

Mini proyecto

Juan Sebastian Yepes Sanchez

Ingeniera de Software

Universidad Manuela Beltrán

Ingeniería web I

Diana Marcela Toquica

Ingeniera de Sistemas

2025

Tabla de contenido	
<b>Introducción .....</b>	<b>3</b>
<b>Objetivo.....</b>	<b>3</b>
<b>Informe.....</b>	<b>3</b>
<b>Definir el alcance y arquitectura del proyecto: .....</b>	<b>3</b>
<b>Desarrollar los servicios web: .....</b>	<b>4</b>
<b>Configurar integración básica:.....</b>	<b>6</b>
<b>Implementar integración continua:.....</b>	<b>9</b>
<b>Bibliografía .....</b>	<b>13</b>

## **Introducción**

El desarrollo actual de aplicaciones web es imprescindible para la comunicación entre el cliente y el servidor. Las API RESTful hacen más fácil la interoperabilidad y la escalabilidad de los sistemas. El objetivo de este proyecto es crear una API RESTful para el registro y gestión de usuarios utilizando buenas prácticas de programación, seguridad y arquitectura junto con herramientas modernas para el control de versiones de integración continua.

La solución está desarrollada utilizando Node.js y Express.js, dos tecnologías ampliamente utilizadas para la creación de servicios web debido a su efectividad y versatilidad. Además, se incluyen mecanismos de validación de datos básica autenticación, y gestión de errores, asegurando la calidad y confiabilidad de los servicios prestados. Y se incluye la gestión de errores, garantizando la calidad y confiabilidad de los servicios prestados.

Se utilizan un sistema de control de versiones y GitHub como plataforma de colaboración y automatización para garantizar la legibilidad y el mantenimiento del código. Esto se logra configurando una canalización de integración continua (CI) mediante GitHub Actions, que permite la ejecución automática de pruebas unitarias y de integración antes de cada lanzamiento.

## **Objetivo**

- Permitir que los usuarios se registren desde una página web sencilla y que la información se procese y almacene en el servidor a través de una API REST.
- Configurar un entorno de desarrollo con Git y GitHub para el control de versiones y la automatización de pruebas mediante GitHub Actions.

## **Informe**

### **Definir el alcance y arquitectura del proyecto:**

El proyecto consiste en el desarrollo de una API RESTful para la gestión de usuarios en una aplicación web. Esta API permitirá realizar operaciones básicas relacionadas con el registro de usuarios, las cuales forman parte del CRUD (Create, Read, Update, Delete).

Funcionalidades incluidas:

Registro de nuevos usuarios mediante un formulario web.

Validación de datos enviados desde el cliente (nombre, correo y contraseña).

Almacenamiento temporal de los usuarios en memoria (array interno).

Visualización de los usuarios registrados a través de una ruta GET.

Comunicación entre el frontend (HTML y JavaScript) y el backend mediante peticiones HTTP usando fetch().

Para este proyecto se elige el patrón de arquitectura MVC, ya que permite organizar el código separando responsabilidades y facilita la escalabilidad del sistema.

Capa	Descripción	Ejemplo
Modelo (Model)	Gestiona los datos de los usuarios. En este caso, la lista de usuarios (array) o la futura conexión con una base de datos.	models/usuarioModel.js (si se crea más adelante)
Vista (View)	Interfaz del usuario. Es la página web (index.html) donde se llenan los datos del registro.	frontend/index.html
Controlador (Controller)	Contiene la lógica de negocio: recibe los datos del usuario, los valida y responde al cliente.	Funciones dentro de server.js o en controllers/usuarioController.js

### Desarrollar los servicios web:

El proyecto implementa una API RESTful utilizando Express.js, un framework moderno para Node.js, ampliamente usado en el desarrollo de servicios web por su simplicidad, rendimiento y compatibilidad con el estándar REST.

Los servicios desarrollados permiten la gestión de usuarios, específicamente el registro y la consulta de usuarios registrados, siguiendo buenas prácticas de calidad, seguridad y manejo de errores.

EndPoints implementados

Método	Ruta	Descripción
POST	/api/usuarios	Registra un nuevo usuario.
GET	/api/usuarios	Devuélvela la lista de usuarios registrados.
GET	/	Verifica que la API esté en funcionamiento.

En las siguientes imágenes podemos validar las respectivas pruebas de los EndPoints solicitados.

