What is API?

In [computer programming](https://en.wikipedia.org/wiki/Computer_programming), an **application programming interface** (**API**) is a set of [subroutine](https://en.wikipedia.org/wiki/Subroutine) definitions, [protocols](https://en.wiktionary.org/wiki/Protocol), and tools for building [application software](https://en.wikipedia.org/wiki/Application_software). In general terms, it is a set of clearly defined methods of **communication** between various software components. A good API makes it easier to develop a [computer program](https://en.wikipedia.org/wiki/Computer_program) by providing all the building blocks, which are then put together by the [programmer](https://en.wikipedia.org/wiki/Programmer). An API may be for a web-based system, [operating system](https://en.wikipedia.org/wiki/Operating_system), [database system](https://en.wikipedia.org/wiki/Database_system), [computer hardware](https://en.wikipedia.org/wiki/Computer_hardware) or [software library](https://en.wikipedia.org/wiki/Library_(computing)). An API specification can take many forms, but often includes specifications for [routines](https://en.wikipedia.org/wiki/Subroutine), [data structures](https://en.wikipedia.org/wiki/Data_structure), [object classes](https://en.wikipedia.org/wiki/Class_(computer_programming)), [variables](https://en.wikipedia.org/wiki/Variable_(computer_science)) or [remote calls](https://en.wikipedia.org/wiki/Remote_procedure_call). [POSIX](https://en.wikipedia.org/wiki/POSIX), [Windows API](https://en.wikipedia.org/wiki/Windows_API) and [ASPI](https://en.wikipedia.org/wiki/Advanced_SCSI_programming_interface) are examples of different forms of APIs. Documentation for the API is usually provided to facilitate usage.

Just as a [graphical user interface](https://en.wikipedia.org/wiki/Graphical_user_interface) makes it easier for people to use programs, application programming interfaces make it easier for [developers](https://en.wikipedia.org/wiki/Software_developer) to use certain technologies in building applications. By [abstracting](https://en.wikipedia.org/wiki/Abstraction_(software_engineering)) the underlying implementation and only exposing objects or actions the developer needs, an API simplifies programming. While a graphical interface for an [email client](https://en.wikipedia.org/wiki/Email_client) might provide a user with a button that performs all the steps for fetching and highlighting new emails, an API for file [input/output](https://en.wikipedia.org/wiki/Input/output) might give the developer a [function](https://en.wikipedia.org/wiki/Subroutine) that copies a file from one location to another without requiring that the developer understand the [file system](https://en.wikipedia.org/wiki/Journaling_file_system) operations occurring behind the scenes.

**Libraries and frameworks**

An API is usually related to a [software library](https://en.wikipedia.org/wiki/Library_(computing)). The API describes and prescribes the *expected behavior* (a specification) while the library is an *actual implementation* of this set of rules. A single API can have multiple implementations (or none, being abstract) in the form of different libraries that share the same programming interface. The separation of the API from its implementation can allow programs written in one language to use a library written in another. For example, because [Scala](https://en.wikipedia.org/wiki/Scala_(programming_language)" \o "Scala (programming language)) and [Java](https://en.wikipedia.org/wiki/Java_(programming_language)) compile to compatible [bytecode](https://en.wikipedia.org/wiki/Bytecode" \o "Bytecode), Scala developers can take advantage of any Java API.

API use can vary depending on the type of programming language involved. An API for a [procedural language](https://en.wikipedia.org/wiki/Procedural_programming) such as [Lua](https://en.wikipedia.org/wiki/Lua_(programming_language)" \o "Lua (programming language)) could primarily consist of basic routines to execute code, manipulate data or handle errors, while an API for an [object-oriented language](https://en.wikipedia.org/wiki/Object-oriented_programming) such as [Java](https://en.wikipedia.org/wiki/Java_(programming_language)) would provide a specification of classes and their [class methods](https://en.wikipedia.org/wiki/Class_method).

[Language bindings](https://en.wikipedia.org/wiki/Language_binding) are also APIs. By mapping the features and capabilities of one language to an interface implemented in another language, a language binding allows a library or service written in one language to be used when developing in another language. Tools such as [SWIG](https://en.wikipedia.org/wiki/SWIG) and F2PY, a [Fortran](https://en.wikipedia.org/wiki/Fortran)-to-[Python](https://en.wikipedia.org/wiki/Python_(programming_language)) interface generator, facilitate the creation of such interfaces.

An API can also be related to a [software framework](https://en.wikipedia.org/wiki/Framework_(computer_science)): a framework can be based on several libraries implementing several APIs, but unlike the normal use of an API, the access to the behavior built into the framework is mediated by extending its content with new classes plugged into the framework itself. Moreover, the overall program flow of control can be out of the control of the caller and in the hands of the framework via [inversion of control](https://en.wikipedia.org/wiki/Inversion_of_control) or a similar mechanism.

**Operating systems**

An API can specify the interface between an application and the [operating system](https://en.wikipedia.org/wiki/Operating_system). [POSIX](https://en.wikipedia.org/wiki/POSIX), for example, specifies a set of common APIs that aim to enable an application written for a POSIX conformant operating system to be [compiled](https://en.wikipedia.org/wiki/Compiler) for another POSIX conformant operating system. [Linux](https://en.wikipedia.org/wiki/Linux) and [Berkeley Software Distribution](https://en.wikipedia.org/wiki/Berkeley_Software_Distribution) are examples of operating systems that implement the POSIX APIs.

[Microsoft](https://en.wikipedia.org/wiki/Microsoft) has shown a strong commitment to a backward-compatible API, particularly within their [Windows API](https://en.wikipedia.org/wiki/Windows_API) (Win32) library, so older applications may run on newer versions of Windows using an executable-specific setting called "Compatibility Mode".

An API differs from an [application binary interface](https://en.wikipedia.org/wiki/Application_binary_interface) (ABI) in that an API is source code based while an ABI is [binary](https://en.wikipedia.org/wiki/Binary_file) based. For instance, [POSIX](https://en.wikipedia.org/wiki/POSIX) provides APIs, while the [Linux Standard Base](https://en.wikipedia.org/wiki/Linux_Standard_Base) provides an ABI.

**Remote APIs**

Remote APIs allow developers to manipulate remote resources through [protocols](https://en.wikipedia.org/wiki/Communications_protocol), specific standards for communication that allow different technologies to work together, regardless of language or platform. For example, the Java Database Connectivity API allows developers to query many different types of [databases](https://en.wikipedia.org/wiki/Database) with the same set of functions, while the [Java remote method invocation](https://en.wikipedia.org/wiki/Java_remote_method_invocation) API uses the [Java Remote Method Protocol](https://en.wikipedia.org/wiki/Java_Remote_Method_Protocol) to allow [invocation](https://en.wikipedia.org/wiki/Remote_procedure_call) of functions that operate remotely, but appear local to the developer. Therefore, remote APIs are useful in maintaining the object abstraction in [object-oriented programming](https://en.wikipedia.org/wiki/Object-oriented_programming); a method call, executed locally on a proxy object, invokes the corresponding method on the remote object, using the remoting protocol, and acquires the result to be used locally as return value. A modification on the proxy object will also result in a corresponding modification on the remote object.

**Web APIs**

Web APIs are the defined interfaces through which interactions happen between an enterprise and applications that use its assets. An API approach is an architectural approach that revolves around providing programmable interfaces to a set of services to different applications serving different types of consumers.[[18]](https://en.wikipedia.org/wiki/Application_programming_interface#cite_note-18) When used in the context of [web development](https://en.wikipedia.org/wiki/Web_development), an API is typically defined as a set of [Hypertext Transfer Protocol](https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol) (HTTP) request messages, along with a definition of the structure of response messages, which is usually in an Extensible Markup Language ([XML](https://en.wikipedia.org/wiki/XML)) or JavaScript Object Notation ([JSON](https://en.wikipedia.org/wiki/JSON)) format. An example might be a shipping company API that can be added to an eCommerce-focused website, to facilitate ordering shipping services and automatically include current shipping rates, without the site developer having to enter the shipper's rate table into a web database. While "web API" historically has been virtually synonymous for [web service](https://en.wikipedia.org/wiki/Web_service), the recent trend (so-called [Web 2.0](https://en.wikipedia.org/wiki/Web_2.0)) has been moving away from Simple Object Access Protocol ([SOAP](https://en.wikipedia.org/wiki/SOAP)) based web services and [service-oriented architecture](https://en.wikipedia.org/wiki/Service-oriented_architecture) (SOA) towards more direct [representational state transfer](https://en.wikipedia.org/wiki/Representational_state_transfer) (REST) style [web resources](https://en.wikipedia.org/wiki/Web_resource) and [resource-oriented architecture](https://en.wikipedia.org/wiki/Resource-oriented_architecture) (ROA). Part of this trend is related to the [Semantic Web](https://en.wikipedia.org/wiki/Semantic_Web) movement toward [Resource Description Framework](https://en.wikipedia.org/wiki/Resource_Description_Framework) (RDF), a concept to promote web-based [ontology engineering](https://en.wikipedia.org/wiki/Ontology_engineering) technologies. Web APIs allow the combination of multiple APIs into new applications known as [mashups](https://en.wikipedia.org/wiki/Mashup_(web_application_hybrid)" \o "Mashup (web application hybrid)). In the social media space, web APIs have allowed web communities to facilitate sharing content and data between communities and applications. In this way, content that is created in one place can be dynamically posted and updated in multiple locations on the web.[[21]](https://en.wikipedia.org/wiki/Application_programming_interface#cite_note-Parr16-21) For example, Twitter's REST API allows developers to access core Twitter data and the Search API provides methods for developers to interact with Twitter Search and trends data.

The main policies for releasing an API are:

* Private: The API is for internal company use only.
* Partner: Only specific business partners can use the API. For example, [car service companies](https://en.wikipedia.org/wiki/Transportation_network_company) such as [Uber](https://en.wikipedia.org/wiki/Uber_(company)" \o "Uber (company)) and [Lyft](https://en.wikipedia.org/wiki/Lyft" \o "Lyft) allow approved third-party developers to directly order rides from within their apps. This allows the companies to exercise quality control by curating which apps have access to the API, and provides them with an additional revenue stream.
* Public: The API is available for use by the public. For example, [Microsoft](https://en.wikipedia.org/wiki/Microsoft) makes the [Microsoft Windows](https://en.wikipedia.org/wiki/Microsoft_Windows) API public, and [Apple](https://en.wikipedia.org/wiki/Apple_Inc.) releases its APIs [Carbon](https://en.wikipedia.org/wiki/Carbon_(API)) and [Cocoa](https://en.wikipedia.org/wiki/Cocoa_(API)), so that software can be written for their [platforms](https://en.wikipedia.org/wiki/Computing_platform).

RESTful APIs

Below example talk about one of the approach to explain about REST.

Lets got to history, Talk about the Roy Fielding Research – “[Architectural Styles and the Design of Network-based Software Architectures](http://www.cs.colorado.edu/~kena/classes/7818/f08/lectures/lecture_9_fielding_disserta.pdf)“. Its a big paper and talks a lot of various stuff. But as a standard engineer How you would like to explain the clear meaning of REST, and what is its Architectural Style.

Here is one of the way to explain – “What is REST”.

See this www(world wide web) running on top of various hardwares e.g. routers,servers,browsers,firewalls,cloud infrastructures,LAN,WAN. The overall objective of this www(world wide web) to distribute [hypermedia](https://en.wikipedia.org/wiki/Hypermedia). This world wide web equipped with various services e.g. informational based services, websites, youtube channels, dynamic websites, static websites. This world wide web uses HTTP protocol to distribute hypermedia across the world with a client/server mechanism, on top of TCP/IP or other appropriate network stack.

