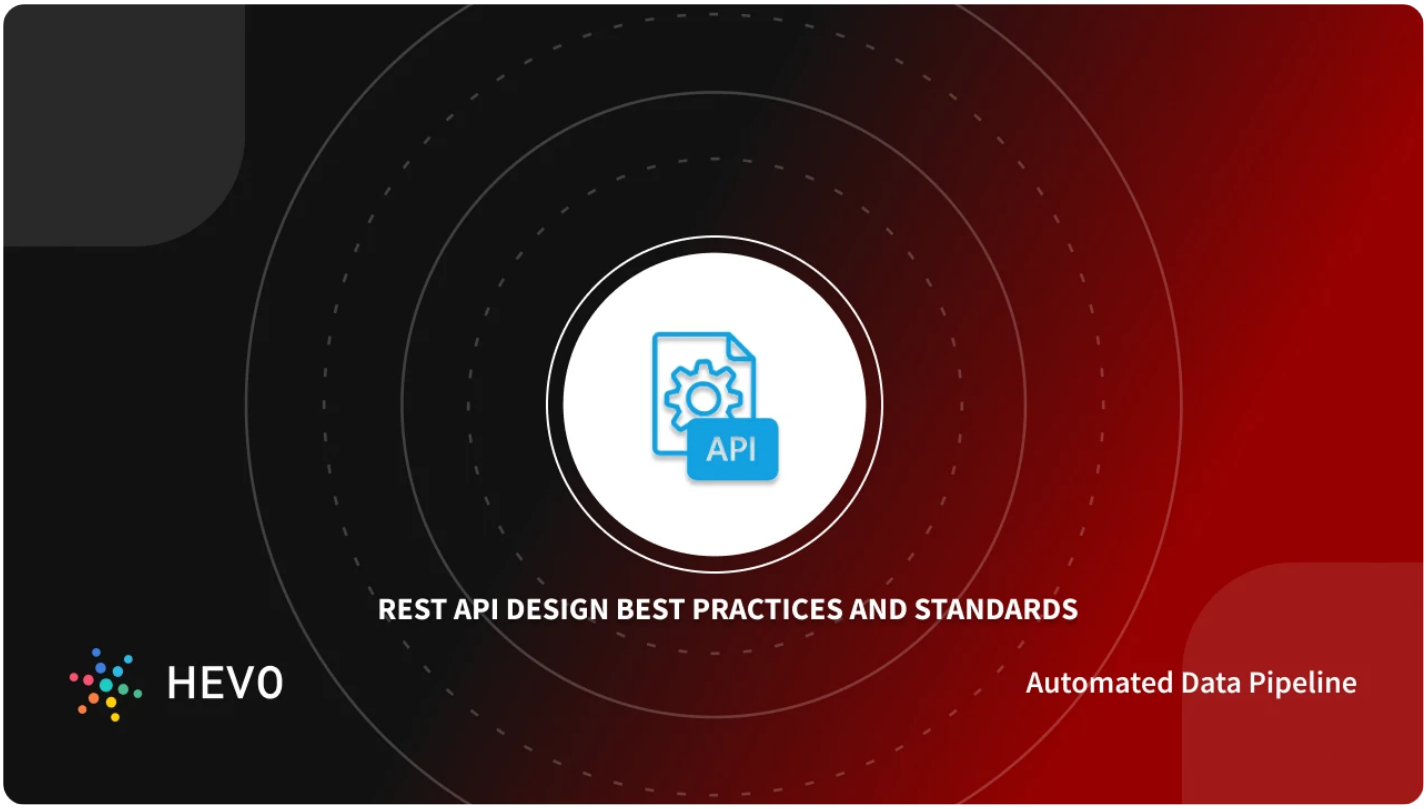
HEVO

Mejores prácticas y estándares de API REST en 2022

Sherly Angel sobre API , REST API , ingeniería de software , tutoriales • 1 de noviembre de 2021 • **ESCRIBIR PARA HEVO**



