

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:

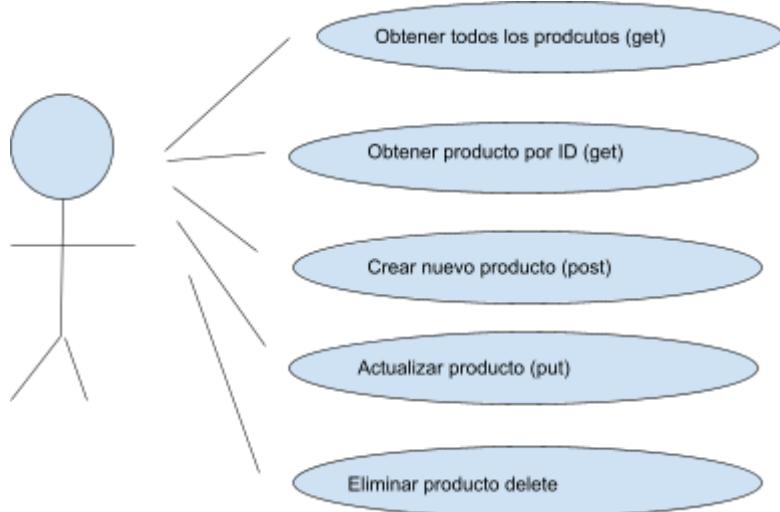
1. **Línea de Estado:** Indica la versión de HTTP y el **Código de Estado**
 2. **Cabeceras (Headers):** Información sobre el servidor y el tipo de contenido que devuelve.
 3. **Cuerpo (Body):** Los datos solicitados
-

5. Características Básicas de la Comunicación HTTP

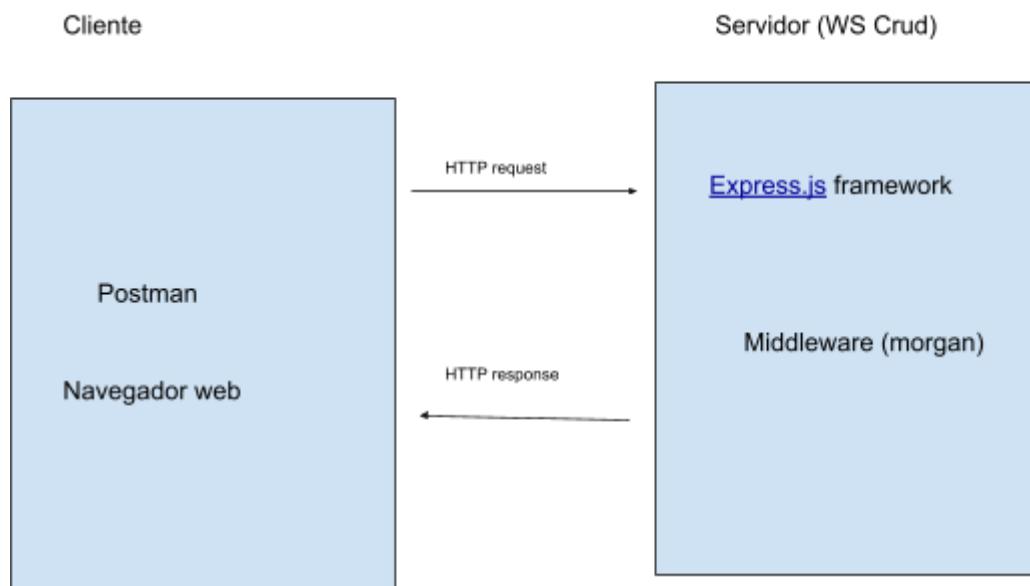
- **Sin Estado (Stateless):** HTTP no "recuerda" nada de la petición anterior. Cada petición es nueva. Por eso usamos *Cookies* o *Tokens* si queremos que el servidor sepa quiénes somos.
- **Independiente del tipo de contenido:** HTTP puede transportar cualquier cosa (JSON, XML, video, texto plano) siempre 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,



