



Universidad del Valle sede Tuluá  
Aplicaciones en el web y redes inalámbricas  
Sistema desacoplado utilizando microservicios y API Gateway

Juan Felipe Arango Guzmán - 2060066  
Miguel Ángel Rivera Reyes - 2059876  
Juan Sebastian Ruiz Aguilar - 2059898

Yilmar Rodriguez Alzate

Tuluá - Valle del Cauca  
2024

## Índice

|   |           |
|---|-----------|
| <b>Introducción.....</b>                                    | <b>3</b>  |
| <b>¿Qué es un microservicio?.....</b>                       | <b>4</b>  |
| <b>¿Qué es una API?.....</b>                                | <b>4</b>  |
| <b>¿Cómo funcionan los microservicios?.....</b>             | <b>5</b>  |
| <b>¿Cómo funcionan las APIs?.....</b>                       | <b>6</b>  |
| <b>¿Qué es un API Gateway?.....</b>                         | <b>6</b>  |
| <b>¿Por qué usar una API Gateway?.....</b>                  | <b>7</b>  |
| <b>Propuesta de desarrollo.....</b>                         | <b>8</b>  |
| <b>Descripción de los microservicios desarrollados.....</b> | <b>9</b>  |
| <b>Funcionamiento de la api Gateway.....</b>                | <b>11</b> |
| <b>Conclusión.....</b>                                      | <b>15</b> |
| <b>Bibliografía.....</b>                                    | <b>16</b> |

## **Introducción**

En el mundo del desarrollo de software moderno, los microservicios, como enfoque arquitectónico, lideran la forma en que se construyen aplicaciones. A diferencia de las aplicaciones monolíticas, los microservicios dividen la funcionalidad en aplicaciones pequeñas, independientes y desacopladas, cada uno encargado de una tarea específica. Estos se comunican entre sí mediante APIs que apuntan a rutas específicas dentro del servidor donde se encuentran desplegadas, pero puede llegar a ser caótico manejar muchas rutas si se tiene un sistema complejo con varios microservicios en ejecución, siendo este uno de los problemas más comunes a la hora de trabajar con este tipo de arquitectura. Para dar solución a esto y simplificar la gestión de microservicios, se puede implementar una API Gateway: Un sistema que actúa como un punto de entrada único y proporciona funcionalidades como enrutamiento, seguridad y transformación de datos.

En este documento se implementará una aplicación construida con el enfoque de microservicios gestionada con una API Gateway, mostrando sus ventajas y cómo puede ayudar a escalar de manera adecuada una aplicación, mejorar la experiencia de desarrollo y de uso del lado del cliente.

## **¿Qué es un microservicio?**

Un microservicio es una arquitectura y una forma de programar que consiste en la descomposición de una aplicación en pequeños fragmentos independientes entre sí, donde cada uno de estos se encarga de una tarea específica y tiene su propio flujo de ejecución y de datos y se comunican a través de APIs.

Cada microservicio se encarga de una tarea en concreto y despliega de forma independiente, así la ejecución y/o error en otros servicios no detienen el funcionamiento completo de la aplicación. La independencia de los servicios hace posible que puedan ser muy diversos en cuanto a las tecnologías que pueden usar, haciendo así que las librerías o frameworks de un servicio no generen conflictos con los de otro servicio.

Una aplicación basada en una arquitectura de microservicios tiene una gran escalabilidad, si se desea agregar una funcionalidad nueva será más sencillo integrarla, ya que se crea un nuevo servicio y se agrega esta característica a la aplicación con el uso de una API, sin la necesidad de modificar la base de la aplicación y evitando conflictos con otros módulos, al contrario que una aplicación monolítica donde crear una nueva funcionalidad puede generar problemas en las características que ya existen.