This [HTTP protocol](http://www.w3.org/Protocols/rfc2616/rfc2616.txt) is using eight methods to manage the ‘protocol of distribution’ or ‘Architectural Style of Distribution’. Those eight methods are namely : OPTIONS,GET,HEAD,POST,PUT,DELETE, TRACE,CONNECT.

But on Top of this HTTP, web applications are using its own way of distributing hypermedia e.g web applications are using web services which are highly tied with clients and servers ‘or’ web applications are using its own way of designed client/server mechanism to make such distribution channel on top of HTTP.

What [Roy Fielding Research](http://www.cs.colorado.edu/~kena/classes/7818/f08/lectures/lecture_9_fielding_disserta.pdf) says , that these eight methods OPTIONS,GET,HEAD,POST,PUT,DELETE,TRACE,CONNECT of HTTP are so successful to deliver HyperMedia to all across the world on top of variety of hardware resources and network stacks with client/server mechanism, Why don’t we use the similar strategy with our web based application as well. On this GET,POST,DELETE and PUT are used the most. so four methods deliver HyperMedia to all across the world.

**Related Answers:**

* **API** means Application Programming Interface In computer programming, API is a set of subroutine definitions, protocols, and tools for building application software.
* A **web service** is any piece of software that makes itself available over the internet and uses a standardized XML messaging system. XML is used to encode all communications to a **web service**.
* **HTTP**means Hypertext Transfer Protocol, which means the underlying protocol used by the World Wide Web and this protocol defines how messages are formatted and transmitted, and what actions Web servers and browsers should take in response to various commands.
* An **architectural style** is characterized by the features that make a building or other structure notable or historically identifiable. A **style**may include such elements as form, method of construction, building materials, and regional character.

2.6k Views · [View Upvoters](https://www.quora.com/api/mobile_expanded_voter_list?key=biyMpZhz3HH&type=answer)

[Upvote**3**Downvote](https://www.quora.com/What-is-a-REST-API)

[Share](https://www.quora.com/What-is-a-REST-API)

[RecommendedAll](https://www.quora.com/What-is-a-REST-API)

[Amruta Phansalker](https://www.quora.com/profile/Amruta-Phansalker)

[Marlon Carvalho](https://www.quora.com/profile/Marlon-Carvalho-1)

[Marlon Carvalho](https://www.quora.com/profile/Marlon-Carvalho-1), worked at Application Programming Interfaces

[Answered Nov 10, 2014](https://www.quora.com/What-is-a-REST-API/answer/Marlon-Carvalho-1)

Originally Answered: [How would you describe RESTful programming/REST APIs to a complete novice?](https://www.quora.com/How-would-you-describe-RESTful-programming-REST-APIs-to-a-complete-novice?no_redirect=1)

The term REST isn't such a new thing. It was forged by Roy Fielding in his thesis, written in 2000. In a few words, REST isn't a pattern, so you won't see it detailed in a RFC (W3C). REST isn't a framework either, so you won't see it as a Python/Java/Ruby library.   
  
It's the Roy Fielding's view of how the Web should work and how every web application should be built to meet this Web architecture. In his thesis, Fielding describes the key principles that you must follow when you're defining your application architecture.  
  
Some people argue that REST is directly related to the HTTP protocol but it's not. Even though HTTP is the main protocol on the web, the REST Principles can be applied to any protocol on the Web. However, the HTTP protocol was created by Roy Fielding and so it entails that this protocol follows Fielding's REST practices somehow.  
  
You'll find a lot of questions about it on Quora, so I won't repeat them here. Instead, I prefer listing some of these questions:  
  
1. [What is a REST API?](https://www.quora.com/What-is-a-REST-API)  
2. [What are the drawbacks of using RESTful APIs?](https://www.quora.com/What-are-the-drawbacks-of-using-RESTful-APIs)  
3. [What is REST API?](https://www.quora.com/What-is-REST-API-1)

3.5k Views · [View Upvoters](https://www.quora.com/api/mobile_expanded_voter_list?key=9dW29G7gccy&type=answer)

[Upvote**10**Downvote](https://www.quora.com/What-is-a-REST-API)

[Share](https://www.quora.com/What-is-a-REST-API)

[RecommendedAll](https://www.quora.com/What-is-a-REST-API)

[Amruta Phansalker](https://www.quora.com/profile/Amruta-Phansalker)

[Ben Nexcess](https://www.quora.com/profile/Ben-Nexcess)

[Ben Nexcess](https://www.quora.com/profile/Ben-Nexcess), Web Hosting Worker and Writer at Nexcess

[Answered Oct 1, 2017](https://www.quora.com/What-is-a-REST-API/answer/Ben-Nexcess)

An API specifies how one piece of software should talk to another, and what responses it can expect. Neither piece of software cares what’s happening inside the other (the implementation). They don’t have to understand each other’s code. As long as the API remains stable, they can communicate.

The WordPress REST API specifies how other software should talk to WordPress and what WordPress will do in response to those requests. An application might use the API to ask WordPress to send a list of blog posts, and WordPress would respond with the blog posts in a format the app can understand.

You can see how useful this is for developers. Before the API, if a developer wanted to build a theme that displays blog posts, they’d have to write code that interfaces with WordPress’ internal code. With the REST API, any application that knows the API can interact with a WordPress site. A theme or front-end app, written in JavaScript or any other language, can ask a WordPress site for content or tell it to perform certain actions, like publishing a post.

The “REST” part of REST API denotes how the API should be interacted with — in the case of REST, requests are sent over HTTP. Each request is a specially formatted web address that WordPress knows how to respond to. REST implies other things too. If you want to deeper understanding of REST APIs, which are used all over the web, take a look at this [excellent video](http://www.restapitutorial.com/lessons/whatisrest.html) or check out this post from my company blog: [What Is An API?](https://blog.nexcess.net/2017/09/12/what-is-an-api/)

1.1k Views · [View Upvoters](https://www.quora.com/api/mobile_expanded_voter_list?key=90MfQ0PhiUZ&type=answer)

[Upvote**5**Downvote](https://www.quora.com/What-is-a-REST-API)

[Share](https://www.quora.com/What-is-a-REST-API)

[RecommendedAll](https://www.quora.com/What-is-a-REST-API)

[Amruta Phansalker](https://www.quora.com/profile/Amruta-Phansalker)

[David Karigithu](https://www.quora.com/profile/David-Karigithu)

[David Karigithu](https://www.quora.com/profile/David-Karigithu), Director Dhana House Limited, Director iSafari Limted

[Answered Sep 5, 2014](https://www.quora.com/What-is-a-REST-API/answer/David-Karigithu)

Originally Answered: [What are Rest APIs, and what are they used for?](https://www.quora.com/What-are-Rest-APIs-and-what-are-they-used-for?no_redirect=1)

They are basically data interfaces to an  application exposed using the the fundamental constructs of HTTP namely: the URL, Methods, Encoding, Formats & the Request/Response cycle as interfaces to the application server.    
  
The main interface is the URL i.e. [http://www.example.com/products?...](http://www.example.com/products?type=garden), could be used to access data pertaining to products filtered by type garden. This could easily be visualized as a banking hall with different lines for different purposes, and forms to handle the request to the teller.   
  
A standard data format is used to transfer data to and from clients, usually in JSON or XML. Clients can consume the service from browsers in JavaScript, mobile phones and other applications consuming the service.

5k Views · [View Upvoters](https://www.quora.com/api/mobile_expanded_voter_list?key=91ZjWDrHTZa&type=answer)

[Upvote**3**Downvote](https://www.quora.com/What-is-a-REST-API)

[Share](https://www.quora.com/What-is-a-REST-API)

[RecommendedAll](https://www.quora.com/What-is-a-REST-API)

[Amruta Phansalker](https://www.quora.com/profile/Amruta-Phansalker)

[Keshav Kashyap](https://www.quora.com/profile/Keshav-Kashyap-16)

[Keshav Kashyap](https://www.quora.com/profile/Keshav-Kashyap-16), worked at REST

[Answered Aug 7, 2016](https://www.quora.com/What-is-a-REST-API/answer/Keshav-Kashyap-16)

***REST API***

**REST: Representational State Transfer.**

* It’s an arrangement of functions on which the testers performs requests and receive responses. In REST API interactions are made via HTTP protocol.
* REST also permits communication between computers with each other over a network.
* For sending and receiving messages, it involves using HTTP methods, and it does not require a strict message definition, unlike Web services.
* REST messages often accepts the form either in form of XML, or JavaScript Object Notation (JSON).

2.3k Views · [View Upvoters](https://www.quora.com/api/mobile_expanded_voter_list?key=FQm4sNkaHQx&type=answer) · Answer requested by [Vimal Kumar](https://www.quora.com/profile/Vimal-Kumar-198)

[Upvote**10**Downvote](https://www.quora.com/What-is-a-REST-API)

[Share](https://www.quora.com/What-is-a-REST-API)

[RecommendedAll](https://www.quora.com/What-is-a-REST-API)

[Amruta Phansalker](https://www.quora.com/profile/Amruta-Phansalker)

[Bob Watson](https://www.quora.com/profile/Bob-Watson-2)

[Bob Watson](https://www.quora.com/profile/Bob-Watson-2), Has been writing and writing about APIs since before they were popular

[Answered May 11, 2016](https://www.quora.com/What-is-a-REST-API/answer/Bob-Watson-2)

Originally Answered: [What is RESTful API? What is the best way to explain it?](https://www.quora.com/What-is-RESTful-API-What-is-the-best-way-to-explain-it?no_redirect=1)

Let's break it into two components:

1) RESTful  
2) API

and start with the last one first.

An API is an interface through which one program or web site talks to another. They are used to share data and services, and they come in many different formats and types.

A RESTful API is one of the many possible ways that programs, servers, and web sites can share data and services. REST (Representational State Transfer) describes the general rules for how the data and services are represented through the API so that other programs will be able to correctly request and receive the data and services that an API makes available.

That's the 20,000-ft view. Diving down to get a closer look...

REST uses HTTP (the protocol used by a browser to get web pages from a server) to represent "resources." The most familiar resource for most people is a web page, but resources can be anything you can describe over the web. In addition to pages, you can use this to share data such as package delivery information, time information, purchasing information...the list is as long as you can imagine. The services that provide the REST API can act on these resources in whatever way makes sense. For example, a service could provide the ability to get information about them, create information about them or create the actual resource, delete them, modify them, to name the most common actions one can take on a web resource.

So, with the ability to create, read, update, and delete anything you can describe over the web, you can see how powerful they can be.

3.2k Views · [View Upvoters](https://www.quora.com/api/mobile_expanded_voter_list?key=qrU58ODs0hP&type=answer) · Answer requested by [Surya Kovur](https://www.quora.com/profile/Surya-Kovur)

[Upvote**11**Downvote](https://www.quora.com/What-is-a-REST-API)

[Share](https://www.quora.com/What-is-a-REST-API)

[RecommendedAll](https://www.quora.com/What-is-a-REST-API)

[Amruta Phansalker](https://www.quora.com/profile/Amruta-Phansalker)

[Keshav Kashyap](https://www.quora.com/profile/Keshav-Kashyap-2)

[Keshav Kashyap](https://www.quora.com/profile/Keshav-Kashyap-2), 2 Years of experience in API Automation

[Updated Jul 28, 2016](https://www.quora.com/What-is-a-REST-API/answer/Keshav-Kashyap-2)

Originally Answered: [What is RESTful API? What is the best way to explain it?](https://www.quora.com/What-is-RESTful-API-What-is-the-best-way-to-explain-it?no_redirect=1)

[**REST: Representational State Transfer**](http://testingalert.com/api-testing/api-manual/)**.**

It’s an arrangement of functions on which the testers performs requests and receive responses. API stands for Application Programming Interface. It is a software to software interface, not a user interface.

Above pic will explain how API works.

