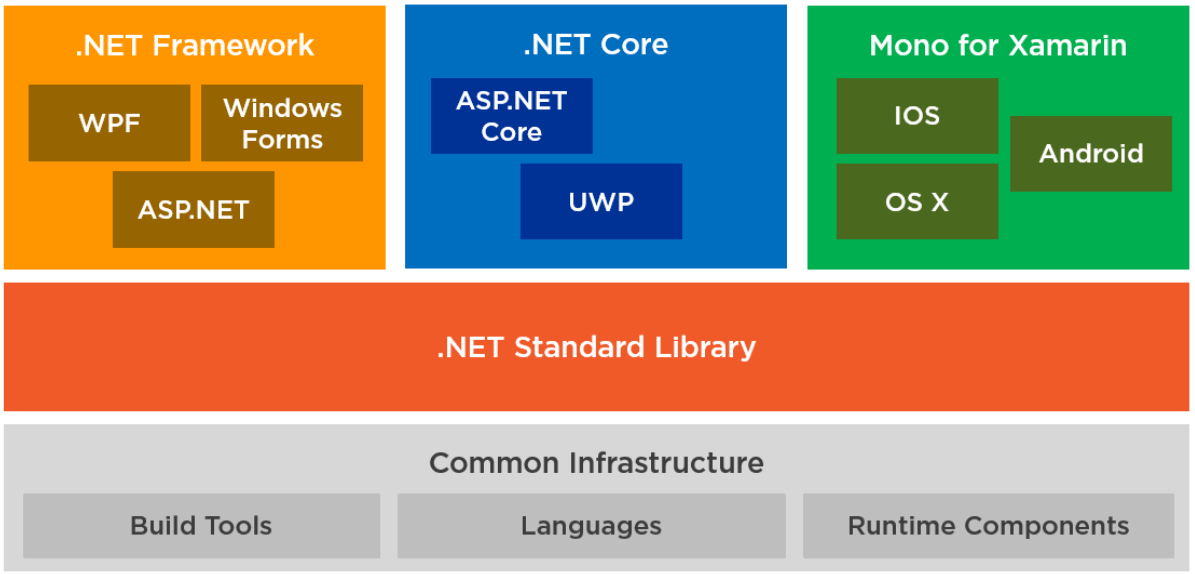
El Ecosistema .NET

.Net es un framework con herramientas para el desarrollo de software creado por Microsoft y lanzado en 2002. A lo largo de todos estos años ha cambiado mucho y se han realizado múltiples versiones funcionando incluso en paralelo. Vamos a presentar las versiones que son necesarias conocer.



Runtimes

NET Framework

La última versión fue la 4.8 fue en abril de 2019. Por mucho tiempo fue el único. El código es abierto para ver pero no se puede contribuir.

Todo el framework funciona solo en SOs de Windows, todo el framework es centrado en windows.

Es side-by-side para las versiones mayores(los cambios mayores introducen modificaciones en el CLR), es decir que puede correr concurrentemente diferentes versiones mayores en un mismo SO.

Ofrece un gran gama de tipos de aplicaciones que se pueden crear. Algunas de estos tipos (Workloads) son los siguientes:

- Console Applications: aplicaciones de consola.
- Windows Communications Foundations (WCF):para comunicación de servidores web

- Windows Workflow Foundation (WF): automatizar procesos de negocios
- Windows Presentation Foundation (WPF): aplicaciones de escritorio con complejas UI
- Windows Form: aplicaciones de escritorio pero sencillas
- ASP .NET: aplicaciones web con Web Forms, Web API o MVC
- Azure (WebJobs, Cloud Services)

Net Core

Completamente Open Source y el código se encuentra disponible en github. Es un framework multiplataforma, más liviano que net framework. La filosofía de Net Core es tener solo las librerías necesarias, y de hacer falta se instalan paquetes NuGet los cuales no son mas que set de librerías que incorporan las librerías que hacen falta para circunstancias particulares.

Net core puede correr SOs de Windows Red Hat(Linux), Ubuntu, Red Hat, Unix y Mac

Es totalmente side-by-side, sin importar si es una versión mayor o menor. También permite realizar aplicaciones self-contained es decir que es una aplicación que ya viene con Net Core incorporada en sí misma con todas las librerías e infraestructura que necesita

Los tipos de aplicaciones son los siguientes:

- Console applications
- ASP.Net Core: MVC y API (una evulucion de net framework)
- UWP: para aplicaciones de escritorio

Mono for Xamarin

Completamente Open Source y el código se encuentra disponible en github. Es un runtime para crear aplicaciones móviles. Es la implementación Open Source y Multiplataforma del CLR original de Net Framework, es de 2001 y solo acepta C#. Por otro lado, la Librería de clases de Xamarin es de 2011, y provee clases para desarrollo móvil en Android y IoS.