## **¿Qué es una API?**

Un Interfaz de Programación de Aplicaciones (API) es el conjunto de herramientas, definiciones y protocolos que permite la comunicación entre aplicaciones, es como un intermediario que hace posible la comunicación entre aplicaciones para intercambiar datos y/o funcionalidades. Las APIs basadas en la web se comunican mediante protocolos HTTP o HTTPS e intercambian información en formatos como JSON o XML.

Algunos ejemplos de APIs son:

- Google Maps API: Proporciona mapas, Street View, indicaciones de ruta y otras funcionalidades de geolocalización.
- OpenWeatherMap API: Ofrece datos meteorológicos actuales, pronósticos y alertas para todo el mundo.
- Spotify API: Permite reproducir música, buscar canciones, crear playlists y otras funcionalidades de Spotify.
- Facebook Graph API: Permite acceder a la información de usuarios de Facebook, publicar en sus muros, crear eventos y más.

- GitHub API: Permite interactuar con repositorios de código en GitHub, crear issues, realizar pull requests y más.

### ¿Cómo funcionan los microservicios?

La arquitectura de microservicios es una evolución de la arquitectura orientada a servicios (SOA), pero con un enfoque más granular y descentralizado. En lugar de tener grandes servicios monolíticos, los microservicios dividen una aplicación en pequeños e independientes componentes, cada uno de los cuales se encarga de una función específica y puede ser desarrollado, desplegado y escalado de forma independiente.

Usar los microservicios implica descomponer una aplicación en funcionalidades que trabajan como pequeños programas que trabajan de forma independiente entre ellas y se comunican a través de APIs para colaborar a llevar a cabo tareas complejas.

Tomemos como ejemplo una aplicación de viajes compartidos, donde algunos servicios que podemos tomar para el ejemplo son:

- **Servicio de geolocalización:** Este servicio podría ser responsable de recuperar y actualizar la ubicación de los conductores y los usuarios en tiempo real.
- **Servicio de procesamiento de pagos:** Este servicio manejaría la lógica relacionada con el procesamiento de pagos para los viajes realizados.
- **Servicio de notificaciones:** Este servicio enviará alertas y notificaciones a los usuarios sobre el estado de sus viajes, como la llegada del conductor o actualizaciones sobre el tiempo estimado de llegada.

Vemos que una aplicación se puede descomponer en pequeñas fracciones que la componen y hacen tareas bien definidas que en conjunto logran hacer tareas complejas como es un servicio de viajes. Si esta aplicación quisiera incluir una funcionalidad nueva como un chat entre el conductor y la persona que desea desplazarse, solo deberían crear un microservicio para el chat e integrarlo con la aplicación sin necesidad de tocar el código de los demás componentes del programa.

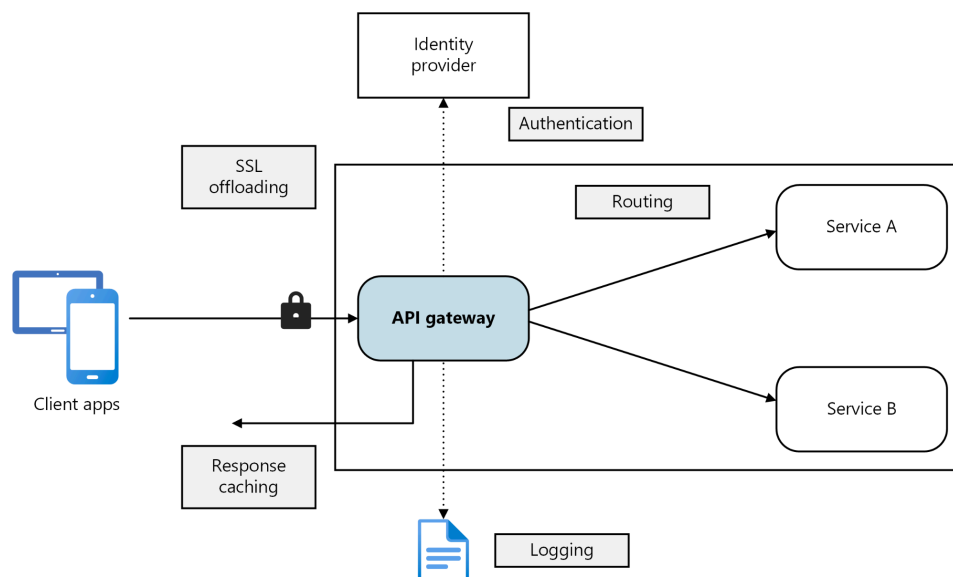
## ¿Cómo funcionan las APIs?