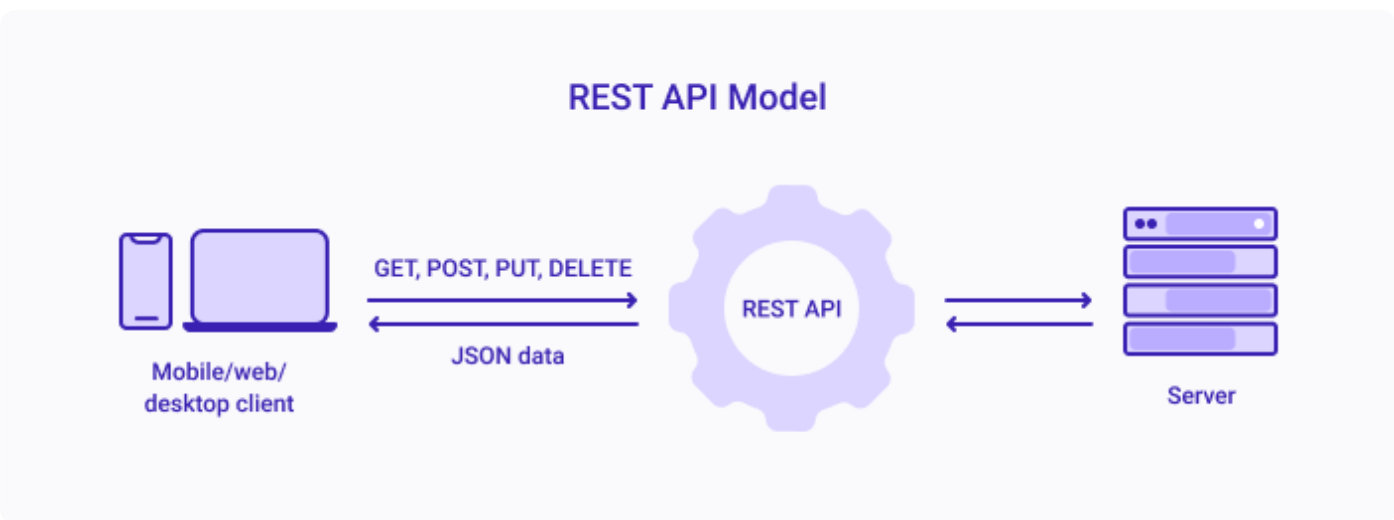
¡Dominar las mejores prácticas de la API REST es un arte! Las API son utilizadas por todos los profesionales del software, pero no todos pueden escribir las mejores. Las preocupaciones de seguridad de las personas debido a las API mal escritas necesitan atención. Cuando diseñe API REST, estas mejores prácticas de API REST lo ayudarán a mejorar sus habilidades de escritura de API. Como diseñador de API REST, escribir una API eficaz facilitará su trabajo.

En este blog, se le presentará la API REST junto con los estándares de la API REST. Se elabora el funcionamiento y características de la API REST. Se discutirán diez mejores prácticas de API REST con ejemplos.

Tabla de contenido

- ¿Qué es la API REST?
- Estándares API REST
- Funcionamiento de la API REST
- Características de una API bien diseñada
- Prácticas recomendadas de la API REST
  - Prácticas recomendadas de la API REST: Priorizar los sustantivos sobre los verbos
  - Prácticas recomendadas de la API REST: prefiera usar convenciones de nomenclatura en plural
  - Prácticas recomendadas de la API REST: utilice el anidamiento de recursos de manera eficiente
  - Prácticas recomendadas de API REST: documentación sistemática
  - Prácticas recomendadas de la API REST: opciones de filtrado de datos
  - Prácticas recomendadas de la API REST: utilice capas de seguridad SSL/TLS
  - Prácticas recomendadas de la API REST: Centrarse en el manejo de errores
  - Prácticas recomendadas de la API REST: elija siempre JSON
- Conclusión

¿Qué es la API REST?



Fuente de la imagen: rlógico

REST API es una API que sigue un conjunto de reglas para que una aplicación y servicios se comuniquen entre sí. Como está restringida a la arquitectura REST, la API REST se denomina API

RESTful. Las API REST proporcionan una forma de acceder a los servicios web de forma flexible sin capacidades de procesamiento masivas.

Estándares API REST

Los estándares de la API REST son obligatorios para todas las API REST. Los estándares de la API REST tienen una lista de restricciones a cumplir. Estas restricciones se explican a continuación.

1) Apatridia

Los sistemas que se alinean con el paradigma REST están destinados a convertirse en sin estado. Para la comunicación Cliente-Servidor, la restricción sin estado obliga a los servidores a permanecer inconscientes del estado del cliente y viceversa. Se aplica una restricción mediante el uso de recursos en lugar de comandos, y son sustantivos de la web que describen cualquier objeto, documento o cosa para almacenar/enviar a otros recursos.