Xamarin permite crear aplicaciones móviles usando el entorno de desarrollo .NET en Visual Studio y usando C#. Por otro lado aplicaciones desarrolladas de esta forma son self-contained y se pueden instalar de forma nativa en dispositivos con Android o IoS y MAC Apple OS X

Comparación entre Runtimes

	.NET Framework	.NET Core	Mono for Xamarin
Workloads	WPF, Windows Forms, ASP.NET	ASP.NET Core, UWP	IOS, Mac OS X, Android
Cross-platform		X	X
Side-by-side	Only major versions	X	X
Self-contained		X	X
Main purpose	Windows desktop apps	Cross-platform web and desktop apps	Cross-platform mobile apps

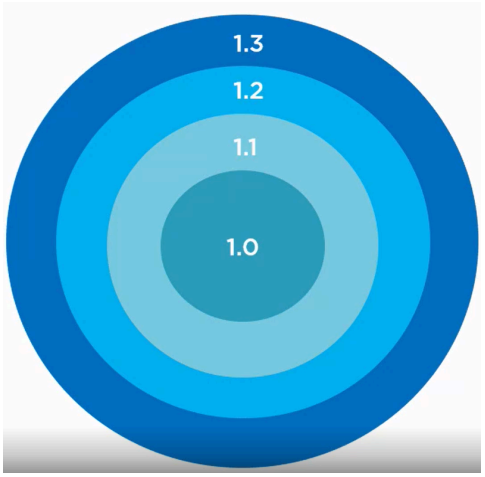
Net Standard

Todos los runtimes que fueron descriptos tienen sus librerías de clases que le permiten realizar funciones específicas para los tipos de aplicaciones que cada uno permite desarrollar. Sin embargo esto hace difícil que se puedan compartir funciones entre los distintos runtimes.

Para solucionar esto está Net Standard el cual es un set de especificaciones que indica qué funciones (implementadas APIs) se pueden usar en todas las plataformas. Por lo tanto no es algo que se pueda instalar, es conjunto formal de especificaciones de APIs de .NET

Todos los runtimes que fueron presentados implementan .Net Standard. Es decir que una versión específica de un runtime implementa una versión específica de .Net Standard. Ejemplo: .Net Framework 4.5 implementa Net Standard 1.1

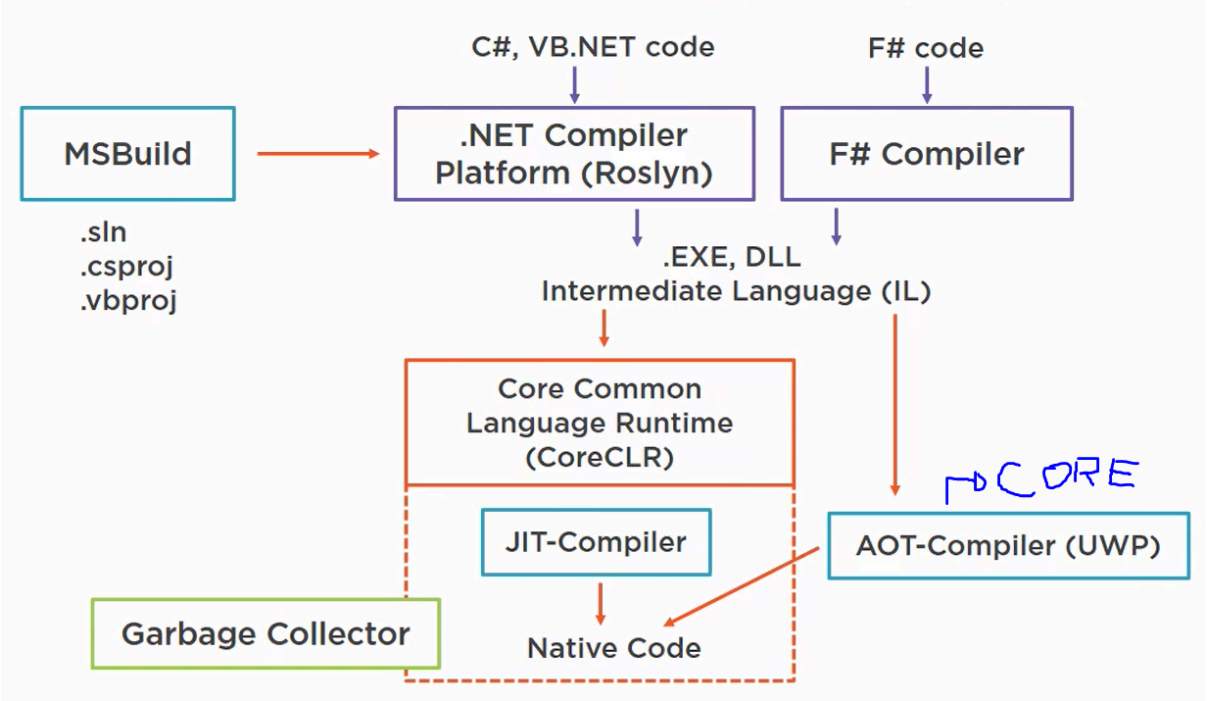
El propósito principal de Net Standard es compartir código entre runtimes, cada nueva versión incluye todas las APIs de la versión anterior más las nuevas propias de la versión actual. Sin embargo por esto mismo las versiones anteriores de Net Standard son compatibles por más versiones de runtimes



