

Universidad Católica Boliviana “San Pablo”

Facultad de Ingeniería

Carrera de Ingeniería de Sistemas

Segunda Evaluación de Tecnologías Web II 1-2025

Estudiante: Christian Kevin Coronel Condori

Objetivo General

Diseñar e implementar un microservicio basado en WebSockets que consuma el endpoint POST /Proyectos del proyecto integrador, permitiendo la gestión en tiempo real de proyectos académicos y trabajos dirigidos, con procesamiento estadístico de datos.

1. Análisis del Proyecto Integrador

1.1 Identificación del Endpoint

- **Endpoint principal:**

POST /Proyectos

Propósito: Realizar búsquedas filtradas de proyectos por año, categoría o título.

1.2 Justificación de la Elección

- **Relevancia:**
 - Permite acceder a datos críticos para análisis estadísticos.
 - Es el único endpoint que proporciona la flexibilidad necesaria para filtrar proyectos académicos y trabajos dirigidos.
- **Impacto:** Facilita la generación de métricas en tiempo real para dashboards institucionales.

1.3 Descripción Técnica del Endpoint

Característica	Detalle
Método HTTP	POST
Body (JSON)	<div><pre>{ "anio": string null, "categoria": string null, "titulo": string null }</pre></div> <div><pre>1 { 2 "anio": "", 3 "categoria": "", 4 "titulo": "" 5 } 6</pre></div>

Característica	Detalle
Respuesta Exitosa	<p>Array de proyectos con estructura { titulo, categoria, date_part (año) }</p> <pre> 1 [2 { 3 "titulo": "Gestión Escolar", 4 "categoria": "Educación", 5 "date_part": 2025 6 }, 7 { 8 "titulo": "Sistema de Bodega", 9 "categoria": "Logística", 10 "date_part": 2025 11 }, 12 { 13 "titulo": "Atención Médica Online", 14 "categoria": "Salud", 15 "date_part": 2025 </pre>
Autenticación	No requerida (según configuración actual)

2. Diseño del Microservicio

2.1 Objetivo del Microservicio

Proveer un sistema de comunicación bidireccional en tiempo real para:

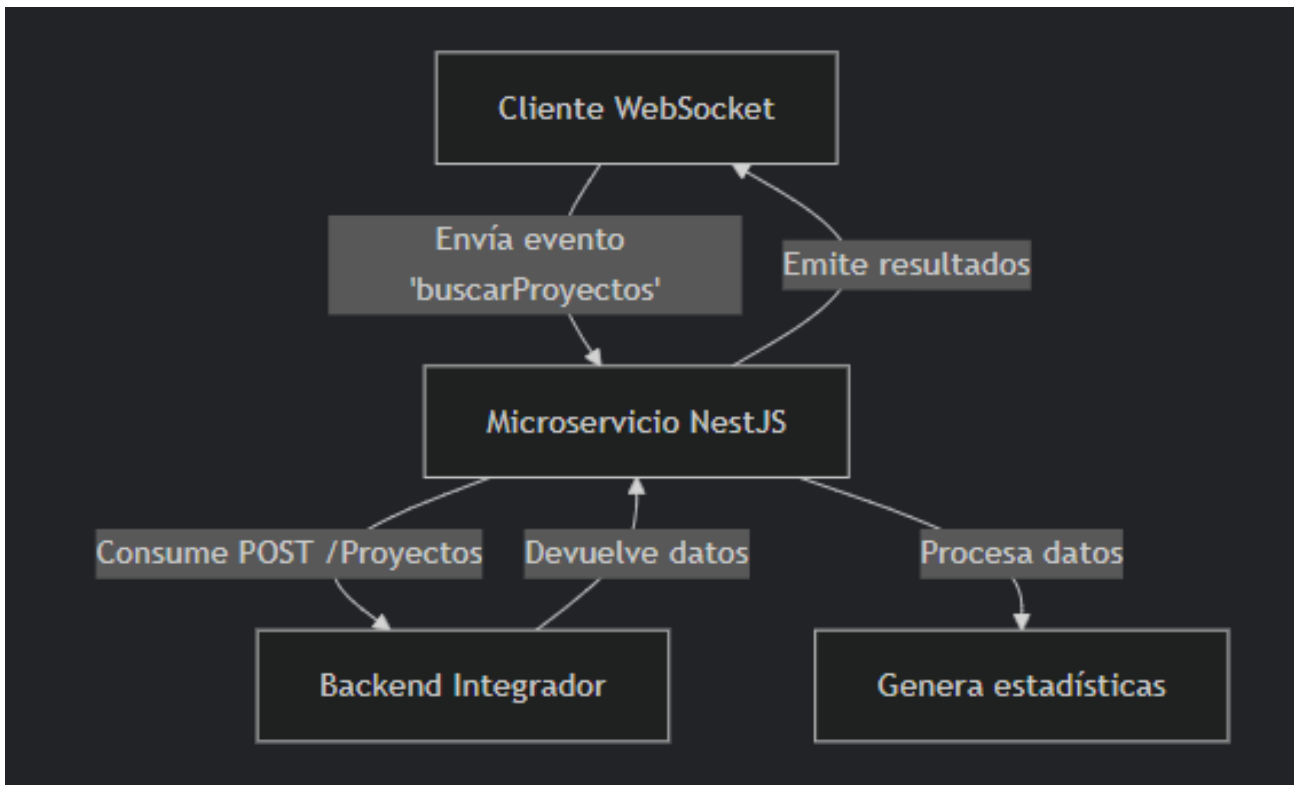
- Búsqueda dinámica de proyectos.
- Generación de estadísticas: conteo por categoría (académico/trabajo dirigido).

2.2 Justificación Tecnológica

Tecnología elegida: WebSockets (Socket.IO + NestJS).

- **Ventajas:**
 - Comunicación en tiempo real ideal para actualizaciones instantáneas.
 - Soporte nativo en NestJS mediante @nestjs/websockets.
 - Escalabilidad para múltiples clientes simultáneos.

2.3 Diagrama de Flujo



3. Implementación Técnica

3.1 Desarrollo del Microservicio Funcional

El microservicio fue desarrollado utilizando NestJS como framework principal, integrando WebSockets (Socket.IO) para la comunicación en tiempo real y Axios para consumir el endpoint externo POST /Proyectos del backend del proyecto integrador. Aunque el microservicio no almacena datos directamente, actúa como un puente inteligente que procesa y transforma la información recibida del backend externo, el cual utiliza su propia base de datos para gestionar los proyectos académicos y trabajos dirigidos.

Arquitectura y Componentes Clave

1. WebSocket Gateway (ProyectosGateway)

- Implementado con el decorador @WebSocketGateway de NestJS.
- Gestiona la conexión bidireccional con los clientes mediante Socket.IO.
- Escucha eventos como buscarProyectos y contarProyectosPorCategoria, delegando la lógica al servicio correspondiente.