For detailed explain please visit, below mentioned link

[REST API Manual Test](http://testingalert.com/api-testing/api-manual/)

[REST API Automation](http://testingalert.com/mobile-testingandroidappium-automation/)

3.3k Views · [View Upvoters](https://www.quora.com/api/mobile_expanded_voter_list?key=8rDifzgvcxP&type=answer)

[Upvote**14**Downvote](https://www.quora.com/What-is-a-REST-API)

[Share](https://www.quora.com/What-is-a-REST-API)

[RecommendedAll](https://www.quora.com/What-is-a-REST-API)

[Amruta Phansalker](https://www.quora.com/profile/Amruta-Phansalker)

[Harshith Keni](https://www.quora.com/profile/Harshith-Keni)

[Harshith Keni](https://www.quora.com/profile/Harshith-Keni), Supergeek, maybe.

[Answered Mar 16, 2014](https://www.quora.com/What-is-a-REST-API/answer/Harshith-Keni)

Originally Answered: [What is REST API?](https://www.quora.com/What-is-REST-API-1?no_redirect=1)

REST - *Representational State Transfer* - is a convention based approach in web apps to define roles of various components. It works by applying some constraints on requests. HTTPS verbs (like GET, POST) and **overloaded** to add some more functionality to them.  
  
Your standard database **CRUD**(Create, Read, Update, Delete) is translated to the *HTTP verbs*: **POST**, **GET**, **PUT**, **DELETE**. Following that convention and passing on the right parameters basically means that you have translated some of the business logic onto HTTP. It's as simple as that.  
  
Now, a REST API, is a set of url routes available to perform some actions, like CRUD, on some entities in a system. Using some convention, like mentioned, along with a simple data transfer mechanism, say **JSON**or **XML**, and possibly some authentication mechanism, you can build a set of APIs to create, read, update and delete records **statelessly** in your system through a simple set of HTTP calls.  
  
Hope that helps.  
  
You can find more literature and examples here: [RESTful Resource Naming](http://www.restapitutorial.com/lessons/restfulresourcenaming.html" \t "_blank)

10.9k Views · [View Upvoters](https://www.quora.com/api/mobile_expanded_voter_list?key=VumGpqQUkA1&type=answer)

[Upvote**27**Downvote](https://www.quora.com/What-is-a-REST-API)

[Share](https://www.quora.com/What-is-a-REST-API)

[RecommendedAll](https://www.quora.com/What-is-a-REST-API)

[Amruta Phansalker](https://www.quora.com/profile/Amruta-Phansalker)

[Ahmetcan Türker](https://www.quora.com/profile/Ahmetcan-T%C3%BCrker)

[Ahmetcan Türker](https://www.quora.com/profile/Ahmetcan-T%C3%BCrker), studied at 50.yıl Tahran Lisesi

[Answered Oct 10, 2016](https://www.quora.com/What-is-a-REST-API/answer/Ahmetcan-T%C3%BCrker)

Originally Answered: [What does RESTful API mean?](https://www.quora.com/What-does-RESTful-API-mean?no_redirect=1)

REST is just a specification and ruleset. An API is called to be RESTful if it satisfies those rules. Simply

Your API needs to be stateless, in that it should not know the user state.

Your operations are Create, Read, Update and Delete based or httpwise post, get, put and delete (similarly).

There are url rules which you should follow.

Aside from what it is, you need to understand the motive in order to decide to use it or not. Or even effectively use it. There are pretty good stackoverflow answers that give you the motive, mostly under questions like soap vs rest. I suggest you to read those.

900 Views · [View Upvoters](https://www.quora.com/api/mobile_expanded_voter_list?key=X9otr9GNmuS&type=answer) · Answer requested by [Sameh Sayed](https://www.quora.com/profile/Sameh-Sayed-2)

[Upvote**4**Downvote](https://www.quora.com/What-is-a-REST-API)

[Share](https://www.quora.com/What-is-a-REST-API)

[RecommendedAll](https://www.quora.com/What-is-a-REST-API)

[Amruta Phansalker](https://www.quora.com/profile/Amruta-Phansalker)

[Lokesh Gupta](https://www.quora.com/profile/Lokesh-Gupta-47)

[Lokesh Gupta](https://www.quora.com/profile/Lokesh-Gupta-47), Extensive knowledge of developing REST APIs

[Answered Jun 5, 2016](https://www.quora.com/What-is-a-REST-API/answer/Lokesh-Gupta-47)

REST revolves around resource where every component is a resource and a resource is accessed by a common interface using HTTP standard methods. REST APIs use [Uniform Resource Identifiers](https://en.wikipedia.org/wiki/Uniform_Resource_Identifier) (URIs) to address resources. REST API designers should create URIs that convey a REST API’s resource model to its potential client developers.

REST defines 6 architectural constraints which make any web service – a true RESTful API.

1. Uniform interface
2. Client–server
3. Stateless
4. Cacheable
5. Layered system
6. Code on demand (optional)

Any single resource should not be too large and contain each and everything in it’s representation. Whenever relevant, a resource should contain **links (HATEOAS) pointing to relative URIs** to fetch related information.

Also, the resource representations across system should follow certain guidelines such as naming conventions, link formats or data format (xml or/and json).

And remember that REST === HTTP analogy is not correct until you do not stress to the fact that it "MUST" be [HATEOAS](http://restfulapi.net/hateoas/) driven.

Roy himself cleared it here: [http://roy.gbiv.com/untangled/20...](http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven)

A REST API should be entered with no prior knowledge beyond the initial URI (bookmark) and set of standardized media types that are appropriate for the intended audience (i.e., expected to be understood by any client that might use the API). From that point on, all application state transitions must be driven by client selection of server-provided choices that are present in the received representations or implied by the user’s manipulation of those representations. The transitions may be determined (or limited by) the client’s knowledge of media types and resource communication mechanisms, both of which may be improved on-the-fly (e.g., code-on-demand).

[Failure here implies that out-of-band information is driving interaction instead of hypertext.]

4.7k Views

[UpvoteDownvote](https://www.quora.com/What-is-a-REST-API)

[Share](https://www.quora.com/What-is-a-REST-API)

[RecommendedAll](https://www.quora.com/What-is-a-REST-API)

[Amruta Phansalker](https://www.quora.com/profile/Amruta-Phansalker)

[Robert Arles](https://www.quora.com/profile/Robert-Arles)

[Robert Arles](https://www.quora.com/profile/Robert-Arles), SDET, Software Development Engineer in Test

[Answered Apr 9, 2016](https://www.quora.com/What-is-a-REST-API/answer/Robert-Arles)

Originally Answered: [What is a REST API in plain English?](https://www.quora.com/What-is-a-REST-API-in-plain-English?no_redirect=1)

A REST API is a set of web addresses that respond with pure information, not a formatted web page. An API would be custom for it's use, but made up of web addresses structured very logically and might be so well designed that you could guess at how to request information, see below for simple examples. That structure would follow the REST design concepts. Developers use APIs as a way to retrieve and store information. An HR web app could display employee information, and allow changes to employee information. Hidden behind that web app, a REST API could be in use.

Two simple, theoretical,  REST API examples:

[http://example.com/api/employee/...](http://example.com/api/employee/update) *(some hidden data would be passed describing the change wanted and the employee ID to apply the update)*

[***http://example.com/api/employee/1***](http://example.com/api/employee/1)

*might respond with info for employee number 1:*

*Joe Doe, Manager of Accounting, Salaried, Male, blah blah blah*

The format of the data returned wouldn't look like that, because developers need it in a format that a programming language can predictably decipher. JSON is a common format. You'll see all of the above info surrounded with quotation marks, {}, [] and descriptive titles for each bit of info.

1.6k Views · [View Upvoters](https://www.quora.com/api/mobile_expanded_voter_list?key=Xo6YGVRxfRd&type=answer)

[Upvote**1**Downvote](https://www.quora.com/What-is-a-REST-API)

[Share](https://www.quora.com/What-is-a-REST-API)

[RecommendedAll](https://www.quora.com/What-is-a-REST-API)

[Amruta Phansalker](https://www.quora.com/profile/Amruta-Phansalker)

[Nani Kaufmann](https://www.quora.com/profile/Nani-Kaufmann)

[Nani Kaufmann](https://www.quora.com/profile/Nani-Kaufmann), Co-Founder at Be Brave - Creative Sales Solutions (2018-present)

[Answered Jun 12, 2017](https://www.quora.com/What-is-a-REST-API/answer/Nani-Kaufmann)

Wow! Good question! I just wrote about this at Medium, so if you’d like to read the full post, [**click here**](https://medium.com/@sheetsu.com/rest-api-for-dummies-the-store-metaphor-c9e531bcd425).

**REST API for Dummies: “The Store Metaphor”**

In order to be able to give you a REST API definition, let’s go shopping. Let’s say you go to your favorite store to **buy a new pair of jeans**.

You look around and find **exactly what you were looking for! But in the wrong size**: It’s small while you need medium. But, unless you are a shop lifter, **you do not have access to that mysterious room in the back of the store** that holds all items they sell, at all sizes and colors.

To get access to this wonderland, you need to **make a request** to the **sales person** so you **receive back** exactly what you need: those pair of jeans you loved, but the right size for you.

Since the**sales person has this special access**, he goes inside, gets exactly what you need, **comes back and delivers what you requested**.

Yes! It’s done!

In this metaphor, **the sales person is the messenger,** right? It took a request you made, used the access he had in order to give you what you wanted or at least worked to give you the answer to your question.

**Now get ready for this: the sales person is the API!!**

Whaaat? Yes!

**An API is the messenger that takes a request, tells a system what you want to do and then returns the response back to you!**

It looks something like this:

… Did I just blow your mind? Let’s get a big deeper than…

**Let’s say I gained a few pounds.**

You need [**the best real-time update**](http://sheetsu.com/) you can imagine, right? I tried the pants, size medium, but they didn’t fit. At this stage you just want to get the right pants size, pay and leave the store to move on with your life and not waste any more time!

It would be amazing if we could just**type what we needed and the pants you were holding in your hands would magically become your size,** and that whereSheetsucomes in.

[**Sheetsu**](http://sheetsu.com/)**allows you to create a REST API from a Google Spreadsheet.** In other words: it connects whatever data you need to a simple Google Spreadsheet in seconds.

Paste the link from your Google Spreadsheet into **[Sheetsu](http://sheetsu.com/" \t "_blank)**and you will get your REST API link. You can update the content in your Sheet as many times as you want and it will work in real-time. I would look a bit like this:

PS: I actually explained what is an **API.** So if you ever asks a developer to show you an a REST API, that’s all you will see: **links**. **REST** is the way you organize these links, like it was mentioned before.

Hope this metaphor helped someone out there :)

12.7k Views · [View Upvoters](https://www.quora.com/api/mobile_expanded_voter_list?key=bFGrhgY6Y7m&type=answer)

[Upvote**57**Downvote](https://www.quora.com/What-is-a-REST-API)

[Share](https://www.quora.com/What-is-a-REST-API)

[RecommendedAll](https://www.quora.com/What-is-a-REST-API)

[Amruta Phansalker](https://www.quora.com/profile/Amruta-Phansalker)

[Aurel Kurtula](https://www.quora.com/profile/Aurel-Kurtula-1)

[Aurel Kurtula](https://www.quora.com/profile/Aurel-Kurtula-1), I like learning

[Answered Apr 9, 2016](https://www.quora.com/What-is-a-REST-API/answer/Aurel-Kurtula-1)

Originally Answered: [What is a REST API in plain English?](https://www.quora.com/What-is-a-REST-API-in-plain-English?no_redirect=1)

REST stands for **re**presentational **s**tate **t**ransfer. It is a style which you are recommended to follow, especially if you want it to be public, hence other developers to be on the same page.

