SIT 378 about API

Outline

* API Protocol
* API Framework
* URL link list

1 API protocol

a brief description of

### RESTful API Protocols(REST) Representational State Transfer

### Bubble chart Description automatically generated

Introduced in 2000[, REST](https://www.ibm.com/topics/rest-apis) is one of the most popular forms of the API. REST sends data in multiple formats: JSON, PYTHON, HTML, or media. However, it can only be sent over the HTTP/HTTPS protocol. REST is more flexible than SOAP (outlined below), but is limited by the send type.

Components of a REST call:

* HTTP methods (POST, GET, PUT, DELETE): The action that the API performs.
* Endpoint (URL): The location where the data is sent.
* Header: Data about the data, such as the originating server.
* Body: Information that is actually held in the transmission.

Websites can communicate with databases using [REST APIs](https://getstream.io/docs_rest/). You can obtain information from the Website, post information to the Website, replace old information with new information, or delete old information.

For example, whenever a user submits a contact form on a Web site, the Web site typically POSTS the form data to the server. The server then sends that information to another process. Another process records information in a database or sends an email.

* **SOAP (Simple Object Access Protocol)**

**Diagram

Description automatically generated**

Introduced in 1998[, SOAP](https://stoplight.io/api-types/soap-api) is one of the simplest forms of the API. SOAP typically sends XML files over HTTP/HTTPS, but also uses SMTP, TCP, and Also sent over UDP. SOAP submission is simple and broad, making it easy to deploy but difficult to debug.

SOAP provides an overview of a simple transmission mechanism. It is packaged into an XML file (a standard file format). It is easy to create an XML file to complete the SOAP submission, but because SOAP is so simple, errors are high.

SOAP Components:

* Envelope: The beginning/end of the message.
* Headers: Information about the message, such as authentication requirements.
* Body: The actual message.
* Fault: Error message.

The SOAP system is frequently used for e-mail. A Web site can send an e-mail embedded in a SOAP envelope to someone. The header describes the source/sender of the information, the body is the email message, and the fault code is related to the failure of the email (for example, the email bounced).

However, this highlights one of the shortcomings of SOAP systems. It's simple, so you can also take advantage of it. In emails, headers can be "spoofed" to appear to come from a completely different recipient. These API protocols do not have built-in authentication. This must be handled by the sending and receiving devices.

### Remote Procedure Calls(RPC)

[RPC may](https://www.techtarget.com/searchapparchitecture/definition/Remote-Procedure-Call-RPC) be the oldest type of API and was used in various forms in the 70s and 80s . There are actually multiple types of RPC APIs, such as JSON-RPC and gRPC implementations. Anyway, RPC focuses on calling a process or action. Basically, the app tells another app to complete the process internally.

RPC is much faster than REST, but the details depend on the implementation. Unlike REST and SOAP, messages come in a variety of formats. RPC is tailored for client-server architectures and is typically used over the network.

RPC System Components:

* Client: The requesting device.
* Client stub: How the client packages/unpackages its materials.
* RPC runtime: A messaging system (client-server courier).
* Server stub: How the server packages/unpackages that material.
* Server: Feeding device.

E-commerce platforms can use RPC solutions to send information to inventory systems. The e-commerce platform aggregates sales information and sends it to the inventory system via the RPC runtime. The inventory system unpackages the information and performs an internal data-logging process.

The inventory system then bundles the response (including information such as updated inventory totals) with the ecommerce platform. The e-commerce platform then unpackages that information and uses it to update the listing.

Note, however, that RPC calls are likely to be network-based. For example, you can call your company's security system from your personal office phone. The security system then verifies that the corporate phone is sufficiently protected to connect to the organization's internal network. If the phone is sufficiently protected, it is authenticated and passed to the network.

* **XML-RPC**

XML stands for "Extensible Markup Language" and RPC stands for "Remote Procedure Call." [XML-RPC](https://www.tutorialspoint.com/xml-rpc/xml_rpc_intro.htm) uses XML to encode API calls and forward them using HTTP. The use of the XML format is useful because it is designed to be readable by both humans and machines, so you have minimal code knowledge and want to learn more. WordPress is one of the companies that use XML-RPC on their platform.

* **JSON-RPC**

It is a protocol similar to XML-RPC, except that it uses JSON instead of XML. JSON stands for "JavaScript Object Notation" and is similar in that it is relatively easy for humans to read. It provides maximum browser compatibility and is easier to implement and use. Unlike REST and SOAP, both XML-RPC and JSON-RPC can start processes on the server. This means that you can run scripts, start applications, initiate data transfers, and perform other actions on the server. Naturally, this raises security concerns. For this reason, RPC-style APIs are most often used in internal systems where there is no concern for external threats.

### gRPC

### gRPC was created by Google and made generally available in 2015. It is a modern, robust open-source RPC framework. Cross-platform and contract-based. Communicate using HTTP/2 and Transport Layer Security (TLS). Here, data is transferred using a protocol buffer consisting of three components. First, start defining a service or contract using a ".proto" file. This is followed by function code and language-specific libraries. Today, gRPC is used by many well-known organizations outside of Google. It has also proven to be lighter and faster compared to REST. This requires a proxy because gRPC makes complex use of HTTP/2, so it is impossible to implement a gRPC client in a browser .

### GraphQL APIs

### Diagram Description automatically generated

If you're looking for a fast API solution, consider GraphQL. This state-of-the-art approach leverages specialized query definition capabilities to eliminate the need for excessive queries from clients. Therefore, it provides much-needed flexibility during data aggregation.

Unlike traditional REST APIs, developers can easily take advantage of an interface that allows for on-demand "data queries." Essentially, with [the GraphQL API](https://konghq.com/learning-center/api-gateway/graphql), users request everything they need from a backend service in a single API endpoint. This avoids unnecessary API calls. The GraphQL API has the potential to minimize data requests and make applications more efficient as a result by enabling accurate data retrieval. In addition, developers can quickly incorporate new fields and types into their applications, creating a quick and streamlined development process.

### Webhook APIs

The functionality of the webhook API revolves primarily around facilitating real-time transfer of data and seamless integration between various web applications. An example use case is to send a push notification to alert the user about incoming email. As soon as certain preset events occur, these APIs automatically trigger messages (or "hooks") that initiate communication between different software programs. The webhook API allows applications to stay in sync and respond to events instantly. These APIs are widely used for event-based programming that allows software to exchange real-time data. It serves various purposes, such as notifying users after clicking a link or updating your online store with new items without manual intervention.

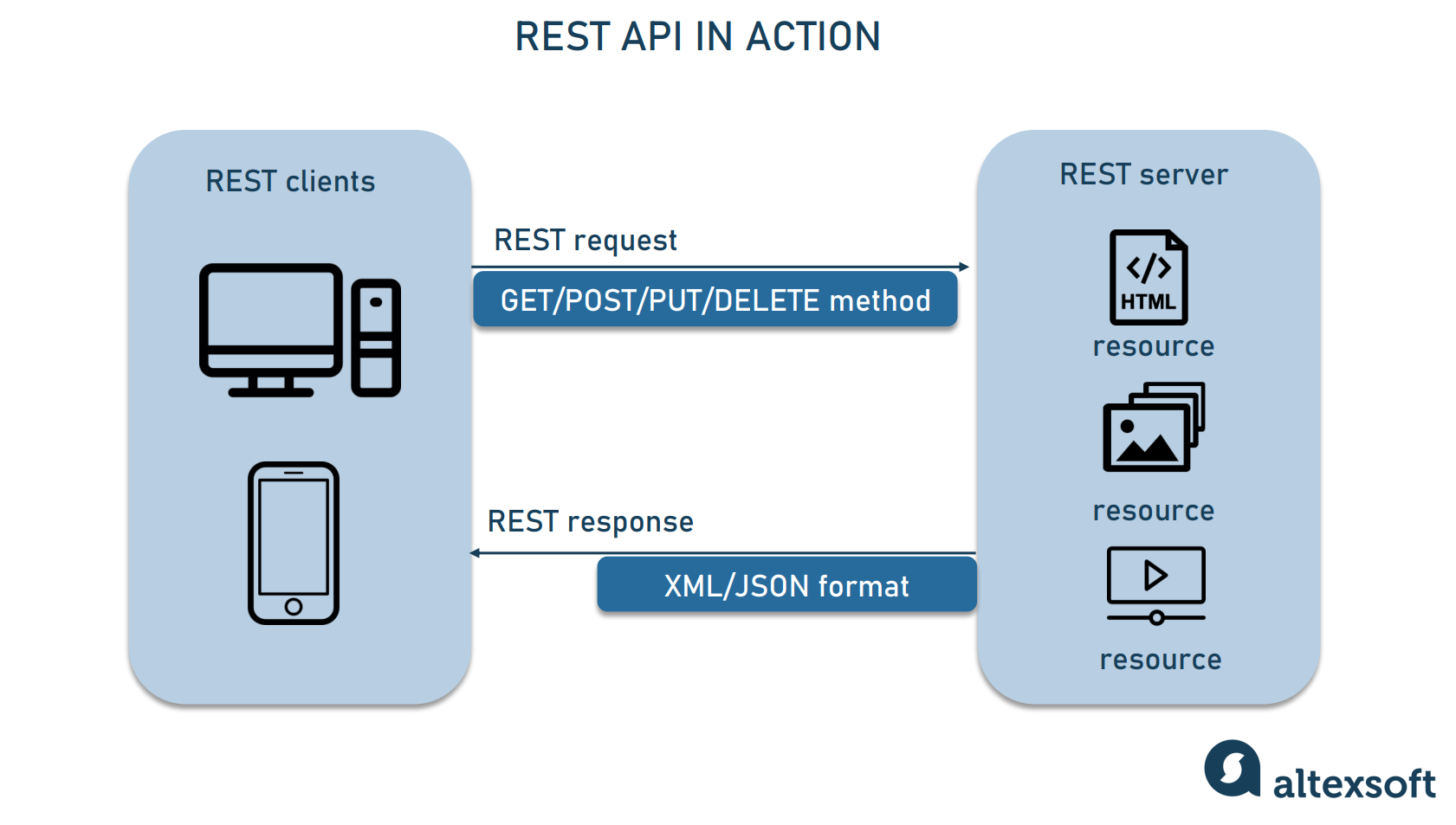
Detailed merits and demerits

Here's not everything but going into more detail

* **RESTful API (REST; Representation State Transfer)**

## **What is the REST API and how does it work**?

**REST stands** for **Representational State Transfer** and is an architectural style for building Web services that interact over the HTTP protocol. The principle was formulated by computer scientist Roy Fielding in 2000 and became popular as a scalable and flexible alternative to the old methods of machine-to-machine communication. This is still the gold standard for public APIs .

[](https://content.altexsoft.com/media/2021/03/rest_api_works.png)

REST clients can interact with each resource by sending HTTP requests.

### REST API concepts

The key elements of the REST API paradigm are:

* A client or software that runs on the user's computer or smartphone and initiates communication**.**
* A server that provides an API as a means of accessing data or functionality. and
* Any content that the server can provide to clients, such as videos or text files.

To access a resource, the client sends **an HTTP request**. Instead, the server generates an HTTP response that contains the encoded data for the resource**.** Both types of REST messages are self-descriptive. That is, it contains information about how to interpret and process the message.

### REST API methods and request structure

A REST request contains four important parts: HTTP method, endpoint, header, and body.

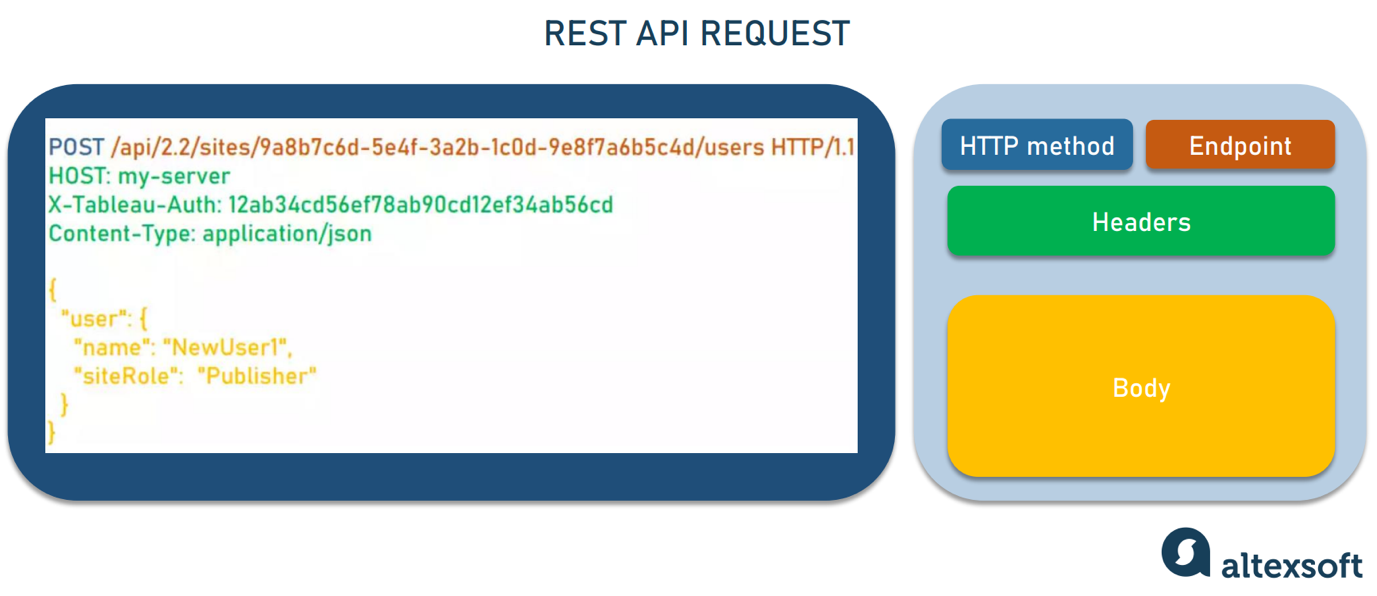
**HTTP methods** describe what a resource does. There are four basic methods, also known as CRUD operations.

* POST to create a resource,
* Get the resource with GET,
* Update the resource with PUT,
* DELETE deletes the resource.

**The end point** contains a Uniform Resource Identifier (URI) that indicates where to find the resource on the Internet and how to find it. The most common type of URI is a Unique Resource Location (URL) that serves as a full web address.

**The header** contains information related to both the client and the server. Primarily, headers provide authentication data such as the API key, the name or IP address of the computer where the server is installed, and information about the response format.

**The body is** used to convey additional information to the server. For example, it may be part of the data you want to add or replace.