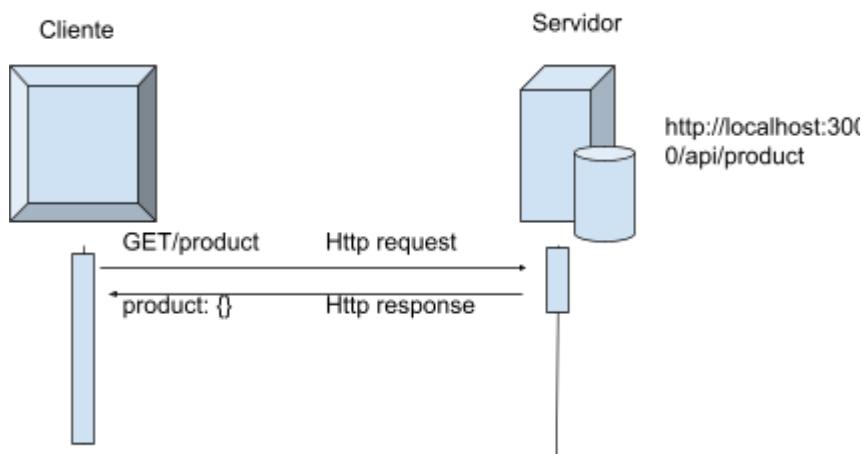
Arquitectura Distribuida del sistema,



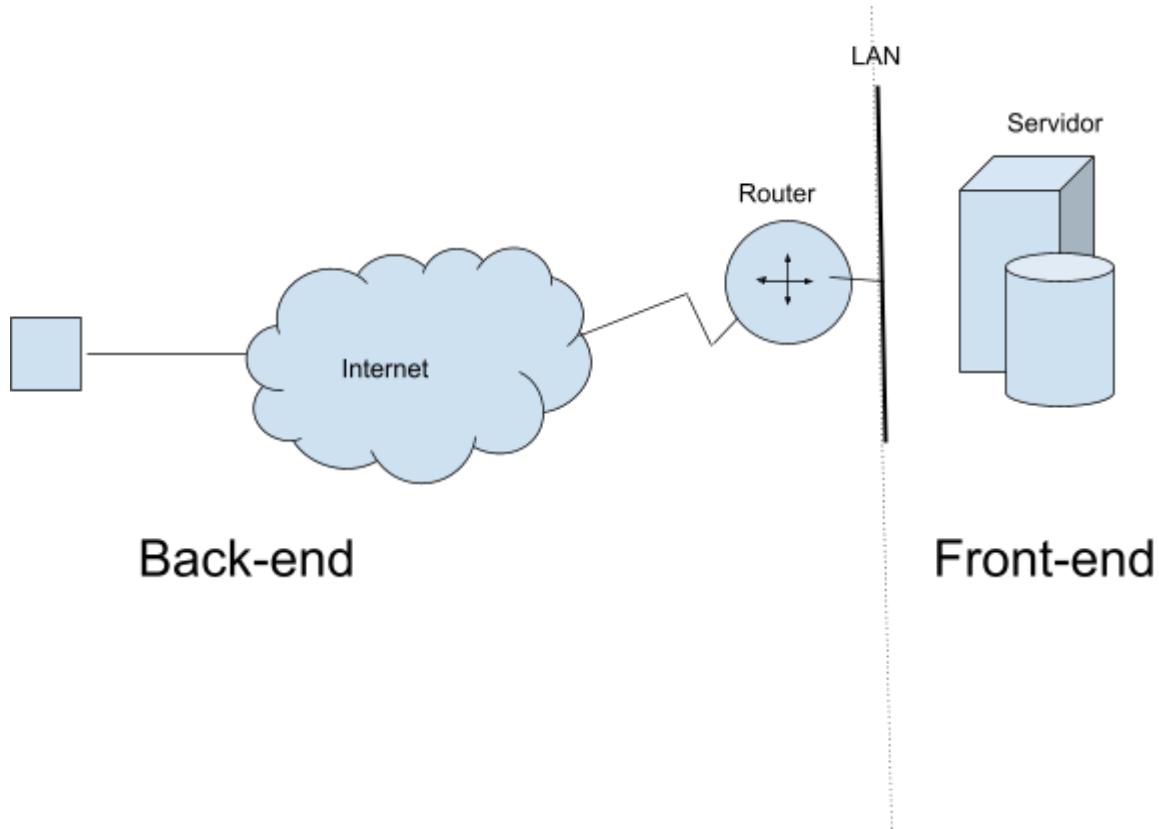
Definición del API,

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 para 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 métodos 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

