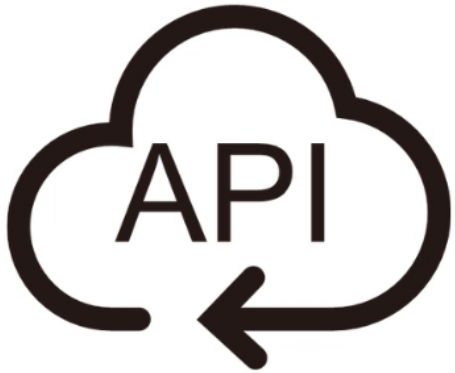


Popular API Architecture Patterns

Kanani Nirav
Posted on 18 Sep 2023 • Edited on 19 Sep 2023

Top 6 Most Popular API Architecture Styles You Need to

Kanani Nirav

Software Engineer 💻
Passionate About Continuous Growth And Development.

LOCATION
Tsukuba, Japan

WORK
Software Engineer (Ruby On Rails)

JOINED
8 Aug 2022

More from Kanani Nirav

The Power of SEO: How my website ranked higher in just 48 hours

#seo #webdev #productivity #beginners

Know (with Pros, Cons, and Use Cases)

#api #webdev #beginners #softwareengineering

In this article, we will discuss about 6 Most Popular API Architecture Styles: **REST, SOAP, GraphQL, gRPC, WebSocket, and Webhooks.**

APIs are ways for software to talk to each other and share data. There are different ways to design and build APIs. Let's see the top 6 API Architecture Styles and their pros, cons, and Use Cases.

REST

REST is a way to use web standards and addresses to work with data on a server. They're popular, easy to implement, and use HTTP methods. Most of the web services you interact with daily, like Twitter or YouTube, are powered by Restful APIs. For example, a client can use GET to get data, POST to make new data, PUT to

Page Views Counter: Netlify Dynamic Site Challenge using Netlify Blobs
#netlifychallenge #devchallenge #webdev #javascript

Monorepo vs Microrepo: How to Choose the Best Repository Structure for Your Code
#git #developer #beginners #softwareengineering

change data, or DELETE to remove data. The data are usually in formats like JSON or XML.

Pros

- It is simple and easy to use and understand.
- It follows the rules of the web and uses existing standards and protocols.
- It is fast and can handle many requests, as it supports caching and statelessness.
- It is flexible and can use different formats and media types.

Cons

- It does not have a clear contract or schema, which can make it unclear and inconsistent.
- It does not support complex queries or operations, which can make it need many requests and get too much or too little data.
- It does not handle errors or exceptions well, as it uses HTTP status codes that are not always clear or correct.

Usage

REST is good for situations where:

- The data model is simple and stable.
- The clients and servers are not dependent on each other.
- The speed and scalability are important.

SOAP

SOAP is s style that uses XML messages and a predefined contract to exchange information between applications. It has a clear and strict contract, and it supports complex queries and operations, but it is complex, verbose, and not scalable or performant.

Pros

- It has a clear and strict contract that ensures interoperability and compatibility.
- It supports complex queries and operations, such as transactions, security, or authentication.
- It handles errors and exceptions well, as it uses SOAP faults that provide detailed information.

Cons

- It is complex and verbose to use and understand.
- It does not follow the principles of the web and adds overhead to the existing protocols.
- It is not scalable or performant, as it does not support caching or statelessness.
- It is not flexible or extensible, as it requires changes to the contract for any modifications.

Usage

SOAP is suitable for scenarios where:

- The data model is complex and dynamic.
- The clients and servers are tightly coupled and dependent.
- It’s heavily used in financial services and payment gateways where security and reliability are key

GraphQL

GraphQL is not just an architectural style but also a query language, allowing clients to ask for specific data as they need. This means no more over-fetching or under-

fetching of data. You ask for exactly what you need. This leads to more efficient network communication and faster responses. Facebook developed GraphQL to deliver efficient and precise data to its billions of users. Now it's used by companies like GitHub and Shopify. Its flexibility and efficiency make it a strong choice for applications with complex data requirements. It also supports mutations to change the data, and subscriptions to get live updates when the data changes.