API stands For application interface. It allows apps to talk to each other.

Every time you go on a web page, you are making an HTTP request. When ever navigate the web normally, you are making HTTP request using the GET method. In human terms, when you go to "[google.com](http://google.com/)" you want to get the google search page.

Now as a developer, using the REST architecture, you know that the way to implement a CRUD API  (API to **C**reate, **R**ead, **U**pdate, and **D**elete data) is to use the following methods to perform the following actions

GET - to read content from server  
PUT - update content in the server  
DELETE - delete content from the server

Back to the meaning of REST. There is nothing stopping you from using the GET method to delete content. But it would be confusing to other developers, hence using the REST architecture is better.

You can see how useful this is for developers. Before the API, if a developer wanted to build a theme that displays blog posts, they’d have to write code that interfaces with WordPress’ internal code. With the REST API, any application that knows the API can interact with a WordPress site. A theme or front-end app, written in JavaScript or any other language, can ask a WordPress site for content or tell it to perform certain actions, like publishing a post.

**Rule 1: Offer access through *resources*.**

In typical computer systems, clients and servers exchange *commands*: do this, do that. Suppose we want to model a to-do list in a non-REST way. In a technical language, that might look like this:

***NOT REST***

* /changeTodoList.php?item=35&action=changeTitle&title=new\_title

Note how this is indeed an instruction: change something.  
But a “changeTodoList” is not a thing, it's not a resource.

In the REST architectural style, servers only offer resources. Resources are conceptual things about which clients and servers communicate.  
***REST***

* /todolists/7/items/35/

This above thing is not a command, it is the address of a resource, a thing.  
You can then use this address to manipulate the to-do list using standard operations, instead of interface-specific commands.

**Rule 2: Represent resources by *representations*.**

A resource is a thing—and we can describe those things in different formats. For instance, humans might want to see an [HTML](http://en.wikipedia.org/wiki/HTML) version, which your browser transforms into a readable layout. But sometimes, interfaces on the Web are used by machines, too. They need a different format, such as [JSON](http://en.wikipedia.org/wiki/JSON).

In a non-REST way, different formats have different addresses:  
***NOT REST***

* **browser:**/showTodoList.php?format=html
* **application:** /showTodoList.php?format=json

The problem is then that systems using different formats cannot communicate with each other, because they use different addresses for the same things!

In a REST system, addresses identify *things*, not formats, so all systems use the same address for the same thing. How can they get different formats then? They explicitly ask for it!  
***REST***

* **browser:**“I want /todolists/7/, please give me HTML.”
* **application:**“I want /todolists/7/, please give me JSON.”

The technique that enables this is called [content negotiation](http://en.wikipedia.org/wiki/Content_negotiation).

**Rule 3: Exchange *self-descriptive* messages.**

In a REST system, we should be able to interpret any message without having seen the previous one.

Imagine the following conversation:  
***NOT REST***

1. /search-results?q=todo
2. /search-results?page=2
3. /search-results?page=3

The first request gets search results for “todo”; the second request gets the second page of that. Now imagine that you only see the second request. How would you know as a server what to do?

In REST, each message stands on its own:  
***REST***

1. /search-results?q=to-**do**
2. /search-results?q=todo&page=2
3. /search-results?q=todo&page=3

Note how each request can be interpreted by itself.  
Another aspect of this, is that REST clients and servers only use standard operations, which are defined in a *specification*. For the Web, this specification is called [HTTP](http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol).

**Rule 4: Connect resources through *links*.**

How can you navigate a website you've never seen before? You use links!  
You don't have to manually edit the address bar in your browser every time you go to a new page.

In machine interfaces, this is not always the case. Suppose an application asks for your to-do list. It might receive the following representation:  
***NOT REST***

* /todolists/7/

1. {
2. "name": "My to-dos",
3. "items": [35, 36]
4. }

Now how can you get the items of the list?  
Good question! We'd have to read the documentation for that.

In REST, resources connect to each other through hyperlinks:  
***REST***

* /todolists/7/

1. {
2. "name": "My to-dos",
3. "items": ["/todolists/7/items/35/", "/todolists/7/items/36/"]
4. }

Note how you don't have to read the manual to know how you can retrieve the items of your list. You just follow the links.

**APIs that follow those rules are REST APIs**

**Follow those 4 rules in your interface, and you have a REST API.**

**The six constraints are: (click the constraint to read more)**

* [Uniform Interface](http://www.restapitutorial.com/lessons/whatisrest.html)

The uniform interface constraint defines the interface between clients and servers. It simplifies and decouples the architecture, which enables each part to evolve independently. The four guiding principles of the uniform interface are:

**Resource-Based**

Individual resources are identified in requests using URIs as resource identifiers. The resources themselves are conceptually separate from the representations that are returned to the client. For example, the server does not send its database, but rather, some HTML, XML or JSON that represents some database records expressed, for instance, in Finnish and encoded in UTF-8, depending on the details of the request and the server implementation.

**Manipulation of Resources Through Representations**

When a client holds a representation of a resource, including any metadata attached, it has enough information to modify or delete the resource on the server, provided it has permission to do so.

**Self-descriptive Messages**

Each message includes enough information to describe how to process the message. For example, which parser to invoke may be specified by an Internet media type (previously known as a MIME type). Responses also explicitly indicate their cache-ability.

**Hypermedia as the Engine of Application State (HATEOAS)**

Clients deliver state via body contents, query-string parameters, request headers and the requested URI (the resource name). Services deliver state to clients via body content, response codes, and response headers. This is technically referred-to as hypermedia (or hyperlinks within hypertext).

* [Stateless](http://www.restapitutorial.com/lessons/whatisrest.html)

As REST is an acronym for REpresentational State Transfer, statelessness is key. Essentially, what this means is that the necessary state to handle the request is contained within the request itself, whether as part of the URI, query-string parameters, body, or headers. The URI uniquely identifies the resource and the body contains the state (or state change) of that resource. Then after the server does it's processing, the appropriate state, or the piece(s) of state that matter, are communicated back to the client via headers, status and response body.

Most of us who have been in the industry for a while are accustomed to programming within a container which provides us with the concept of “session” which maintains state across multiple HTTP requests. In REST, the client must include all information for the server to fulfill the request, resending state as necessary if that state must span multiple requests. Statelessness enables greater scalability since the server does not have to maintain, update or communicate that session state. Additionally, load balancers don't have to worry about session affinity for stateless systems.

So what's the difference between state and a resource? State, or application state, is that which the server cares about to fulfill a request—data necessary for the current session or request. A resource, or resource state, is the data that defines the resource representation—the data stored in the database, for instance. Consider application state to be data that could vary by client, and per request. Resource state, on the other hand, is constant across every client who requests it.

Ever had back-button issues with a web application where it went AWOL at a certain point because it expected you to do things in a certain order? That's because it violated the statelessness principle. There are cases that don't honor the statelessness principle, such as three-legged OAuth, API call rate limiting, etc. However, make every effort to ensure that application state doesn't span multiple requests of your service(s).

* [Cacheable](http://www.restapitutorial.com/lessons/whatisrest.html)

As on the World Wide Web, clients can cache responses. Responses must therefore, implicitly or explicitly, define themselves as cacheable, or not, to prevent clients reusing stale or inappropriate data in response to further requests. Well-managed caching partially or completely eliminates some client–server interactions, further improving scalability and performance.

* [Client-Server](http://www.restapitutorial.com/lessons/whatisrest.html)

The uniform interface separates clients from servers. This separation of concerns means that, for example, clients are not concerned with data storage, which remains internal to each server, so that the portability of client code is improved. Servers are not concerned with the user interface or user state, so that servers can be simpler and more scalable. Servers and clients may also be replaced and developed independently, as long as the interface is not altered.

* [Layered System](http://www.restapitutorial.com/lessons/whatisrest.html)

A client cannot ordinarily tell whether it is connected directly to the end server, or to an intermediary along the way. Intermediary servers may improve system scalability by enabling load-balancing and by providing shared caches. Layers may also enforce security policies.

* [Code on Demand (optional)](http://www.restapitutorial.com/lessons/whatisrest.html)

Servers are able to temporarily extend or customize the functionality of a client by transferring logic to it that it can execute. Examples of this may include compiled components such as Java applets and client-side scripts such as JavaScript.

Complying with these constraints, and thus conforming to the REST architectural style, will enable any kind of distributed hypermedia system to have desirable emergent properties, such as performance, scalability, simplicity, modifiability, visibility, portability and reliability.

**NOTE:** The only optional constraint of REST architecture is code on demand. If a service violates any other constraint, it cannot strictly be referred to as RESTful.

# HTTP Methods

RESTful APIs enable you to develop any kind of web application having all possible CRUD (create, retrieve, update, delete) operations. REST guidelines suggest you to use specific HTTP method on specific type of call made to server (though technically it is possible to violate this guideline, yet it is highly discouraged).

Use below given information to find suitable HTTP method for the action performed by API.

Table of Contents

[HTTP GET](https://restfulapi.net/http-methods/#get)

[HTTP POST](https://restfulapi.net/http-methods/#post)

[HTTP PUT](https://restfulapi.net/http-methods/#put)

[HTTP DELETE](https://restfulapi.net/http-methods/#delete)

[HTTP PATCH](https://restfulapi.net/http-methods/#patch)

[Summary](https://restfulapi.net/http-methods/#summary)

[Glossary](https://restfulapi.net/http-methods/#glossary)

## HTTP GET

Use GET requests **to retrieve resource representation/information only** – and not to modify it in any way. As GET requests do not change the state of resource, these are said to be **safe methods**. Additionally, GET APIs should be **idempotent**, which means that making multiple identical requests must produce same result everytime until another API (POST or PUT) has changed the state of resource on server.

If the Request-URI refers to a data-producing process, it is the produced data which shall be returned as the entity in the response and not the source text of the process, unless that text happens to be the output of the process.

For any given HTTP GET API, if resource is found on server then it must return HTTP response code 200 (OK) – along with response body which is usually either XML of JSON content (due to their platform independent nature).

In case resource is NOT found on server then it must return HTTP response code 404 (NOT FOUND). Similarly, if it is determined that GET request itself is not correctly formed then server will return HTTP response code 400 (BAD REQUEST).

#### Example request URIs

* HTTP GET http://www.appdomain.com/users
* HTTP GET http://www.appdomain.com/users?size=20&page=5
* HTTP GET http://www.appdomain.com/users/123
* HTTP GET http://www.appdomain.com/users/123/address

## HTTP POST

Use POST APIs **to create new subordinate resources**, e.g. a file is subordinate to a directory containing it or a row is subordinate to a database table. Talking strictly in terms of REST, POST methods are used to create a new resource into the collection of resources.

Ideally, if a resource has been created on the origin server, the response SHOULD be HTTP response code 201 (Created) and contain an entity which describes the status of the request and refers to the new resource, and a [Location](https://en.wikipedia.org/wiki/HTTP_location) header.

Many times, the action performed by the POST method might not result in a resource that can be identified by a URI. In this case, either HTTP response code 200 (OK) or 204 (No Content) is the appropriate response status.

Responses to this method are **not cacheable**, unless the response includes appropriate [Cache-Control](https://en.wikipedia.org/wiki/Web_cache#Cache_control) or [Expires](https://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html) header fields.

Please note that POST is **neither safe nor idempotent** and invoking two identical POST requests will result in two different resources containing the same information (except resource ids).

#### Example request URIs

* HTTP POST http://www.appdomain.com/users
* HTTP POST http://www.appdomain.com/users/123/accounts

## HTTP PUT

Use PUT APIs primarily **to update existing resource** (if resource does not exist then API may decide to create a new resource or not). If a new resource has been created by the PUT API, the origin server MUST inform the user agent via the HTTP response code 201 (Created) response and if an existing resource is modified, either the 200 (OK) or 204 (No Content) response codes SHOULD be sent to indicate successful completion of the request.