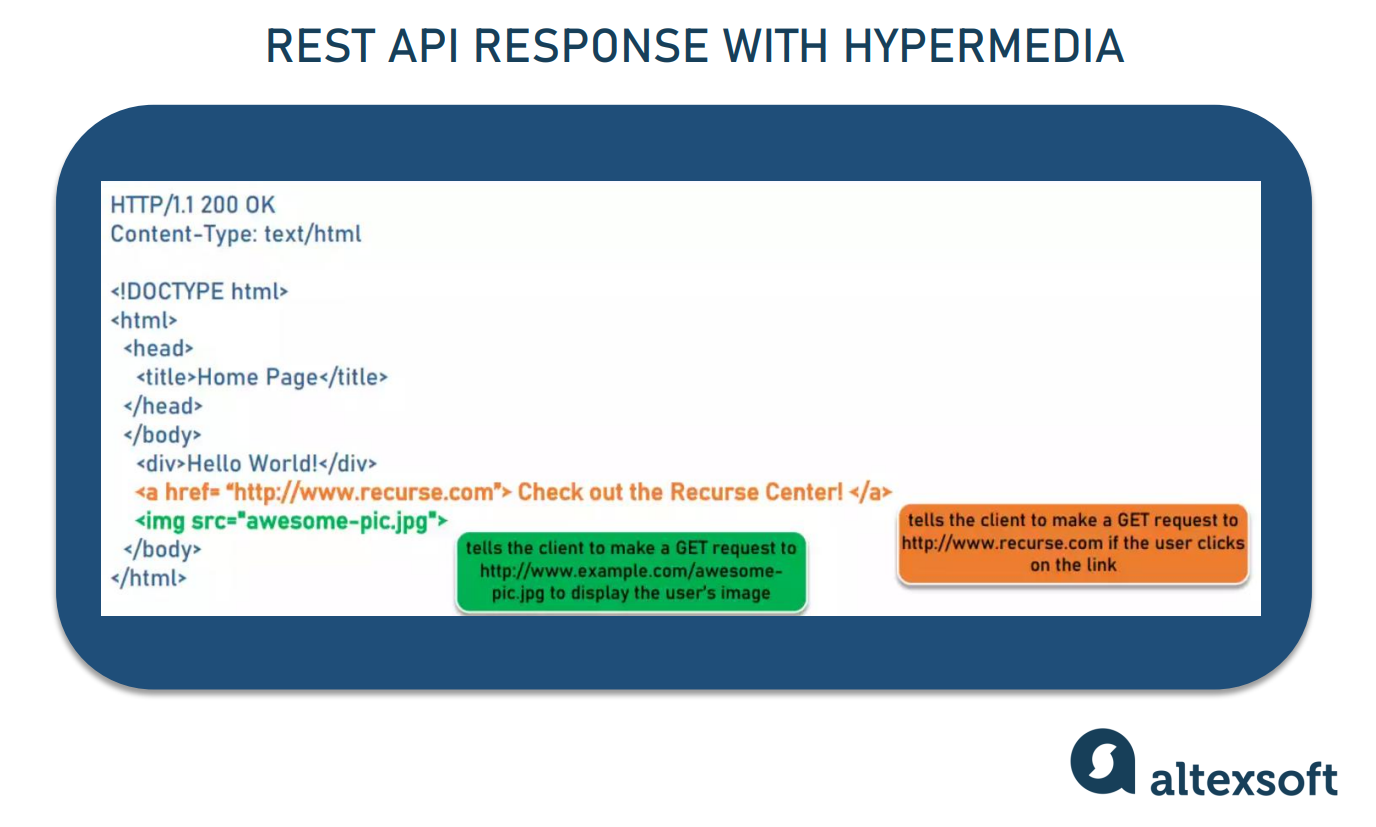
[](https://content.altexsoft.com/media/2021/03/rest_request.png)

A REST request to create a new user that returns the ID of the resource for which the response was created. Source: [Tableau API](https://help.tableau.com/current/api/rest_api/en-us/REST/rest_api_concepts_example_requests.htm)

### REST response structure

In response, the server sends a **representation** of the resource being asked (a machine-readable description of its current state) rather than the resource itself. The same resource can be represented in different formats, but the most common are XML and [JSON](https://www.json.org/json-en.html).

**Whenever relevant, the server includes hyperlinks or hypermedia in the response that link to other related resources**. In this way, the server gives instructions about what the client can do next and what further requests it can make.

[](https://content.altexsoft.com/media/2021/03/rest_response.png)

Example of a self-describing server response using hypermedia. Source: [Lauren Long](https://codewords.recurse.com/issues/five/what-restful-actually-means)

## **REST Best Practices**: **What to Make** **Your API RESTful**

REST is not tied to any particular technology or platform. It also doesn't dictate exactly how to build an API. Instead, best practices called constraints are introduced. They describe how the server processes requests and responds to them. Operating within these constraints, the system acquires the desired characteristics.

### Client-server autonomy

**Characteristics obtained:** changeability, increased system reliability

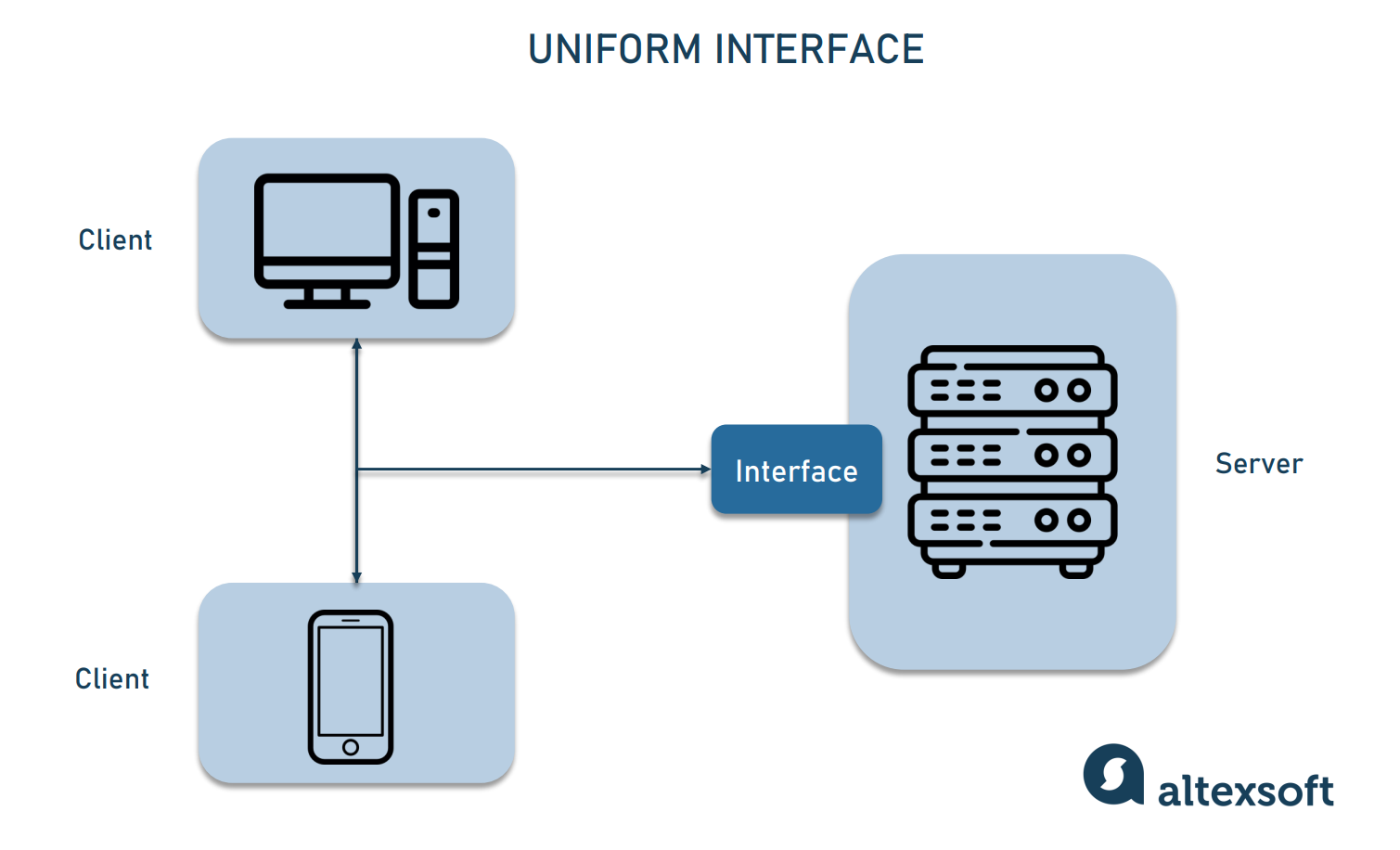
In a REST API system, the client and server operate independently using different technical stacks. The client doesn't need to know anything about the business logic, but the server knows nothing about the user interface. Separation of responsibilities means that API providers and API consumers can change, and their communication is not counterproductive.

### Unified Interface

**Characteristics obtained:** ease of use, shared understanding

Unified Interface is an important attribute that distinguishes REST APIs from non-REST APIs. Regardless of the client app or device that runs it, it specifies a standardized way to communicate with a particular server. We've already covered some of the basics that support this practice.

* A unique identifier (URI) assigned to each resource,
* self-descriptive messages explaining how to interpret them and what to do next,
* The ability to manipulate resources through their representation in JSON or XML, and
* Hypermedia links to related resources.

[](https://content.altexsoft.com/media/2021/03/uniform_interface.png)

The server uses the same interface, regardless of the client.

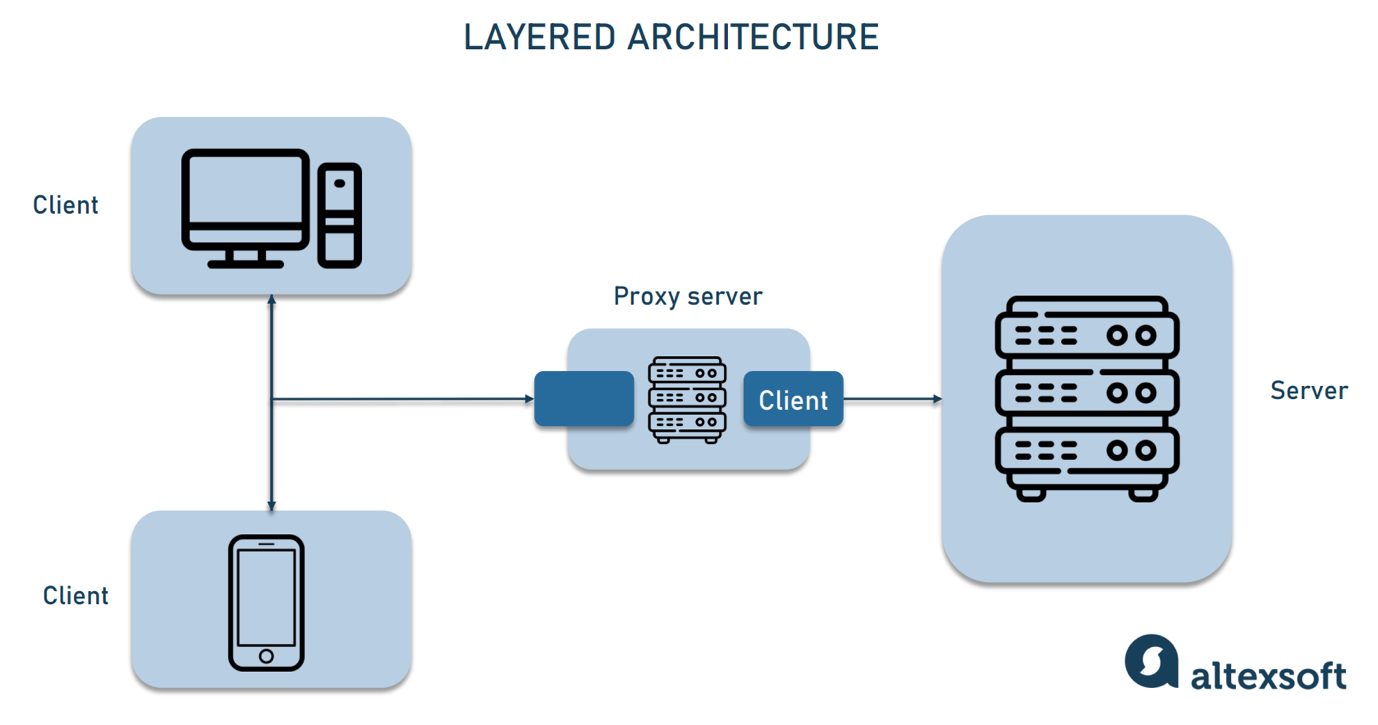
A unified interface makes it easy for developers to understand the logic of the API. [Todd Main](https://envysion.com/), [Envysion's director of software development,](https://www.linkedin.com/in/todd-main-a956a51/) acknowledges the relief that partners feel when they choose the REST approach. Or supply." Todd adds that implementing code using RESTful APIs is also easy. "The passed object is directly translated into a data structure in the programming language."

### Layered architecture

**Characteristics obtained:** Increased system scalability and security

RESTful systems have a layer structure in which each layer operates independently and only interacts with directly connected layers. When calling the server, the client does not know if there is an intermediary on the way.

The layered architecture allows you to place a proxy or load balancer between the client and server to improve scalability. Adding security as a separate layer makes your system more secure. These services are involved in generating the response, but the client doesn't have to worry about what's behind the interface.

**[](https://content.altexsoft.com/media/2021/03/layered-architecture.png)**

The client interacts with the API layer, which reaches the server through a proxy.

### caching

**Characteristics:** Low server latency, increased app speed and responsiveness

REST APIs allow clients to store frequently accessed data on their side rather than requesting it over and over again. As a result, there are fewer app calls, which reduces server load and its latency. As a result, the application becomes more responsive and reliable.

### Stateless interaction

**Characteristics obtained:** better performance, reliability of the app

The term stateless indicates that the API does not store information related to previous sessions and handles each request individually. All data about the current client state is contained in the request body.

Because the REST API is stateless, you don't need to deal with server-side state synchronization logic. Another advantage of session independence is that any server can process requests. This improves application performance and reduces the risk of going down.

"Being stateless means fewer side effects," argues Pál Váradi  [Nagy, developer at Hanna Instruments](https://www.linkedin.com/in/p%C3%A1l-v%C3%A1radi-nagy-07966aa0/?originalSubdomain=ro). "In FTP, for example, you have a session in progress using a command to change the state of the session. This state can and may be lost. Therefore, it was decided to make REST as pure as possible. This means relying on a PURE function that always returns the same output when given the same input, and doesn't affect anything else."

### Code on Demand (CoD)

**Characteristics obtained:** customization of functions, extensions

Instead of sending back a JSON representation, the server might return a piece of executable code at the request of the client. CoD practices give clients more control over functionality and enable extensibility.

## **REST API Principles: Prioritizing** **Business** Needs

**Obtained properties:** flexibility

Still, REST is about flexibility. Implementing a REST architecture allows developers to deviate, extend, or only partially cover the standard set of constraints. Let's consider the basic constraints of stateless interaction. You can ignore this and keep your application stateful for [sessions that need to be stored on the server side](https://stackoverflow.com/questions/3105296/if-rest-applications-are-supposed-to-be-stateless-how-do-you-manage-sessions).

As a result, you'll hear that there aren't actually any REST APIs that actually follow the results of Fielding.

"For me, some constraints (like client/server architecture and statelessness) are great, but they're pretty standard application design, and others are like plague that I want to avoid!" Garry Taylor, [senior software developer and technical consultant,](https://www.linkedin.com/in/garrytaylormelbourne/) said: In particular, Garry says that code-on-demand is a bad idea.