Pros

- It works with any server language or framework, as it uses its own schema definition language.
- It has a single endpoint, which makes it more efficient than REST, where multiple requests might be needed to get enough data.
- It is strongly typed, which makes sure the data is consistent and compatible between the client and the server.
- It lets clients fetch only the data they need, which avoids over-fetching or under-fetching of data.

Cons

- It is complex and hard to use and understand, as it requires learning a new syntax and logic.
- It does not support caching by default, which can affect performance and scalability.
- It does not handle errors well, as it always returns HTTP 200 status code, even if there are errors in the GraphQL layer.

Usage

GraphQL is good for situations where:

- The data model is complex and dynamic, as it can handle nested and relational data.
- The clients and servers are not dependent on each other, as it lets clients define their own data requirements.
- The bandwidth and performance are important, as it reduces the amount of data transferred.

gRPC

gRPC is modern, high-performance, and uses protocol buffers. It's a favorite for microservices architectures,

and companies like Netflix use gRPC to handle their immense interservice communication. However, if you're dealing with browser clients, gRPC might pose some challenges due to limited browser support.

Pros

- It has a clear and strict contract, which ensures interoperability and compatibility between services.
- It supports complex queries and operations, such as streaming, bidirectional communication, authentication, or encryption.
- It is fast and efficient, as it uses binary format and HTTP/2 features to reduce latency and bandwidth.

Cons

- It is complex and hard to use and understand, as it requires generating and compiling protocol buffer files.
- It does not follow the principles of the web, as it uses custom headers and methods that are not compatible with standard web tools or browsers.

Usage

gRPC is good for situations where:

- The data model is complex and dynamic, as it can handle structured and unstructured data.
- The services are dependent on each other, as it allows services to invoke each other's methods directly.
- The speed and efficiency are important, as it minimizes the overhead and maximizes the throughput.

WebSocket

WebSocket is all about real-time, bidirectional, and persistent connections. It's perfect for live chat applications and real-time gaming, where low-latency data exchange is crucial.

Pros

- It is faster and more efficient than HTTP, as it uses a single connection and does not need headers or cookies for each message.
- It can send and receive different types of messages over the same connection, such as text, binary, or

streaming data.
- It can push data from the server to the client without waiting for a request, which enables real-time and event-driven communication.

Cons

- It’s not supported by some older browsers or proxies that do not understand the WebSocket protocol or the upgrade header.
- It’s not secure by default, as they do not use encryption or authentication unless they use the `wss://` scheme, which is similar to `https://` for HTTP.
- It does not store any information about the connection or the messages on either side, which means they are not stateful.

Usage

WebSocket is good for situations where:

- The web application needs fast and interactive data exchange, such as chat, gaming, or streaming.
- The web application needs bidirectional and

multiplexed communication, where both sides can send and receive multiple messages of different types at the same time.
- The web application needs real-time and event-driven communication, where the server can send data to the client without waiting for a request.

Webhooks

Webhooks are a way for servers to send messages to clients when something happens. They use HTTP callbacks or POST requests to deliver payloads that contain information about the events. The clients register their webhooks with the servers by providing URLs that can receive the payloads. Webhook is all about event-driven, HTTP callbacks, and asynchronous operation.

Pros

- It is simple and easy to use and understand, as it uses standard HTTP methods and formats.
- It follows the principles of the web, as it leverages existing standards and protocols.
- It is scalable and performant, as it supports

asynchronous communication without polling or waiting.

Cons

- It does not have a clear or strict contract, which can lead to inconsistency and ambiguity between servers and clients.
- It does not support complex queries or operations, it only sends one-way notifications without confirmation or feedback.
- It does not handle errors well, It does not provide retries or acknowledgments in case of failures.

Usage

Webhooks are good for situations where:

- Event-driven notifications (GitHub uses webhooks to notify your other systems whenever a new commit is pushed.)
- The data model is simple and stable, as it can handle basic types and fields.
- The servers and clients are not dependent on each other, as they do not require direct communication or

coordination.
- The performance and scalability are important, as they reduce the load and latency of the communication.