If the request passes through a cache and the Request-URI identifies one or more currently cached entities, those entries SHOULD be treated as stale. Responses to this method are **not cacheable**.

*The difference between the POST and PUT APIs can be observed in request URIs. POST requests are made of resource collections whereas PUT requests are made on individual resource.*

#### Example request URIs

* HTTP PUT http://www.appdomain.com/users/123
* HTTP PUT http://www.appdomain.com/users/123/accounts/456

## HTTP DELETE

As the name applies, DELETE APIs are used **to delete resources** (identified by the Request-URI).

A successful response of DELETE requests SHOULD be HTTP response code 200 (OK) if the response includes an entity describing the status, 202 (Accepted) if the action has been queued, or 204 (No Content) if the action has been performed but the response does not include an entity.

DELETE operations are **idempotent**. If you DELETE a resource, it’s removed from collection of resource. Repeatedly calling DELETE API on that resource will not change the outcome – however calling DELETE on a resource a second time will return a 404 (NOT FOUND) since it was already removed. Some may argue that it makes DELETE method non-idempotent. It’s matter of discussion and personal opinion.

If the request passes through a cache and the Request-URI identifies one or more currently cached entities, those entries SHOULD be treated as stale. Responses to this method are **not cacheable**.

#### Example request URIs

* HTTP DELETE http://www.appdomain.com/users/123
* HTTP DELETE http://www.appdomain.com/users/123/accounts/456

## HTTP PATCH

HTTP PATCH requests are **to make partial update on a resource**. If you see PUT requests also modify a resource entity so to make more clear – PATCH method is the correct choice for partially updating an existing resource and PUT should only be used if you’re replacing a resource in it’s entirety.

Please note that there are some challenges if you decide to use PATCH APIs in your application:

* Support for PATCH in browsers, servers and web application frameworks is not universal. IE8, PHP, Tomcat, django, and lots of other software has missing or broken support for it.
* Request payload of PATCH request in not straightforward as it is for PUT request. e.g.

HTTP GET /users/1

produces below response:

{id: 1, username: 'admin', email: 'email@example.org'}

A sample patch request to update the email will be like this:

HTTP PATCH /users/1

[  
{ “op”: “replace”, “path”: “/email”, “value”: “new.email@example.org” }  
]

There may be following possible operations are per HTTP specification.

[  
{ "op": "test", "path": "/a/b/c", "value": "foo" },  
{ "op": "remove", "path": "/a/b/c" },  
{ "op": "add", "path": "/a/b/c", "value": [ "foo", "bar" ] },  
{ "op": "replace", "path": "/a/b/c", "value": 42 },  
{ "op": "move", "from": "/a/b/c", "path": "/a/b/d" },  
{ "op": "copy", "from": "/a/b/d", "path": "/a/b/e" }  
]

PATCH method is not a replacement for the POST or PUT methods. It applies a delta (diff) rather than replacing the entire resource.

## Summary of HTTP Methods for RESTful APIs

Below table summarises the use of HTTP methods discussed above.

| **HTTP METHOD** | **CRUD** | **ENTIRE COLLECTION (E.G. /USERS)** | **SPECIFIC ITEM (E.G. /USERS/123)** |
| --- | --- | --- | --- |
| POST | Create | 201 (Created), ‘Location’ header with link to /users/{id} containing new ID. | Avoid using POST on single resource |
| GET | Read | 200 (OK), list of users. Use pagination, sorting and filtering to navigate big lists. | 200 (OK), single user. 404 (Not Found), if ID not found or invalid. |
| PUT | Update/Replace | 404 (Not Found), unless you want to update every resource in the entire collection of resource. | 200 (OK) or 204 (No Content). Use 404 (Not Found), if ID not found or invalid. |
| PATCH | Partial Update/Modify | 404 (Not Found), unless you want to modify the collection itself. | 200 (OK) or 204 (No Content). Use 404 (Not Found), if ID not found or invalid. |
| DELETE | Delete | 404 (Not Found), unless you want to delete the whole collection — use with caution. | 200 (OK). 404 (Not Found), if ID not found or invalid. |
| Markup language |  |  |  |

In computer text processing, a **markup language** is a system for [annotating](https://en.wikipedia.org/wiki/Annotation) a [document](https://en.wikipedia.org/wiki/Document) in a way that is [syntactically distinguishable](https://en.wikipedia.org/wiki/Syntax_(logic)) from the text.[[1]](https://en.wikipedia.org/wiki/Markup_language#cite_note-1) The idea and terminology evolved from the "marking up" of paper manuscripts, i.e., the revision instructions by editors, traditionally written with a [blue pencil](https://en.wikipedia.org/wiki/Blue_pencil_(editing)) on authors' [manuscripts](https://en.wikipedia.org/wiki/Manuscript).[[*citation needed*](https://en.wikipedia.org/wiki/Wikipedia:Citation_needed)] In [digital media](https://en.wikipedia.org/wiki/Digital_media), this "blue pencil instruction text" was replaced by [tags](https://en.wikipedia.org/wiki/HTML_element), that is, instructions are expressed directly by tags or "instruction text encapsulated by tags."

Examples include typesetting instructions such as those found in [troff](https://en.wikipedia.org/wiki/Troff" \o "Troff), [TeX](https://en.wikipedia.org/wiki/TeX" \o "TeX) and [LaTeX](https://en.wikipedia.org/wiki/LaTeX" \o "LaTeX), or structural markers such as [XML](https://en.wikipedia.org/wiki/Extensible_Markup_Language) tags. Markup instructs the software that displays the text to carry out appropriate actions, but is omitted from the version of the text that users see.

Some markup languages, such as the widely used [HTML](https://en.wikipedia.org/wiki/HTML), have pre-defined [presentation semantics](https://en.wikipedia.org/wiki/Presentation_semantics)—meaning that their specification prescribes how to present the [structured data](https://en.wikipedia.org/wiki/Structured_data). Others, such as XML, do not have them and are general purpose.

[HyperText](https://en.wikipedia.org/wiki/Hypertext) Markup Language (HTML), one of the document formats of the [World Wide Web](https://en.wikipedia.org/wiki/World_Wide_Web), is an instance of [Standard Generalized Markup Language](https://en.wikipedia.org/wiki/Standard_Generalized_Markup_Language) or SGML, and follows many of the markup conventions used in the publishing industry in the communication of printed work between authors, editors, and printers.

**Wikitext**, also known as *Wiki markup* or *Wikicode*, consists of the syntax and keywords used by the [MediaWiki](https://en.wikipedia.org/wiki/MediaWiki" \o "MediaWiki) software to format a page. To learn how to see this markup, and to save an edit, see: [Help:Editing](https://en.wikipedia.org/wiki/Help:Editing" \o "Help:Editing). Generally, coding can be copied and pasted, without writing new code. There is a short list of markup and tips at [Help:Cheatsheet](https://en.wikipedia.org/wiki/Help:Cheatsheet" \o "Help:Cheatsheet).

## What is HTML?

HTML is the standard markup language for creating Web pages.

* HTML stands for Hyper Text Markup Language
* HTML describes the structure of Web pages using markup
* HTML elements are the building blocks of HTML pages
* HTML elements are represented by tags
* HTML tags label pieces of content such as "heading", "paragraph", "table", and so on
* Browsers do not display the HTML tags, but use them to render the content of the page

## A Simple HTML Document

### Example

<!DOCTYPE html>  
<html>  
<head>  
<title>Page Title</title>  
</head>  
<body>  
  
<h1>My First Heading</h1>  
<p>My first paragraph.</p>  
  
</body>  
</html>

# What is HyperText

Hypertext is text which is not constrained to be linear.

Hypertext is text which contains [links](https://www.w3.org/Terms.html" \l "link) to other texts. The term was coined by [Ted Nelson](https://www.w3.org/Xanadu.html" \l "Nelson) around 1965 (see [History](https://www.w3.org/History.html) ).

HyperMedia is a term used for hypertext which is not constrained to be text: it can include graphics, video and [sound](https://www.w3.org/Talks/YesWeCan.snd) , for example. Apparently Ted Nelson was the first to use this term too.

Hypertext and HyperMedia are concepts, not products.

XML

Extensible Markup Language (XML) is used to describe [data](https://searchdatamanagement.techtarget.com/definition/data). The XML standard is a flexible way to create information formats and electronically share structured data via the public [Internet](https://searchwindevelopment.techtarget.com/definition/Internet), as well as via corporate [networks](https://searchnetworking.techtarget.com/definition/network).

XML [code](https://whatis.techtarget.com/definition/code), a formal recommendation from the [World Wide Web Consortium](https://searchmicroservices.techtarget.com/definition/W3C-World-Wide-Web-Consortium) (W3C), is similar to [Hypertext Markup Language](https://searchmicroservices.techtarget.com/definition/HTML-Hypertext-Markup-Language) (HTML). Both XML and HTML contain [markup symbols](https://searchmicroservices.techtarget.com/definition/markup) to describe page or file contents. HTML code describes Web page content (mainly text and graphic images) only in terms of how it is to be displayed and interacted with.

XML data is known as self-describing or self-defining, meaning that the structure of the data is embedded with the data, thus when the data arrives there is no need to pre-build the structure to store the data; it is dynamically understood within the XML. The XML format can be used by any individual or group of individuals or companies that want to share information in a consistent way. XML is actually a simpler and easier-to-use subset of the [Standard Generalized Markup Language](https://searchmicroservices.techtarget.com/definition/SGML-Standard-Generalized-Markup-Language) (SGML), which is the standard to create a document structure.

The basic building block of an XML document is an element, defined by[tags](https://searchmicroservices.techtarget.com/definition/tag). An element has a beginning and an ending tag. All elements in an XML document are contained in an outermost element known as the root element. XML can also support [nested](https://whatis.techtarget.com/definition/nested) elements, or elements within elements. This ability allows XML to support hierarchical structures. Element names describe the content of the element, and the structure describes the relationship between the elements.

# What is the difference between XML and HTML?

[0](http://www.xmlobjective.com/what-is-the-difference-between-xml-and-html/#respond)

By [admin](http://www.xmlobjective.com/author/admin/)

July 11th, 2013

* XML is the acronym from Extensible Markup Language (meta-language of noting/marking). XML is a resembling language with HTML. It was developed for describing data.
* The XML tags are not pre-defined in XML. You will have to create tags according to your needs.
* XML is self descriptive.
* XML uses DDT principle (Defining the Document Type) to formally describe the data.
* The main difference between XML and HTML: XML is not a substitute for HTML.

### XML and HTML were developed with different purposes:

* XML was developed to describe data and to focalize on what the data represent.
* HTML was developed to display data about to focalize on the way that data looks.
* HTML is about displaying data, XML is about describing information.
* XML is extensible.

The tags used to mark the documents and the structures of documents in HTML are pre-defined. The author of HTML documents can use only tags that were previously defined in HTML. The Standard XML gives you the possibility to define personal structures and tags.

### XML is a complement of the HTML language

It is important to understand that XML is not a substitute for HTML. In the future development of the Web, XML will be the main language to describe the structure and the Web data, and the HTML language will be responsible for displaying the data.

### XML in the future development of web

We participated to the evolution of XML since its appearance. It is amazing to observe its rapid evolution, and how fast it was adopted by the majority of software developers. We strongly believe that XML will become as important as HTML for the future web evolution, especially when it comes to data manipulation.

### How can you use XML?

* XML can store data separately from HTML.
* XML can be used to store data inside the HTML documents.
* XML can be used as a format for exchanging information.
* XML can be used to store data in files and databases.

The HTML pages are used to display data. The data are sometimes stored in the interior of HTML pages. Using XML, you can store data in a separated file. This way, you can easily concentrate on using HTML for formatting and displaying, and you can also be certain that the modifications won’t bring any modifications to any HTML code.