The screenshot shows the Postman interface with a successful API call. The URL is `http://localhost:3000`. The response status is `200 OK`, time is `7 ms`, and size is `303 B`. The response body is:

```
1 API de registro de usuarios funcionando ✓
```

The screenshot shows the Postman interface with a successful API call. The URL is `http://localhost:3000/api/usuarios`. The method is `POST`. The response status is `201 Created`, time is `5 ms`, and size is `412 B`. The response body is:

```
1 {"mensaje": "Usuario registrado correctamente", "usuario": {"id": 2, "nombre": "Juan Pérez", "correo": "juan@example.com", "contraseña": "123456"}}
```

The screenshot shows a Postman interface with the following details:

- Header bar: JS usuarioController.js, JS usuarioRoutes.js, JS server.js, HTTP http://localhost:3000/api/usuarios, JS auth.js.
- Request URL: http://localhost:3000/api/usuarios
- Method: POST
- Body tab selected, showing raw JSON input:

```
1 [  
2   {"nombre": "Juan Pérez",  
3   "correo": "juan@example.com",  
4   "contraseña": "123456"  
5 ]
```
- Response status: Status: 201 Created Time: 5 ms Size: 412 B
- ResponseBody: [{"mensaje": "Usuario registrado correctamente", "usuario": {"id": 2, "nombre": "Juan Pérez", "correo": "juan@example.com", "contraseña": "123456"}}]

The screenshot shows a Postman interface with the following details:

- Header bar: JS usuarioRoutes.js, HTTP http://localhost:3000/api/usuarios.
- Request URL: http://localhost:3000/api/usuarios
- Method: GET
- Authorization tab selected, showing Basic Auth configuration with Username: admin and Password: 1234.
- Response status: Status: 200 OK Time: 3 ms Size: 343 B
- ResponseBody: [{"id": 1, "nombre": "juan", "correo": "juan@seguro.com", "contraseña": "123456"}]

La API cumple con las buenas prácticas de diseño REST y seguridad, asegurando la integridad y coherencia de los datos procesados. Las pruebas se realizaron con la herramienta Postman y el formulario web conectado al backend, verificando respuestas correctas ante datos válidos y manejo adecuado de errores en entradas inválidas o accesos no autorizados.

### Configurar integración básica:

Para llevar un control de los cambios del proyecto, se inicializó un repositorio Git dentro de la carpeta del proyecto con el siguiente comando lo realizamos en un terminal: **git init**

Luego se configuró la identidad del desarrollador:

```
git config --global user.name "Nombre"  
git config --global user.email "Correo electronico"
```

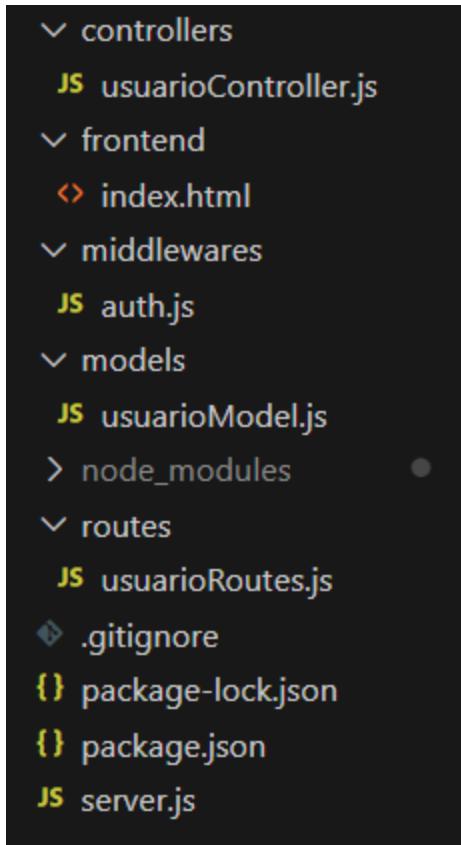
Se realizó el primer seguimiento de archivos y commit:

```
git add .  
git commit -m "Inicialización del proyecto con API REST de usuarios"
```

Esto permitió comenzar a registrar todas las versiones del proyecto y mantener un historial de cambios seguro y ordenado.

El proyecto se organizó en carpetas según las buenas prácticas de desarrollo de APIs REST con Express.js:

/controllers	→ Controladores de la lógica de negocio
/frontend	→ Página web de interacción
/middlewares	→ Middleware de autenticación
/models	→ Modelos de datos
/routes	→ Definición de rutas de la API
.gitignore	→ Archivo para excluir dependencias y logs
package.json	→ Configuración del proyecto y dependencias
server.js	→ Punto de entrada principal del servidor



Dentro del archivo package.json se agregaron scripts para simplificar la instalación y ejecución del proyecto.