2) almacenable en caché

Cache ayuda a los servidores a mitigar algunas limitaciones de la apatridia. Es un factor crítico que ha mejorado el rendimiento de las aplicaciones web modernas. El almacenamiento en caché no solo mejora el rendimiento en el lado del cliente, sino que también escala resultados significativos en el lado del servidor. Un mecanismo de caché bien establecido reduciría drásticamente el tiempo de respuesta promedio de su servidor.

3) desacoplado

REST es un enfoque distribuido, donde las aplicaciones de cliente y servidor están desacopladas entre sí. Independientemente de dónde se inicien las solicitudes, la única información que conoce la aplicación cliente es el Identificador **Uniforme de Recursos (URI)** del recurso solicitado. Una aplicación de servidor debe pasar los datos solicitados a través de HTTP, pero no debe intentar modificar la aplicación cliente.

4) Sistema en capas

Un sistema en capas hace que una arquitectura REST sea escalable. Con la arquitectura RESTful, las aplicaciones Cliente y Servidor están desacopladas, por lo que las llamadas y respuestas de las API REST pasan por diferentes capas. Como la API REST está en capas, debe diseñarse de tal manera que ni el Cliente ni el Servidor identifiquen su comunicación con las aplicaciones finales o un intermediario.

5) Cliente-Servidor

Las aplicaciones del cliente y del servidor deben poder funcionar sin la ayuda de los demás. Tanto la aplicación del servidor como la aplicación del cliente deben cumplir con el acuerdo de separación de intereses. Por este acuerdo, al alterar el extremo del cliente, no debería haber ningún impacto en la aplicación del servidor. Y además, cuando se altera el código del servidor, no debería afectar al extremo del cliente. Esto mejora la escalabilidad y la flexibilidad de la interfaz entre plataformas.

6) Código bajo demanda (opcional)

De todas las restricciones, esta es opcional. El formato habitual utilizado al enviar recursos es JSON REST API o XML. Pero siempre que sea necesario, se le proporciona una opción para devolver el código ejecutable. Esto apoyará la parte principal de su aplicación.

- **Transferencia de datos en tiempo real:** Hevo proporciona migración de datos en tiempo real desde fuentes de datos como Google Analytics y Shopify, para que siempre pueda tener datos listos para el análisis.
- **Transferencia de datos 100% completa y precisa:** la sólida infraestructura de Hevo garantiza una transferencia de datos confiable sin pérdida de datos.
- **Infraestructura escalable:** Hevo tiene integraciones incorporadas para más de 100 fuentes que pueden ayudarlo a escalar su infraestructura de datos según sea necesario.
- **Soporte en vivo las 24 horas, los 7 días de la semana:** el equipo de Hevo está disponible las 24 horas del día para brindarle un soporte excepcional a través de chat, correo electrónico y llamadas de soporte.
- **Gestión de esquemas:** Hevo elimina la tediosa tarea de la gestión de esquemas y detecta automáticamente el esquema de los datos entrantes y lo asigna al esquema de destino.
- **Monitoreo en vivo:** Hevo le permite monitorear el flujo de datos para que pueda verificar dónde están sus datos en un momento determinado.

¡REGÍSTRESE AQUÍ PARA UNA PRUEBA GRATUITA DE 14 DÍAS!

Funcionamiento de la API REST

Una API REST requiere una URL de host que actúe como la dirección principal para sus interacciones. Las API REST también necesitan un conjunto de puntos finales, que son direcciones únicas dentro de las URL del host responsables de su funcionalidad. Además, es una buena práctica documentar los puntos finales, el valor devuelto, los tipos de datos y otros elementos esenciales de una API REST.