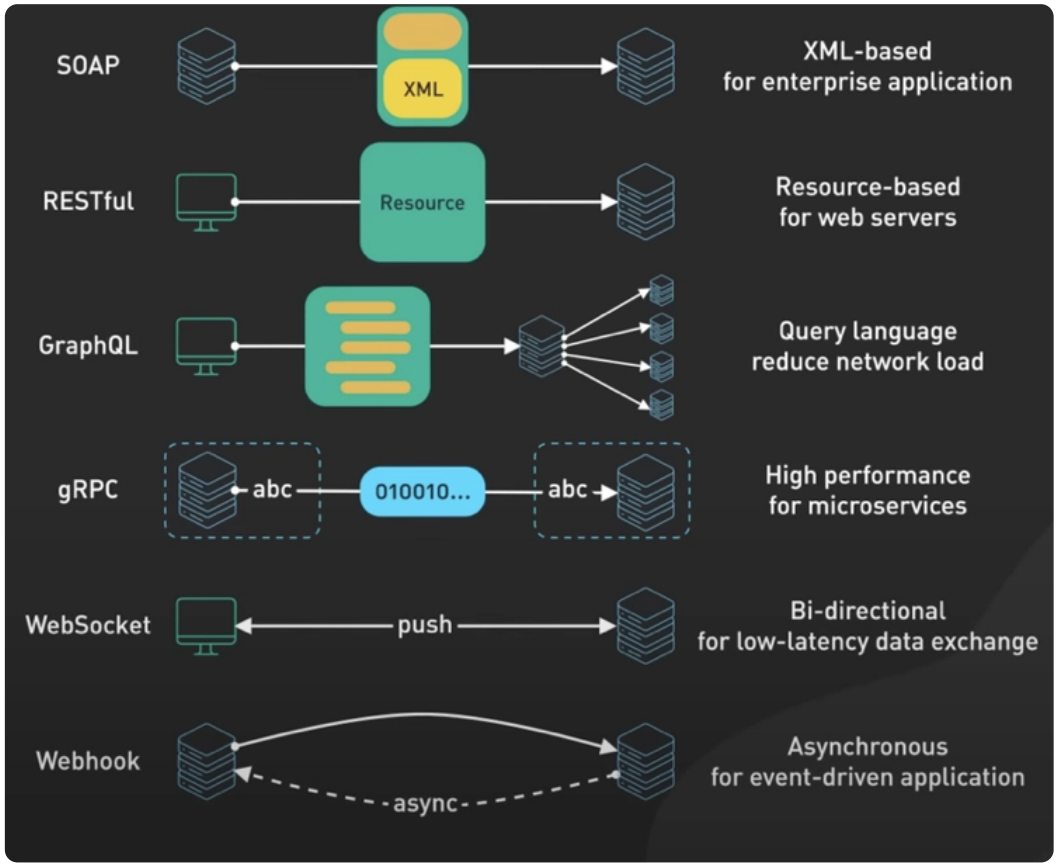
Here is the final summarized table

API Style	Pros	Cons	Use Cases
REST	Simple, flexible, scalable, web-friendly	No clear contract, no complex queries, poor error handling	Data-orient applications web service caching, statelessne
SOAP	Clear contract, complex queries, good error handling	Complex, verbose, not scalable, not web-friendly	Transaction applications security, authenticati interoperab
GraphQL	Language agnostic, single endpoint,	Complex, hard to use, no	Complex an dynamic da models, clie driven

	strongly typed, data efficiency	caching, poor error handling	requiremen bandwidth a performanc
gRPC	Language agnostic, clear contract, complex queries, fast and efficient	Complex, hard to use, not web-friendly, not flexible	Complex an dynamic da models, microservic communica speed and efficiency
WebSocket	Fast and efficient, bidirectional and multiplexed communication, real-time and event-driven communication	Not supported by some older browsers or proxies, not secure by default, not stateful	Fast and interactive c exchange s as chat, gar or streamin
	Simple, easy to use, web-	No clear contract, no complex	Simple and stable data models, eve driven