XML can also store data inside the HTML documents. The XML data will store data in HTML documents as “data islands.” You can concentrate on HTML to format and display data.

In the real world, the calculation systems and the databases are containing data in incompatible formats. One of the most soliciting provocations for developers was exchanging data between incompatible systems over the internet. Converting XML data can reduce the complexity, and it can also create data that are easy to read by any kind of application.

XML can be used to store data in files or databases. You can write applications to store and recover information from the hard disks, and you can write generic application to display certain types of data.

# Tuple

In [mathematics](https://en.wikipedia.org/wiki/Mathematics), a **tuple** is a finite ordered list (sequence) of [elements](https://en.wikipedia.org/wiki/Element_(mathematics)). An ***n*-tuple** is a [sequence](https://en.wikipedia.org/wiki/Sequence) (or ordered list) of *n* elements, where *n*is a non-negative [integer](https://en.wikipedia.org/wiki/Integer). There is only one 0-tuple, an empty sequence, or empty tuple, as it is referred to. An *n*-tuple is [defined inductively](https://en.wikipedia.org/wiki/Recursive_definition) using the construction of an [ordered pair](https://en.wikipedia.org/wiki/Ordered_pair).

Mapreduce

## What is MapReduce?

MapReduce™ is the heart of [Apache™ Hadoop®](https://www.ibm.com/analytics/hadoop). It is this programming paradigm that allows for massive scalability across hundreds or thousands of servers in a Hadoop cluster. The MapReduce concept is fairly simple to understand for those who are familiar with clustered scale-out data processing solutions.

For people new to this topic, it can be somewhat difficult to grasp, because it’s not typically something people have been exposed to previously. If you’re new to Hadoop’s MapReduce jobs, don’t worry: we’re going to describe it in a way that gets you up to speed quickly.

The term MapReduce actually refers to two separate and distinct tasks that Hadoop programs perform. The first is the map job, which takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs).

The reduce job takes the output from a map as input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce job is always performed after the map job.

## An example of MapReduce

Let’s look at a simple example. Assume you have five files, and each file contains two columns (a key and a value in Hadoop terms) that represent a city and the corresponding temperature recorded in that city for the various measurement days. Of course we’ve made this example very simple so it’s easy to follow. You can imagine that a real application won’t be quite so simple, as it’s likely to contain millions or even billions of rows, and they might not be neatly formatted rows at all; in fact, no matter how big or small the amount of data you need to analyze, the key principles we’re covering here remain the same. Either way, in this example, city is the key and temperature is the value.  
  
Toronto, 20  
Whitby, 25  
New York, 22  
Rome, 32  
Toronto, 4  
Rome, 33  
New York, 18  
  
Out of all the data we have collected, we want to find the maximum temperature for each city across all of the data files (note that each file might have the same city represented multiple times). Using the MapReduce framework, we can break this down into five map tasks, where each mapper works on one of the five files and the mapper task goes through the data and returns the maximum temperature for each city. For example, the results produced from one mapper task for the data above would look like this:  
  
(Toronto, 20) (Whitby, 25) (New York, 22) (Rome, 33)  
  
Let’s assume the other four mapper tasks (working on the other four files not shown here) produced the following intermediate results:  
  
(Toronto, 18) (Whitby, 27) (New York, 32) (Rome, 37)(Toronto, 32) (Whitby, 20) (New York, 33) (Rome, 38)(Toronto, 22) (Whitby, 19) (New York, 20) (Rome, 31)(Toronto, 31) (Whitby, 22) (New York, 19) (Rome, 30)  
  
All five of these output streams would be fed into the reduce tasks, which combine the input results and output a single value for each city, producing a final result set as follows:  
  
(Toronto, 32) (Whitby, 27) (New York, 33) (Rome, 38)  
  
As an analogy, you can think of map and reduce tasks as the way a census was conducted in Roman times, where the census bureau would dispatch its people to each city in the empire. Each census taker in each city would be tasked to count the number of people in that city and then return their results to the capital city.  
  
There, the results from each city would be reduced to a single count (sum of all cities) to determine the overall population of the empire. This mapping of people to cities, in parallel, and then combining the results (reducing) is much more efficient than sending a single person to count every person in the empire in a serial fashion.

Hadoop

When you learn about Big Data you will sooner or later come across this odd sounding word: Hadoop - but what exactly is it?

Put simply, Hadoop can be thought of as a set of open source programs and procedures (meaning essentially they are free for anyone to use or modify, with a few exceptions) which anyone can use as the "backbone" of their big data operations.

I'll try to keep things simple as I know a lot of people reading this aren't software engineers, so I hope I don't over-simplify anything - think of this as a brief guide for someone who wants to know a bit more about the nuts and bolts that make big data analysis possible.

**The 4 Modules of Hadoop**

Hadoop is made up of "modules", each of which carries out a particular task essential for a computer system designed for big data analytics.

**1. Distributed File-System**

The most important two are the Distributed File System, which allows data to be stored in an easily accessible format, across a large number of linked storage devices, and the MapReduce - which provides the basic tools for poking around in the data.

(A "file system" is the method used by a computer to store data, so it can be found and used. Normally this is determined by the computer's operating system, however a Hadoop system uses its own file system which sits "above" the file system of the host computer - meaning it can be accessed using any computer running any supported OS).

**2. MapReduce**

MapReduce is named after the two basic operations this module carries out - reading data from the database, putting it into a format suitable for analysis (map), and performing mathematical operations i.e counting the number of males aged 30+ in a customer database (reduce).

**3. Hadoop Common**

The other module is Hadoop Common, which provides the tools (in Java) needed for the user's computer systems (Windows, Unix or whatever) to read data stored under the Hadoop file system.

**4. YARN**

The final module is YARN, which manages resources of the systems storing the data and running the analysis.

Various other procedures, libraries or features have come to be considered part of the Hadoop "framework" over recent years, but Hadoop Distributed File System, Hadoop MapReduce, Hadoop Common and Hadoop YARN are the principle four.

**How Hadoop Came About**

Development of Hadoop began when forward-thinking software engineers realised that it was quickly becoming useful for anybody to be able to store and analyze datasets far larger than can practically be stored and accessed on one physical storage device (such as a hard disk).

This is partly because as physical storage devices become bigger it takes longer for the component that reads the data from the disk (which in a hard disk, would be the "head") to move to a specified segment. Instead, many smaller devices working in parallel are more efficient than one large one.

It was released in 2005 by the Apache Software Foundation, a non-profit organization which produces open source software which powers much of the Internet behind the scenes. And if you're wondering where the odd name came from, it was the name given to a toy elephant belonging to the son of one of the original creators!

**The Usage of Hadoop**

The flexible nature of a Hadoop system means companies can add to or modify their data system as their needs change, using cheap and readily-available parts from any IT vendor.

Today, it is the most widely used system for providing data storage and processing across "commodity" hardware - relatively inexpensive, off-the-shelf systems linked together, as opposed to expensive, bespoke systems custom-made for the job in hand. In fact it is claimed that more than half of the companies in the Fortune 500 make use of it.

Just about all of the big online names use it, and as anyone is free to alter it for their own purposes, modifications made to the software by expert engineers at, for example, Amazon and Google, are fed back to the development community, where they are often used to improve the "official" product. This form of collaborative development between volunteer and commercial users is a key feature of open source software.

In its "raw" state - using the basic modules supplied here http://hadoop.apache.org/ by Apache, it can be very complex, even for IT professionals - which is why various commercial versions have been developed such as Cloudera which simplify the task of installing and running a Hadoop system, as well as offering training and support services.

So that, in a (fairly large) nutshell, is Hadoop. Thanks to the flexible nature of the system, companies can expand and adjust their data analysis operations as their business expands. And the support and enthusiasm of the open source community behind it has led to great strides towards making big data analysis more accessible for everyone.

**Apache Hadoop** ( [/həˈduːp/](https://en.wikipedia.org/wiki/Help:IPA/English" \o "Help:IPA/English)) is a collection of [open-source](https://en.wikipedia.org/wiki/Open_source) software utilities that facilitate using a network of many computers to solve problems involving massive amounts of data and computation. It provides a [software framework](https://en.wikipedia.org/wiki/Software_framework) for [distributed storage](https://en.wikipedia.org/wiki/Clustered_file_system) and processing of [big data](https://en.wikipedia.org/wiki/Big_data) using the [MapReduce](https://en.wikipedia.org/wiki/MapReduce" \o "MapReduce)[programming model](https://en.wikipedia.org/wiki/Programming_model). Originally designed for [computer clusters](https://en.wikipedia.org/wiki/Computer_cluster) built from [commodity hardware](https://en.wikipedia.org/wiki/Commodity_hardware)[[3]](https://en.wikipedia.org/wiki/Apache_Hadoop#cite_note-3)—still the common use—it has also found use on clusters of higher-end hardware.[[4]](https://en.wikipedia.org/wiki/Apache_Hadoop#cite_note-4)[[5]](https://en.wikipedia.org/wiki/Apache_Hadoop#cite_note-5) All the modules in Hadoop are designed with a fundamental assumption that hardware failures are common occurrences and should be automatically handled by the framework.[[2]](https://en.wikipedia.org/wiki/Apache_Hadoop#cite_note-homepage-2)

The core of Apache Hadoop consists of a storage part, known as Hadoop Distributed File System (HDFS), and a processing part which is a MapReduce programming model. Hadoop splits files into large blocks and distributes them across nodes in a cluster. It then transfers [packaged code](https://en.wikipedia.org/wiki/JAR_(file_format)) into nodes to process the data in parallel. This approach takes advantage of [data locality](https://en.wikipedia.org/wiki/Data_locality),[[6]](https://en.wikipedia.org/wiki/Apache_Hadoop" \l "cite_note-6) where nodes manipulate the data they have access to. This allows the dataset to be [processed](https://en.wikipedia.org/wiki/Distributed_processing) faster and more efficiently than it would be in a more conventional [supercomputer architecture](https://en.wikipedia.org/wiki/Supercomputer_architecture) that relies on a [parallel file system](https://en.wikipedia.org/wiki/Parallel_file_system) where computation and data are distributed via high-speed networking.[[7]](https://en.wikipedia.org/wiki/Apache_Hadoop#cite_note-7)[[8]](https://en.wikipedia.org/wiki/Apache_Hadoop#cite_note-8)

The base Apache Hadoop framework is composed of the following modules:

* *Hadoop Common* – contains libraries and utilities needed by other Hadoop modules;
* *Hadoop Distributed File System (HDFS)* – a distributed file-system that stores data on commodity machines, providing very high aggregate bandwidth across the cluster;
* *Hadoop YARN* – introduced in 2012 is a platform responsible for managing computing resources in clusters and using them for scheduling users' applications;[[9]](https://en.wikipedia.org/wiki/Apache_Hadoop#cite_note-9)[[10]](https://en.wikipedia.org/wiki/Apache_Hadoop#cite_note-10) and
* *Hadoop MapReduce* – an implementation of the MapReduce programming model for large-scale data processing.

Key value pair

A **key**-**value pair** (KVP) is a set of two linked data items: a**key**, which is a unique identifier for some item of data, and the **value**, which is either the data that is identified or a pointer to the location of that data. **Key**-**value pairs** are frequently used in lookup tables, hash tables and configuration files.

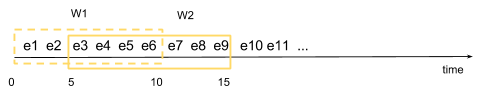
### Understanding Sliding and Tumbling Windows

This subsection describes how sliding and tumbling windows work. Both types of windows move across continuous streaming data, splitting the data into finite sets. Finite windows are helpful for operations such as aggregations, joins, and pattern matching.