## **REST API examples**

REST API concepts and principles may seem abstract until you actually try them. [Below are examples of real-world APIs to help you understand the RESTful approach and understand](https://www.altexsoft.com/blog/api-documentation/) how to write API documentation.

### Trello API

Widely used [project management tools provide a simple](https://www.altexsoft.com/blog/project-management-tools-trello-basecamp-jira/)  API that allows you to quickly understand REST resources and the HTTP methods that apply to them.

[The first thing to do in introducing](https://developer.atlassian.com/cloud/trello/guides/rest-api/api-introduction/) [Trello's API](https://developer.atlassian.com/cloud/trello/guides/rest-api/api-introduction/)  is to send a GET request to the most basic resource: the board.



Use cURL to get the Board message. This is a client-side program that makes an HTTP request to the specified URL.

This will give you a better understanding of how to manipulate other basic resources such as lists, cards, and actions using the methods applicable to them. For example, there are two types of POST requests available for cards:

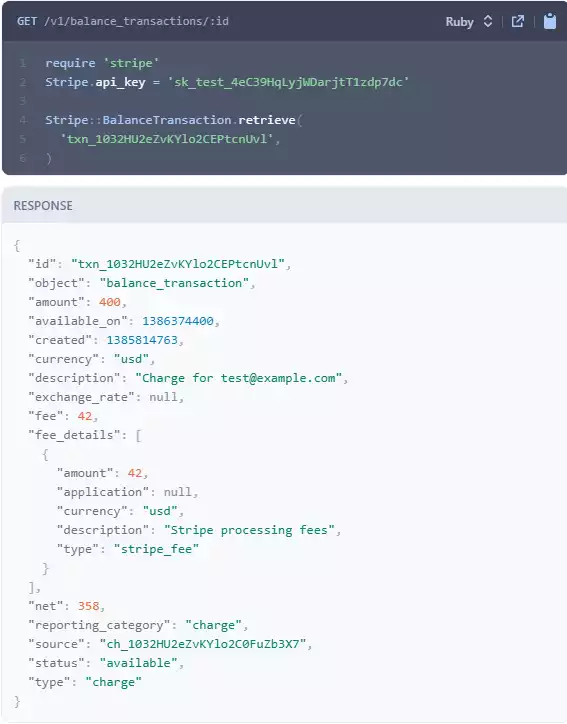
POST /1/cards/[card id or shortlink]/actions/comments means "add a comment to a card"

POST /1/cards/[card id or shortlink]/actions/idMembers means "add a member to a card"

The full list of resources includes 18 objects that can be accessed via the API. Each comes with a detailed description that includes all request types and example query parameters.

### Stripe API

[One of the most popular](https://www.altexsoft.com/blog/business/how-to-choose-and-integrate-payment-gateway-online-payments-transaction-processing-and-payment-gateways-providers/) online payment solutions boasts probably the best API documentation [you can find on the internet.](https://stripe.com/docs/api) Stripe has a dedicated team that creates a complete guide with code snippets and API request and response examples for all resources. "Our philosophy is to create documentation for how you use APIs, not how you build them," says Stripe's Payments and Platform [Cristina Cordova, former head of the partnership,](https://www.linkedin.com/in/cristinajcordova/) [explains](https://getputpost.co/a-look-inside-stripes-api-platform-92ba19ca9751).



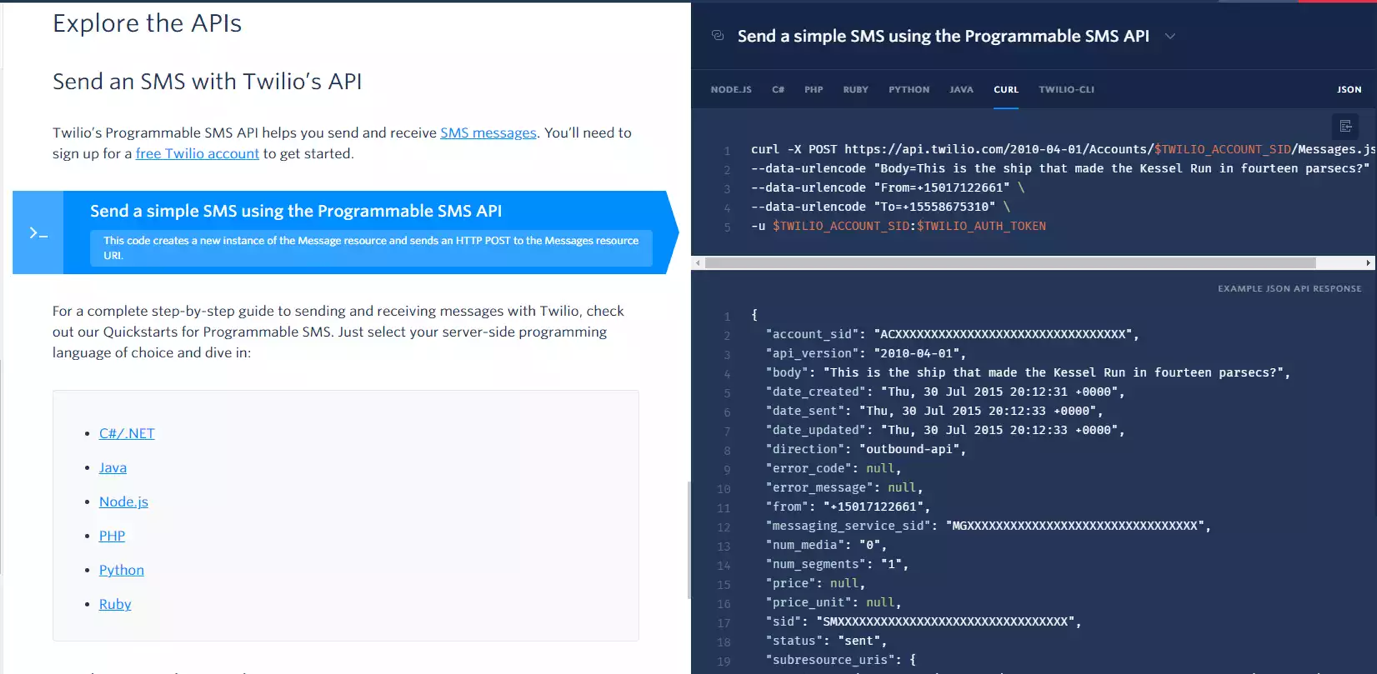
Stripe Rest API requests and responses for balance transactions.

First,there is a step-by-step development quick start guide. Engineers who like to learn by example can use [Stripe Samples to](https://github.com/stripe-samples) run ready-to-work code from the reels.

### Twilio API

Twilio [provides](https://www.twilio.com/) APIs for integrating voice and video calls with SMS, MMS, and other messages into web and mobile apps It is a driven platform. To enable developers at all levels to create communication tools using [Twilio, the company devised comprehensive REST API best practices](https://www.twilio.com/docs/usage/rest-api-best-practices). Also, beginners should read the short explanation "What is [a REST API anyway?" before starting.](https://www.twilio.com/docs/usage/api)

Twilio offers a free trial account to try and test API integrations [.](https://www.twilio.com/try-twilio?_ga=2.128058510.322708128.1614681008-1111485117.1613468016) For added convenience, step-by-step guides are supported in code snippets.



Use example API requests and JSON API responses to explain in text how to send an SMS.

### Jira REST API

[Jira is](https://www.atlassian.com/software/jira) one of the most popular tools among [project managers](https://www.altexsoft.com/blog/business/product-managers-vs-project-managers-difference-responsibilities-and-kpis/) [used by more than 65,000](https://www.atlassian.com/software/jira)  teams worldwide. Its [REST API allows](https://developer.atlassian.com/cloud/jira/platform/rest/v3/intro/) enterprises to programmatically interact with equipment, integrate its functionality into enterprise software and other apps, build add-ons, and automate interactions with Jira. There are comprehensive guidelines on the resources available and how to access them through the [API](https://developer.atlassian.com/cloud/jira/software/rest/intro/).

[](https://content.altexsoft.com/media/2021/03/jira-api.png)

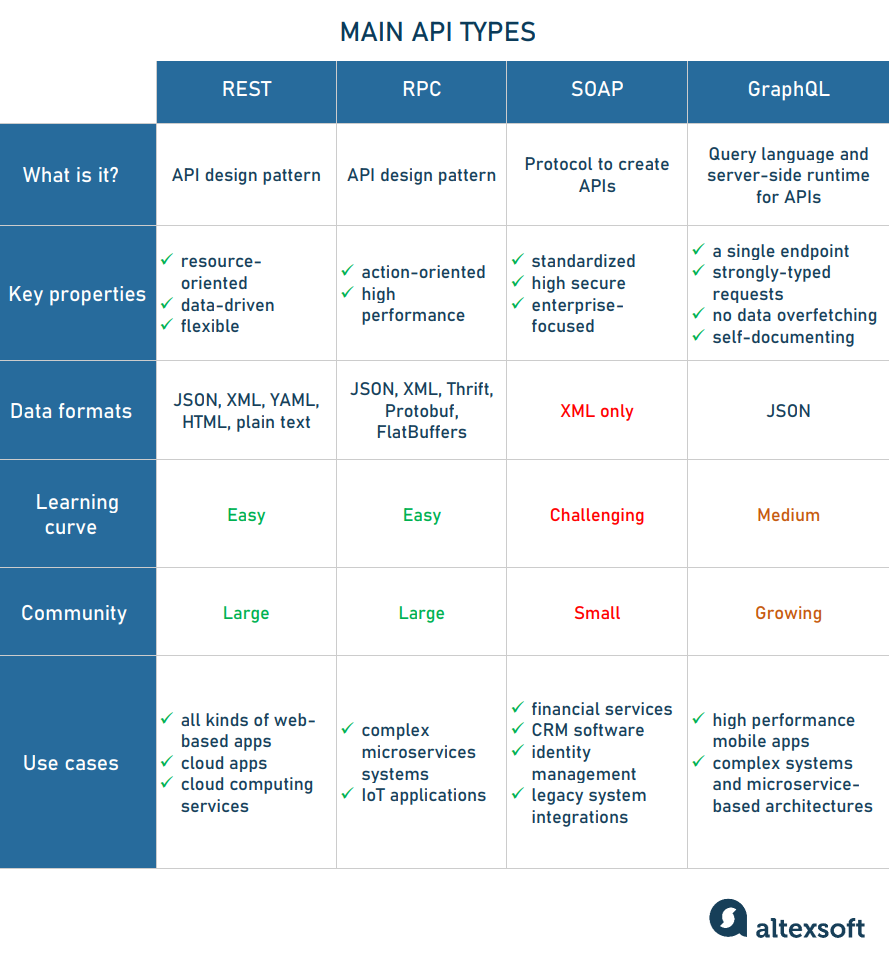
Jira REST API response to create a new Scrum board.

### Power BI REST API

According to Gartner, Microsoft Power BI has been an industry leader in [analytics and business](https://www.altexsoft.com/blog/bi-developer-role-responsibilities-skills/) intelligence for five consecutive years. This self-service platform is the top choice for data analysts, [BI developers](https://www.altexsoft.com/blog/bi-developer-role-responsibilities-skills/), and decision makers across all industries. The suite of [Power BI REST APIs enables](https://learn.microsoft.com/en-us/rest/api/power-bi/) any type of client to add interactive [data visualizations,](https://www.altexsoft.com/blog/data-visualization-tools-types-techniques/) dashboards, and reports directly to existing applications.

## **REST** compared to other **API paradigms**

There is room for debate about direct comparisons of approaches to building APIs. [That's why REST for command-oriented remote procedure calls (RPC), standardized SOAP](https://www.altexsoft.com/blog/engineering/graphql-core-features-architecture-pros-and-cons/), and schema-based GraphQL We decided to check out the key features that make it stand out.

[](https://content.altexsoft.com/media/2021/03/api_types.png)

A comparison of the four main API paradigms.

### REST and RPC

RPC has been around for a long time and can be considered the core of REST. Pál Váradi Nagy describes REST as "a limited subset of what is already being done globally — remote Procedure Call".

The procedural part of RPC executes a function on input and returns output. This makes RPC easy to use over the network and provides high performance. Therefore, for large microservices systems, facilitating short and clear internal communication is the preferred option. [RPC](https://www.altexsoft.com/blog/iot-architecture-layers-components/) is also suitable for IoT applications, especially low-power applications.

The latest RPC version is [gRPC](https://grpc.io/). Because it uses binary data instead of text, communication is [more compact and efficient than REST.](https://www.yonego.com/nl/why-milliseconds-matter/#gref) gRPC is also type-safe. That is, it sends only the expected data types.

However, gRPC requires you to set up a client. This means that gRPC-generated code must be embedded in the client process. It's cumbersome for dynamic languages (JavaScript, Python, etc.) where the build process might not exist. REST does not require it. You can also make API calls by simply entering the URL in your browser. This is  [very useful for API testing](https://www.altexsoft.com/blog/api-testing/) because the basic functionality of the API can only be tested in the browser.

In addition, REST provides better abstraction than RPC. Observe RESTful constraints to separate client and server as much as possible. RESTful connections do not depend on existing state, but RPC has no such requirement. If REST says, "Let's do this and then forget about each other," RPC might say, "Let's do this and then do that." This depends on the output of the previous action.

### REST and SOAP

According to Cloud Elements'  [2017 State of API Integration](https://cloud.google.com/blog/products/api-management/the-state-of-apis-report-2017) report, 83% of APIs use REST, compared to [SOAP 15% of APIs use it.](https://www.altexsoft.com/blog/engineering/what-is-soap-formats-protocols-message-structure-and-how-soap-is-different-from-rest/) This proves that SOAP is not dead yet.

Rob James, who has been [in software development since the '80s, points out that while SOAP has some drawbacks, it has some important advantages](https://www.quora.com/profile/Rob-James-179). Web Services Description Language ( WSDL for short, where SOAP API logic is described) provides more information than a typical JSON object provides."

Integrated with the WS-Security [protocol, the SOAP API](https://www.altexsoft.com/glossary/soap-apis/) sends messages with a high level of privacy and integrity. This is where financial services, payment gateways ([PayPal public APIs](https://developer.paypal.com/docs/nvp-soap-api/PayPalSOAPAPIArchitecture/)), CRM software, identity management, and why it remains the best option for communication services.

Still, Rob James admits that his goto is REST. This is because SOAP is not easily modifiable and nearly impossible to circumvent. and vendor-specific tags. 」

SOAP was primarily based on the first version of RPC, XML. Therefore, the biggest advantage of REST over XML-bound SOAP is its support for multiple forms.

### REST and GraphQL

To get information about a request, the REST API client must combine and match multiple endpoints. This snowballs into another problem — data overfetching. This means that the response contains unnecessary information. This can slow down the processing of requests.

[GraphQL emerged in](https://www.altexsoft.com/blog/engineering/graphql-core-features-architecture-pros-and-cons/)  2015 with a new philosophy of endpoint customization. The GraphQL API begins by defining a schema that describes how data is structured on the server. This schema allows clients to understand how to create a single query to get an accurate response.

Mobile devices are unreliable networks. Therefore, if your RESTful API needs to make multiple requests, it is much more likely to fail. Therefore, efficient queries in GraphQL are very suitable for mobile APIs.

