

1 **Node.js** es un entorno de ejecución de JavaScript que permite ejecutar código JavaScript en el servidor.

2 **Ventajas**

**Rendimiento:** Node.js es muy rápido debido a su arquitectura asíncrona orientada a eventos que lo hace ideal para aplicaciones en tiempo real.

**Escalabilidad:** Node.js se escala bien horizontalmente permitiendo distribuir la carga en múltiples CPUs/máquinas.

**Ecosistema rico:** Hay un gran ecosistema de módulos npm que facilitan el desarrollo.

**Mismo lenguaje cliente y servidor:** Al usar JavaScript tanto en frontend como backend se necesita aprender solo un lenguaje.

**Facilidad para coder streams:** Node.js facilita el manejo de streams de datos como la lectura/escritura de archivos, requests HTTP, etc.

**Activo Open Source:** Gran comunidad open source con rápida evolución y adopción de nuevas tecnologías.

3 **Node.js se usa principalmente para desarrollar:**

- A APIs y backends que necesitan alta concurrencia.
- B Aplicaciones en tiempo real como chats, juegos online, apps colaborativas.
- C Herramientas de línea de comandos y automatización.

4 **Algunas empresas que usan Node.js son:**

1 2 3 4 5 6 7  
Netflix, Uber, Trello, LinkedIn, eBay, PayPal, Twitter.

5 **Express**

Express es el framework web más popular para Node.js. Proporciona una capa delgada para manejar solicitudes HTTP y rutas en aplicaciones Node.js.

6 **Algunas razones para usar Express:**

**Fácil de usar:** Tiene una sintaxis simple y minimalista para enrutamiento y manejo de requests/responses. Ideal para APIs y sitios web..

Una API (Interfaz de Programación de Aplicaciones) es un conjunto de definiciones y protocolos que se utilizan para desarrollar e integrar software de aplicaciones.

**Robusto:** Incluye varias funcionalidades como middlewares, renderizado de vistas, etc.

Un middleware es básicamente un trozo de código que se ejecuta entre el momento en que se recibe una petición (request) y el momento en que se envía una respuesta (response) en una aplicación.

**Popular:** Es el framework web más usado en Node.js. Fácil encontrar recursos y soporte de la comunidad.

**MVC:** Soporta arquitecturas MVC para separar la lógica de negocio, vistas y acceso a datos.

La arquitectura de software es la estructura general de un sistema, que incluye sus componentes, las relaciones entre ellos y cómo interactúan para lograr los objetivos del sistema.

## 7 npm (Node Package Manager):

Es el gestor de paquetes por defecto para Node.js. Permite a los desarrolladores:

Instalar paquetes/módulos de Node.js - npm cuenta con el mayor ecosistema de librerías open source del mundo.

Publicar paquetes/módulos propios para compartir con otros desarrolladores.

Gestionar las dependencias y versiones de los paquetes que utiliza una aplicación Node.js.

Actualizar fácilmente paquetes a sus últimas versiones..

Manejar diferentes versiones de paquetes para diferentes proyectos y automatizar tareas con scripts npm.

## 8 Backend

El backend de una aplicación es la parte que se ejecuta en el servidor, por oposición al frontend que se ejecuta en el cliente (navegador web o app móvil).

En resumen, el backend se encarga de:

- Procesar las peticiones del frontend.
- Acceder a la base de datos.
- Ejecutar la lógica del negocio.
- Autenticar y autorizar usuarios.
- Manejar la sesión de usuarios.
- Integrar con otros sistemas externos.

```
backend
├── src
│   ├── controllers // Controladores, manejadores de rutas y lógica
│   │   ├── users.controller.ts
│   │   ├── products.controller.ts
│   │   └── orders.controller.ts
│   ├── models // Modelos y esquemas de base de datos
│   │   ├── User.ts
│   │   ├── Product.ts
│   │   └── Order.ts
│   ├── services // Servicios y lógica de negocio
│   │   ├── users.service.ts
│   │   ├── products.service.ts
│   │   └── orders.service.ts
│   ├── middlewares // Middlewares y funciones helper
│   │   ├── auth.middleware.ts
│   │   └── validators.ts
│   ├── config // Archivos de configuración
│   │   ├── database.ts
│   │   └── env.ts
│   └── app.ts // Punto de entrada principal
├── tests // Pruebas unitarias y de integración
│   └── example.test.ts
├── docs // Documentación del API
│   └── openapi.yaml
└── package.json
```

La idea es separar la lógica en capas:

- Controllers: capa de entrada, manejo de rutas y peticiones HTTP.
- Services: capa de lógica de negocio y funciones del core.
- Models: capa de datos y modelos de base de datos.
- Middlewares: funciones helper reutilizables.
- Config: archivos de configuración.
- Tests: pruebas automatizadas.
- Docs: documentación del API.