2. Servicio de Lógica de Negocio (ProyectosService)

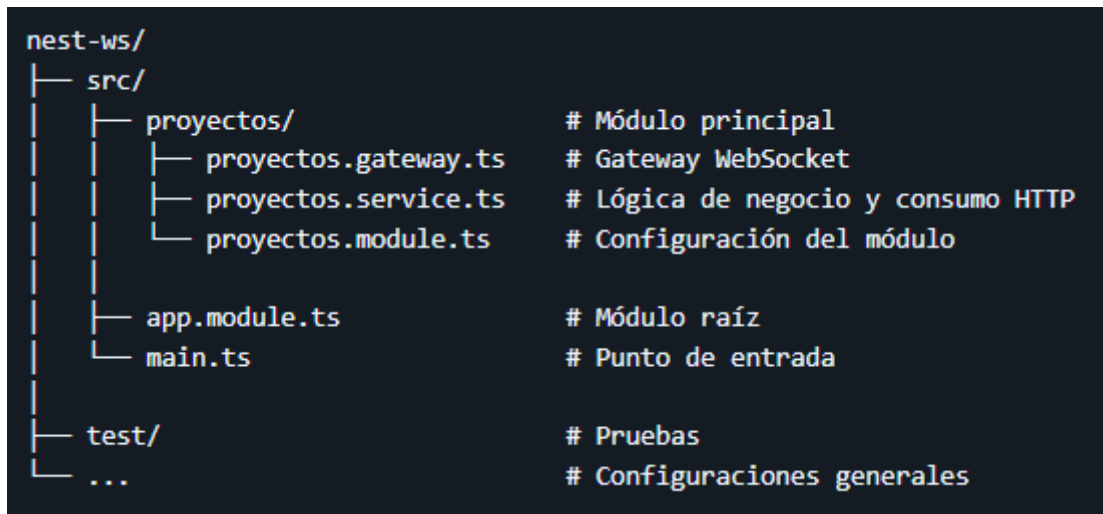
- Utiliza Axios para realizar solicitudes HTTP al backend externo.
- Procesa los datos recibidos (ej: conteo de proyectos por categoría, filtrado por año).

- Transforma los resultados en formatos útiles para los clientes (ej: estadísticas agrupadas).

3. Módulo Principal (ProyectosModule)

- Organiza las dependencias
- Registra el gateway y el servicio como proveedores principales.

3.2 Estructura del Proyecto



3.3 Código Clave

Gateway WebSocket (proyectos.gateway.ts):

```

import { WebSocketGateway, SubscribeMessage, MessageBody, WebSocketServer } from '@nestjs/websockets';
import { ProyectosService } from './proyectos.service';
import { Server } from 'socket.io';

@WebSocketGateway({
  cors: {
    origin: '*',
    methods: ['GET', 'POST']
  }
})
export class ProyectosGateway {
  @WebSocketServer()
  server: Server;

  constructor(private readonly proyectosService: ProyectosService) {}

  @SubscribeMessage('buscarProyectos')
  async buscarProyectos(@MessageBody() data: any) {
    try {
      const resultados = await this.proyectosService.buscarProyectos(data);
      this.server.emit('resultadosProyectos', resultados);
      return resultados;
    } catch (error) {
  
```

```

        this.server.emit('errorBusqueda', error.message);
        throw error;
    }
}

Tabnine | Edit | Test | Explain | Document
@SubscribeMessage('contarProyectosPorCategoria')
async contarPorCategoria() {
    try {
        const conteo = await this.proyectosService.contarProyectosPorCategoria();
        this.server.emit('resultadoConteoCategorias', conteo);
        return conteo;
    } catch (error) {
        this.server.emit('errorConteo', error.message);
        throw error;
    }
}
}

```

Servicio de Proyectos (proyectos.service.ts):

```

import { Injectable } from '@nestjs/common';
import axios from 'axios';

@Injectable()
export class ProyectosService {
    Tabnine | Edit | Test | Explain | Document
    async buscarProyectos(data: any) {
        try {
            const response = await axios.post('http://localhost:3000/Proyectos', data);
            return response.data;
        } catch (error) {
            throw new Error(error.response?.data?.message || 'Error al buscar proyectos');
        }
    }

    Tabnine | Edit | Test | Explain | Document
    async contarProyectosPorCategoria() {
        try {
            const response = await axios.post('http://localhost:3000/proyectos', {
                anio: "",
                categoria: "",
                titulo: ""
            });
        }
    }
}

