Requisitos:

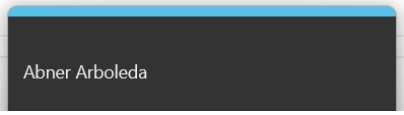
- ID generado con Date.now() o UUID

```
  }
  // Crear usuario
  const newUser = {
    id: Date.now(),
    name,
    email,
  };

  users.push(newUser);
  res.status(201).json(newUser);
}
```



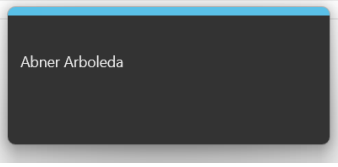
- Almacén: arreglo en memoria

```
// Arreglo en memoria
let users = [];

function errorResponse(status, message, path) {
  return {
```



- Si ID no existe → 404 con JSON {message, path}

```
//Obtener un usuario por ID
function getUserById(req, res) {
  const user = users.find((u) => u.id === parseInt(req.params.id));
  if (!user) {
    return res.status(404).json(errorResponse(404, 'Usuario no encontrado', req.originalUrl));
  }
  res.json(user);
}
```

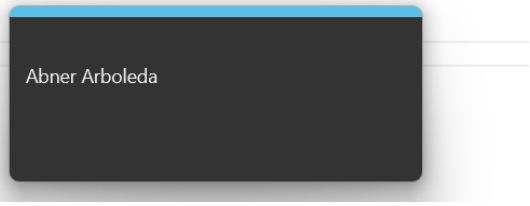


Evidencia PDF: Capturas claras del código cada endpoint.

Puntaje: 2.0 pts

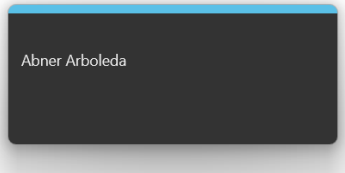
Get de todos los usuarios

```
}

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

//Crear un nuevo usuario
```



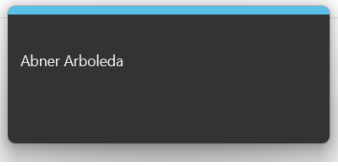
Post Crear usuario

```
//Crear un nuevo usuario
function createUser(req, res) {
  const { name, email } = req.body;
  // Validación de email obligatorio
  if (!email) {
    return res.status(400).json(errorResponse(400, 'Email es requerido', req.originalUrl));
  }
  // Validación de que el nombre no puede ser un número y es obligatorio
  if (!name || !isNaN(name)) {
    return res.status(400).json(errorResponse(400, 'Nombre es requerido y no puede ser un numero', req.originalUrl));
  }
  // Validación de email debe ser único
  const emailExists = users.find((u) => u.email === email);
  if (emailExists) {
    return res.status(409).json(errorResponse(409, 'Email ya existe', req.originalUrl));
  }
  // Crear usuario
  const newUser = { id: Date.now(), name, email };
  users.push(newUser);
  res.status(201).json(newUser);
}
```



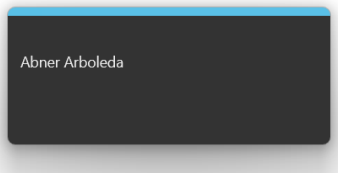
Get obtener usuario por id

```
//Obtener un usuario por ID
function getUserById(req, res) {
  const user = users.find((u) => u.id === parseInt(req.params.id));
  if (!user) {
    return res.status(404).json(errorResponse(404, 'Usuario no encontrado', req.originalUrl));
  }
  res.json(user);
}
```



Delete eliminar usuario por id

```
//Eliminar un usuario por ID
function deleteUser(req, res) {
  const index = users.findIndex((u) => u.id === parseInt(req.params.id));
  if (index === -1) {
    return res.status(404).json(errorResponse(404, 'Usuario no encontrado', req.originalUrl));
  }
  const deleted = users.splice(index, 1);
  res.json(deleted[0]);
}
```