## **RESTful or** **RESTish, that's the question**

The basic idea and semantics of interaction between machines have been around for a very long time. But with the advent of REST, it brought order to web APIs.

"REST services became important because they were an attempt to standardize the interface," says Todd Main. He emphasizes that neither simple old RPC calls nor SOAP had such a structure. Instead, REST brings a more standard way to the table to browse the system programmatically, or at least interact with it without having to refer to the manual at every step."

In RPC, a URL indicates an action. This is because the primary purpose of the URL is to process the request. The idea behind REST is to organize interactions between independent systems.

REST means a lot more than using HTTP. "You don't even have to use HTTP to implement the REST architecture, but HTTP makes it easier," says the man with [over 30 years of programming experience. Claude Wilbur](https://www.quora.com/profile/Claude-Wilbur-1) says.

But if developers can't understand the whole concept, you might end up with a system that's a bit more than an RPC with HTTP verbs and pretty URLs. There is no cacheability, wacky rules, or zero links to discover the next available action (hypermedia). People who notice the difference sarcastically refer to these APIs as RESTish.

REST, on the other hand, is not a stereotyped specification, unlike SOAP. Its implementation, which is somewhat standardized, can be objectively considered RESTful. So, to be safe, developers can describe their APIs as conforming to the REST architecture rather than RESTful.

Top of Form

Bottom of Form

* **SOAP (Simple Object Access Protocol)**

## Advantages of SOAP

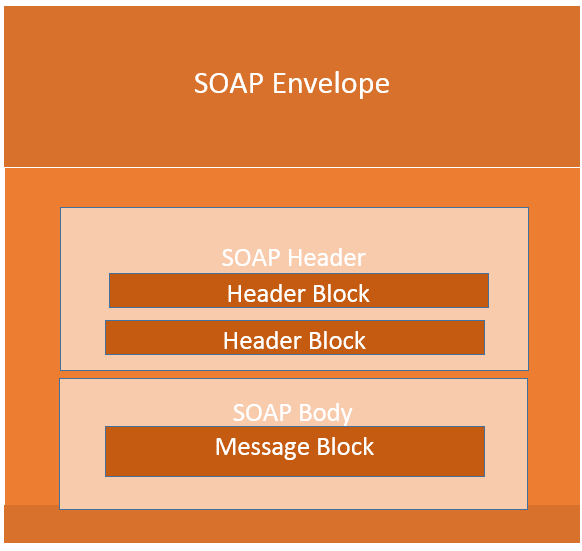
SOAP is a protocol used to exchange data between applications. The following are some of the reasons why SOAP is used:

* When you develop SOAP-based Web services, you need a language that the Web service can use to interact with client applications. SOAP is the perfect medium developed to achieve this goal. This protocol is also recommended by the W3C Consortium, the governing body for all Web standards.
* SOAP is a lightweight protocol used to exchange data between applications. Pay attention to the keyword "light". SOAP as a protocol also falls into the same category because SOAP programming is based on the XML language, which is itself a lightweight data interchange language.
* SOAP is designed to be platform independent and not to operate system. Therefore, the SOAP protocol can run programming language-based applications on both Windows and [Linux](https://www.guru99.com/unix-linux-tutorial.html) platforms.
* Works with the HTTP protocol. SOAP works with the HTTP protocol, which is the default protocol used by all web applications. Hence SOAP
* There are no customizations required to run Web services built on the protocol to work on the World Wide Web .

## SOAP Building Block

The SOAP specification defines what is called a  **"SOAP**  message" that is sent to a Web service and client application**.**

The following SOAP architecture diagram shows the different components of a SOAP message.

Building Blocks for SOAP Messages

A SOAP message is simply an XML document with the following components:

* Envelope element that identifies the XML document as a SOAP message – This is the containing part of the SOAP message and is used to encapsulate all the details of the SOAP message. This is the root element of the SOAP message.
* Header element with header information – The header element can contain information such as authentication credentials that can be used by the calling application. It can also contain definitions of complex types that can be used in SOAP messages. By default, SOAP messages can contain parameters of simple types, such as strings and numbers, but they can also be complex object types.

The following is an example of a simple SOAP service of a complex type:

Assuming you want to send a structured data type with a combination of "tutorial name" and "tutorial description", define a complex type as shown below.

Complex types are defined by the element tag <xsd:complexType> All required elements of the structure and their respective data types are defined in the complex type collection.

<xsd:complexType>

<xsd: Sequence >

<xsd:Element Name="Tutorial Name" Type="String"/>

<xsd:element name="tutorial description" type="string"/>

</xsd: Sequence >

</xsd:complexType>

* Body element that contains call and response information – This element contains the actual data that needs to be sent between the Web service and the calling application. The following is an example of a SOAP Web service with a SOAP body that actually works with the complex types defined in the header section. The following is the tutorial name and tutorial description response sent to the calling application that calls this Web service.

< soap: main body>

<GetTutorialInfo>

<TutorialName>Web Services</TutorialName>

<TutorialDescription> All about Web services</TutorialDescription>

</GetTutorialInfo>

</soap: Main unit >

## Structuring SOAP messages

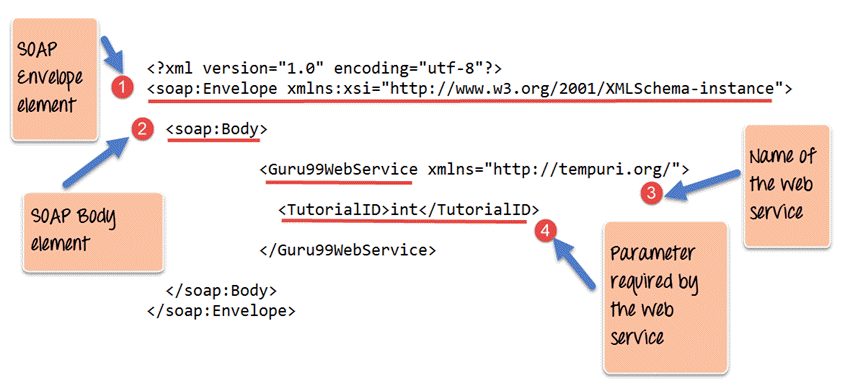
One thing to note is that SOAP messages are typically auto-generated by the Web service when it is invoked.

Each time the client application calls a method of the Web service, the Web service receives SOAP containing the necessary details of the data sent from the Web service to the client application. Generate messages automatically.

As discussed in the previous topic of this SOAP tutorial, a simple SOAP message has the following elements:

* Envelope Essentials
* Header element and
* Body Essentials
* Fault element (optional)

Let's take a look at the following simple SOAP message example to see what the element actually does.

Structuring SOAP messages

1. As you can see from the SOAP message above, the first part of the SOAP message is the envelope element used to encapsulate the entire SOAP message.
2. The next element is a SOAP body that contains the details of the actual message.
3. The message contains a web service named Guru99WebService.
4. "Guru99Webservice" accepts a parameter of type "int" and has the name TutorialID.

The above SOAP message is now passed between the Web service and the client application.

You can see how useful the above information is for client applications. The SOAP message informs the client application of the name of the Web service, the expected parameters, and the type of each parameter that the Web service retrieves.

## SOAP envelope

The first part of the building block is the SOAP Envelope .

The SOAP envelope is used to encapsulate all the necessary details of the SOAP message exchanged between the Web service and the client application.

SOAP envelope elements are used to indicate the beginning and end of a SOAP message. This allows the client application that calls the Web service to know when the SOAP message ends.

For SOAP envelope elements, you can note the following:

* All SOAP messages must have a root envelope element. It is absolutely mandatory to include an envelope element in a SOAP message.
* Every Envelope element must have at least one SOAP body element.
* If the Envelope element contains a header element, it must contain only one and appear as the first child of the Envelope before the body element.
* When the SOAP version changes, the envelope changes.
* A v1.1-compliant SOAP processor generates an error when it receives a message that contains a v1.2 envelope namespace.
* A v1.2-compliant SOAP processor generates a version mismatch error when it receives a message that does not contain a v1.2 envelope namespace.

The following is an example of the SOAP API in version 1.2 of the SOAP envelope element.

<?xml version="1.0"?>

<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://www.w3.org/2001/12/soap-envelope" SOAP-ENV:encodingStyle=" http://www.w3.org/2001/ 12/ soap-encoding">

< soap: main body>

<Guru99WebService xmlns="http://tempuri.org/">

<TutorialID>int</TutorialID>

</Guru99WebService>

</soap: Main unit >

</SOAP-ENV: Envelope >

**Fault messages**

When a request is made to a SOAP Web service, the response returned can be in one of two forms: a success response or an error response. When success is generated, the response from the server is always a SOAP message. However, if a SOAP fault is generated, it is returned as an "HTTP 500" error.

A SOAP Fault message consists of the following elements:

1. **<faultCode>** – This is the code that specifies the code of the error. The fault code can be one of the following values:
   1. SOAP-ENV:VersionMismatch – This is when an invalid namespace for a SOAP envelope element is encountered.
   2. SOAP-ENV:MustUnderstand – The immediate child element of a Header element with the mustUnderstand attribute set to "1" was not recognized.
   3. SOAP-ENV:Client – The message was malformed or contained incorrect information.
   4. SOAP-ENV:Server – The message could not be continued because there was a problem with the server.
2. **<faultString>** – This is a text message that provides a detailed description of the error.
3. **<faultActor> (optional)** – This is a text string that indicates the cause of the failure.
4. **<detail> (optional)** – This is an element of an application-specific error message. Therefore, your application might display specific error messages for different business logic scenarios.

**Example failure message**

The following is an example of a fault message: Scenarios where a client tries to use a method called TutorialID in class GetTutorial generates an error.

If the method does not exist in the class in which it is defined, the following failure message is generated:

<?xml version='1.0' encoding='UTF-8'?>

<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi=" http://www.w3.org/1999/XMLSchema-instance" xmlns :xsd="http://www.w3.org/1999/XMLSchema">

<SOAP-ENV:Main unit >

<SOAP-ENV: Fault >

<faultcode xsi:type="xsd:string">SOAP-ENV:Client</faultcode>

<FaultString XSI:Type="XSD:String">

Could not find method (GetTutorialid) in class (GetTutorial)

</fault string>

</SOAP-ENV: Fault >

</SOAP-ENV: Main unit >

</SOAP-ENV: Envelope >

**Output:**

When I run the above code, I get an error similar to "The method (GetTutorialID) in the class (GetTutorial) could not be found".

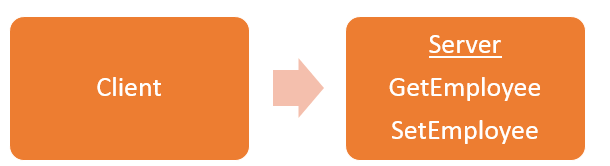
## SOAP Communication Model

All communication with SOAP occurs over the HTTP protocol. Before SOAP, many Web services used the standard RPC (remote procedure call) style for communication. This was the simplest type of communication, but with a number of limitations.

In this SOAP API tutorial, let's examine the following diagram to see how this communication works. This example assumes that the server is hosting a Web service that provides two methods.

* **GetEmployee** – This retrieves all employee details
* **SetEmployee** – This sets the values of the employee's department, salary, and other details accordingly.

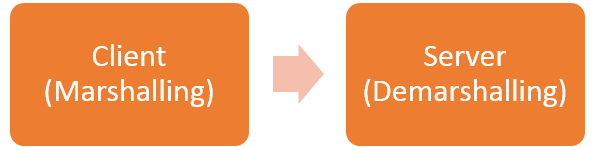
In normal RPC-style communication, the client simply calls a method in the request, sends the required parameters to the server, and the server sends the necessary response.



the above communication model has the following significant limitations

1. **Language independent** – The server that hosts the method is a specific programming language, and calls to the server are usually made only in that programming language.
2. **Non-standard protocols** – When a call is made to a remote procedure, the call is not made over the standard protocol. This was a problem because almost all communication over the web had to take place over the HTTP protocol.
3. **Firewall** – RPC calls do not go through normal protocols, so you must open a different port on the server to allow the client to communicate with the server. Typically, all firewalls block this type of traffic, and a lot of configuration was usually required to ensure that this type of communication between the client and server worked.

To overcome all of the above limitations, SOAP uses the following communication model:



1. The client formats the information about the procedure call and arguments into a SOAP message and sends it to the server as part of the HTTP request. This process of encapsulating data in a SOAP message **was known as marshaling.**
2. The server then unwraps the message sent by the client, verifies what the client requested, and sends the appropriate response back to the client as a SOAP message. Unwrapping a request sent by a client is **known as demarshalling.**

## SOAP example in action

In this SoapUI tutorial, let's look at a real-world SOAP example.

Perhaps one of the best ways to see how SOAP messages are generated is to actually see the Web service in action.

This topic describes how to build ASMX Web services using the Microsoft.Net framework. This type of Web service supports both SOAP version 1.1 and version 1.2.

ASMX Web services automatically generate Web Services Definition Language (WSDL) documents. This WSDL document is required so that the calling client application knows what it can do with the Web service.

This example creates a simple Web service. This is used to return a string to the application that calls the Web service.

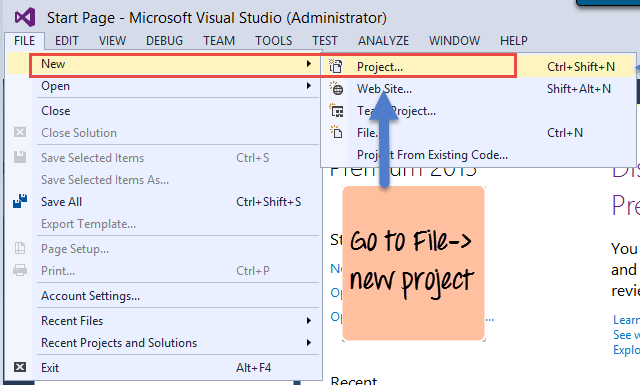
[This Web service is](https://www.guru99.com/asp-net-tutorial.html)  hosted in a Asp.Net Web application. Next, call the Web service to see the results returned by the Web service.

Visual Studio also displays the SOAP message that is passed between the Web service and the calling application.

The first prerequisite for setting up a Web service application can be done by following these steps:

For this example, make sure Visual Studio 2013 is installed on your system.

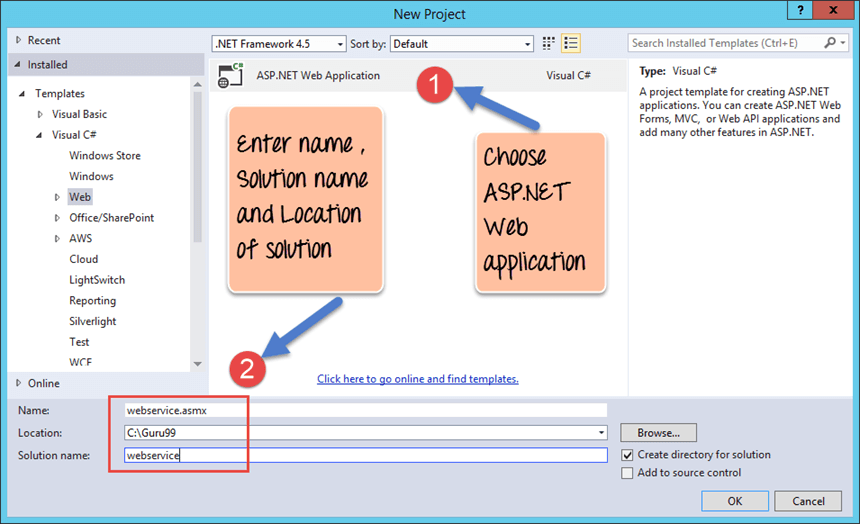
**Step 1)** The first step is to create an empty ASP.Net web application. From Visual Studio 2013, click the menu option File > New Project.