En una arquitectura de microservicios, cada uno de ellos expone una API a través de un servidor web utilizando protocolos estándar como HTTP, gRPC, WebSockets, entre otros; y es accesible a través de una URL específica. Para hacer una petición a una API de un microservicio, se deben seguir una serie de pasos:

1. **Solicitud (request):** Se envía una solicitud al microservicio en cuestión desde el lado del cliente. La respuesta o acción a realizar dependerá del tipo de solicitud (GET, POST, PUT, DELETE) y los parámetros especificados.
2. **Procesamiento de la solicitud:** La aplicación asociada a la API procesa la solicitud y ejecuta las acciones correspondientes. Esto puede involucrar cálculos con ciertos datos, peticiones a bases de datos, llamadas a otras APIs, etc.
3. **Respuesta (Response):** Una vez que se ha procesado la solicitud, la aplicación devuelve un código de respuesta o los datos solicitados, según sea el caso.

## ¿Qué es un API Gateway?

Una API Gateway (puerta de enlace de APIs) es una herramienta que se sitúa entre el lado del cliente y un conjunto de aplicaciones o microservicios del lado del servidor. Se encarga de todas las tareas que involucren aceptar y procesar hasta cientos de miles de llamadas simultáneas a diferentes APIs, control de acceso, monitoreo, control de versiones, entre otras funciones.



Funcionamiento de una API Gateway - Microsoft Azure

### **¿Por qué usar una API Gateway?**

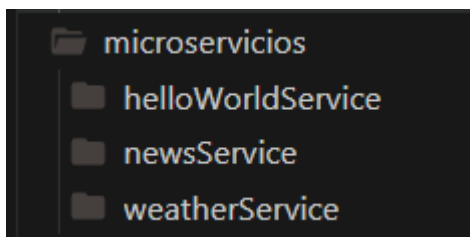
- Proteger APIs en contra de peticiones excesivas y limitar su uso
- Monitorear y analizar la actividad de las personas
- Conectar APIs específicas con servicios de cobro por uso
- Conectar a distintas APIs si la respuesta a una petición hace uso de varios microservicios
- Eliminar APIs obsoletas e introducir nuevas sin afectar las rutas en uso

## Propuesta de desarrollo

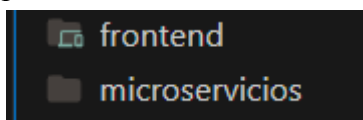
Para la creación del prototipo hemos desarrollado una aplicación la cual consta de microservicios y api gateway, ambos creados en el framework node js, utilizando dependencias como express, express-gateway y nodemon, además de un frontend desarrollado en el framework React con vite, con la finalidad de mostrar un uso práctico de estas tecnologías.

Como primer punto, debemos de tener descargado el node js en nuestro computador, para esto nos dirigimos a <https://nodejs.org/en/download> y decargamos su version LTS (long-term Support) ya que en palabras generales, es su versión más estable para el desarrollo.