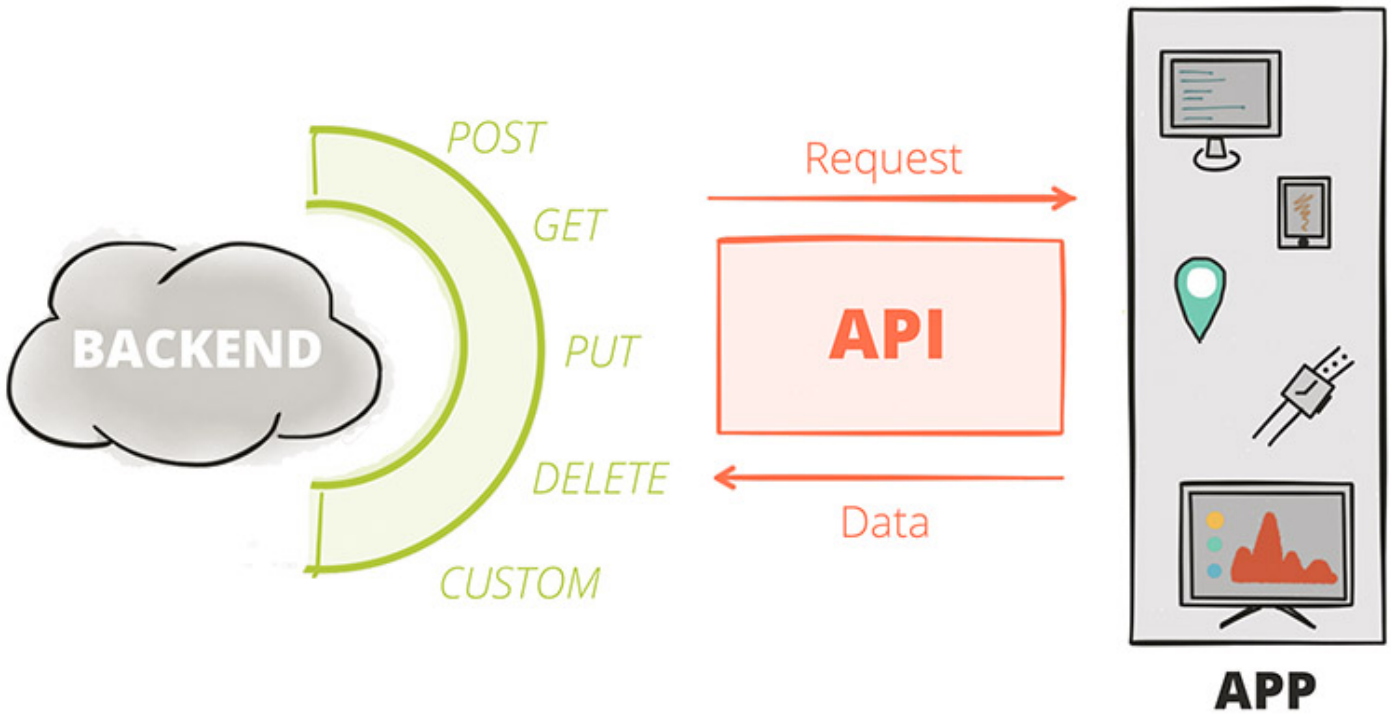
El siguiente diagrama es una representación de alto nivel de la organización requerida de su código para crear una API REST. Puede tener una o más bases de datos que contengan datos que otras aplicaciones podrían necesitar. Entonces, usarán la API REST que usa SQL y JDBC para interactuar con la base de datos. Las API REST le permiten centralizar toda su lógica básica en un solo lugar en lugar de volver a escribirla cada vez que desee crear una nueva aplicación, como se muestra en la imagen a continuación.

Fuente de la imagen: codificación feliz

Ahora, las API están diseñadas para devolver los datos requeridos cada vez que un usuario las llama. Sin embargo, cuando usa REST APIS, no solo devuelve los datos solicitados, sino que también los presenta en una forma bien estructurada para su representación. Una API REST utiliza una arquitectura cliente-servidor que permite que diferentes aplicaciones se comuniquen. El software del cliente realiza una llamada a la aplicación del servidor mediante una API REST. La aplicación del servidor envía los datos solicitados de forma estructurada y organizada utilizando parámetros clave sobre el protocolo HTTP.

Características de una API bien diseñada

- **Flexible** : la API REST es flexible con múltiples tipos de llamadas, como devolver diferentes formatos de datos y cambiar estructuralmente con la implementación correcta de hipermedia. Permite a los usuarios comunicarse de un lado a otro con clientes y servidores, incluso si están alojados en servidores diferentes.
- **Adaptable** : la API REST se adapta a cualquier modificación realizada en los datos que residen en la base de datos, incluso cuando están alojados en los diferentes servidores back-end y front-end. Dado que depende hasta cierto punto de los códigos, ayuda a sincronizar los datos dentro de los sitios web sin ningún problema.
- **Facilidad de comprensión** : como REST utiliza métodos de verbos HTTP (GET, POST, PUT o DELETE) para la comunicación, estos métodos se explican por sí mismos. Además, la arquitectura REST ayuda a aumentar la productividad de los desarrolladores, permitiéndoles mostrar la información en el lado del cliente y almacenar o manipular los datos en el lado del servidor.



Fuente de la imagen: tethys-staging

Prácticas recomendadas de la API REST

Al diseñar las API de REST, debe centrarse en todas estas prácticas recomendadas para que su API de REST sea la mejor. Como diseñador de API REST, debe centrarse tanto en la seguridad como en el funcionamiento de la API.