```

```

    if (!Array.isArray(response.data)) {
      throw new Error('Formato de respuesta inválido');
    }

    const conteoCategorias = response.data.reduce((acumulador, proyecto) => {
      const categoria = proyecto.categoria || 'Sin categoría';
      acumulador[categoria] = (acumulador[categoria] || 0) + 1;
      return acumulador;
    }, {});

    return conteoCategorias;
  } catch (error) {
    throw new Error(`Error al contar categorías: ${error.message}`);
  }
}

```

Module de Proyectos (proyectos.module.ts):

```

import { Module } from '@nestjs/common';
import { ProyectosGateway } from '../proyectos.gateway';
import { ProyectosService } from '../proyectos.service';

@Module({
  providers: [ProyectosGateway, ProyectosService],
})
export class ProyectosModule {}

```

3.4 Consumo Correcto del Endpoint Externo

El microservicio se integra con el backend del proyecto integrador mediante el consumo del endpoint POST /Proyectos, utilizando la biblioteca Axios para realizar solicitudes HTTP. Este endpoint permite realizar búsquedas filtradas de proyectos académicos y trabajos dirigidos, aceptando parámetros como el año, la categoría y el título. Por ejemplo, al recibir una solicitud de búsqueda con el año 2025 y la categoría "Inteligencia Artificial", el servicio envía una petición POST al backend externo con estos filtros. La comunicación se realiza de manera eficiente, garantizando que los datos siempre estén actualizados y sincronizados con la fuente principal.

3.5 Procesamiento y Transformación de los Datos

Una vez recibidos los datos del backend externo, el microservicio realiza un procesamiento riguroso. Primero, valida que la respuesta sea un array válido y filtra proyectos con campos incompletos. Luego, transforma los datos crudos en información estructurada y útil. Por ejemplo, agrupa los proyectos por categoría (como "Investigación" o "Social") convirtiendo valores numéricos en etiquetas legibles. Este procesamiento garantiza que los clientes reciban respuestas claras y listas para su visualización, sin detalles técnicos innecesarios. Los resultados

se emiten en formatos estandarizados, como objetos con conteos específicos, facilitando su uso en interfaces frontend.

3.6 Exposición de Endpoints Propios y Documentación


Aunque el microservicio no expone endpoints REST tradicionales, define eventos WebSocket documentados que funcionan como puntos de interacción. Estos eventos incluyen buscarProyectos (para búsquedas filtradas) y contarProyectosPorCategoria (para estadísticas por categoría). Cada evento acepta y devuelve datos en formato JSON, siguiendo estructuras predefinidas. Por ejemplo, al emitir buscarProyectos con un año específico, los clientes reciben un array de proyectos relevantes. La documentación en el README.md detalla estos eventos, incluyendo ejemplos de payloads y respuestas, lo que permite a los desarrolladores integrar el servicio sin necesidad de acceder al código fuente.


4. Pruebas y Documentación

4.1 Evidencias Funcionales

Prueba en Postman (WebSocket):

1. Conexión con el microservicio

 ws://localhost:3009

 Save

ws://localhost:3009

Disconnect

Message

Events (4)

Params

Headers

Settings

Response

Listening to 4 events

Connected

Search

All Messages

Clear Messages

Listening to

resultadoConteo...

23:09:09

Listening to

contarProyectos...

23:09:09

Listening to

resultadosProyec...

23:09:09

Listening to

buscarProyectos

23:09:09

Connected to ws://localhost:3009

23:09:09

2. Solicitud de buscar proyectos con año=2025 y categoría=Salud

ws://localhost:3009

Save

ws://localhost:3009

Disconnect

Message Events (4) Params Headers Settings

```
1 {
2   "anio": "2025",
3   "categoria": "Salud",
4   "titulo": ""
5 }
```

+ Arg JSON ☐ Ack buscarProyectos **Send**

Response Listening to 4 events ● Connected

Search All Messages Clear Messages

↓ resultadosProyec...	[{"titulo": "Atención Médica Online", "categoria": "Salud", "date_part": 2025}, {"titulo": "Sistema de Citas", "categoria": "Salud", "date_part": 2025}]	23:09:58
↑ buscarProyectos	{ "anio": "2025", "categoria": "Salud", "titulo": "" }	23:09:58