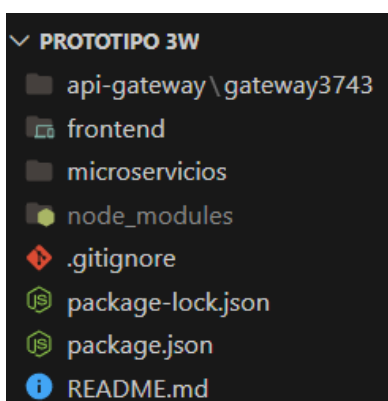
Luego de tener node js en nuestros computadores debemos de crear la jerarquía de carpetas bien diseñada, se debe de tener en cuenta que cada microservicio es un proyecto de node, entonces se debe crear una carpeta de microservicios general, y dentro de ella una carpeta que sea el microservicio el cual vamos a desarrollar, un ejemplo de esto es:



A su vez, el frontend también es un proyecto aparte, por ende debe de tener una carpeta solo para el:



Por último, la api-gateway también debe de estar aparte, sin embargo al instalar la dependencia express-gateway desde nuestra carpeta madre, y seguimos los pasos de la creación de la api, esta dependencia nos creará la carpeta al mismo nivel que las demás:



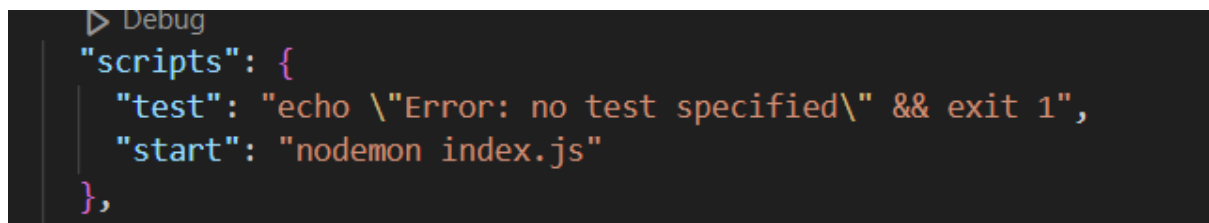


## Descripción de los microservicios desarrollados

Es importante aclarar que para cada microservicio se debió hacer individualmente la ejecución de los siguientes comandos:

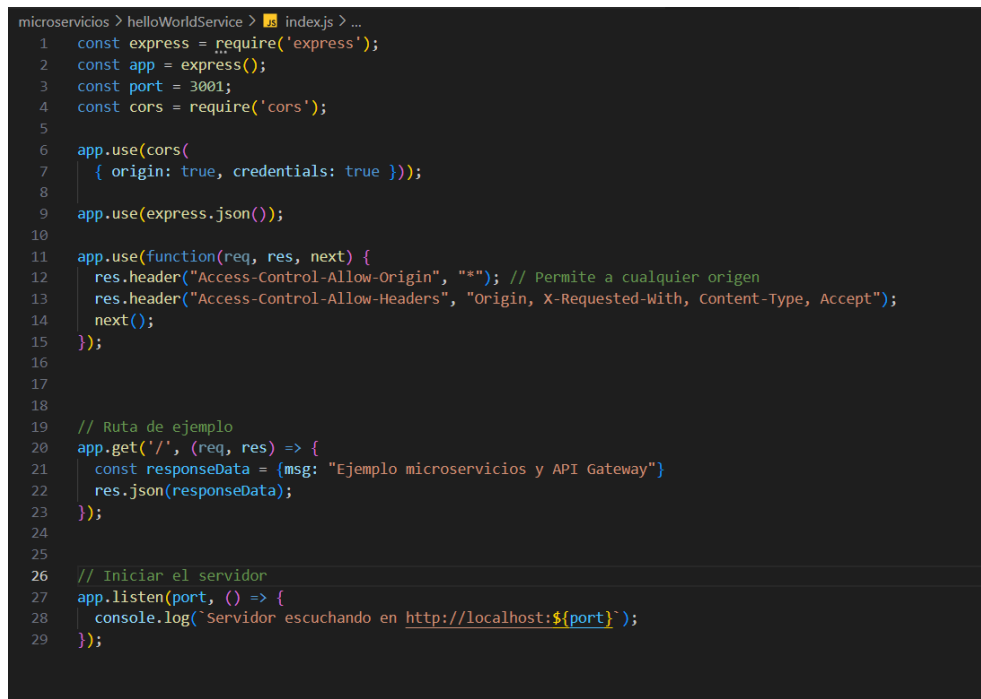
- npm install
- npm install express
- npm install nodemon