Prácticas recomendadas de la API REST: Priorizar sustantivos sobre verbos en URI

Dado que la API REST se desarrolla principalmente para recursos como servicios, es esencial usar sustantivos y no verbos. Por lo tanto, es mejor usar solo nombres para representar una entidad en las rutas de los puntos finales REST. Esto se debe a que el método de solicitud HTTP ya consta de verbos. Por lo tanto, tener un verbo en los puntos finales de la API REST no generará ninguna información nueva. Debe usar etiquetas para cambiar el estado del recurso.

La siguiente tabla lo ayuda a comprender los verbos de la API REST:

DESCANSAR Verbo	**Acción**
OBTENER	Obtiene un registro o conjunto de recursos del servidor

DESCANSAR Verbo	Acción
OPCIONES	Obtiene todas las operaciones REST disponibles
CORREO	Crea un nuevo conjunto de recursos o un recurso
PONER	Actualiza o reemplaza el registro dado
PARCHE	Modifica el registro dado
ELIMINAR	Elimina el recurso dado

Aquí hay algunos ejemplos para mostrar cómo deberían verse los puntos finales,

- GET/libros/123
- ELIMINAR/libros/123
- POST/libros
- PUT/libros/123
- PARCHE/libro/123

Prácticas recomendadas de la API REST: prefiera usar convenciones de nomenclatura en plural

En general, la mejor práctica es usar sustantivos en plural para las colecciones. Esta convención de nomenclatura plural se convierte en un código global. Esto también ayuda a las personas normales a comprender que estos grupos de API forman una colección.

La siguiente tabla lo ayuda a comprender el uso correcto e incorrecto de los nombres plurales en la API REST:

hacer	no hacer
OBTENER/bicicletas/123	OBTENER/bicicleta/123

hacer	no hacer
POST/bicicletas	POSTE/bicicleta
OBTENER/bicicletas	OBTENER/bicicleta

Prácticas recomendadas de la API REST: utilice el anidamiento de recursos de manera eficiente

El anidamiento de recursos es una práctica de agrupar dos funciones que tienen cierta jerarquía o están vinculadas entre sí. Anidar a un nivel es una de las mejores prácticas para agrupar recursos que son lógicamente coherentes. Por ejemplo, 'pedido' y 'usuarios' son dos recursos de la misma categoría en una tienda online. El 'usuario' hace el 'pedido' y el 'pedido' pertenece al 'usuario'. El siguiente código explica el escenario discutido anteriormente.

```
/users // list all users
/users/123 // specific user
/users/123/orders //list of orders that belong to a specific user
/users/123/orders/0001 // specific orders of a specific users order list
```

Overusing Nesting is not good in any case. When overused, Nesting loses its appeal and creates unwanted dependency issues. So the REST API best practice that can be followed is limiting the use of nesting to one level.

REST API Best Practices: Systematic Documentation

Another important REST API best practice is to document all the solutions in a very systematic manner. The utilization of framework, application, or software usage requires proper documentation. This document will act as a reference while troubleshooting an issue. This API documentation needs to be precise and simple enough for non-technical people to understand it. Doing such systematic documentation will help your users indulge and understand all the necessary aspects like error handling, security, and authentication.

REST API Best Practices: Data Filtering options

When the database grows, it becomes a great challenge to manage it. The main challenge in this huge database is to retrieve only the requested data. The entire database should not be exposed while retrieving data. For fulfilling this, you need to use a filter that will pull data that satisfies the required criteria. By filtering the data while retrieving, huge bandwidth is saved in the client's end. REST API provides you with 4 types of filtering options. The REST API filtering options include:

- Filtering
- Sorting
- Paging
- Field Selection

Filtering

Using this you can filter results that satisfy your required conditions. You can use search parameters like country, creation, date and etc for this.

```
ET /users?country=UK
GET /users?creation_date=2021-10-11
GET /users?creation_date=2021-10-11
```

Sorting

You can sort your results in ascending and descending order using this option.

```
GET /users?sort=birthdate_date:asc
GET /users?sort=birthdate_date:desc
```

Paging

Using the 'limit' option, you can narrow down the results to the required number. You can also use 'offset' to show the part of the overall results displayed.

```
GET /users?limit=120
GET /users?offset=3
```

Field Selection

Using the field selection function, you can request to display a specific part of the data available for that object. While you query an object with many fields, you can specify the fields in your response. An object will have ‘Name’, ‘Surname’, ‘Birthdate’, ‘Email’, ‘Phone’ as its fields.