Endpoints

```
const express = require('express');
const {
  getAllUsers,
  createUser,
  getUserById,
  deleteUser,
} = require('../controllers/user.controller');

const router = express.Router();

// Ruta GET para obtener todos los usuarios
router.get('/', getAllUsers);

// Ruta POST para crear un nuevo usuario
router.post('/', createUser);

// Ruta GET para obtener un usuario por ID
router.get('/:id', getUserById);

// Ruta DELETE para eliminar un usuario por ID
router.delete('/:id', deleteUser);

module.exports = router;
```

```
const express = require('express');
const userRoutes = require('./routes/user.routes');

const app = express(); // Crea una instancia de la aplicación Express

// Middleware para parsear JSON del cuerpo de las solicitudes
app.use(express.json());

app.use('/api/abnerarboleda/users', userRoutes);

// Manejador de rutas no encontradas (404)
app.use((req, res) => {
  res.status(404).json({
    message: 'Route not found',
    path: req.originalUrl,
  });
});

module.exports = app;
```

2. Reglas avanzadas de ESLint (1.5 pts)

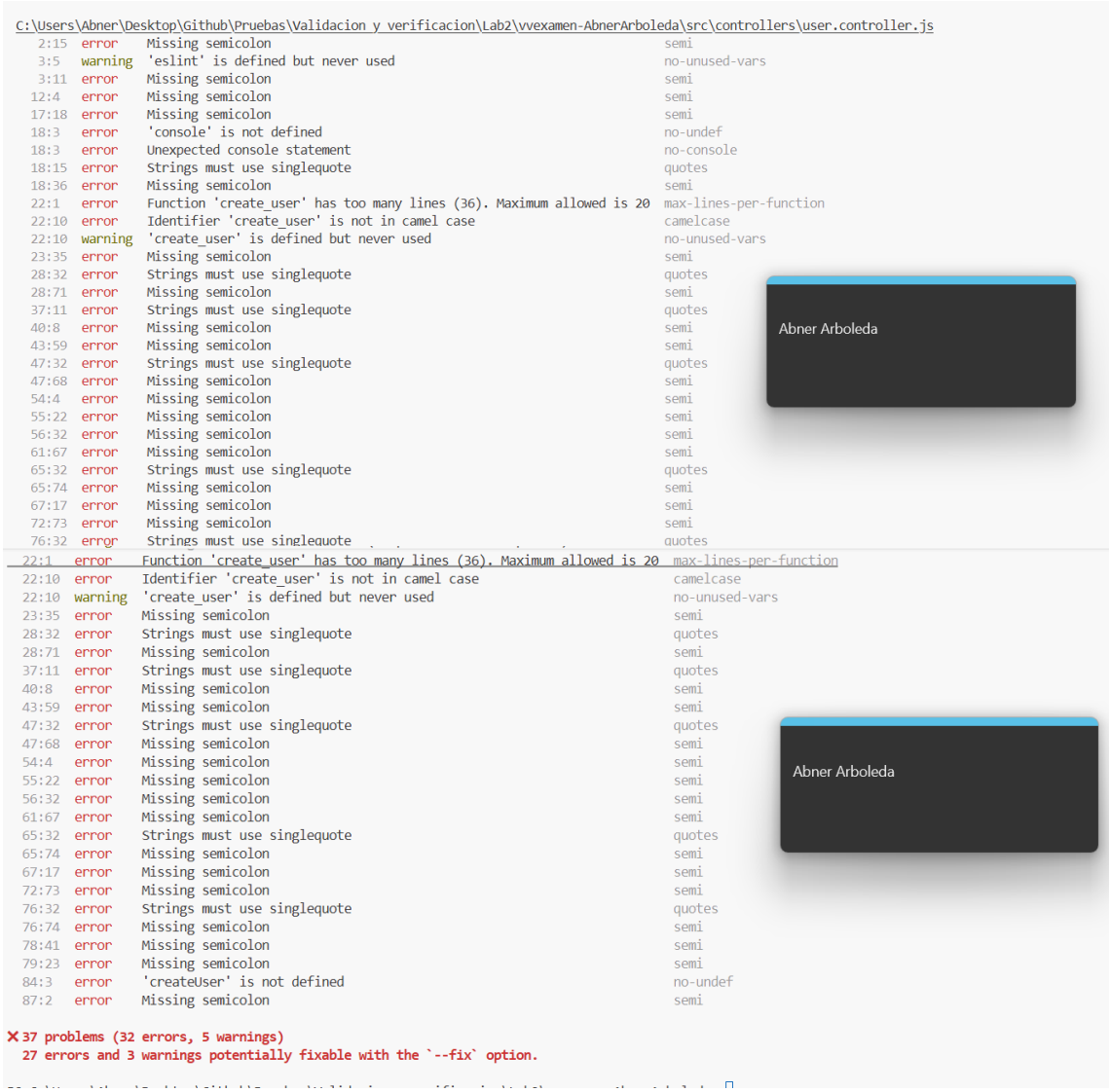
Configurar eslint.config.js con las reglas:

- no-console
- eqeqeq
- camelcase
- no-unused-vars
- quotes: 'single'
- semi: 'always'
- max-lines-per-function: 20

```
const js = require('@eslint/js');

module.exports = [
  {
    files: ['src/**/*.js'],
    languageOptions: {
      ecmaVersion: 2021,
      sourceType: 'commonjs',
    },
    rules: {
      ...js.configs.recommended.rules,
      'no-console': 'error',
      eqeqeq: 'error',
      camelcase: 'error',
      'no-unused-vars': 'warn',
      quotes: ['error', 'single'],
      semi: ['error', 'always'],
      'max-lines-per-function': ['error', 20],
    },
  },
];
```


Evidencia PDF:

- Captura **ANTES** del lint (con al menos un error por cada regla)



- Captura **DESPUÉS** del lint (sin errores)

Puntaje: 1.5 pts

3. Validación + manejo de errores (1.5 pts)

Reglas:

- email obligatorio → si falta: 400
- email debe ser único → 409
- name no puede ser un número → 400

Respuestas de error deben incluir:

{

"timestamp": "...",

"status": 400|409,

"message": "...",

"path": "/api/nombreapellido/users/..."

}

```
//Respuesta de Error
function errorResponse(status, message, path) {
  return {
    timestamp: new Date().toISOString(),
    status,
    message,
    path,
  };
}
```


Evidencias PDF (3 capturas):

- Error 409 (email duplicado)

```
  );
}
// Validación de email debe ser único
const emailExists = users.find((u) => u.email === email);
if (emailExists) {
  return res
    .status(409)
    .json(errorResponse(409, 'Email ya existe', req.originalUrl));
}
// Crear usuario
```


- Error 400 (email vacío)

```
const { name, email } = req.body;
// Validación de email obligatorio
if (!email) {
  return res
    .status(400)
    .json(errorResponse(400, 'Email es requerido', req.originalUrl));
}
```


- Error 400 (name inválido)

```
// Validación de que el nombre no puede ser un número
if (!name || !isNaN(name)) {
  return res
    .status(400)
    .json(
      errorResponse(
        400,
        'Name es requerido y no puede ser un numero',
        req.originalUrl
      )
    );
}
```


Puntaje: 1.5 pts

4. Pruebas unitarias (2.0 pts)

Escribir mínimo **6 pruebas completas** usando Jest + Supertest:

Cobertura mínima obligatoria:

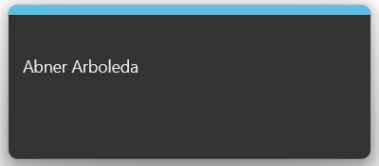
Evidencias PDF:

- Captura del código de las pruebas generadas
  - ✓ Crear usuario válido

```
// Crear usuario válido
test('POST - should create a valid user - Abner Arboleda', async () => {
  const newUser = {
    name: 'Abner Arboleda',
    email: 'abnerarboleda@ejemplo.com',
  };
  const res = await request(app).post(baseUrl).send(newUser);
  expect(res.statusCode).toBe(201);
  expect(res.body).toHaveProperty('id');
  expect(res.body.name).toBe('Abner Arboleda');
  expect(res.body.email).toBe('abnerarboleda@ejemplo.com');
});
```

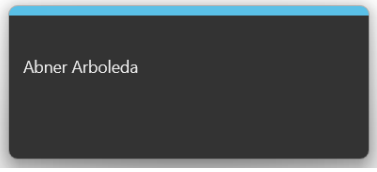
  - ✓ Error por datos inválidos

```
// Error por datos inválidos (email vacío)
test('POST - should return 400 if email is missing', async () => {
  const res = await request(app).post(baseUrl).send({ name: 'Test' });
  expect(res.statusCode).toBe(400);
  expect(res.body).toHaveProperty('status', 400);
  expect(res.body).toHaveProperty('message', 'Email es requerido');
  expect(res.body).toHaveProperty('timestamp');
  expect(res.body).toHaveProperty('path');
});

// Error por nombre inválido (número)
test('POST - should return 400 if name is a number', async () => {
  const res = await request(app)
    .post(baseUrl)
    .send({ name: '123', email: 'test@example.com' });
  expect(res.statusCode).toBe(400);
  expect(res.body).toHaveProperty('status', 400);
  expect(res.body).toHaveProperty('message');
});

// Error por email duplicado (409)
test('POST - should return 409 if email already exists', async () => {
  const user = { name: 'Usuario Uno', email: 'duplicado@test.com' };
  await request(app).post(baseUrl).send(user);
  const res = await request(app).post(baseUrl).send(user);
  expect(res.statusCode).toBe(409);
  expect(res.body).toHaveProperty('status', 409);
  expect(res.body).toHaveProperty('message', 'Email ya existe');
});
```