En el caso de nodemon luego de instalarlo se deberá por cada servicio modificar el package.json para que éste haga efecto, solo se debe añadir en la sección de scripts, el tag de start tal cual como aparece a continuación



```
Debug
"scripts": {
  "test": "echo \"Error: no test specified\" && exit 1",
  "start": "nodemon index.js"
},
```

**HelloWorldServices:** Servicio para dar un mensaje de bienvenida al momento de ingresar al aplicativo web.



```
microservicios > helloWorldService > index.js > ...
1  const express = require('express');
2  const app = express();
3  const port = 3001;
4  const cors = require('cors');
5
6  app.use(cors({
7    origin: true, credentials: true }));
8
9  app.use(express.json());
10
11 app.use(function(req, res, next) {
12   res.header("Access-Control-Allow-Origin", "*"); // Permite a cualquier origen
13   res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept");
14   next();
15 });
16
17
18
19 // Ruta de ejemplo
20 app.get('/', (req, res) => {
21   const responseData = {msg: "Ejemplo microservicios y API Gateway"}
22   res.json(responseData);
23 });
24
25
26 // Iniciar el servidor
27 app.listen(port, () => {
28   console.log(`Servidor escuchando en http://localhost:${port}`);
29 });
```

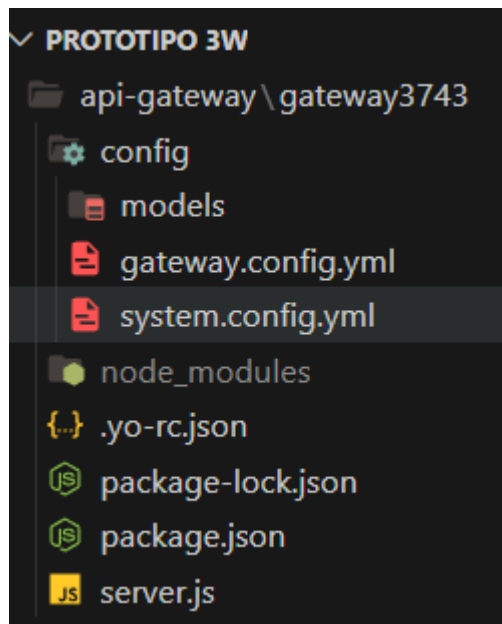
**newsService:** Microservicio que utiliza una api de terceros para importar noticias relevantes del mundo, dependiendo de la categoría en la que se desee filtrar

```
20
21 // Función para obtener noticias
22 const getNews = async (category) => { //las categorías pueden ser -> { general, business, entertainment, health,
23   const apiKey = '9491f04f4e3241cc8d030e0b49bcf87f';
24   const url = `https://newsapi.org/v2/top-headlines?country=us&category=${category}&pageSize=3&apiKey=${apiKey}`;
25   const response = await axios.get(url);
26
27   //console.log(response.data.articles.source)
28   //const newsData = JSON.parse(response.data);
29   //console.log(newsData)
30   return response.data.articles;
31 };
32
33 // Ruta para obtener noticias por categoría
34 app.get('/:category', async (req, res) => {
35   const category = req.params.category;
36   console.log(category)
37   console.log("entra")
38   try {
39     const newsData = await getNews(category);
40     console.log(newsData)
41     res.json(newsData);
42   } catch (error) {
43     res.status(500).json({ error: 'Internal Server Error' });
44   }
45 });
```