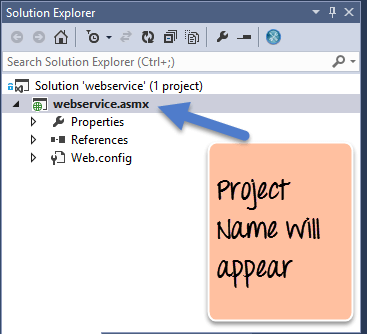
New Project When you click an option, Visual Studio displays another dialog box that allows you to select the project type and provide the required details for the project. This will be discussed in the next step.

**Step 2)** In this step,

1. First ASP.NET select [the C#](https://www.guru99.com/c-sharp-tutorial.html) web template for your web application. To create a SOAP service project, the project must be of this type. When you select this option, Visual Studio performs the necessary steps to add the files that are required by your Web-based application.
2. In this case, name the project specified as webservice.asmx. Next, specify the location where the project file will be saved.



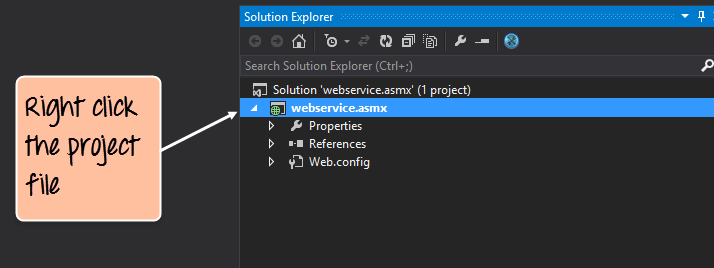
When complete, you should see the project file that was created in Solution Explorer in Visual Studio 2013 .



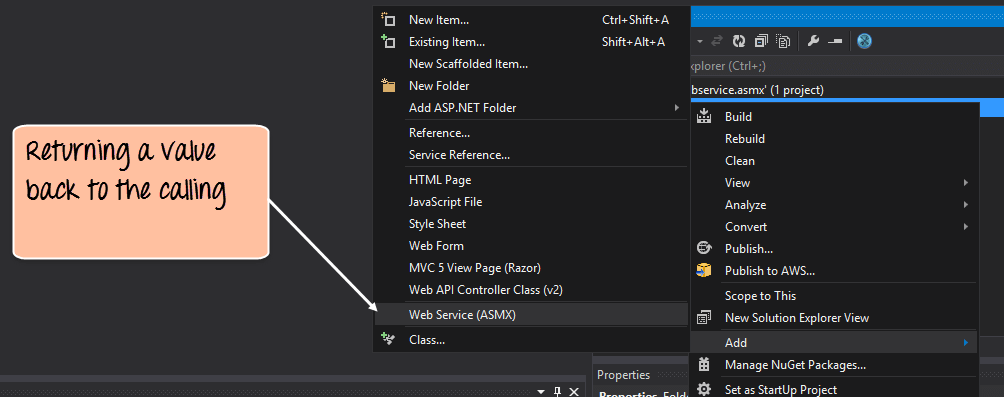
**Step 3)** In this step,

Add a Web service file to the project .

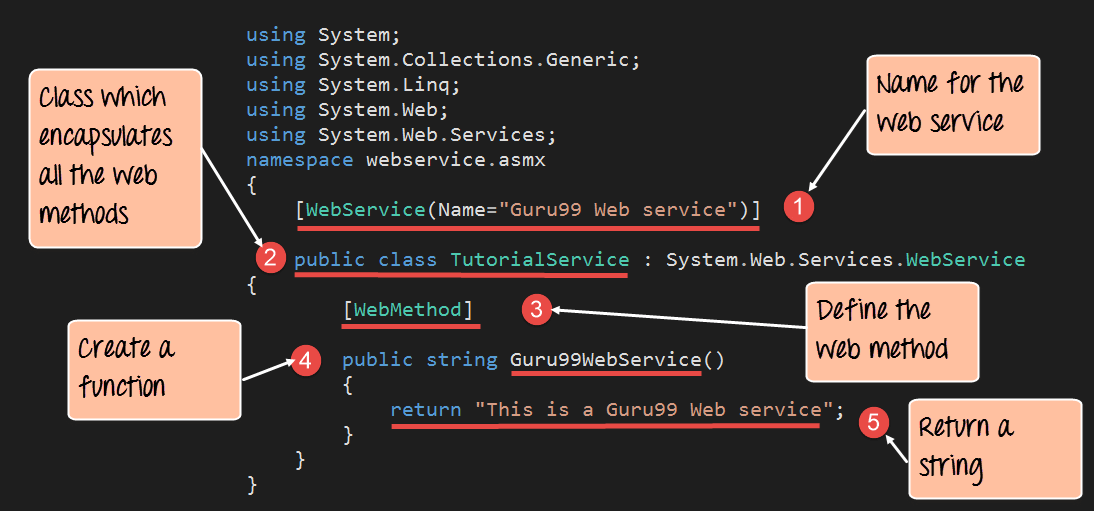
1. First, right-click the project file as shown below.



1. You can add a Web service file by right-clicking the project file and selecting option Add -> Web Service (ASMX ). Simply specify the name of the Tutorial Service in the Web Service Name file.



**Step 4)** Add the following code to the tutorial service asmx file:

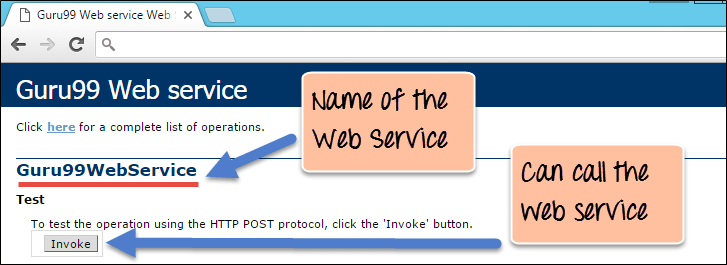


**Code description:**

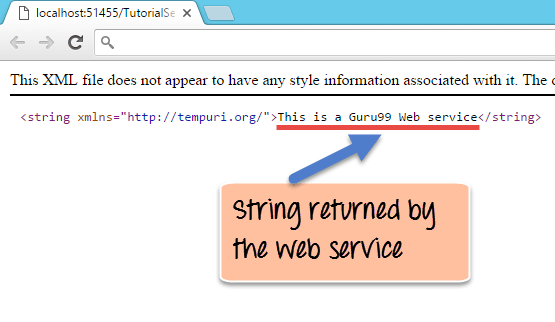
1. This line of code provides the name of the Web service file. This is an important step because it provides a way for client applications to call the Web service through the name of the Web service.
2. Class files are typically used to encapsulate the functionality of a Web service. Therefore, the class file contains definitions for all Web methods that provide some functionality to the client application.
3. Here, [WebMethod] is known as the attribute that describes the function. The following procedure creates a function called Guru99WebService, but you can add a [WebMethod] attribute so that it can be called by a client application. If this attribute is not set, the method cannot be called from the client application.
4. Here we define a function called "Guru99WebService" that is used to return a string to the calling client application. This function is a Web service that can be called from any client application.
5. Use the return statement to return the string "This is a Guru99 Web service" to the client application.

If the code runs successfully, you should see the following output when you run the code in a browser:

**Output:**



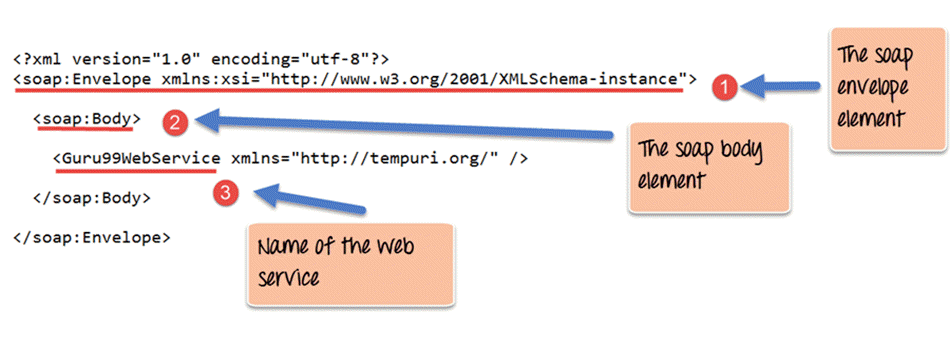
* The output clearly shows that the web service is named Guru99 Web Service. This is the result of naming the Web service.
* You can also see that you can call the Web service. When you click the Invoke button, your web browser displays the following response:



The above output,

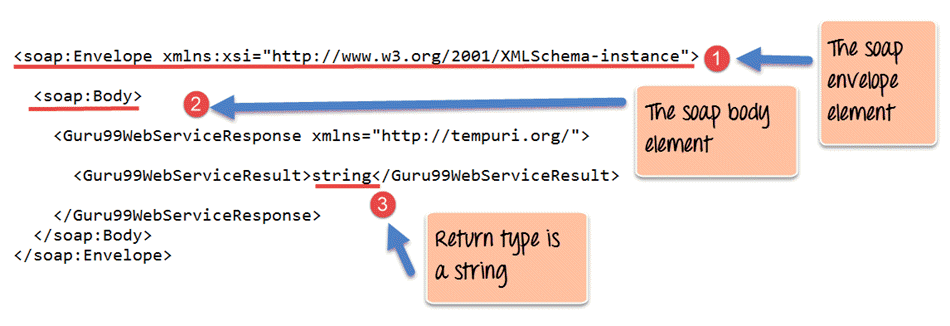
* It clearly shows that calling the Web method returns the string "This is a Guru99 Web service".
* Visual Studio can also view the request and response of the SOAP message that is generated when the above Web service is invoked.

The following SOAP request is generated when the web service is invoked:



**Code description:**

1. The first part of the SOAP message is the envelope element described in the previous chapter. This is an encapsulating element that is present in all SOAP messages.
2. The SOAP body is the next element and contains the actual details of the SOAP message.
3. The third part is an element that specifies that the service called "Guru99WebService" should be called.



<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

< soap: main body>

<Guru99WebServiceResponse xmlns="http://tempuri.org/">

<Guru99WebServiceResult> string </Guru99WebServiceResult>

</Guru99WebServiceResponse>

</soap: Main unit >

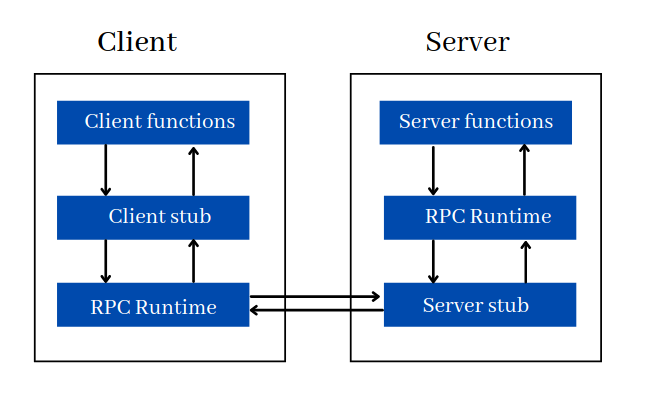
</soap: Envelope >

**Code description:**

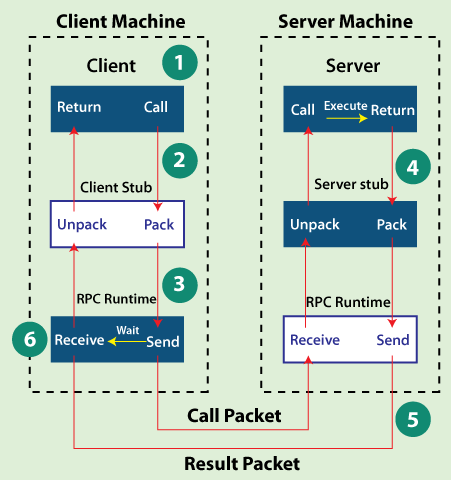
1. The first part of the SOAP message is the envelope element described in the previous chapter. This is an encapsulating element that is present in all SOAP messages.
2. The SOAP body is the next element and contains the actual details of the SOAP message.
3. The 'string' attribute is interesting here. This notifies the client application that the called Web service returns an object of type string. This is very useful when the client application does not know what the Web service returns.

### Remote Procedure Calls(RPC)

**How** **RPC works**

A picture containing table

Description automatically generated



1. The client invokes a stub that has the same signature as the function in the server.
2. **A stub, called marshaling**, is a process that encodes function arguments in messages that are passed over the network to the server.
3. The server decodes the message and calls the function with the specified arguments. This process is called **unmarshalling**.
4. This function returns a value, which is returned to the client as a response.

For Web services, RPC-style APIs focus on actions. Each endpoint represents an action that a client can perform on the server.

An action is a function that is called on the server. Like a normal function that takes parameters and returns a value that is sent as a response to the client.

For RPC APIs that use HTTP as the transport protocol, the method or function is placed in the URL and the arguments are placed in the query string or body.

Example RPC call:

GET /getMovie/12 HTTP/1.1

Host: api.moviedb.com

Content-Type: application/json

Python

/\* Function definition \*/

def getMovie(id) {

// ...

}

/\* Function call \*/

getMovie(id)

Paison

Kopi

The RPC API supports only GET and POST requests.

**RPC API examples**

Let's take a look at the RPC-based Slack Web API.

**In the user API**

1. To get all users on a specific Slack team:

Method: GET

Endpoint: https://slack.com/api/users.list

1. To get information about a user

Method: GET

ENDPOINT: https://slack.com/api/users.info

body: {"token": xxxx-xxxxxxxxx-xxxx, user: "W1234567890"}

3. **Use the Chat API**  To send a message to a specific channel

Method: POST

ENDPOINT: https://slack.com/api/chat.postMessage

1. Delete a message from a chat

Method: DELETE

ENDPOINT: https://slack.com/api/chat.delete

**5. Use the** Conversations API   
 Get all channels for a Slack team.

Method: GET

ENDPOINT: https://slack.com/api/conversations.list

RPC Highlights

The basic characteristics of remote procedure calls are as follows:

* The called procedure is in a different process and may reside on a different machine.
* Processes do not share address space.
* Parameters are passed only by value.
* RPC runs within the environment of the server process.
* It does not provide access to the environment of the calling procedure.