3. Respuesta del microservicio:

ws://localhost:3009

Save

ws://localhost:3009

Disconnect

Message Events (4) Params Headers Settings

Response Listening to 4 events ● Connected

Search All Messages Clear Messages

↓ resultadosProyec... [{"titulo": "Atención Médica Online", "categoria": "Salud", "date_part": 2025}, {"titulo": "Sistema de Citas", "categoria": "Salud", "date_part": 2025}] 23:09:58

JSON Show Hexdump

```
1 [
2   {
3     "titulo": "Atención Médica Online",
4     "categoria": "Salud",
5     "date_part": 2025
6   },
7   {
8     "titulo": "Sistema de Citas",
9     "categoria": "Salud",
10    "date_part": 2025
11  },
12 ]
```


4. Solicitud de contar proyectos por categoría.

The screenshot shows a WebSocket client interface connected to `ws://localhost:3009`. The interface includes a message input field, a "Disconnect" button, and tabs for "Message", "Events (4)", "Params", "Headers", and "Settings". The "Message" tab is active, showing a list of messages. The "Response" section is expanded, displaying a message from the server with the following JSON payload: `{"Educación":8,"Logística":1,"Salud":6,"Tecnología":3,"Medio Ambiente":3,"Constru..."}`. The message is timestamped 23:12:29. The client is also showing a "Send" button with the text `contarProyectosPorCategoria` and an "Ack" checkbox.

5. Respuesta del microservicio


The screenshot shows the same WebSocket client interface as in the previous image, but with the "Response" section expanded to show the full JSON payload. The payload is a JSON object with the following structure: `{"Educación":8,"Logística":1,"Salud":6,"Tecnología":3,"Medio Ambiente":3,"Construcción":2,"Finanzas":1,"Agricultura":3,"Transporte":4,"Deportes":3,"Comercio":2}`. The message is timestamped 23:12:29. The client is also showing a "Send" button with the text `contarProyectosPorCategoria` and an "Ack" checkbox.

4.2 Pruebas Manuales


Escenario	Resultado Esperado
Búsqueda sin filtros	Retorna todos los proyectos.
Búsqueda con categoría inexistente	Array vacío
Conteo de categorías	Objeto con { "Investigación": X, ... }

4.3 Documentación clara del microservicio (README.md o PDF)


README

 **Microservicio de Gestión de Proyectos**

Este microservicio desarrollado con NestJS permite gestionar proyectos académicos y trabajos dirigidos mediante WebSocket (Socket.IO). Se encuentra en la carpeta nest-ws, la cual forma parte del backend principal. El microservicio consume endpoints HTTP externos para realizar búsquedas y generar estadísticas en tiempo real.

 **Estructura del Proyecto**

```
nest-ws/
├── src/
│   ├── proyectos/
│   │   ├── proyectos.gateway.ts    # Módulo principal
│   │   ├── proyectos.service.ts   # Gateway WebSocket
│   │   └── proyectos.module.ts    # Lógica de negocio y consumo HTTP
│   ├── app.module.ts              # Configuración del módulo
│   └── main.ts                    # Módulo raíz
├── test/                          # Punto de entrada
└── ...                           # Pruebas
                                # Configuraciones generales
```

 **Instalación y Ejecución**

Requisitos previos:

- Node.js v18+
- npm v9+
- Backend externo corriendo en `http://localhost:3000` con:
 - `POST /Proyectos` (Búsqueda de proyectos)

Pasos:

```
npm install    # Instalar dependencias
npm run start  # Iniciar microservicio (puerto 3009)
```