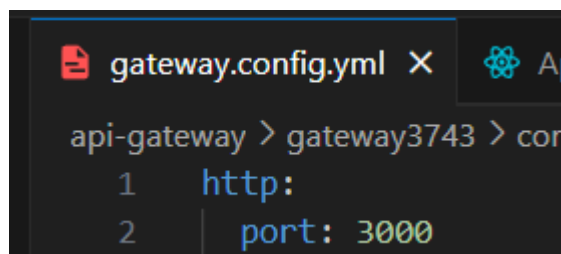
**WeatherService:** Servicio que utiliza una API de terceros para importar el clima según la ciudad en la que se desee consultar.

```
7   });
8
9 // Función para obtener el clima
10 const getClima = async (ciudad) => {
11   const urlAPIClima = `https://api.openweathermap.org/data/2.5/weather?q=${ciudad}&appid=ce26a0bc0211a75ef010c41fc92639d6`;
12   const respuesta = await axios.get(urlAPIClima);
13   console.log(respuesta.data)
14   return respuesta.data;
15 };
16
17 // Ruta para obtener el clima de una ciudad
18 app.get('/:ciudad', async (req, res) => {
19   const ciudad = req.params.ciudad;
20   const datosClima = await getClima(ciudad);
21   if (datosClima.cod !== 200) {
22     return res.status(400).json({ error: 'Ciudad no encontrada' });
23   }
24   res.json(datosClima);
25 });
26
27 // Inicio del servidor
28 app.listen(port, () => {
29   console.log(`Microservicio de clima iniciado en el puerto ${port}`);
30 });
31
```

## Funcionamiento de la api Gateway



En los archivos creados al momento de utilizar la dependencia express-gateway, tenemos que el archivo principal y en el que trabajaremos en este proyecto, será el archivo gateway.config.yml, pues ahí estará toda la lógica necesaria para que se realice las conexiones con los microservicios, entre toda esta configuración podemos destacar los siguientes puntos:



El número de puerto en el cual se ejecutará nuestra api Gateway

```
apiEndpoints:
  welcomeMessage:
    host: localhost
    path: '/mensajeBienvenida'

  weather:
    host: localhost
    paths: '/clima/:ciudad'

  news:
    host: localhost
    paths: '/noticias/:category'
```

En la sección de apiEndpoints encontraremos los endpoint de cada microservicio, algo a tener en cuenta es que están divididos por palabras claves para mantener un orden, pero además para poder añadirlas al flujo de trabajo (pipelines) que se definirá más adelante. Como partes a destacar, cada endpoint debe de tener su nombre clave, y dentro de él su host y por último el camino (path) sobre el cual tomará para entrar a un servicio en específico.

```
serviceEndpoints:
  welcomeMessageServices:
    url: 'http://localhost:3001'

  newsService:
    url: 'http://localhost:3002'

  weatherService:
    url: 'http://localhost:3003'
```

En la sección de serviceEndpoints encontraremos separados por nombres clave de servicios, cada microservicio que hemos desarrollado, además de la url directa para la comunicación con cada servicio.

```

weatherPipeline:
  apiEndpoints:
    - weather
  policies:
    - cors:
    - proxy:
      - action:
        serviceEndpoint: weatherService
        ignorePath: false
        prependPath: true
        stripPath: true
        changeOrigin: true
    - log:
      - action:
        message: "Procesando solicitud de clima"

```

En la sección de pipelines están los flujos de comunicación, también tendrá un nombre clave para diferenciarlo de los demás, aquí se tendrá que poner el nombre clave de cada endpoint, además, en la parte de action, se pondrá el nombre clave del servicio, de esta manera, cuando se encuentre en la petición http un endpoint que tenga un flujo de comunicación, entrara en dicho flujo y conectara con el microservicio deseado.

Por último, queda ejecutar la aplicación para poder mostrar su funcionalidad, además de la vista creada con React.

Para dar un buen funcionamiento, se deberán tener abiertas 5 terminales al tiempo, las cuales van en:

1. Terminal de ejecución de la api Gateway.
2. Terminal de ejecución del frontend
3. Terminal de ejecución del microservicio **HelloWordServices**
4. Terminal de ejecución del microservicio **newsService**
5. Terminal de ejecución del microservicio **WeatherService**

Nota: Para cada uno de los servicios antes mencionados, deben de ejecutarse en la carpeta correspondiente.