For example, when you want to retrieve the birthdate and email to automate birthday wishes. You can use a query like this:

For a specific user:

```
GET/  users/123?fields=name,birthdate,email
```

For a full list of users:

```
 GET/ users?fields=name,birthdate,email
```

REST API Best Practices: Utilize SSL/TLS security layers

One of the REST API Best practices is to encrypt the communication using SSL/TLS. It is very essential to ensure database security for any API developer. The earned trust of the customers to keep their sensitive details private is a must. To avoid security breaches, you need to use SSL (Secure Socket Layer) and TLS (Transport Layer Security). SSL/TSL provides a public and private key to give a secured connection. TSL is an advanced version of SSL and hence provides better protection and security.

REST API Best Practices: Focus on Error Handling

Handling error with care is one essential skill of an API developer. The HTTP error code will point to the nature of the individual error when the API is effective. REST API has 71 unique errors with HTTP status

codes with error messages. Along with the standard error handling of HTTP statuses, and elaborated message on the internal code will help the user to understand better.

Idel error handling code consists of 3 parts:

- **Error –** a unique identifier of the error
- **Message –** a comprehensive, readable message
- **Detail –** lengthier explanation of the message

For example, when you receive a login response with an incorrect password, you can send a 401 response with a code like this,

```
{ "error": "auth-0001", "message": "Incorrect username and password", "detail": "Ensure that the username and password included in the request are correct" }
```

REST API Best Practices: Always choose JSON

La notación de objetos de JavaScript es uno de los lenguajes más fáciles y un formato fácil de usar. Una de las mejores prácticas importantes a seguir es elegir siempre JSON. La característica clave de JSON es que es muy fácil de analizar y es compatible con la mayoría de los marcos. JSON puede ser utilizado por cualquier lenguaje de programación.

Conclusión

En este blog, habría aprendido sobre la API REST junto con los estándares de la API REST. El funcionamiento y las características de la API REST serán claros para usted ahora. Diez prácticas recomendadas de API REST con ejemplos son todas suyas. ¡Que estas esperando! Úsalos y domina este ARTE. **El conector API REST nativo** de Hevo puede ayudar a conectarse con una variedad de fuentes no nativas/personalizadas en su Data Warehouse para visualizarlas en una herramienta de BI.

Hevo Data ofrece una forma más rápida de mover datos de más de 100 fuentes de datos , como la **API REST de forma gratuita** , a su almacén de datos para visualizarlos en una herramienta de BI. Hevo

está totalmente automatizado y, por lo tanto, no requiere que codifiques. Hevo proporciona un conector API REST nativo preconstruido que le permitirá integrar datos de una gran cantidad de fuentes personalizadas y no nativas. Todo esto sin escribir una sola línea de código y sin costo alguno.

¿Quieres llevar a Hevo a dar una vuelta? INSCRIBIRSE para una **prueba gratuita de 14 días** y experimente la suite Hevo rica en funciones de primera mano. También puede echar un vistazo a los precios imbatibles que lo ayudarán a elegir el plan adecuado para las necesidades de su negocio.

Comparta sus conocimientos sobre el tema de las mejores prácticas de la API REST. ¡Cuéntanos en los comentarios a continuación!

Prácticas recomendadas de la API REST Características de la API REST Diseñador de API REST Estándares API REST API REST

Sigue leyendo

Manjiri Gaikwad sobre integración de datos, almacenes de datos , Firebase Analytics , Snowflake , tutoriales

Integración de Firebase Analytics a Snowflake: 2 métodos sencillos

Kamya en API , MongoDB

Integración FastAPI MongoDB: 5 sencillos pasos

PLATAFORMA

Producto

integraciones

Precios

Prueba gratis

registro de cambios

Próximas funciones

Estado

CONCEPTOS

ETL

Desplazamiento al rojo de Amazon

Google Big Query

Copo de nieve

GUÍAS DE COMPARACIÓN

Herramientas ETL

Herramientas de canalización de datos

Herramientas de integración de datos

Desplazamiento al rojo frente a BigQuery

BigQuery frente a copo de nieve

Copo de nieve vs corrimiento al rojo

TUTORIALES

Redshift ETL

ETL de BigQuery

Copo de nieve ETL

Cambiar la captura de datos

ESCRIBIR PARA HEVO

Puede contribuir con cualquier cantidad de publicaciones detalladas sobre todos los datos.

Saber más

HEVO