## 10 API REST

Un **API REST (RESTful API)** es una interfaz de programación de aplicaciones (API) que cumple con los principios de arquitectura REST. **REST** significa Transferencia de Estado Representacional. Los principios más importantes de un API REST son:

- 1 Utiliza **HTTP métodos** (GET, POST, PUT, DELETE, etc) para operaciones CRUD (Create, Read, Update, Delete).
- 2 Es stateless, es decir, no guarda estado entre solicitudes. Cada solicitud contiene toda la información necesaria para ejecutarse.
- 3 Utiliza URLs para identificar recursos. Por ejemplo, <https://ejemplo.com/api/users> podría representar el recurso de usuarios.
- 4 Retorna representaciones JSON, XML, etc de los recursos solicitados. No retorna HTML para renderizarse en un navegador.
- 5 Tiene una estructura uniforme, siguiendo el estilo arquitectónico REST.

Paso a paso para crear el proyecto:

1. Cree una carpeta para el proyecto.

 proyecto1

2. Inicializa el proyecto con npm

```
npm init -y
```

Se crea este archivo:

```
{ } package.json
```

3. Instala TypeScript y Express como dependencias de desarrollo

```
npm install typescript @types/node @types/express --save-dev
```

4. Instala Express como dependencia

```
npm install express
```

5. Crea un archivo tsconfig.json para configurar TypeScript

```
tsc --init
```

6. Modifica el tsconfig.json para que compile a ES6 y permita imports

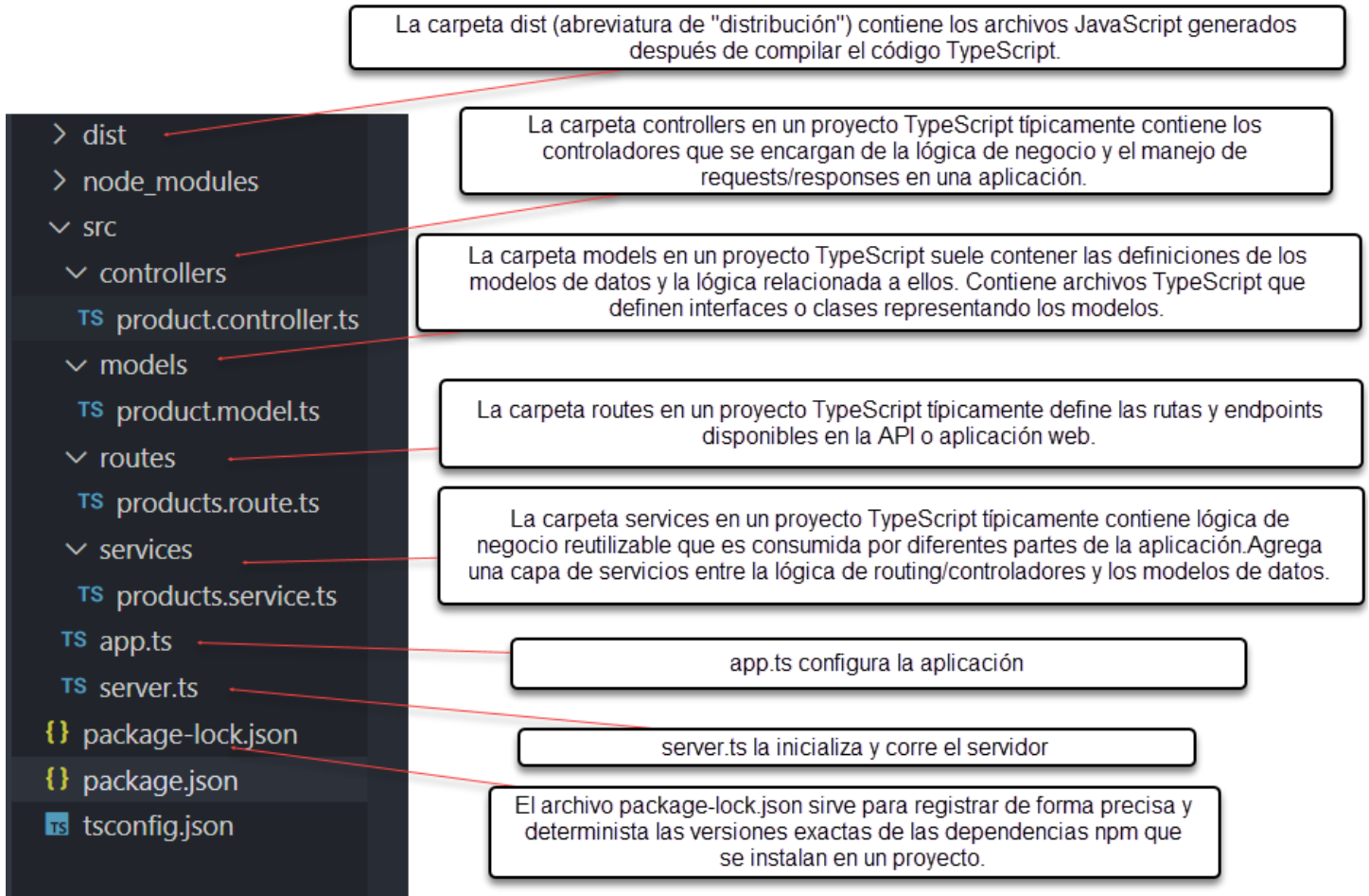
```
"target": "es6",  
"module": "commonjs",
```

```
"skipLibCheck": true,  
"outDir": "dist"
```