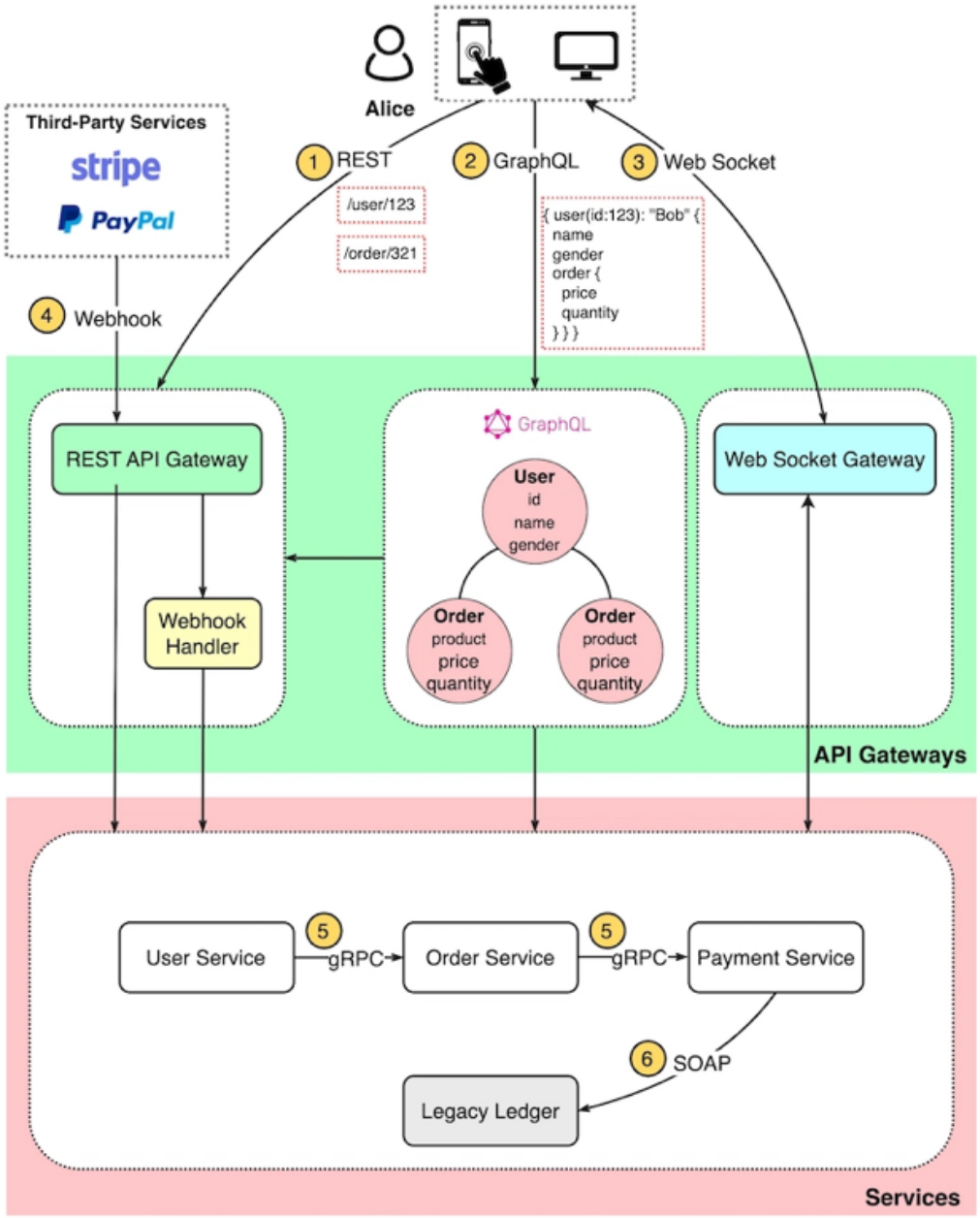
Webhooks	friendly, scalable	queries, poor error handling	notifications performanc and scalabil

API-architecture-styles-summary.md hosted with ❤ by GitHub view raw



Sample API architectural Styles

API Architectural Styles



Reference

- Top 6 Most Popular API Architecture Styles - ByteByteGo
- https://blog.bytebytego.com/p/ep49-api-architectural-styles
- https://www.postman.com/state-of-api/api-technologies/#api-technologies