Se pueden crear Librerías de Clases que usen una determinada versión de .Net Standard como objetivo y de esta forma se puede reutilizar estas clases en todos los runtimes que implementen la versión de Net Standard objetivo de nuestra Librería de Clases creada.

Ejemplo: Realizó una Librería de Clases que implementa una clase MonedaArgentina y está implementa métodos de conversión a otras divisas esta clase. Esta librería de clases tiene como objetivo Net Standard 1.1, de esta forma para todos mi proyectos de Net Framework 4.5 van a tener disponible esta Librería de Clases

Common Infrastructure



En la estructura común de .NET podemos ver en la primera imagen de la unidad que en la hay herramientas de compilación, lenguajes y componentes de runtimes. Todo esto es común a todo el ecosistema de .NET.

Todas estos elementos de la infraestructura común permiten construir la aplicación para que se ejecute de forma correcta en el dispositivo (PC o Servidor) en donde se ejecuta.

El proceso de compilación para Net Core y para Net Framework y en términos generales también para Mono, es el siguiente:

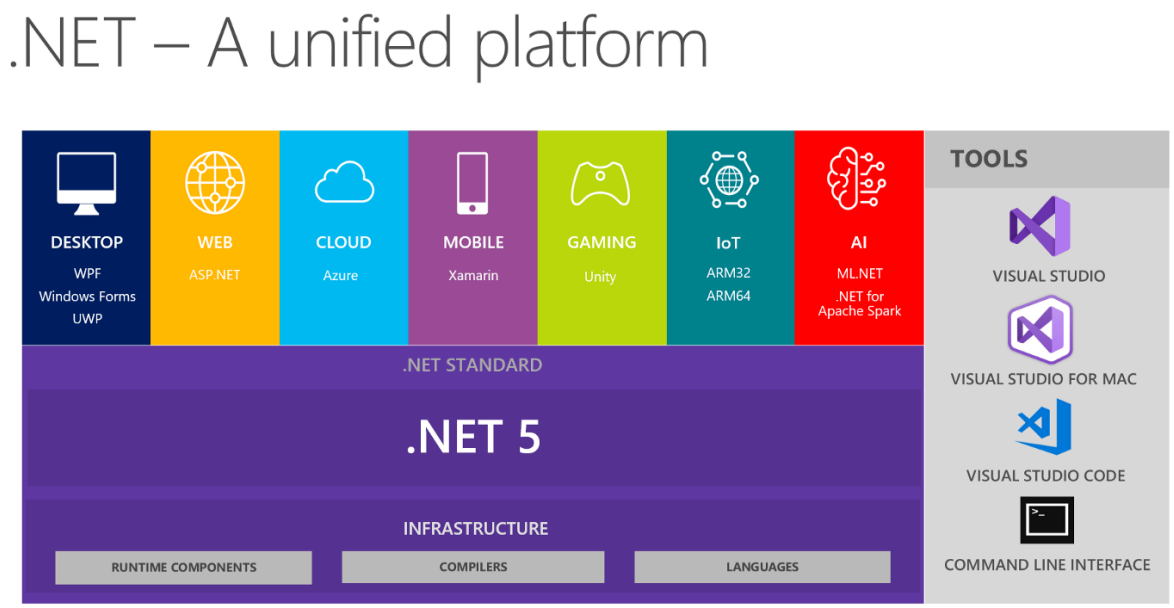
1. La solución que nosotros generamos produce tres archivos que se usan para compilar. El archivo sln que es el ejecutable de la solución (que se usa para abrir la solución en visual studio), el archivo .csproj junto con .vbproj
2. Cuando le damos a ejecutar se invoca al compilador de la plataforma .Net para c# (también conocido como Roslyn) o bien el F# Compiler, en el caso de que se haya programado en ese lenguaje. Estos compiladores generan un archivo ejecutable (.Exe, .DLL) el cual esta escrito en IL (Intermediate Language)
3. El common language runtime o CLR (en el caso de Net Core es el Core CLR) interpreta estos ejecutables en IL y ejecuta el JIT Compiler (Just in Time) apropiado en tiempo de ejecución (el JIT es específico para el hardware en donde se ejecuta la aplicación). El JIT pasa de IL a lenguaje de máquina, es decir lenguaje nativo para el