7. Crea un script en package.json para compilar y ejecutar

```
"scripts": {  
  "test": "echo \"Error: no test specified\" && exit 1",  
  "start": "tsc && node dist/server.js"  
},
```

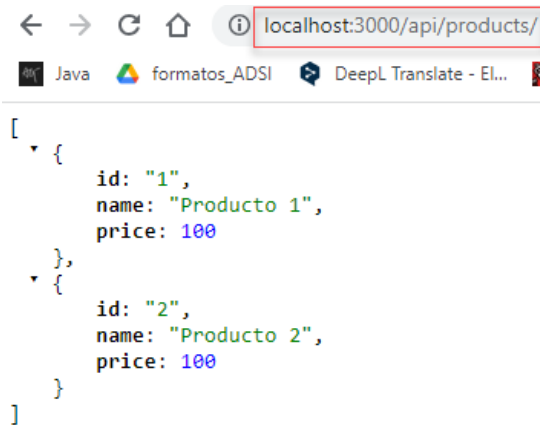
8. Genere la estructura del proyecto según indicaciones



9. Se levanta todo el servidor

```
PS D:\proyectoTypescript\proyectoProductos> npm start  
  
> proyectoproductos@1.0.0 start  
> tsc && node dist/server.js  
  
Server listening on port 3000
```





```

[
  {
    id: "1",
    name: "Producto 1",
    price: 100
  },
  {
    id: "2",
    name: "Producto 2",
    price: 100
  }
]

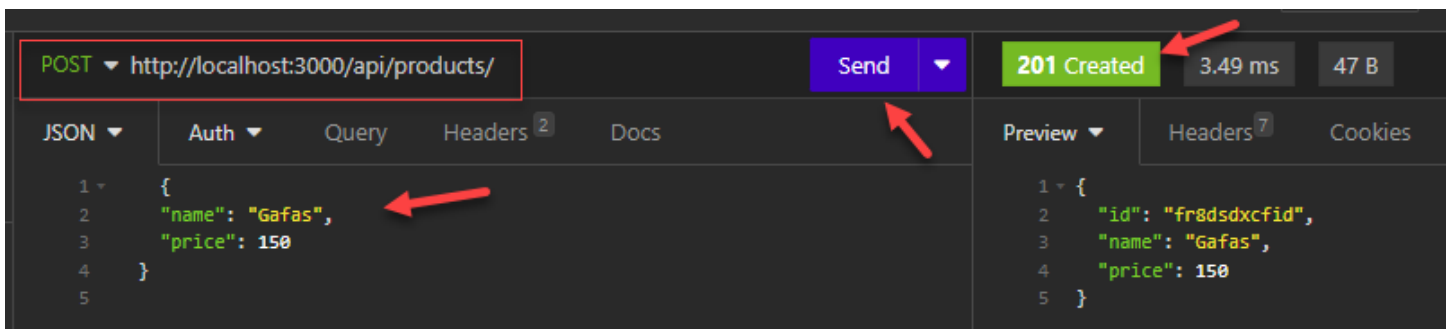
```