If You are using Medium Please support and follow me for interesting articles. **Medium Profile**

Subscribe to My Newsletter:

If you're ready to subscribe, simply click the link below:

Subscribe to My Newsletter

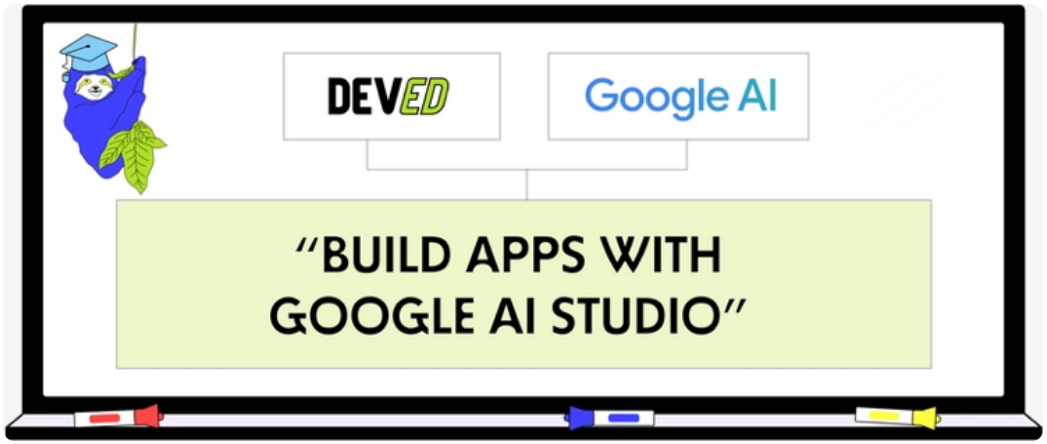
By clicking the link, you'll be directed to the newsletter page, where you can easily subscribe and receive regular updates delivered directly to your inbox. Don't miss out on the latest trends and expert analysis in software development.

Stay updated with my latest and most interesting

articles by following me.

If this guide has been helpful to you and your team please share it with others!

DEV Community



Build Apps with Google AI Studio

This track will guide you through Google AI Studio's new "Build apps with Gemini" feature, where you can turn a simple text prompt into a fully functional, deployed web application in minutes.

Read more →

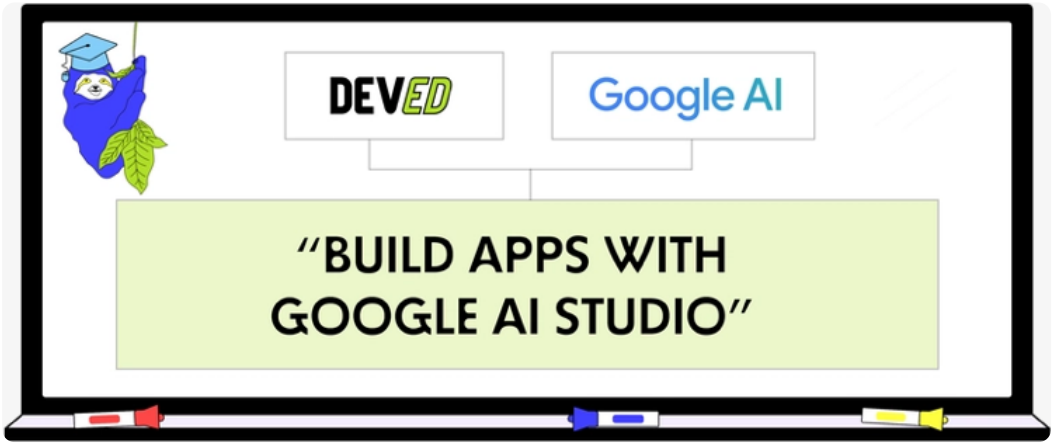
Top comments (2)

Sirajul Muneer • 22 Sep 23

Well summarized!

Raí B. Toffoletto • 20 Sep 23

Where is SSE!?



Build Apps with Google AI Studio 🥾

This track will guide you through Google AI Studio's new "Build apps with Gemini" feature, where you can turn a simple text prompt into a fully functional, deployed web application in minutes.

Read more →