hardware y lo hace en tiempo de ejecución, es decir que si hay algún error que no fue detectado por el compilador de C# o F# que que paso el codigo a IL, entonces es detectado por el JIT cuando se lo invoca para que compile la parte de código errónea que se requiere ejecutar

Ejemplos

https://github.com/bmaluijb/Dot-Net-Ecosystem

NET 5 en adelante



Microsoft decidió realizar un gran esfuerzo para unificar los diferentes runtimes incorporando las características de todos el único framework. De esta forma nace .NET 5 en noviembre de 2020 y la meta es crear una nueva versión de .NET anualmente por lo que hoy en día (Febrero de 2022) la última versión de .NET es NET 6 lanzada en noviembre de 2021. Más sobre las novedades de NET 5

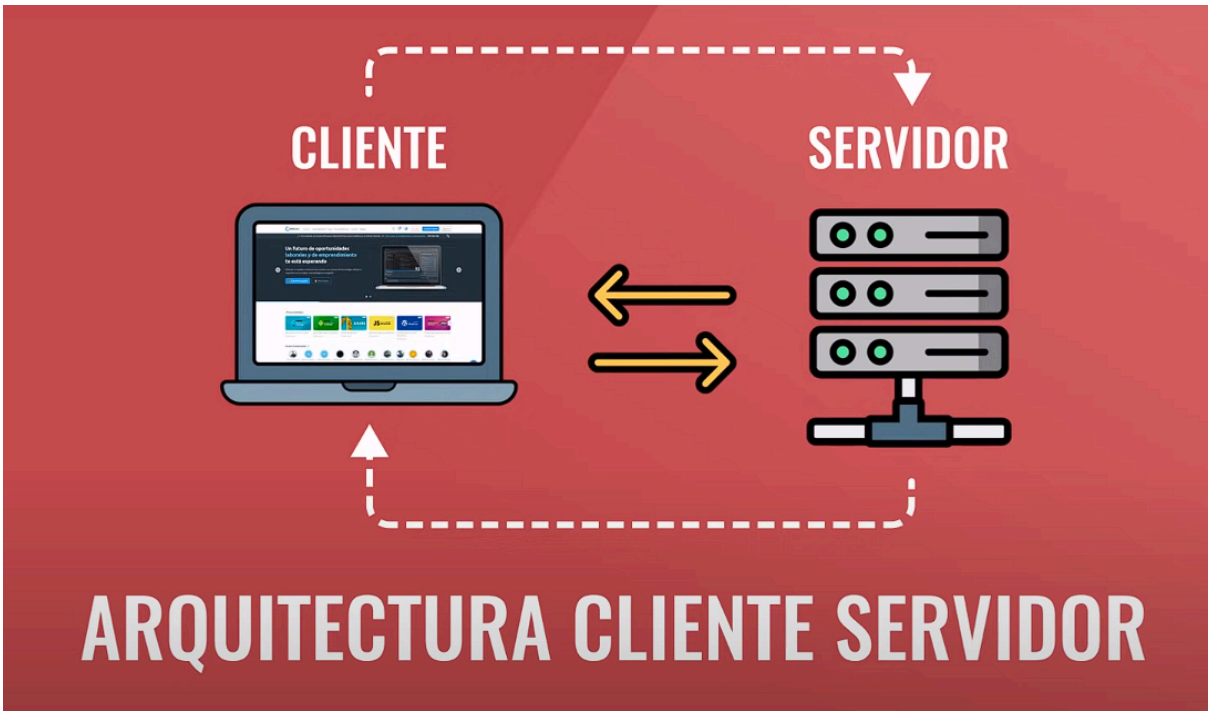
El desarrollo de todos los runtimes en las páginas anteriores tiene como objetivo final entender cómo funciona NET es su completitud y también por consecuencia el de su sucesor NET 5 y sus siguientes versiones. Ya que estas versiones incorporan las características de todos los runtimes mejorando y añadiendo además nuevas características