Feb 3 17:12

```

Commits - Rato: x Documentos de x Memoria 1 (rev. x) localhost:8888/hola/prueba1
JSON Raw Data Headers
Save Copy Collapse All | Filter JSON
mensaje: "Hola Prueba! desde Express!"

Roberto@Ubuntu24:~/node/api-crud$ npm i -D nodemon
[100] added 27 packages, and audited 93 packages in 3s
[200] 26 packages are looking for funding
[300]   run 'npm fund' for details
[400]
[500] found 0 vulnerabilities
Roberto@Ubuntu24:~/node/api-crud$ npm start
> api-crud@1.0.0 start
> nodemon index.js

[nodemon] 3.1.11
[nodemon] to restart at any time, enter 'rs'
[nodemon] watching path(s): *
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting 'node index.js'
API REST ejecutándose en http://localhost:8888/hola/:unNombre

```

index.json

```

{
  "name": "api-crud",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "debug": true,
  "scripts": {
    "start": "nodemon index.js",
    "test": "echo \"Error: no test specified\" > test.log"
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/RatonDeGranja/api-crud.git"
  },
  "author": "",
  "license": "ISC",
  "bugs": {
    "url": "https://github.com/RatonDeGranja/api-crud/issues"
  },
  "homepage": "https://github.com/RatonDeGranja/api-crud",
  "dependencies": {
    "express": "^5.2.1"
  },
  "devDependencies": {
    "nodemon": "^3.1.11"
  }
}

```

SUGGESTED ACTIONS

- Build Workspace
- Show Config
- + package.json
- Describe what's new
- A. v A. v T

prueba morgan

Feb 3 17:18

```

Commits - RatonDeGranja: x Documentos de Google x Memoria 1 (rev2). Backer: x localhost:8888/hola/pedro
JSON Raw Data Headers
Save Copy Collapse All | Filter JSON
mensaje: "Hola petro desde Express!"

Roberto@Ubuntu24:~/node/api-crud$ npm i -S morgan
added 7 packages, and audited 100 packages in 2s
26 packages are looking for funding
  run 'npm fund' for details
found 0 vulnerabilities
Roberto@Ubuntu24:~/node/api-crud$ npm start
> api-crud@1.0.0 start
> nodemon index.js

[nodemon] 3.1.11
[nodemon] to restart at any time, enter 'rs'
[nodemon] watching path(s): *
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting 'node index.js'
API REST ejecutándose en http://localhost:8888/hola/:unNombre
GET /hola/juan 200 8.461 ms - 38
GET /favicon.ico 404 1.937 ms - 150
GET /hola/pedro 200 0.835 ms - 39
GET /favicon.ico 404 0.477 ms - 150

```

index.js

```

'use strict';
const port = process.env.PORT || 8888;
const express = require('express');
const logger = require('morgan');
const app = express();

//Middleware
app.use(logger('dev'));

//Se declara el endpoint
app.get('/hola/:unNombre', (req, res) => {
  res.status(200).send(`mensaje: ${req.params.unNombre}`);
});

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

```

SUGGESTED ACTIONS

- Build Workspace
- Show Config
- + package.json
- Describe what's new
- A. v A. v T

Primera API REST CRUD básica

The terminal window shows the execution of the `index.js` file, which defines a simple REST API for products. The browser window shows the JSON response from the `/api/product` endpoint, which returns an empty array.

```

const port = process.env.PORT || 3000;
const express = require('express');
const logger = require('morgan');
const app = express();
//Middleware
app.use(logger('dev'));
//Implementamos api restful
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);
  res.status(200);
  res.send({
    mensaje: 'Producto creado',
    producto: 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
  });
});
  
```

Probando el api

The screenshot shows a desktop environment with multiple windows open. In the foreground, a terminal window displays the execution of the `index.js` file and its output. A browser window shows the JSON response of the API. In the background, a Postman application window is open, showing a POST request to `http://localhost:3000/api/product` with a JSON body containing `name: "mi producto"`, `price: 200`, `photo: miProducto.png`, and `category: general`. The response status is 200 OK.