# Memoria 1 (rev2). Backend - WS CRUD

## Investigación

### 1. El Patrón Arquitectónico Cliente-Servidor

La arquitectura **Cliente-Servidor** es un modelo de diseño de software en el que las tareas se reparten entre los proveedores de recursos o servicios (servidores) y los demandantes (clientes).

- **El Cliente:** Es el proceso que inicia la comunicación. Solicita un recurso o servicio. Como el navegador
- **El Servidor:** Es el proceso que espera peticiones, las procesa y entrega un resultado.
- **Características clave:**
    - **Centralización:** Los recursos se gestionan en un punto central (el servidor).
    - **Escalabilidad:** Se pueden añadir más clientes sin afectar drásticamente el funcionamiento del servidor (hasta cierto límite).
    - **Separación de responsabilidades:** El cliente se encarga de la interfaz y la experiencia de usuario; el servidor de la lógica de negocio y los datos.

---

### 2. El Patrón de Intercambio: Petición-Respuesta (Request-Response)

HTTP funciona bajo un ciclo de vida muy estricto de **ida y vuelta**.

1. **Petición (Request):** El cliente abre una conexión TCP y envía un mensaje formateado al servidor.
2. **Procesamiento:** El servidor recibe el mensaje, entiende qué se le pide (gracias a las rutas como `/api/product`) y prepara una respuesta.
3. **Respuesta (Response):** El servidor envía el mensaje de vuelta y, generalmente, cierra la conexión (o la mantiene en espera para la siguiente petición).

---

### 3. Formato de una Petición HTTP (HTTP Request)

Cuando tu navegador intenta acceder a una API, envía un bloque de texto con esta estructura:

### Estructura:

1. **Línea de Solicitud:** Contiene el **Método** (GET, POST, etc.), el **Path** (la ruta) y la **Versión** de HTTP.
2. **Cabeceras (Headers):** Metadatos que dan contexto
3. **Cuerpo (Body):** Opcional. Se usa en métodos como POST o PUT para enviar información (por ejemplo, el JSON de un nuevo producto).

---

## 4. Formato de una Respuesta HTTP (HTTP Response)

Es lo que el servidor de Express genera cuando haces un `res.send()` o `res.status()`.

### Estructura:

1. **Línea de Estado:** Indica la versión de HTTP y el **Código de Estado**
2. **Cabeceras (Headers):** Información sobre el servidor y el tipo de contenido que devuelve.
3. **Cuerpo (Body):** Los datos solicitados