Introducción a la programación web

Cliente y Servidor

El modelo cliente-servidor, también conocido como “principio cliente-servidor”, es un modelo de comunicación que permite la distribución de tareas dentro de una red de ordenadores.

Un servidor es un hardware que proporciona los recursos necesarios para otros ordenadores o programas, pero un servidor también puede ser un programa informático que se comunica con los clientes. Un servidor acepta las peticiones del cliente, las procesa y proporciona la respuesta solicitada. También existen diferentes tipos de clientes. Un ordenador o un programa informático se comunica con el servidor, envía solicitudes y recibe respuestas del servidor. En cuanto al modelo cliente-servidor, representa la interacción entre el servidor y el cliente.



Hay una clara distribución de tareas entre los clientes y los servidores. El servidor es el responsable de proporcionar los servicios. Se encarga de ejecutar los servicios solicitados y entrega la respuesta esperada. El cliente, en cambio, utiliza y solicita los servicios proporcionados. Finalmente, recibe la respuesta del servidor (Por ejemplo renderiza los datos en la ventana del navegador).

En el modelo cliente-servidor, un servidor sirve a varios clientes y, por ende, procesa múltiples peticiones de diferentes clientes. Para ello, presta su servicio de forma permanente y pasiva. Por su parte, el cliente solicita activamente los servicios del servidor e inicia las tareas del servidor.

Siguiendo este modelo, un ordenador físico puede ser tanto cliente como servidor. El único factor decisivo es su papel dentro de una red y el hecho de que el ordenador envíe o reciba solicitudes de servicios y recursos.

Las normas que definen la comunicación entre clientes y servidores se comunican en forma de protocolos. Y según la tarea, se utilizan diferentes protocolos de red para la transmisión de datos. Además, según el ámbito de aplicación, existen diferentes tipos de red.
.

Ventajas

El modelo cliente-servidor es uno de los conceptos arquitectónicos más utilizados en la tecnología de redes, dado que ofrece algunas ventajas significativas.

Administración central

La administración central es una de las principales ventajas. El servidor está en el centro de la red. Todos los usuarios o clientes lo utilizan. Los recursos importantes, como bases de datos, se encuentran en el servidor y son accesibles de forma centralizada. Esto simplifica la administración y el mantenimiento de los recursos importantes que requieren protección. La ubicación central del servidor hace que la realización de actualizaciones sea cómoda y de bajo riesgo.

Derechos de acceso controlados globalmente

El almacenamiento central de recursos importantes permite una gestión segura y global de los derechos de acceso. Cuando se trata de datos sensibles, es importante saber quién puede ver los datos y quién puede manipularlos. Para proteger los datos de la mejor manera posible, hay que establecer derechos de acceso.

Un servidor para muchos clientes

El número de clientes puede ampliarse. Varios clientes trabajan simultáneamente utilizando un único servidor. Los clientes comparten los recursos del servidor. También es posible que el servidor esté situado en un lugar distinto al de los clientes. Lo más importante es que el servidor y los clientes estén conectados a través de una red. Y por ende, no es necesario que los recursos estén en el mismo sitio.

Inconvenientes

Caída del servidor

Debido a la disposición centralizada y a la dependencia en un modelo cliente-servidor, la caída del servidor conlleva la caída de todo el sistema. Si el servidor se cae, los clientes dejan de funcionar porque no pueden recibir las respuestas necesarias del servidor.

Recursos de un servidor

El servidor realiza las tareas que requieren muchos recursos. La demanda de recursos de los clientes es mucho menor. Si el servidor tiene muy pocos recursos, afecta a todos los clientes. Por eso es importante elegir un proveedor que proporcione estos recursos de forma fiable.