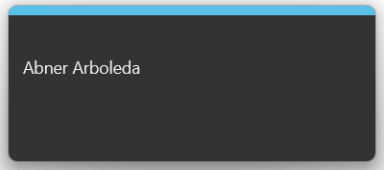
  - ✓ Listado de usuarios

```
// Listado de usuarios
test('GET - should return list of users - Abner Arboleda', async () => {
  const res = await request(app).get(baseUrl);
  expect(res.statusCode).toBe(200);
  expect(Array.isArray(res.body)).toBe(true);
});
```



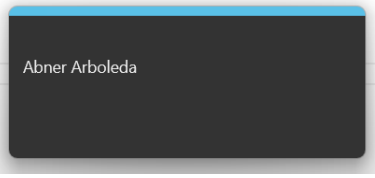
- ✓ Obtener usuario por ID

```
// Obtener usuario por ID
test('GET/:id - should return user by ID - Abner Arboleda', async () => {
  const newUser = {
    name: 'Test User',
    email: 'testuser@example.com',
  };
  const createRes = await request(app).post(baseUrl).send(newUser);
  const userId = createRes.body.id;

  const res = await request(app).get(`${baseUrl}/${userId}`);
  expect(res.statusCode).toBe(200);
  expect(res.body).toHaveProperty('id', userId);
  expect(res.body.name).toBe('Test User');
});
```



- ✓ Error al obtener usuario inexistente

```
// Error al obtener usuario inexistente
test('GET/:id - should return 404 for non-existent user', async () => {
  const res = await request(app).get(`${baseUrl}/10`);
  expect(res.statusCode).toBe(404);
  expect(res.body).toHaveProperty('status', 404);
  expect(res.body).toHaveProperty('message', 'Usuario no encontrado');
});
```



- ✓ Eliminar usuario (y verificar que ya no existe)

```
// Eliminar usuario y verificar que ya no existe
test('DELETE/:id - should delete user - Abner Arboleda', async () => {
  const newUser = {
    name: 'Abner Arboledaa',
    email: 'adarboleda@example.com',
  };
  const createRes = await request(app).post(baseUrl).send(newUser);
  const userId = createRes.body.id;

  const deleteRes = await request(app).delete(`${baseUrl}/${userId}`);
  expect(deleteRes.statusCode).toBe(200);
  expect(deleteRes.body).toHaveProperty('id', userId);

  const getRes = await request(app).get(`${baseUrl}/${userId}`);
  expect(getRes.statusCode).toBe(404);
});
```

- Captura del resultado exitoso de npm test

```
PS C:\Users\Abner\Desktop\Github\Pruebas\Validacion y verificacion\Lab2\vvexamen-AbnerArboleda> npm test

> lab2@1.0.0 test
> jest

 PASS  test/user.test.js
  User API - Abner Arboleda
    √ POST - should create a valid user - Abner Arboleda (67 ms)
    √ POST - should return 400 if email is missing (10 ms)
    √ POST - should return 400 if name is a number (6 ms)
    √ POST - should return 409 if email already exists (17 ms)
    √ GET - should return list of users - Abner Arboleda (10 ms)
    √ GET/:id - should return user by ID - Abner Arboleda (18 ms)
    √ GET/:id - should return 404 for non-existent user (13 ms)
    √ DELETE/:id - should delete user - Abner Arboleda (27 ms)

Test Suites: 1 passed, 1 total
Tests:       8 passed, 8 total
Snapshots:   0 total
Time:        1.574 s
Ran all test suites.
```

Puntaje: 2.0 pts

Rúbrica (Total 7 pts)

Criterio	Pts
4 endpoints implementados correctamente	2.0
Reglas ESLint avanzadas funcionando	1.5
Validaciones + manejo de errores	1.5
Pruebas unitarias (6) completas	2.0
Total	**7.0 pts**