10. Hacer pruebas unitarias de la API.

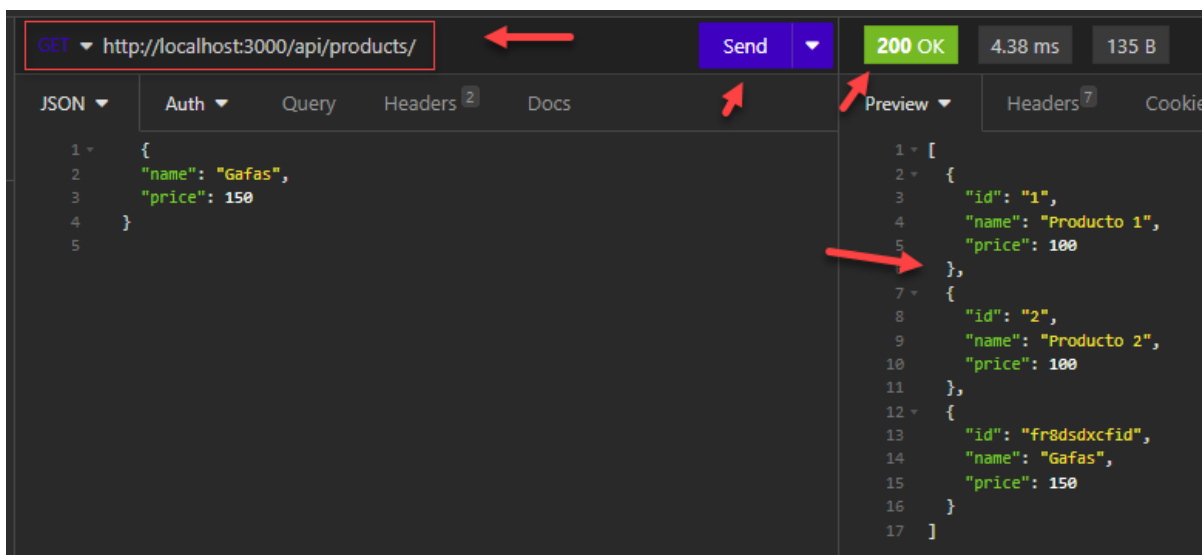
**Insomnia y Postman** son programas muy utilizados para realizar pruebas de API REST:

1. Permiten crear y enviar requests HTTP (GET, POST, PUT, DELETE, etc) a APIs de forma sencilla.
2. Se pueden escribir URLs y endpoints para hacer requests a un servidor backend.
3. Facilitan la construcción de queries, headers y bodies para las llamadas.
4. Permiten guardar environments, variables y collections de requests para reutilizar.
5. Generan code snippets para integrar con tests automatizados.
6. Permiten inspeccionar responses con detalle, incluyendo headers, bodies, tiempos, etc.
7. Ofrecen utilidades para mockear/simular APIs y endpoints.
8. Cuentan con funciones para monitorizar y documentar APIs.
9. Integran autenticación mediante tokens, OAuth, etc.
10. Permiten probar APIs rápidamente sin necesidad de escribir código.

Insertar o crear



Consultar todo



Consultar uno

GET `http://localhost:3000/api/products/2` Send 200 OK 4.56 ms 42 B

JSON Auth Query Headers 2 Docs Preview Headers 7 Cookies

```
1 {
2   "name": "Gafas",
3   "price": 150
4 }
5
```

```
1 {
2   "id": "2",
3   "name": "Producto 2",
4   "price": 100
5 }
```

Actualizar:

PUT `http://localhost:3000/api/products/2` Send 200 OK 4.08 ms 40 B

JSON Auth Query Headers 2 Docs Preview Headers 7

```
1 {
2   "name": "sombrero",
3   "price": 150
4 }
5
```

```
1 {
2   "id": "2",
3   "name": "sombrero",
4   "price": 150
5 }
```

Borrar:

DELETE `http://localhost:3000/api/products/1` Send 200 OK 4.18 ms

JSON Auth Query Headers 2 Docs Preview Headers 7

```
1 {
2   "name": "sombrero",
3   "price": 150
4 }
5
```

Deleted