**Sliding Windows**

In a sliding window, tuples are grouped within a window that slides across the data stream according to a specified interval. A time-based sliding window with a length of ten seconds and a sliding interval of five seconds contains tuples that arrive within a ten-second window. The set of tuples within the window are evaluated every five seconds. Sliding windows can contain overlapping data; an event can belong to more than one sliding window.

In the following image, the first window (w1, in the box with dashed lines) contains events that arrived between the zeroth and tenth seconds. The second window (w2, in the box with solid lines) contains events that arrived between the fifth and fifteenth seconds. Note that events e3 through e6 are in both windows. When window w2 is evaluated at time t = 15 seconds, events e1 and e2 are dropped from the event queue.

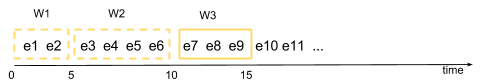


An example would be to compute the moving average of a stock price across the last five minutes, triggered every second.

**Tumbling Windows**

In a tumbling window, tuples are grouped in a single window based on time or count. A tuple belongs to only one window.

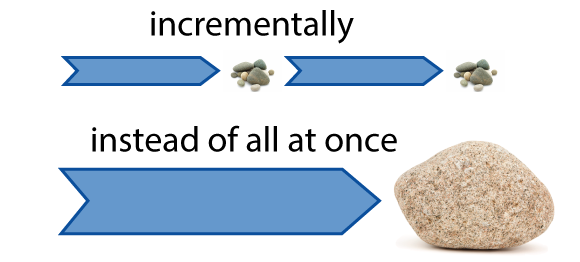
For example, consider a time-based tumbling window with a length of five seconds. The first window (w1) contains events that arrived between the zeroth and fifth seconds. The second window (w2) contains events that arrived between the fifth and tenth seconds, and the third window (w3) contains events that arrived between tenth and fifteenth seconds. The tumbling window is evaluated every five seconds, and none of the windows overlap; each segment represents a distinct time segment.



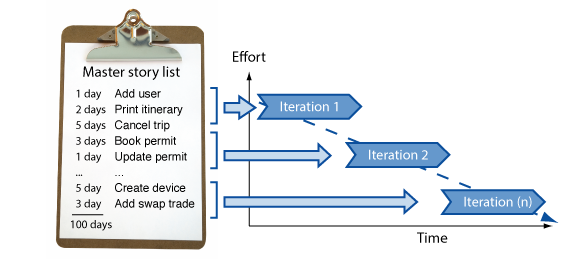
An example would be to compute the average price of a stock over the last five minutes, computed every five minutes.

# What is Agile?

Agile is a time boxed, iterative approach to software delivery that builds software incrementally from the start of the project, instead of trying to deliver it all at once near the end.



It works by breaking projects down into little bits of user functionality called [user stories](http://www.agilenutshell.com/user_stories), prioritizing them, and then continuously delivering them in short two week cycles called [iterations](http://www.agilenutshell.com/iterations).



# How does it work?

At its core, Agile does the same thing you and I do when faced with too much to do and not enough time.

### You make a list

Sitting down with your customer you make a list of features they would like to see in their software. We call these things [user stories](http://www.agilenutshell.com/user_stories) and they become the To Do list for your project.



### You size things up

Then, using Agile [estimation](http://www.agilenutshell.com/estimation) techniques, you size your stories relatively to each other, coming up with a guess as to how long you think each user story will take.



### You set some priorities

Like most lists, there always seems to be more to do than time allows. So you ask your customer to prioritize their list so you get the most important stuff done first, and save the least important for last.



### You start executing

Then you start delivering some value. You start at the top. Work your way to the bottom. Building, iterating, and getting feedback from your customer as you go.

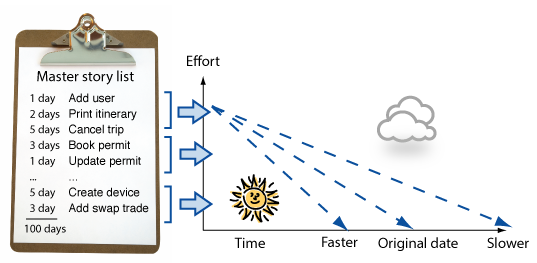


### You update the plan as you go.

Then, as you and your customer starting delivering, one of two things is going to happen. You'll discover:

1. You're going fast enough. All is good. Or,
2. You have too much to do and not enough time.

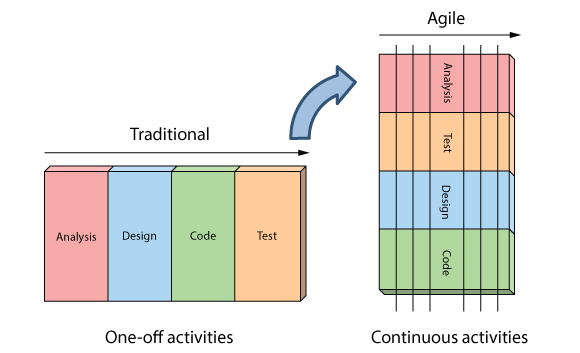
At this point you have two choices. You can either a) do less and cut scope (recommended). Or you can b) push out the date and ask for more money.



# How is Agile different?

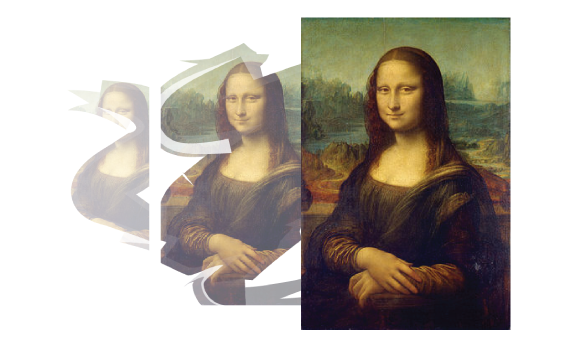
### Analysis, design, coding, and testing are continuous activities

You are never done analysis, design, coding and testing on an Agile project. So long as there are features to build, and the means to deliver them, these activities continue for the duration of the project.



### Development is iterative

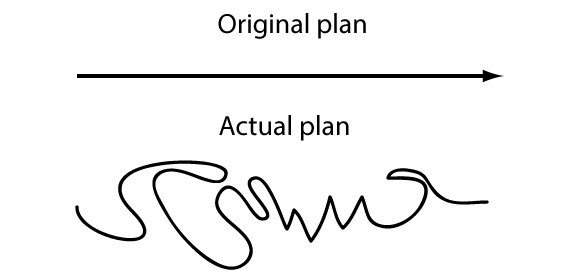
Iterative development means starting with something really simple, and adding to it incrementally over time.



It means evolving the architecture, accepting that your requirements are going to change, and continuously refining and tweaking your product as you go.

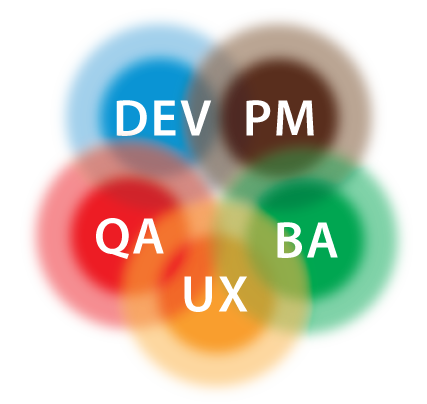
### Planning is adaptive

When reality disagrees with their plans, Agilists find it easier to change their plans than reality. They call this adaptive planning.



And while there are many ways to changes plans, the preferred way is to flex on scope.

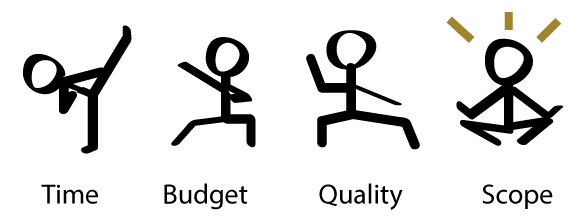
### Roles blur



Roles really blur on Agile projects. When it’s done right, joining an Agile team is a lot like working in a mini-startup. People pitch in and do whatever it takes to make the project successful—regardless of title or role.

Yes, people still have core competencies, and, yes, they generally stick to what they are good at. But on an agile project, narrowly defined roles like analyst, programmer, and tester don’t really exist - at least not in the traditional sense.

### Scope can vary

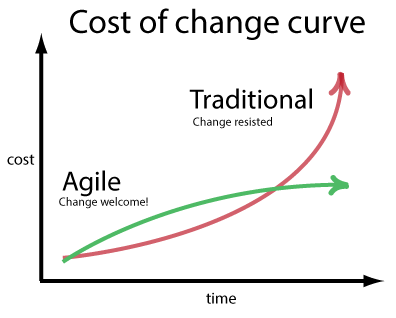


Agile deals with the age old problem of having too much to do and not enough time by doing less.

By fixing time, budget, and quality, and being flexing around scope, Agile teams maintain the integrity of their plans, work within their means, and avoid the burn out, drama, and dysfunction traditionally associated with our industry.

### Requirements can change

Traditionally change has been shunned on software projects because of it's high perceived cost late in the game. Agile challenges this notion and believes the cost of change can be relatively flat.



Through a combination of modern software engineering practices, and open and honest planning, Agilsts accept and embrace change even late in delivery process.

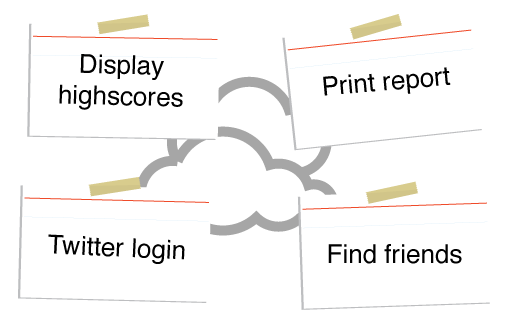
### Working software is the primary measure of success



The rate at which teams can turn their customer's wishes into working software is how Agilists measure productivity. Project plans, test plans, and analysis artifacts are all well and good but Agilists understand they in themselves are of no value to the end customer.

# User Stories

Because life's too short to write everything down



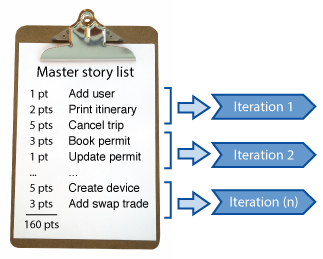
User stories are features our customers might one day like to see in their software.

User stories are like Agile requirements except that they’re not. For one there’s no guarantee all these features are going to make it into the final version of the software. Secondly, Agilists know their customers are going to change their mind - and that’s OK. Because they weren’t really requirements to begin with.



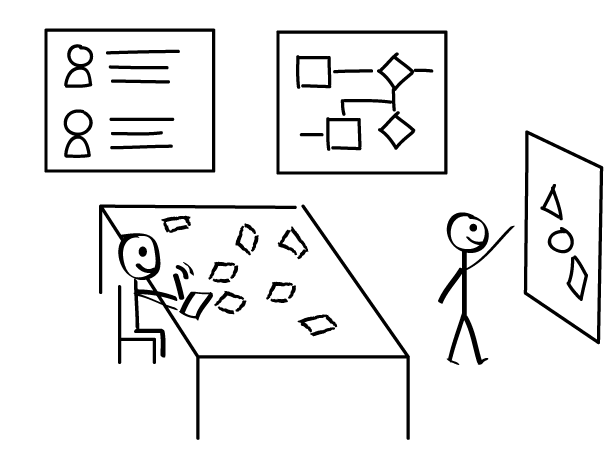
They are written on index cards to encourage face-to-face communication.