RPC Advantages

The following are some of the benefits or benefits of RPC:

* RPC methods help clients communicate with servers through the traditional use of procedure calls in high-level languages.
* RPC methods are modeled after local procedure calls, but procedures are more likely to run in a separate process and typically on a separate computer.
* RPC supports a process and thread-oriented model.
* RPC hides its internal message-passing mechanism from the user.
* The effort required to rewrite and redevelop the code is minimal.
* Remote procedure calls can be used for distribution and local environments.
* Commit much of the protocol layer to improve performance.
* RPC provides an abstraction. For example, the message-passing nature of network communications remains invisible to the user.
* RPC enables applications to be used in distributed environments as well as local.
* Using RPC code minimizes rewrite and redevelopment efforts.
* The process-oriented and thread-oriented models supported by RPC.

RPC missing

The following are some of the drawbacks or drawbacks of using RPC:

* Remote procedure calls pass parameters as values only and pointer values, which is not allowed.
* Remote procedures can have much shorter call (and return) times (that is, overhead) than local procedures.
* This mechanism is highly vulnerable to failure because it involves a communication system, another machine, and another process.
* RPC concepts can be implemented in a variety of ways, but this is not a standard.
* Because RPC is mostly interaction-based, the hardware architecture of RPC is inflexible.
* Because of remote procedure calls, the cost of the process increases.

### GraphQL APIs

## **GraphQL Advantages**

GraphQL has many advantages over REST APIs.  [One of the main benefits is that the client can tell you exactly what it needs from the server](https://stablekernel.com/mobile-app-development-services-atlanta/backend-development/) and receive that data in a predictable way.

For example, consider the following schema:

type Query {

me: User

}

type User {

id: ID

name: String

city: String

state: String

friends: [User]

}

copy

Now, let's say we need to get the name of **User**. **If** you're using the User REST API, you typically receive the entire object in the response. This can cause [overfetching](https://stackoverflow.com/questions/44564905/what-is-over-fetching-or-under-fetching) and cause performance issues. When you use GraphQL, you receive only the data that you explicitly request.

The name query looks like User:

{

me {

name

}

}

copy

And our response looks like this:

{

"me": {

"name": "John Doe"

}

}

copy

Another big advantage is that you can get more resources in a single request. Continuing with the schema example, let's say you want to get the name of your User friend as well. **/friends?id=X** For traditional REST APIs, you would probably make an additional request to the endpoint and send an additional request to the User ID of must be passed.

Simply add the friends property to your query to receive all the names in a single request.

{

me {

name

friends {

name

}

}

}

copy

And our response looks like this:

{

"me": {

"name": "John Doe"

"friends": [

{

"name": "Jane Doe"

},

{

"name": "Jim Doe"

}

]

}

}

copy

Another benefit is that they are strongly typed, so API consumers know exactly what data is available and in what format. According to the [documentation,](https://graphql.org/learn/schema/) "Every GraphQL service defines a set of types that fully describe the set of possible data that can be queried by that service, and then, when the query arrives, it is validated and executed against that schema."

## **GraphQL highlights**

GraphQL has many advantages over traditional REST APIs , but it also has some important drawbacks.

One of the 200 drawbacks is that the query always returns an HTTP status code, regardless of whether the query succeeds or not. If the query fails, the response JSON includes the top-level **errors** key with the associated error message and stack trace. This makes error handling much more difficult and can add complexity to monitoring, etc.

Another drawback is the lack of built-in caching support. Because the REST API has multiple endpoints, you can take advantage of the native HTTP cache to avoid reacquiring resources. GraphQL requires you to set up your own caching support. This means you either need to rely on another library or set up a [globally unique ID in the backend](https://graphql.org/learn/caching/), etc.

This leads to **the** final drawback: complexity. If you're using a simple REST API and want to work with data that is relatively consistent over time, you might want to use the REST API . For companies that have the engineering resources to work with rapidly changing data and focus on redesigning their API platform, GraphQL can solve many of the pain points experienced with REST APIs.

GraphQL provides an interesting solution to common hurdles faced when using REST APIs. There are some drawbacks to using GraphQL, but if you're dealing with rapidly changing data at scale, it can be a great solution for your business. For more information, see [the documentation](https://graphql.org/learn). And if you're looking for a more serious GraphQL solution, [Apollo makes](https://www.apollographql.com/) it easy to get started on both the client and server sides.

2 About the framework

How to choose an API framework

Typically, the first factor in choosing an API framework is deciding which language to work in. Since you're reading this post, chances are you've already decided to build a Python project. Next, you need to sort the available framework choices. At this point, verify that you need the type of functionality that your API requires and that the framework you choose can support these needs. Some frameworks have plugins and dependencies that allow for easy integration with other platforms, some more accurately support use cases, while others automatically disqualify with limited functionality required. It's important to ensure that your use cases and features are supported by the framework.

Last but not least, you should also consider the learning curve and available materials and documentation. As a Python developer, the availability of great documentation and examples is a big factor in how quickly you and your team can scale up your API. Before deciding on a framework, consult the documentation and do a quick search so that you can find examples that can guide and guide you through the work required to build an API with your chosen framework. There are a few factors to consider, so let's take a look at some of the options for common Python frameworks.

pyramid

**Pyramid** is a full-stack Python web framework that lets you start building with just Pyramid. As you grow, Pyramid will grow with you, scaling as needed, adding new features, and handling different aspects of web application development. The Pyramid framework is a versatile and constantly evolving Python web development framework. It includes many of the standard features of the most popular frameworks, including MVC patterns, routing, templates, security, sessions, and databases. There is also a production web server called preserve.

Pyramid starts with a minimal installation that can be expanded as needed. It is worth noting that it is part of the Pylons project that integrates web-related technologies.   
These are some of the most important features of Pyramid:

* Provides synchronous request processing,
* The context of a view can be defined by a function as well as a class.
* There is no specific ORM, but SQLAlchemy is recommended.
* Does not enforce coding style or project architecture – [TIMTOWTDI](https://docs.pylonsproject.org/projects/pyramid/en/1.10-branch/designdefense.html?highlight=MVC),
* It provides excellent documentation with tutorials and examples.
* No specific HTML templating engine is provided, but [Chameleon](https://docs.pylonsproject.org/projects/pyramid-chameleon/en/latest/) is recommended.
* An interesting custom routing system allows you to match multiple views to a single URL.
* WSGI compliant.
* Extensive static file support – file serving, static file URL routing,
* It can be extended with external modules such as [Cornice](https://cornice.readthedocs.io/en/latest/) for REST API and [aiopyramid with asynchronous support](https://github.com/housleyjk/aiopyramid).

This framework is a good choice if you don't want to spend time learning custom framework solutions (such as ORMs) but need extensive tools to build software. That's because Pyramid supports the use of many well-known standalone solutions.   
It has great scaling capabilities and advertises itself as a framework that allows you to "start small and finish big."

**Pros:**

* Using Ajax Requests
* Flexible and easy to configure
* Suitable for SQL-based projects

**Cons:**

* Easy to set up configuration to add templates
* Lack of documentation
* No built-in database management

Web2py

**Web2py is** a Python-based framework that serves as a backend for database-driven web applications. Web2py has many useful features, such as HTTP handling and URL management. In addition, there is a built-in scheduler for background tasks that run separately from the Web server thread. Finally, web2py has a database abstraction layer where Python objects are mapped to database objects such as tables, queries, and records. The database abstraction layer interacts with one or more drivers to support a variety of databases, such as SqLite and Oracle.

**Web2py is** a Python-based web framework for database-driven web applications. The core is all about HTTP and URL handling, but there is also a scheduler for running background tasks and a Python object in the database It also has a database abstraction layer for mapping to objects. It also comes with MVC support, forms validation, security, and access control.

**Pros:**

* Extensive documentation
* Adaptable to your team's needs
* Suitable for large projects
* Built-in management interface: Web2Py includes a built-in management interface that makes it easy to manage your database and perform other administrative tasks.
* Automatic Database Migration: Web2Py includes a database migration system that allows you to easily change your database schema without writing SQL code.
* Ease of use: Web2Py is designed to be easy to use. This means that the learning curve is relatively shallow. If you are new to web development or are new to Python, Web2Py may be a good choice.

**Cons:**

* Not supported in Python 3
* Inflexible plug-in system
* Unable to load data model
* Scalability limitations: Web2Py is not as scalable as other frameworks. So if you're building a high-traffic website, it may not be the best choice.
* Less popular: Web2Py is not as popular as other frameworks, so there are fewer resources available for learning and troubleshooting.
* Web2py supports doctests, but not unit tests. Due to its limited scope, doctest is not the best choice.
* There is no difference between production and development mode. If an exception occurs, a ticket is always generated and you should go to the ticket to check for errors. This may be useful in the case of production servers, but it is difficult in a development environment because developers need to check for errors immediately rather than checking the ticket number.
* Web2py has a good database abstraction layer (DAL) that can abstract away many types of database engines, but it lacks a powerful ORM. If you're dealing with a relatively large model, all nested definitions and attributes clutter and complicate your code.
* web2py has very poor IDE support, so you can't use the standard python development tools without modification.

Django and the Web2py framework are full-stack frameworks. This means that you can provide all the code you need, from form generators to template layout and form validation, and write them according to your specific needs.

However, with non-stack frameworks like Flask and Pyramid, if you want to create a full-featured website, you have to add a lot of code and extra bits yourself. This requires a lot of skills and time.

Fast API

**FastAPI** is a new Python API framework created by Sebastián Ramírez and based on Python 3.6 asynchronous features. Quickly gaining popularity in Silicon Valley, we ship Starlettle and Pydantic to make development fast, simple, and scalable. FastAPI has customers all over the world, including Uber, Netflix, and Microsoft.

**Pros:**

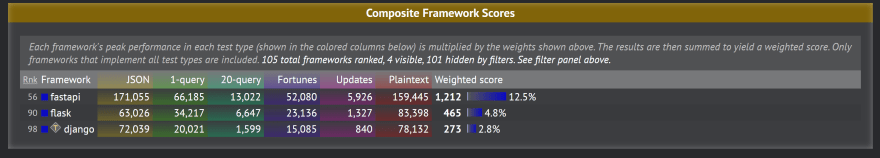
* Cording quickly
* High Performance
* Easy creation of plug-ins
* **Excellent performance**   
  [FastAPI](https://fastapi.tiangolo.com/#performance/) [surpasses Flask in terms of performance](https://fastapi.tiangolo.com/#performance/) and is one of the fastest Python web frameworks. Only [Starlette](https://www.starlette.io/) and [Uvicorn](https://www.uvicorn.org/) are fast. ASGI enables FastAPI to support concurrent and asynchronous code by declaring endpoints.
* Built-in concurrency For concurrent programming, Python 3.4 introduced Async I/O. FastAPI simplifies concurrency by eliminating the need for event loops or async/wait management. The initial path function can be specified by the developer as a coroutine using async def and can wait for a specific location.
* **Dependency injection support**   
  FastAPI supports a simple and easy-to-use dependency injection solution. This approach ensures that different classes do not depend directly on each other. It makes it easier to change the code, and it can be useful. This technique improves code modularity and system scalability by achieving inversion of control. Path manipulation functions in FastAPI allow developers to declare relevant dependencies.
* **Built-in documentation Documentation generated by**   
  FastAPI is useful. Documentation helps developers explain their software to others, simplifies backend usage by front-end engineers, and simplifies testing API endpoints.
* **Validation built-in**   
  built-in data validation allows developers to skip proofs and write more compact code. Detects an incorrect data type and returns the underlying reason in JSON. FastAPI uses [Pydantic](https://pydantic-docs.helpmanual.io/) modules to simplify validation and speed up typing. According to the creators of FastAPI, developer errors are reduced by 40%.

**Cons:**

* Community Support
* We use Pydamtic to validate requests , which can be difficult to understand and you may need to create your own custom validator.
* **Insufficient security**   
  FastAPI is not secure. Instead, fastapi.security handles security. At the same time, OAuth 2.0 is supported.
* FastAPI, a small group of developers , is eight years younger than Flask. Therefore, its community and educational materials are still modest. When searching, you can hardly find any books, guidelines, or lessons. Due to its growing popularity, this may change in the future.

### 1. Exactly ****Fast**** API

Compared to other major Python frameworks such as FastAPIFlask and Django, it's reasonably fast. The following [Techempower](https://www.techempower.com/benchmarks/) score chart shows the performance differences between frameworks.

[](https://res.cloudinary.com/practicaldev/image/fetch/s--oW5khIvq--/c_limit%2Cf_auto%2Cfl_progressive%2Cq_auto%2Cw_880/https:/dev-to-uploads.s3.amazonaws.com/i/7yxcdbt72bwrj0cwm89y.PNG)

I also did a small test myself to see which framework was the fastest, and the results were actually very interesting. In this test, we set up a basic "Hello world" API for all three frameworks. We tested the response time by calling the API and took the average response time. The results can be divided into two scenarios:

**The.**  Average time of first call after server startup   
**b.**  Average duration of consecutive calls from the first call

DjangoFastAPI responds slower than usual on the first API call. Flask is always consistent, but during all API calls, it is much slower than the other two.   
 Here's the average time it takes for all three APIs:

| **framework** | **Case A** | **Case B** |
| --- | --- | --- |
| FastAPI | 17 ms | 6.2 ms |
| Django | 517.2 ms | 5.834 ms |
| flask | 507.2 ms | 508.9 ms |

One interesting thing to note here is that after the first call, Django actually runs a bit faster than FastAPI. However, in certain scenarios, such as serverless environments, Django's initial invocation and startup time can be an issue. Note that the measurements were made in certain environments with little data. My experience with Flask and Django is very limited, so results may vary.

### 2. Asynchronous code support

The most exciting feature of FastAPI is that it supports out-of-the-box asynchronous code using the Python keyword async/await. Here's an example API that retrieves data from Reddit asynchronously. (See example Python  [async/await tutorial](https://stackabuse.com/python-async-await-tutorial/) [by](https://stackabuse.com/python-async-await-tutorial/) [Scott Robinson](https://stackabuse.com/python-async-await-tutorial/).)

app = FastAPI()

async def get\_json(client: ClientSession, url: str) -> bytes:

async with client. get(url) as response:

assert response. status == 200

return await response. read()

async def get\_reddit\_top(subreddit: str, client: ClientSession, data: dict):

data1 = await get\_json(client, 'https://www.reddit.com/r/' + subreddit + '/top.json?sort=top&t=day&limit=5')

j = json. loads(data1. decode('utf-8'))

subreddit\_data = []

for i in j['data']['children']:

score = i['data']['score']

title = i['data']['title']

link = i['data']['url']

print(str(score) + ': ' + title + ' (' + link + ')')

subreddit\_data. append(str(score) + ': ' + title + ' (' + link + ')')

data[subreddit] = subreddit\_data

print('DONE:', subreddit + '\n')

@app. get("/")

async def get\_reddit\_data\_api() -> dict:

start\_time: float = time. time()

client: ClientSession = aiohttp. ClientSession()

data: dict = {}

await asyncio. gather(

get\_reddit\_top('python', client, data),

get\_reddit\_top('programming', client, data),

get\_reddit\_top('compsci', client, data),

)

await client. close()

print("Got reddit data in ---" + str(time. time() - start\_time) + "seconds ---")

return data

The magic of asynchronous code is that coroutines run get\_reddit\_top at the same time, so the execution time of the API is significantly reduced compared to the execution time of running serially.