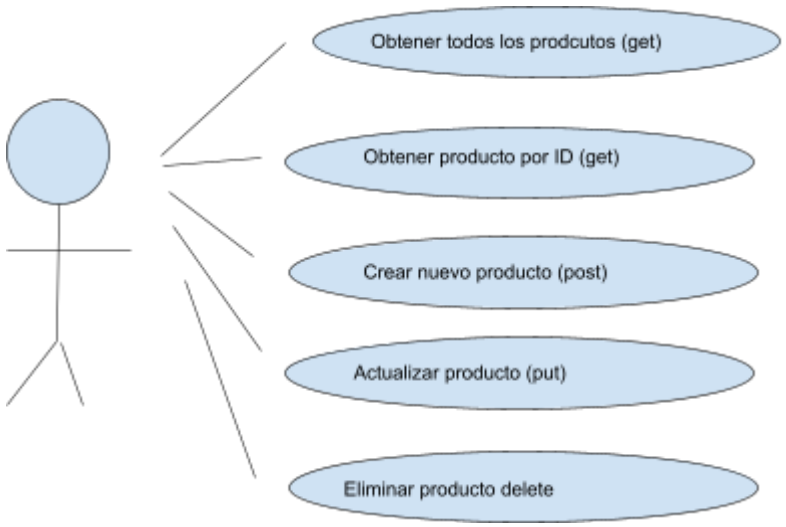
---

## 5. Características Básicas de la Comunicación HTTP

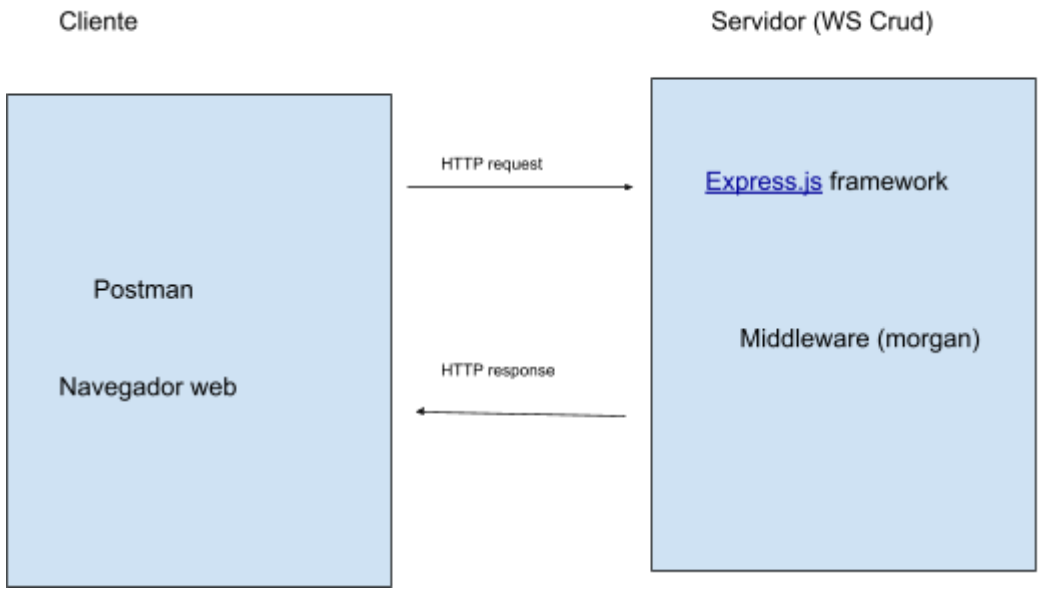
- **Sin Estado (Stateless):** HTTP no "recuerda" nada de la petición anterior. Cada petición es nueva. Por eso usamos *Cookies* o *Tokens* si queremos que el servidor sepa quiénes somos.
- **Independiente del tipo de contenido:** HTTP puede transportar cualquier cosa (JSON, XML, video, texto plano) siempre que el cliente y el servidor acuerden el `Content-Type`.
- **Basado en texto:** Aunque los cuerpos pueden ser binarios, el protocolo en sí es legible por humanos