Words are slippery things. Get a comma wrong and it can [cost you a million dollars](http://www.nytimes.com/2006/10/25/business/worldbusiness/25comma.html?_r=0). That’s why Agilists love index cards. They make it impossible to write everything down and instead force you to get off your butt and go talk to your customers about the features they’d like to see in their software.



Typically no more than a couple days work, they form the basis of our Agile plans.

User stories form the basis of the Agile plan. They are sized and prioritized like any other wish list. You simply start at the top and work your way down. Nothing big or complex. Just a prioritized todo list and a desire to get things done.

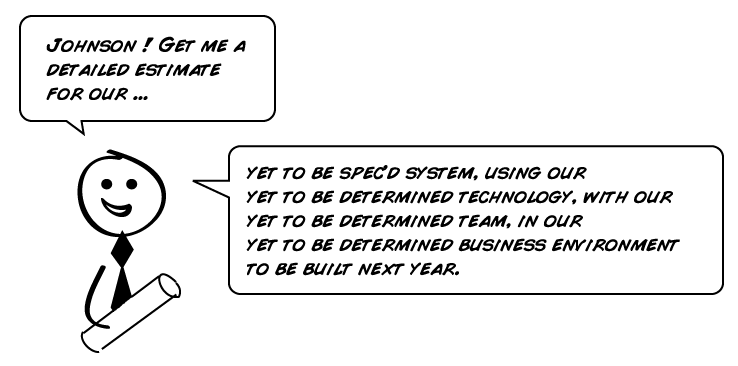


We get them by sitting down with our customers and asking lots of questions.

Big rooms with lots of white space to draw are great for gathering user stories. In these story gathering workshops we draw lots of pictures (flowcharts, screens, storyboards, mockups, anything that helps) and break the functionality down into simple easy to understand words and phrases our customers understand. User stories!

# Estimation

The fine art of expectation guessing



While we aren’t very good at estimating things absolutely, it turns out we are pretty good at estimating things relatively.



Sizing stories relatively means not worrying about *exactly* how big a story is, and worrying more how this story's size compares to others.

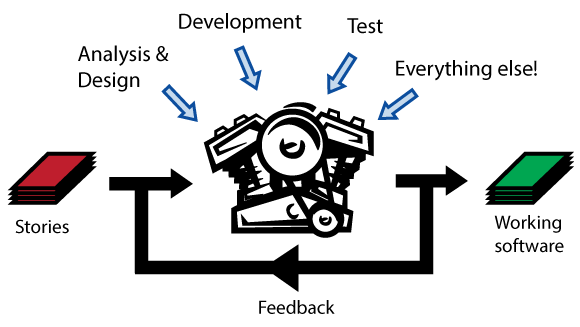


This style of estimation (relative over absolute) forms the corner stone of Agile Planning. By sizing our stories relatively, and feeding actuals back into our plan, we can make some really accurate predictions about the future while based on what we've done in the past.

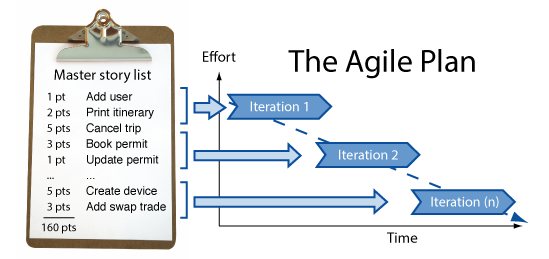
# Iterations

Agile's engine for getting things done

An Agile iteration is a short one to two week period where a team takes a couple of their customers most important user stories and builds them completely as running-tested-software.



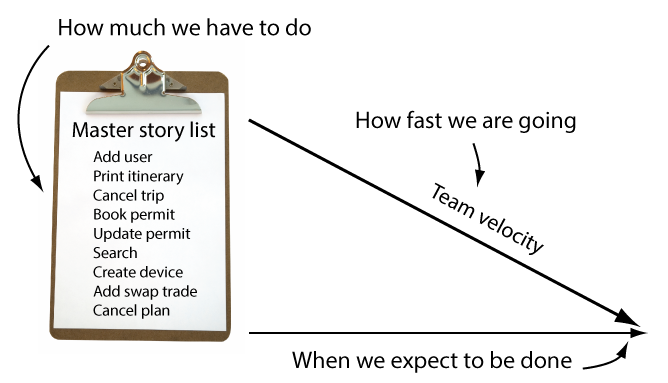
This means everything happens during an iteration. Analysis, design, coding, testing. It all happens here. The beauty of working this way, is every couple weeks the customer gets something of great value (working software), but it's also a great way to track progress (measuring the rate at which the team can turn user stories into production ready working software).



# Planning

The fine art of expectation setting

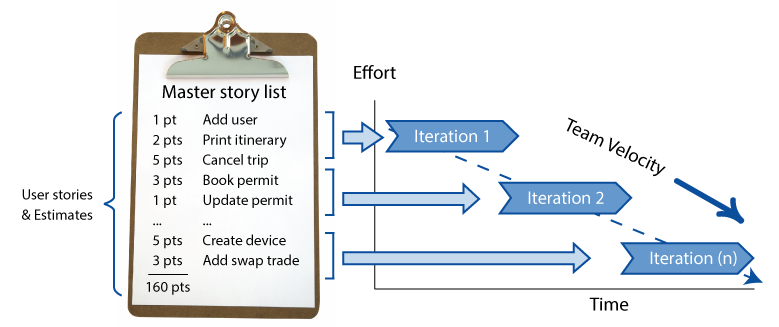
In its simplest form, agile planning is nothing more than measuring the speed a team can turn user stories into working, production-ready software and then using that to figure out when they’ll be done.



Our to-do list on an agile project is called the master story list. It contains a list of all the features our customers would like to see in their software.

The speed at which we turn user stories into working software is called the team velocity. It’s what we use for measuring our team’s productivity and for setting expectations about delivery dates in the future.

The engine for getting things done is the agile iteration - one to two week sprints of work where we turn user stories into working, production-ready software.



To give us a rough idea about delivery dates, we take the total effort for the project, divide it by our estimated team velocity, and calculate how many iterations we think we’ll require to deliver our project. This becomes our project plan.

# iterations = total effort / estimated team velocity

For example:

# iterations = 100 pts / 10 pts per iteration = 10 iterations

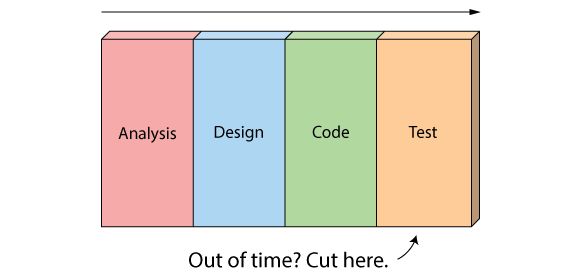
Now, as we start delivering, one of two things is going to happen. We are going to discover that a) we are going faster than expected or b) we are going slower than we originally thought.

# Agile vs Waterfall

### Waterfall challenges

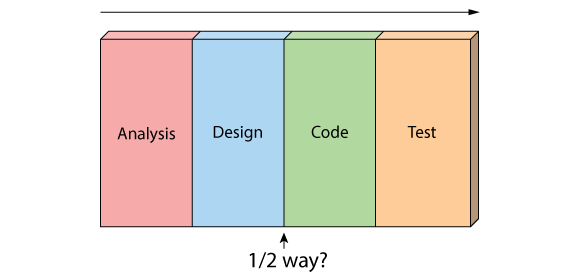
Traditional Waterfall treats analysis, design, coding, and testing as discrete phases in a software project. This worked OK when the cost of change was high. But now that it's low it hurts us in a couple of ways.

### Poor quality



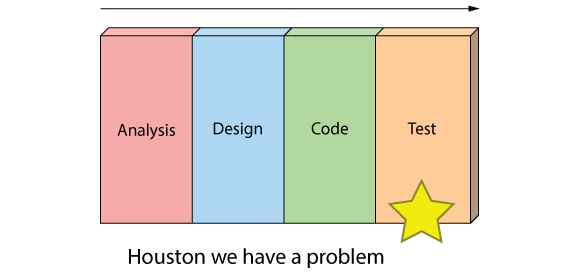
First off, when the project starts to run out of time and money, testing is the only phase left. This means good projects are forced to cut testing short and quality suffers.

### Poor visibility



Secondly, because working software isn't produced until the end of the project, you never really know where you are on a Waterfall project. That last 20% of the project always seems to take 80% of the time.

### Too risky



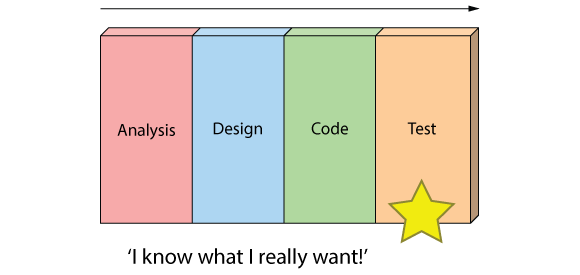
Thirdly you've got schedule risk because you never know if you are going to make it until the end.

You've got technical risk because you don't actually get to test your design or architecture until late in the project.

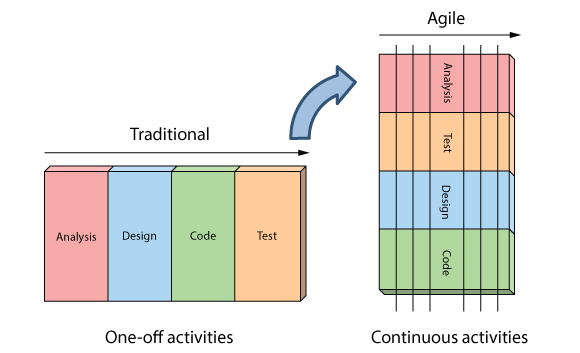
And you've got product risk because don't even know if you are building the right until it's too late to make any changes.

### Can't handle change

And finally, most importantly, it's just not a great way for handling change.



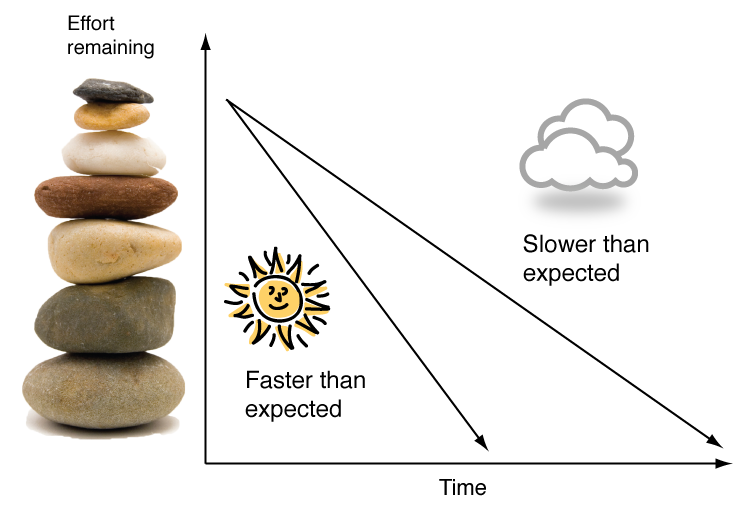
## The Agile Approach



Instead of treating these fixed stages Agilists believe these are continuous activities.

By doing them continuously:

* Quality improves because testing starts from day one.
* Visibility improves because you are 1/2 way through the project when you have built 1/2 the features.
* Risk is reduced because you are getting feedback early, and
* Customers are happy because they can make changes without paying exorbitant costs.



Faster than expected means you and your team are ahead of schedule. Slower than expected (more the norm) means you have too much to do and not enough time.

When faced with too much to do, agile teams will do less (kind of like what you and I do when faced with a really busy weekend). They will keep the most important stories, and drop the least important. This is called *adaptive planning* and it’s how Agile teams work within their budgets and keep their projects real.