Estos comandos permiten iniciar el servidor y realizar pruebas de forma rápida:

```
"scripts": {  
  "start": "node server.js",  
  "dev": "nodemon server.js",  
  "test": "echo \"Ejecutando pruebas automáticas\" && exit 0"  
}
```

```
{
  "name": "nueva-carpetita",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "node server.js",
    "dev": "nodemon server.js",
    "install": "npm install",
    "test": "echo \"Ejecutando pruebas automáticas (futuro)\" && exit 0"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "type": "commonjs",
  "dependencies": {
    "body-parser": "^2.2.0",
    "cors": "^2.8.5",
    "express": "^5.1.0",
    "express-validator": "^7.3.0"
  },
  "devDependencies": {
    "nodemon": "^3.1.10"
  }
}
```

Para ejecutar los comandos:

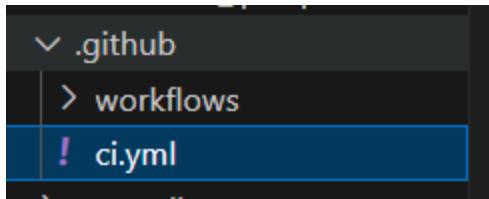
- npm install → Instala las dependencias del proyecto.
- npm run dev → Inicia el servidor en modo desarrollo.
- npm test → Ejecuta pruebas automáticas o simuladas.

### **Implementar integración continua:**

Para la integración continua se utilizó GitHub Actions, una herramienta que permite automatizar flujos de trabajo directamente desde el repositorio de GitHub.

Se creó el archivo de configuración del pipeline en la ruta:

.github/workflows/ci.yml



Este archivo define el proceso automatizado que se ejecuta cada vez que se realiza un **push** o **pull request** en la rama principal del repositorio.

## Automatización de pruebas y análisis de código

En el pipeline se configuraron las siguientes acciones automáticas:

1. **Instalación del entorno Node.js** (versión 18 o superior).
2. **Instalación de dependencias del proyecto** mediante npm install.
3. **Ejecución de pruebas automáticas** usando Jest, a través del script definido en el archivo package.json:

```
"test": "jest"
```

```
"scripts": {  
  "start": "node server.js",  
  "dev": "nodemon server.js",  
  "install": "npm install",  
  "test": "jest"
```

Con esta configuración, cada vez que se suben cambios al repositorio, GitHub ejecuta automáticamente las pruebas del proyecto y verifica que el código funcione correctamente.

El pipeline se configuró para ejecutarse en un entorno de pruebas proporcionado por GitHub Actions. De esta manera, se asegura que cualquier modificación en el código sea evaluada antes de integrarse a la versión principal del proyecto.

El archivo ci.yml contiene las siguientes etapas del flujo de integración continua:

```
name: Integración continua - API Usuarios
```

```
on:
```

```
  push:
```

```
    branches: [ main ]
```

```
  pull_request:
```

```
branches: [ main ]
```

```
jobs:
```

```
  build-and-test:
```

```
    runs-on: ubuntu-latest
```

```
steps:
```

```
  - name: ✨ Clonar el repositorio
```

```
    uses: actions/checkout@v3
```

```
  - name: 🛡 Configurar Node.js
```

```
    uses: actions/setup-node@v3
```

```
    with:
```

```
      node-version: '18'
```

```
  - name: 📦 Instalar dependencias
```

```
    run: npm install
```

```
  - name: 💚 Ejecutar pruebas
```

```
    run: npm test
```

```
  - name: ✅ Verificar formato de código
```

```
    run: npx eslint . || true
```

```
.github > ! ci.yml
1   name: Integración continua - API Usuarios
2
3   < on:
4     push:
5       branches: [ main ]
6     pull_request:
7       branches: [ main ]
8
9   < jobs:
10  build-and-test:
11    runs-on: ubuntu-latest
12
13  < steps:
14    - name: ✨ Clonar el repositorio
15      uses: actions/checkout@v3
16
17    - name: ⚙ Configurar Node.js
18      uses: actions/setup-node@v3
19    with:
20      node-version: '18'
21
22    - name: 📦 Instalar dependencias
23      run: npm install
24
25    - name: 💚 Ejecutar pruebas
26      run: npm test
27
28    - name: ✅ Verificar formato de código
29      run: npx eslint . || true
```

Este flujo garantiza que:

- El código se descargue desde GitHub.
- Se configure el entorno Node.js adecuado.
- Se instalen las dependencias del proyecto.
- Se ejecuten las pruebas automáticas de manera continua.

```
● PS C:\Users\Juany\OneDrive\Documentos\UMB 4 SEMESTRE\INGENIERIA WEB 1\ACTIVIDAD 3\Nueva carpeta> npm test
> nueva-carpeta@1.0.0 test
> jest

  PASS  tests/app.test.js
    ✓ Servidor responde correctamente (2 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        0.394 s
Ran all test suites.
⚡ PS C:\Users\Juany\OneDrive\Documentos\UMB 4 SEMESTRE\INGENIERIA WEB 1\ACTIVIDAD 3\Nueva carpeta>
```

## Bibliografía

- Flanagan, D. (2020). *JavaScript: The definitive guide: Master the world's most-used programming language* (7th ed.). O'Reilly Media.
- Chacon, S., & Straub, B. (2014). *Pro Git* (2nd ed.). Apress.