# Documentación del Servicio (o Aplicación)

## Diagrama de Casos de Uso,



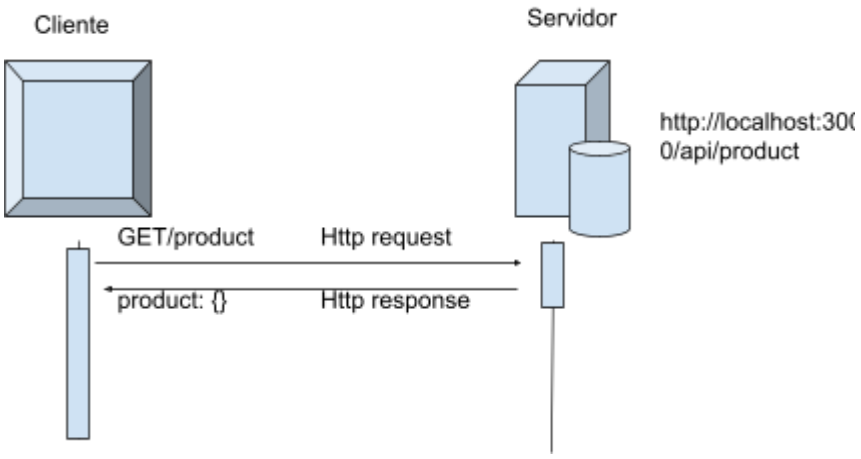

## Arquitectura Distribuida del sistema,



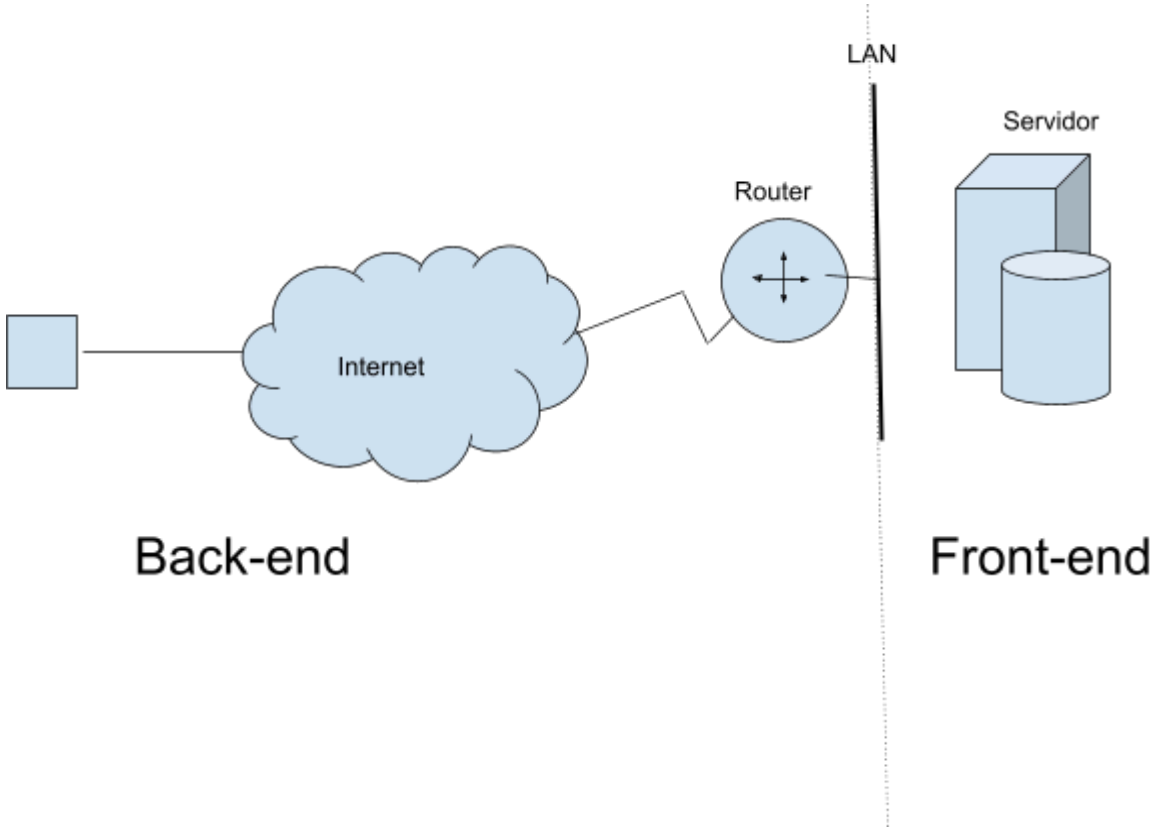
Definición del API,

| Método HTTP | Ruta (*endpoints*) | Descripción |
|---|---|---|
| GET | /api/product | Obtenemos todos los elementos de la colección product. |
| GET | /api/product/{id} | Obtenemos el elemento indicado en {id} de la colección product. |
| POST | /api/product | Creamos un nuevo elemento en la colección product. |
| PUT | /api/product/{id} | Modificamos el elemento {id} de la colección product. |
| DELETE | /api/product/{id} | Eliminamos el elemento {id} de la colección product. |

Diagrama de Secuencia,



Arquitectura física del Escenario de Despliegue.




# Blog o Bitácora

Primero descargamos lo que va ha hacer falta, en este caso Postman y Nodemon

npm i -D nodemon

Además, instalamos Morgan ara tener un registro de los logs, después aplicamos el logger:

```
const logger = require('morgan');

const app = express();

// Declaramos los middleware
app.use(logger('dev'));     // probar con: tiny, short, dev, common, combined
```