### 3. Very short development time

To create a basic "Hello world" API, the framework requires the following number of lines of code (considering the entire project):

| **framework** | **Lines of code** |
| --- | --- |
| FastAPI | 8 lines |
| flask | 7 lines |

I don't consider Django because I think it's structurally different than the other two.   
 Scaling a FastApi app is similar to Flask. Both have the concept of modularity through Flask's Blueprint and FastAPI's router. In other words, the development times for Flask and FastAPI are very similar.

### 4. Simple test

[Testing the FastAPI endpoint is very simple and](https://fastapi.tiangolo.com/tutorial/testing) can be done using  [TestClient provided by](https://fastapi.tiangolo.com/tutorial/testing) [FastAPI](https://fastapi.tiangolo.com/tutorial/testing). This makes test-driven development (TDD) very easy.

app = FastAPI()

@app. get("/")

async def read\_main():

return {"msg": "Hello World"}

client = TestClient(app)

def test\_read\_main():

response = client. get("/")

assert response. status\_code == 200

assert response. json() == {"msg": "Hello World"}

You can easily mock service calls or code in API-defined functions and test them using read\_main TestClient.

### 5. Seamless central exception handling

@app.exception\_handler To handle exceptions with theFastAPI, simply register the response to the app.add\_exception\_handler using annotations or functions. Handled by FastAPI.

app = FastAPI()

@app. exception\_handler(SomeException)

async def http\_exception\_handler(request: Request, exc: SomeException) -> PlainTextResponse:

return PlainTextResponse(str(exc. detail), status\_code=exc. status\_code)

async def request\_exception\_handler(request: Request, exc: SomeOtherException ) -> PlainTextResponse:

return PlainTextResponse(str(exc. detail),status\_code=exc. status\_code)

app. add\_exception\_handler(exc\_class\_or\_status\_code=SomeOtherException,

handler=request\_exception\_handler)

### 6. Excellent documentation

FastAPI has a very extensive and rich [sample documentation](https://fastapi.tiangolo.com/) that makes your life easier. If you need to find out about FastAPI, you usually don't need to look anywhere else.

### 7. Easy introduction

You can easily deploy FastAPI apps via Docker using [the docker image provided by](https://github.com/tiangolo/uvicorn-gunicorn-fastapi-docker) FastAPI. You can also use [Mangum](https://pypi.org/project/mangum/) to deploy to AWS Lamdba.

# **Bad people**

### 1. Crowded main file

With FastAPI, everything is available in the FastAPI app. So, main.py files can get crowded very easily. Here is an example.

app = FastAPI()

app. include\_router(users. router)

app. include\_router(items. router)

app. include\_router(shops. router)

app. include\_router(other. router)

@app. exception\_handler(SomeException)

async def http\_exception\_handler(request: Request, exc: SomeException) -> PlainTextResponse:

return PlainTextResponse(str(exc. detail), status\_code=exc. status\_code)

@app. exception\_handler(SomeOtherException)

async def http\_exception\_handler(request: Request, exc: SomeOtherException) -> PlainTextResponse:

return PlainTextResponse(str(exc. detail), status\_code=exc. status\_code)

Now, suppose that if you handle 10 routers and 20 exceptions, main.py file can be very difficult to maintain. Fortunately, this is easy to solve.

app = FastAPI()

include\_routers(app);

add\_exception\_handlers(app);

include\_routersadd\_exception\_handlers can be stored in separate files.

### 2. Absence of singletons in dependency injection

According to this [Github thread](https://github.com/tiangolo/fastapi/issues/504) singleton, FastAPI dependency injection does not support instances, but does support a single instance per HTTP request. You must create your own singleton class or use a different DI library.

# **Busaiku**

### Request validation

My worst experience when I was using FastAPI was handling request validation. I'll use validation from Pydantic, but as far as I know, there's no easy way to pass validation messages from the point of validation to the response. Write a RequestValidationError or custom validator to handle the Pydantic passed message. For instance

app = FastAPI()

class SomeDto(BaseModel):

data: str = Field(min\_length=1, description="Minimum length must be greater than 1",

title="Minimum length must be greater than 1")

@app. post(path="/")

async def get\_response(request: SomeDto):

return "some response"

@app. exception\_handler(RequestValidationError)

async def handle\_error(request: Request, exc: RequestValidationError) -> PlainTextResponse:

return PlainTextResponse(str(exc. errors()), status\_code=400)

Pydantic returns a list of validation violations, including hardcoded messages from exc.errors(). I couldn't find a way to change it from the documentation for both FastAPI and Pydantic. and title parameter values are also lost.

Django

**Django is** the most popular Python framework and is frequently used in web development. Developers and designers love the battery-inclusive approach that the Django REST framework provides everything you need to build applications. This means that Django developers don't have to rely on separate libraries for functionality. The Django rest framework includes many features for different types of applications, so it works for all kinds of websites.

The Django REST framework includes a wide range of out-of-the-box features, but the core view classes are very simple and the framework is generally easy to use. The main idea behind the DRF is to create a model, a generalized wire representation (e.g. [JSON](http://quintagroup.com/cms/technology/json), XML, etc.) and a specific It's about clearly dividing the set of generic class-based views that can be customized to satisfy API endpoints. Mapping between them.

One of the main differences between DRF and other frameworks is that developers can define URL structures without relying on auto-generated ones, while other frameworks use the Django model to REST It automates many of the conversions to endpoints, which makes it less flexible. In addition, the Django REST Framework includes a built-in API browser for testing newly developed APIs. Web APIs developed in   
DRF are rich and enable web browsing, out-of-the-box support for a wide range of media types, authentication, and permission policies. The display data of the API can use a standard function-based view or use a powerful class-based view to detail more complex functionality. Because authentication as a mechanism for associating an incoming request with a set of identifying credentials is one of its key features, DRF provides a set of out-of-the-box authentication policies (OAuth1a and OAuth2 ) and provideopportunities to develop custom schemes.

**Pros:**

* Machine Learning Capabilities
* Fast Execution
* CDN connectivity and content management
* Source code simplicity, flexibility, quality, and test coverage.
* Powerful serialization engine compatible with both ORM and non-ORM data sources.
* Pluggable and easy customization of emitters, parsers, validators, and authenticators.
* The generic class for CRUD operations.
* A clean and simple view of resources using Django's new class-based view.
* Support for ModelResources with out-of-the-box default implementations and input validation (optional support for forms as input validation).
* HTTP response processing, content type negotiation using HTTP Accept headers.
* Pagination simplifies the process of returning paginated data in a way that it can be rendered to any media type.
* Publishing metadata and query sets.
* Permission classes and throttling management (APIs may have RESTrictive throttling for unauthenticated requests, more RESTrictive throttling for authenticated requests, and so on) 。

**Cons:**

* Monolithik
* Not suitable for small projects
* Learning Curve

**There are two main tools** for Django **to work with MongoDB: Djongo and MongoEngine.**

**Djongo is** the  **MongoDB database** engine **for** **Django . It's still under development, but** has **the big advantage of being able to use** **Django's native ORM.**

**MongoEngine,** on the other hand, is an independent library that introduces its own  **ORM (actually ODM - more on that later). This is very robust, but** **should be used instead** of  **Django's ORM**, which is expensive**.**

Depending on your purpose, it is better to choose one or the other. However, if you're working on a simple project or just learning MongoDB, we recommend Djongo. This saves you quite a bit of hassle.

Let's take a closer look at each tool to help you make an informed decision. It also uses the Django REST Framework to create simple APIs to demonstrate how they work.

## **What is Django?**

As the official [GitHub](https://github.com/nesdis/djongo)  states , Django will use Django in MongoDB without changing the Django ORM It is the only connector that can be used.

Internally, Django converts the SQL queries generated by Django into MongoDB queries. This is how you maintain your ORM.

Maintaining a native ORM has several advantages.

* Stick with the familiar Django model API or migrate existing projects from SQL to NoSQL databases
* Core Django modules such as user authentication and permissions, sessions and admin panel work fine
* Third-party packages that depend on Django's ORM should also work

However, while Djongo appears to be fairly stable, it's still **under development**.

So, if you plan to work on a large, long-term project, you may encounter unexpected problems at some point, as complex projects often require unusual solutions.

Check out [open issues on](https://github.com/nesdis/djongo/issues) GitHub to get a better idea of where things are.

## What is the Mongo engine?

According to [official GitHub](https://github.com/MongoEngine), MongoEngine is a Python object document mapper for working with MongoDB . It's very similar to Django's ORM (in fact, even its API has many similarities), but For document-oriented databases (hence the name ODM).

It is actively developed, has a large community, and integrates with various frameworks such as Flask. Overall, MongoEngine is a fairly robust solution.

But the main downside of MongoEngine with respect to Django is that it does exactly that. MongoEngine must use its own ODM, so some issues are to be expected.

* Core Django modules such as sessions, user authentication and permissions, and the admin panel do not work out of the box.
* The same applies to third-party packages that rely on Django's ORM.
* If you migrate an existing project from SQL to a NoSQL database, you must recreate all models

[There](https://github.com/MongoEngine/django-mongoengine) have been attempts to address Django's [core modules](https://github.com/MongoEngine/django-mongoengine), but their integration seems to be incomplete yet.

As you probably guessed, you might find yourself rebuilding a lot of things that would otherwise have been covered by the existing package.

So, if your project is relatively complex and you need a non-relational database, you should ask yourself if Django (and the Django REST Framework) is the right choice in the first place.

flask

**Flask** is a popular web framework for Python projects. Over the years, Flask has added many new features, on par with other full-stack frameworks such as Django and Ruby on Rails. It also takes a minimal approach to building Web applications.

**Pros:**

* Documentation
* Scaleravirity
* Light dose
* It is a lightweight framework that provides hassle-free development.
* Provide developers with flexibility to experiment with modules and architectures
* Suitable for small projects
* Provides built-in development server and fast debugger
* Easy to scale applications
* Secure cookie support
* Using the Ninja2 Template Engine
* Integrated support for unit tests
* API is consistent and neat
* Provide strong WSGI support

**Cons:**

* High maintenance and implementation costs
* Complex technical stucking
* Security Lisk
* Not suitable for large applications and projects
* It does not provide a built-in administration site to maintain a record of insertions, deletions, and changes.
* Lack of database and ORM
* Complex maintenance of large-scale applications
* The library must be installed manually
* Limited community support compared to Django
* Slow minimum viable product (MVP) development process
* No login, no authentication.

## Not suitable for large applications

For large projects, using Flask can be time-consuming. It is only suitable for small applications. If you want to add a simple and innovative use case to an existing application, you should choose Flask because of its flexibility.

Each project can be in a single application, but multiple models and views can be added to a single application. Django, on the other hand, allows users to split a single project into multiple smaller applications, making it easier to develop and maintain.

# Community

The flask community is not as large as Django. Therefore, it is difficult to find a solution to the problem because there are not many active users answering the question.

Django has been around since 2005. Flask stepped in about five years later, in 2010. At the time of editing this article, Django has 244615 questions about StackOverflow, while Flask has 40800 questions.

# Full-stack experience

Django has a default web template engine that provides a full-stack experience . Flask does not have a default template engine. For this purpose, we will use the Jinja2 template engine. The flask is freeloaded. Therefore, it does not need to be installed separately.

# No admin site

One of the main advantages of Django is the availability of an admin site. Here you can view all models, insert new records, and update and delete old records. You can't do that with a flask. There is no administration site. But don't worry because there is a workaround. You can solve this problem by installing a library called flask-admin to create an administration site.

In the administration site, you can create/view/edit, and delete records from the database. You can also customize this administration site to add filters, search fields, and sorting to database records.

# No login or authentication

Flask does not provide authentication. There is no login function. However, Django provides login and authentication that can be used in web applications for login purposes.

Django has a built-in login mechanism. You can use Django's default User table to create users, set passwords, and more.

# ORM

ORMs are the next key thing that every web application needs: ORMs provide APIs for performing CRUD operations on databases. Without an ORM, you would have to manually create SQL queries to perform CRUD operations. This can also be fixed by installing another library called SQLAlchemy.

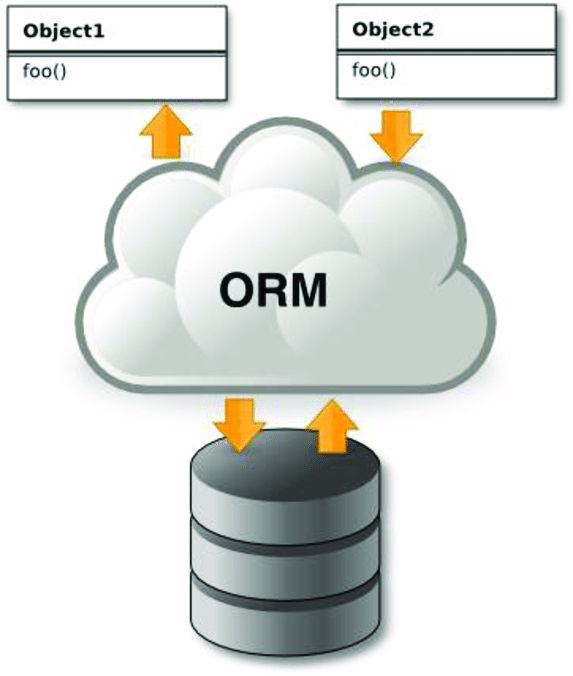
By default, Django supports ORMs. Let's say you have a table called Student. If you want to get all the records from the student table, you can just do the following in Django:

Students. objects.all()

This will give you all the records in the student table. This is equivalent to the following query in SQL:

SELECT \* FROM students;

This is a simple query, so there are no major changes. But do you get the idea right? Complex queries are easy to create using an ORM. Of course, flasks do not have this function either.



Django's ORM

# Migration can be difficult

Flask is also difficult to migrate. Unlike Django, there is no direct way to migrate databases. This can be fixed by installing the Flask-migrate library.