Y este es el resultado final:

Ejemplo microservicios y API Gateway

Aplicación que cuenta con 2 microservicios que corren en node junto con la librería express. Para implementar el API Gateway, se usó la librería de código abierto express-gateway, permitiendo configurar de manera declarativa características como el enrutamiento de solicitudes, entre otras.

Microservicio 1

Este microservicio consulta una API de [openweathermap](#) con información relevante del clima y la ubicación de la ciudad en el momento.

Bogotá

Consultar


City: Bogotá

Latitude: 4.6097

Longitude: -74.0817

Temperature: 12°C

Weather condition: Few clouds



Microservicio 2

Este microservicio consulta una API de [newsapi](#) y trae noticias relevantes dependiendo de la categoría que se escoja

General

Consultar


Source: ESPN

Published at: 2024-03-19T02:36:00Z

Author: Alden Gonzalez

Title: Source -- Blake Snell agrees to 2-year, \$62M deal with Giants - ESPN

Description: Blake Snell has agreed to a 2-year, \$62 million deal with the Giants, a source told ESPN's Jeff Passan, confirming a report by the New York Post. It includes an opt-out after the first season.



Source: CNN

Published at: 2024-03-19T01:50:00Z

Author: Jason Morris, Zachary Cohen

Title: Trump and co-defendants re-up their efforts to disqualify DA Fani Willis - CNN

Description: Former President Donald Trump and seven of his co-defendants in the January 2020 election subversion case are seeking

## **Conclusión**

Los microservicios y las API Gateway se desarrollan como componentes esenciales dentro de la mejora de las aplicaciones web de vanguardia, anunciando acuerdos exitosos para avanzar en la calidad medida y la adaptabilidad. Los microservicios, al dividir las aplicaciones sólidas en componentes libres y autosuficientes, permiten un avance rápido, un soporte racionalizado y una adaptabilidad más productiva, lo que conlleva a una modularidad y escalabilidad más óptima que una arquitectura monolítica. Por otro lado, las API Gateway actúan como intermediarios entre los clientes y los microservicios, centralizando la administración del acceso, la seguridad, la comprobación y la dirección de las peticiones HTTP, lo que fomenta la organización de numerosas administraciones y el uso de enfoques estratégicos.

La combinación de innovaciones como Node.js, JavaScript, React y Express fomentan la mejora de este diseño, potenciando el desarrollo de microservicios productivos y la creación de interfaces de cliente enérgicas e intuitivas, proporcionan un entorno de ejecución rápido y productivo para la mejora de las administraciones de backend, mientras que React ofrece la creación de interfaces de intuitivas en el frontend.

La introducción de API bien diseñadas y comunicadas fomenta la integración de beneficios y la colaboración entre grupos de avance. Juntos, estos avances y perfeccionamientos promueven el desarrollo de aplicaciones web vigorosas, adaptables y viables en un entorno de comercio en constante avance. La apropiación de microservicios y API Gateway se muestra posteriormente como una metodología clave para impulsar el aislamiento, la adaptabilidad y la productividad en el avance de aplicaciones web.

## Bibliografía

- IBM: ¿Qué son los microservicios?: <https://www.ibm.com/es-es/topics/microservices>
- Red Hat: ¿Qué son y para qué sirven los microservicios?: <https://www.redhat.com/es/topics/microservices>
- Red Hat: ¿Qué es una API?: <https://www.redhat.com/en/topics/api>
- AWS: ¿Qué es una API? <https://docs.aws.amazon.com/appsync/latest/devguide/what-is-an-api.html>
- ¿Qué es una API Gateway?: <https://blog.hubspot.es/website/que-es-api-gateway>
- <https://learn.microsoft.com/es-es/azure/architecture/microservices/design/gateway>
- Documentación de express-gateway: <https://www.express-gateway.io/docs/>
- Documentación de React: <https://react.dev/reference/react>
- Documentación de Node js: <https://nodejs.org/docs/latest/api/>