Ahora ya podemos crear la primera API crud básica, implementamos los metodos http básicos para la comunicación:

```
app.get('/api/product', (req, res) => {
	res.status(200);
	res.send( {productos: []} );
});

app.get('/api/product/:productID', (req, res) => {
	const productID = req.params.productID;

	res.status(200);
	res.send({producto: productID });
});

app.post('/api/product', (req, res) => {
	const queProducto = req.body;
	console.log(queProducto);
```

```
app.put('/api/product/:productID', (req, res) => {
	const queProducto = req.body;
	const productID = req.params.productID;

	res.status(200);
	res.send({
	    mensaje: `Se ha modificado el producto ${productID}`,
	    producto: queProducto
	});
});

app.delete('/api/product/:productID', (req, res) => {
	const productID = req.params.productID;

	res.status(200);
	res.send( { mensaje: `Se ha eliminado el producto ${productID}` } );
});

// Lanzamos nuestro servicio API
app.listen(port, () => {
	console.log(`API REST ejecutándose en http://localhost:${port}/api/product`);
});
```

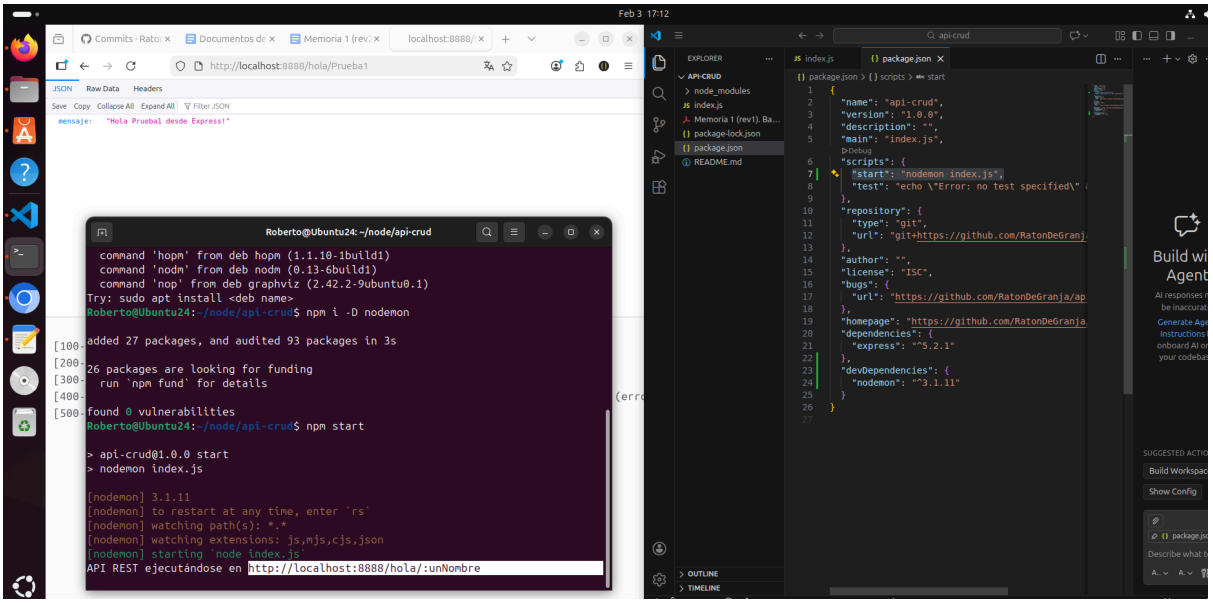
Para probar el API se ejecuta con **npm start** y usamos http://localhost:3000/api/product para hacer la llamada.

Para probar un POST escribimos en el body de postman:

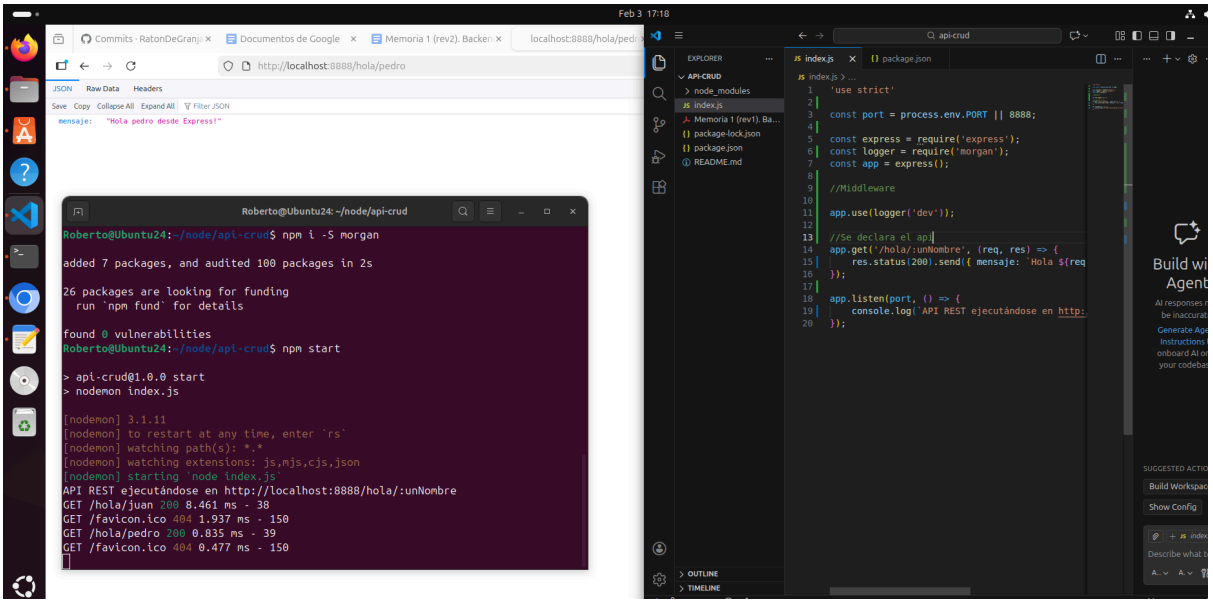
```
name        mi producto
price       200
photo       miProducto.png
category    general
```

# Pruebas

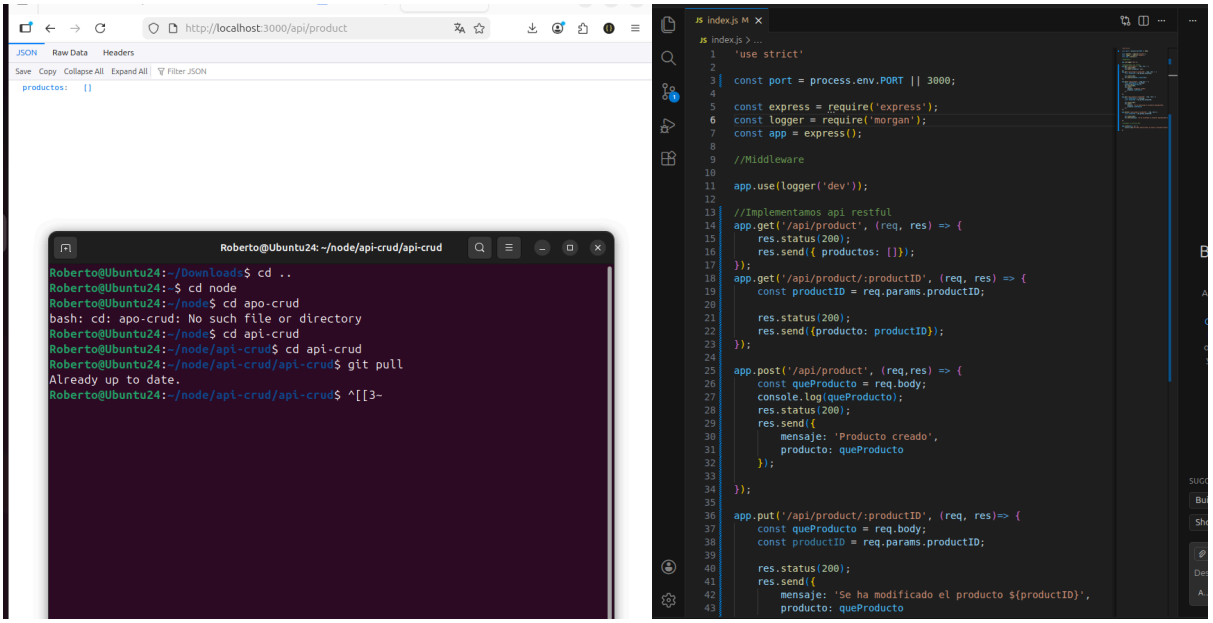
prueba nodemon




prueba morgan




Primera API REST CRUD básica




Probando el api