5. Presentación Final

5.1 Explicación Técnica del Microservicio (PDF Adjunto al Repositorio)

El documento técnico adjunto describe en detalle la arquitectura y funcionamiento del microservicio. Se enfoca en su rol como intermediario entre los clientes y el backend del proyecto integrador, utilizando WebSockets para garantizar comunicación bidireccional en tiempo real. La arquitectura se basa en NestJS, que permite una estructura modular y escalable, integrando el módulo @nestjs/websockets para gestionar eventos mediante Socket.IO. El microservicio consume el endpoint POST /Proyectos del backend principal mediante solicitudes HTTP con Axios, procesando los datos para generar estadísticas como el conteo de proyectos por categoría (académico/trabajo dirigido).

El flujo de datos se divide en tres etapas clave:

1. **Recepción de solicitudes:** Los clientes envían eventos como buscarProyectos o contarProyectosPorCategoria a través de WebSocket.
2. **Procesamiento:** El servicio realiza llamadas al backend, aplica transformaciones a los datos (como agrupaciones o cálculos estadísticos) y prepara las respuestas.
3. **Difusión:** Los resultados se emiten a todos los clientes conectados, asegurando sincronización inmediata.

En el documento también se analiza la integración del microservicio dentro del ecosistema institucional, destacando su capacidad para alimentar dashboards académicos con métricas actualizadas.

5.2 Demostración Funcional

Durante la presentación, se ejecuta un escenario práctico donde el cliente se conecta al microservicio mediante WebSocket. Al realizar una búsqueda con el evento buscarProyectos (ejemplo: filtrar proyectos del año 2025 en la categoría "Inteligencia Artificial"), el cliente recibe simultáneamente los resultados.

Esta demostración ilustra dos aspectos críticos:

- **Actualización en tiempo real:** Cualquier cambio en el backend (como añadir un nuevo proyecto) se refleja instantáneamente en las estadísticas emitidas por el microservicio.
- **Escalabilidad:** El uso de Socket.IO permite manejar conexiones concurrentes sin degradación del rendimiento, ideal para entornos con alta demanda, como periodos de inscripción de proyectos.

El flujo se resume en cuatro pasos claros:

1. Un cliente envía una solicitud.
2. El microservicio consume el backend externo.
3. Los datos se procesan y transforman.
4. Todos los clientes reciben respuestas sincronizadas.

Esta interacción valida la eficacia de la arquitectura orientada a eventos y cumple con los requisitos de integración en tiempo real exigidos por el sistema académico.

ws://localhost:3009

Save

ws://localhost:3009

Disconnect

Message

Events (4)

Params

Headers

Settings

1 {

2 "anio": "2025",

3 "categoria": "Salud",

4 "titulo": ""

5 }

6 }

+ Arg

JSON

☐ Ack

Send

Response

Listening to 4 events

Connected

Search

All Messages

Clear Messages

↓ resultadosProyec...

[{"titulo": "Atenci3n M3dica Online", "categoria": "Salud", "date_part": 2025}, {"titulo": "Sistema de Citas", "categoria": "Salud", "date_part": 2025}]

23:09:58

↑ buscarProyectos

{ "anio": "2025", "categoria": "Salud", "titulo": "" }

23:09:58

ws://localhost:3009

Save

ws://localhost:3009

Disconnect

Message

Events (4)

Params

Headers

Settings

Response

Listening to 4 events

Connected

Search

All Messages

Clear Messages

↓ resultadosProyec...

[{"titulo": "Atenci3n M3dica Online", "categoria": "Salud", "date_part": 2025}, {"titulo": "Sistema de Citas", "categoria": "Salud", "date_part": 2025}]

23:09:58

JSON

Show Hexdump

1 [

2 {

3 "titulo": "Atenci3n M3dica Online",

4 "categoria": "Salud",

5 "date_part": 2025

6 },

7 {

8 "titulo": "Sistema de Citas",

9 "categoria": "Salud",

10 "date_part": 2025

11 },