Comparison table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Django | Flask | Pyramid | Web2py | FastAPI |
| Type | Python all-inclusive megaframework for building web applications. | Python microframework for building web applications. | Python "microframework" for building web applications. | Python full stack for building web applications. | A minimal framework based on starlette and pydantic for building fast web applications using asynchronous IO . |
| Release Date | 2005 | 2010 | 2008 | 2007 | 2018 |
| license | [Three-clause license](https://en.wikipedia.org/wiki/BSD_licenses#3-clause) | [BSD Three-Clause License](https://en.wikipedia.org/wiki/BSD_licenses#3-clause) | [BSD License](https://en.wikipedia.org/wiki/BSD_licenses) | [GNU Lesser General Public License version 3 (LGPLv3)](https://en.wikipedia.org/wiki/GNU_Lesser_General_Public_License) | [My License](https://en.wikipedia.org/wiki/MIT_License) |
| Website | [www.djangoproject.com](https://www.djangoproject.com/) | [palletsprojects.com/p/flask](https://palletsprojects.com/p/flask/) | [trypyramid.com](https://trypyramid.com/) | [www.web2py.com](http://www.web2py.com/) | [fastapi.tiangolo.com](https://fastapi.tiangolo.com/) |
| Popularity | Used in 367,000 projects. | Used in 397,000 projects. | It has been used in 8,000 projects. | Used in 36 projects. | It's a new addition to the Python web framework family, but it's growing in popularity. |
| Used by | nstagram、Pinterest、Coursera、Udemy。 | Netflix, Giraud, Lyft. | Reddit、Yelp、Mozilla、SurveyMoney、NewCars.com。 | It is not used in large companies. | Uber (internal use) Explosion AI, Microsoft (internal use) |
| Jobs | **2074**  jobs **that list Django**  as requirement. | **1067**  jobs **that require** Flask. | **80**  jobs listing pyramids as requirements. | **8** Job **Vacancies that list** **web2py** as requirements. | **100**  jobs **listing Fast API**  as requirement. |
| Performance | It's not as fast as Barebones Flask and other microframeworks, but for many real-world use cases, the difference is negligible. | Because of its minimal and low overhead, Flask performs very well. Extensions can have a negative impact on performance. | It claims to have the best performance, but has the advantage of being a microframework, but [benchmarks prove that it's slower than](https://www.techempower.com/benchmarks/#section=data-r13&hw=ph&test=query&l=4ftgjj&f=zik0zb-zik0zj-v2qiv3-zik0zj-zik0zj-zik0zj-3j)  Django, beyond simple use cases. | Web2py has a unique approach that runs the model and controller in a single global environment that is initialized for each HTTP request. We employ a lot. This approach has advantages such as developers not having to worry about cleanup or avoid conflicts between requests, but the main drawback is that it suffers performance because the code is a model and runs on every request. | High. |
| Flexibility | Unlike microframeworks (like Flask), where developers don't have an opinion on how things are structured, Django expects things to be done a certain way. However, it does this without compromising flexibility. Django has been used to build everything from content management systems to social networks to scientific computing platforms. | It's very flexible and doesn't require users to use a specific project or code layout. (A structured approach is still recommended.) ) | The pyramid is flexible. | It's not as flexible as microframeworks, but it's not always intrusive. | Fast API provides flexibility in writing code and does not restrict users to specific projects or code layouts. |
| Ease of learning | There is a learning curve, especially for those who are not familiar with other web frameworks. However, there are some great online resources, course tutorials, and YouTube videos. | Flask is simple and its core features are not difficult to learn. There are also plenty of online resources to help you learn . | Being a microframework, the learning curve is not that steep. However, the amount of tutorials, courses, and resources available for learning is not the same compared to Django or Flask . | Online tutorials and resources are limited, many of which are several years old. The best resource for learning is web2py author's own "web2py [complete reference manual", which seems to have been written in 2013 .](http://www.web2py.com/books/default/chapter/29/01/introduction) | It's easy to learn, especially for those who are new to Web development. However, there are not many online resources, courses, and tutorials. |
| RDBMS support | built-in  Django comes with a built-in ORM framework [to help developers get started quickly.](https://docs.djangoproject.com/en/2.2/topics/db/models/)  **Verdict** ORM is one of the best features of Django, beloved by developers. | Through plugins or extensions  Flask does not have a built-in ORM framework. Developers can use one of many open source libraries or extensions. [Flask-SQLAlchemy, Flask-Pony](https://flask-sqlalchemy.palletsprojects.com/en/2.x/),etc. | Through plugins or extensions  There is no built-in ORM framework. [Leave the choice of library like](https://docs.pylonsproject.org/projects/pyramid-cookbook/en/latest/database/sqlalchemy.html) SQLAlchemy to the developer | built-in  It comes with the Database Abstraction Layer (DAL) to support MySQL, PostgreSQL, SQLite, and many other relational databases. | Through plugins or extensions  FastAPI does not have a built-in ORM, but is compatible with SQLAlchemy's [Pydantic](https://www.sqlalchemy.org/) ORM mode. |
| NoSQL support | NoSQL databases are not officially supported by Django. Open PynamoDB[,](https://github.com/pynamodb/PynamoDB) Django MongoDB Engine, [Django non-rel,](https://django-mongodb-engine.readthedocs.io/) [etc. to support NoSQL](http://django-nonrel.org/)  You have a source project. Some of these extensions support specific Django versions and do not work well with Django  [ORMs](https://code.djangoproject.com/wiki/NoSqlSupport).  **Verdict We don't recommend using**  NoSQL databases with Django. | NoSQL databases are supported through open source libraries or extensions. To use MongoDB with Flask, Flask-PyMong [is](https://flask-pymongo.readthedocs.io/en/latest/) a common choice. CouchDB, Cassandra, and DynamoDB are also supported via the library.  **Verdict** If you're developing for a NoSQL database, Flask is the way to go. | It does not interfere with the use of NoSQL and is supported similarly to relational databases. | There is no built-in support. Support for NoSQL databases is non-limited at all times. Currently, we only support Google Datastore on Google App Engine. | Fast API includes MongoDb, ElasticSearch, Cassandra, CouchDB, ArangoDB and many Supports NoSQL databases. |
| Admin dashboard | built-in  Django comes with a web-based administration site with an easy-to-use UI. This allows you to quickly perform and test CRUD operations on your model from the browser. | Through plugins or extensions  There is no built-in admin panel, but you can use [the Flask-Admin](https://github.com/flask-admin/flask-admin) extension. It supports many backends such as SQLAlchemy, MongoEngine, and Peewee. | Through plugins or extensions  Although not shipped, [third-party tools](https://awesome-pyramid.readthedocs.io/en/latest/awesome.html#admin-interface) are available. | built-in  Yes, it comes with a built-in management panel. | built-in  Yes, use Swagger as the web user interface for the api docservant |
| REST support | Although not built-in, REST development is supported via the popular and active [Django REST Framework](https://www.django-rest-framework.org/) project.  [API versioning, Browsable APIs to interact with APIs through a web browser](https://www.django-rest-framework.org/topics/browsable-api/), authentication (OAuth1 and OAuth2) Supports serialization support for both ORM and non-ORM sources . | [It is supported via extensions such as Flask-RESTful](https://flask-restful.readthedocs.io/), Flask-Classful[, and](http://flask-classful.teracy.org/) Flask-RESTPlus. | Because it's a minimal framework, users can easily build their own REST APIs or use [third-party libraries like](https://cornice.readthedocs.io/en/latest/) Cornce. | [There is some support](http://web2py.com/books/default/chapter/29/10/services?search=restful#Restful-Web-Services). | Developers can quickly build REST APIs. |
| Security | Built-in protection against several common attack vectors, including CSRF, XSS, and SQL injection. When vulnerabilities are discovered, the Django team has a good [security policy](https://docs.djangoproject.com/en/dev/internals/security/) and a [fix](https://www.djangoproject.com/weblog/2020/mar/04/security-releases/) is released quickly. | [Despite being a minimalist framework, Flask](https://flask.palletsprojects.com/security/)  is [quick](https://flask.palletsprojects.com/security/) to address common security issues such as CSRF, XSS, and JSON security. Does an excellent job of coping. [Third-party extensions like Flask-Security](https://pythonhosted.org/Flask-Security/) can be used for general security measures. However, developers should carefully assess these extensions for security risks and manually apply timely updates when vulnerabilities are discovered. | CSRF protection is built-in, but [XSS](https://snyk.io/vuln/SNYK-PYTHON-PYRAMID-40730) vulnerabilities have been discovered in older versions. | Built-in protection against input injection, XSS, and common vulnerabilities. There are known security vulnerabilities. | FastAPI provides several tools for many security schemes in the fastapi.security module . There is not enough data. |
| Templating library | Use your [own template engine](https://docs.djangoproject.com/en/2.2/ref/templates/). Optionally, you can  [configure](https://docs.djangoproject.com/en/2.2/topics/templates/#django.template.backends.jinja2.Jinja2) Django to use Jinja2. | Flask will immediately use  [Jinja2.](https://flask.palletsprojects.com/templating/) | The default template language is not shipped and must be added by the developer. [Jinja2 can be added to](https://docs.pylonsproject.org/projects/pyramid/en/latest/quick_tutorial/jinja2.html) a Pyramid. Another popular engine among Pyramid users is [Chameleon](https://docs.pylonsproject.org/projects/pyramid-chameleon/en/latest/). | You can use the custom [web2py template language](http://web2py.com/books/default/chapter/29/05/the-views) out of the box. | FastAPI supports Jinja2 for templating and aiofiles for serving static files |
| Web forms | It comes with built-in ModelForms that fully support web forms such as input validation, CSRF, XSS, [and SQL injection.](https://docs.djangoproject.com/en/2.2/topics/forms/modelforms/) | There is no built-in support, but there is a [Flask-WTF](https://flask-wtf.readthedocs.io/) extension. SQLAlchemy support, that is, to create forms based on models, is [WTForms-Alchemy.](https://wtforms-alchemy.readthedocs.io/en/latest/) | There is no built-in support. Prefer third-party Deform libraries. | Built-in support. Please [see here for details](http://web2py.com/books/default/chapter/29/07/forms-and-validators). | It comes with its own [Forms with basic features](https://fastapi.tiangolo.com/tutorial/request-forms/). |
| authentication | Built-in authentication, authorization, account management, and session support. | It only provides support for cookie-based sessions, but has extensions that can be used for [authentication, authorization, and account management](https://pythonhosted.org/Flask-Security/). | Built-in support for authentication, [authorization, and session](https://docs.pylonsproject.org/projects/pyramid/en/latest/narr/security.html). | It includes sufficient [role-based access control mechanisms (RBAC)](https://en.wikipedia.org/wiki/Role-based_access_control)  for many use cases . | The Fast API supports OAuth2, JWT, and simple HTTP authentication. |
| testing | [Built-in support using  Python's](https://docs.python.org/3/library/unittest.html)  unittest framework. | [Built-in support using Python's](https://docs.python.org/3/library/unittest.html) unittest framework. | Use [pytest for unit tests](https://docs.pytest.org/en/latest/index.html#features)  . | [It can be run](https://docs.python.org/3/library/unittest.html) using [Python's](https://docs.python.org/3/library/unittest.html) unittest framework or [doctest](https://en.wikipedia.org/wiki/Doctest). | [Use Starlette and pytest](https://fastapi.tiangolo.com/tutorial/testing/). |

3 URL link list

[**https://stoplight.io/api-types**](https://stoplight.io/api-types)

[**https://www.cioinsight.com/enterprise-apps/types-of-api/**](https://www.cioinsight.com/enterprise-apps/types-of-api/)

[**https://codersera.com/blog/different-types-of-api-for-web-development/amp/**](https://codersera.com/blog/different-types-of-api-for-web-development/amp/)

[**https://getstream.io/blog/api-protocols/**](https://getstream.io/blog/api-protocols/)

[**https://www.valuecoders.com/blog/technology-and-apps/types-of-web-apis-for-building-enterprise-grade-web-applications/**](https://www.valuecoders.com/blog/technology-and-apps/types-of-web-apis-for-building-enterprise-grade-web-applications/)

[**https://www.makeuseof.com/api-architecture-types-how-work/**](https://www.makeuseof.com/api-architecture-types-how-work/)

[**https://blog.dreamfactory.com/types-of-apis-popular-rest-api-protocol/**](https://blog.dreamfactory.com/types-of-apis-popular-rest-api-protocol/)

[**https://konghq.com/learning-center/api-management/different-api-types-and-use-cases**](https://konghq.com/learning-center/api-management/different-api-types-and-use-cases)

[**https://testsigma.com/blog/different-types-of-apis-and-protocols-2022-updated/**](https://testsigma.com/blog/different-types-of-apis-and-protocols-2022-updated/)

[**https://apibakery.com/guides/rest-api/types-of-apis/**](https://apibakery.com/guides/rest-api/types-of-apis/)

[**https://www.soapui.org/learn/api/soap-vs-rest-api/**](https://www.soapui.org/learn/api/soap-vs-rest-api/)

[**https://www.zibtek.com/blog/rest-vs-soap/**](https://www.zibtek.com/blog/rest-vs-soap/)

[**https://www.mailgun.com/blog/it-and-engineering/restful-api/**](https://www.mailgun.com/blog/it-and-engineering/restful-api/)

[**https://hevodata.com/learn/http-api-vs-rest-api/**](https://hevodata.com/learn/http-api-vs-rest-api/)

[**https://smartbear.com/blog/soap-vs-rest-whats-the-difference/**](https://smartbear.com/blog/soap-vs-rest-whats-the-difference/)

[**https://www.altexsoft.com/blog/rest-api-design/**](https://www.altexsoft.com/blog/rest-api-design/)

[**https://raygun.com/blog/soap-vs-rest-vs-json/**](https://raygun.com/blog/soap-vs-rest-vs-json/)

[**https://bejamas.io/blog/graphql-vs-rest-api/**](https://bejamas.io/blog/graphql-vs-rest-api/)

[**https://research.aimultiple.com/graphql-vs-rest/**](https://research.aimultiple.com/graphql-vs-rest/)

[**https://stablekernel.com/article/advantages-and-disadvantages-of-graphql/**](https://stablekernel.com/article/advantages-and-disadvantages-of-graphql/)

[**https://iq.opengenus.org/rpc-vs-rest/**](https://iq.opengenus.org/rpc-vs-rest/)

[**https://www.javatpoint.com/what-is-rpc-in-operating-system**](https://www.javatpoint.com/what-is-rpc-in-operating-system)

[**https://www.guru99.com/remote-procedure-call-rpc.html#7**](https://www.guru99.com/remote-procedure-call-rpc.html#7)

[**https://www.guru99.com/soap-simple-object-access-protocol.html**](https://www.guru99.com/soap-simple-object-access-protocol.html)

[**https://codeahoy.com/compare/python-frameworks.html**](https://codeahoy.com/compare/python-frameworks.html)

[**https://quintagroup.com/cms/python/django-rest-framework**](https://quintagroup.com/cms/python/django-rest-framework)

[**https://sourcery.blog/how-to-build-api-with-django-rest-framework-and-mongodb/**](https://sourcery.blog/how-to-build-api-with-django-rest-framework-and-mongodb/)

[**https://www.netguru.com/blog/python-frameworks-comparison**](https://www.netguru.com/blog/python-frameworks-comparison)

[**https://codeahoy.com/compare/python-frameworks.html**](https://codeahoy.com/compare/python-frameworks.html)

[**https://copyprogramming.com/howto/top-python-frameworks-for-web-development-advantages-disadvantages-and-tips**](https://copyprogramming.com/howto/top-python-frameworks-for-web-development-advantages-disadvantages-and-tips)

[**https://insideaiml.com/blog/How-to-choose-best-framework%3F-992**](https://insideaiml.com/blog/How-to-choose-best-framework%3F-992)

[**https://dev.to/fuadrafid/fastapi-the-good-the-bad-and-the-ugly-20ob**](https://dev.to/fuadrafid/fastapi-the-good-the-bad-and-the-ugly-20ob)

[**https://www.imaginarycloud.com/blog/flask-vs-fastapi/**](https://www.imaginarycloud.com/blog/flask-vs-fastapi/)

[**https://www.mygreatlearning.com/blog/flask-vs-django/**](https://www.mygreatlearning.com/blog/flask-vs-django/)

[**https://allwin-raju-12.medium.com/disadvantages-of-flask-95f1ff8f83bf**](https://allwin-raju-12.medium.com/disadvantages-of-flask-95f1ff8f83bf)