Inversión de tiempo

Otro factor que no hay que subestimar es el tiempo necesario para hacer funcionar tu servidor. Además de los conocimientos técnicos correspondientes, por ejemplo, para proteger y configurar servidores, su uso requiere una considerable inversión de tiempo.

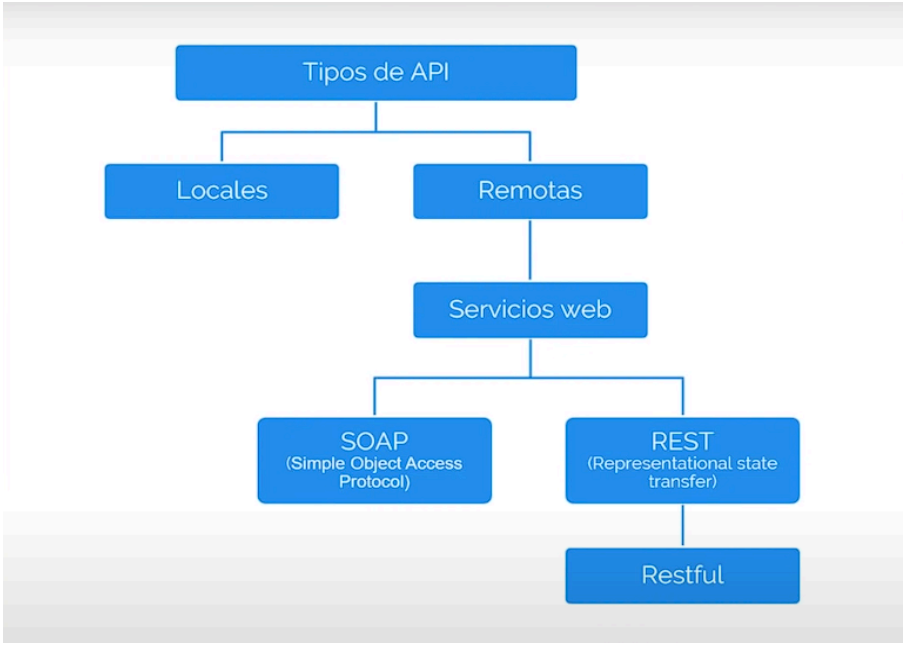
Frontend y Backend

La división Frontend y Backend apareció en el año 2008 con la salida de HTML5 y la aparición de las nuevas APIs. Frontend es la parte del sitio web que interactúa con los usuarios, por eso decimos que está del lado del cliente. Backend es la parte que se conecta con la base de datos y el servidor que utiliza dicho sitio web, por eso decimos que el backend corre del lado del servidor. Estos dos conceptos explican a grandes rasgos cómo funciona un sitio o una aplicación web y son fundamentales para cualquier persona que trabaje en el mundo digital.

¿Qué es una API?

Una API (Application Programming Interface) es una interfaz para que programas de software se comuniquen entre ellos y compartan datos bajo diferentes estándares. Pueden ser locales o remotas estas últimas utilizan un servicio web y sus arquitecturas pueden ser SOAP o REST , el tipo más común hoy en día es REST (Representational State Transfer), generalmente los archivos de información enviados y recibidos son JSON (objetos de

javaScript). Las Api pueden ser públicas o privadas, las privadas requieren un autenticación que se realiza mediante un TOKEN, al momento de por ejemplo loguearte en una página web ésta envía tus datos y en caso de tener permisos se recibe el token, este contiene un objeto cifrado.



Request y Responses

En informática, request–response o request–reply es uno de los métodos básicos que utilizan las computadoras para comunicarse entre sí en una red, en el que la primera computadora envía una solicitud de algunos datos y la segunda responde a la solicitud. Más específicamente, es un patrón de intercambio de mensajes en el que un solicitante envía un mensaje de solicitud a un sistema de respuesta, que recibe y procesa la solicitud y, en última instancia, devuelve un mensaje como respuesta. Es similar a una llamada telefónica, en la que la persona que llama debe esperar a que el destinatario atienda antes de que se pueda hablar de algo. Este es un patrón de mensajería simple pero poderoso que permite que dos aplicaciones tengan una conversación bidireccional entre sí a través de un canal; es especialmente común en arquitecturas cliente-servidor.

Para simplificar, este patrón generalmente se implementa de forma puramente sincrónica, como en las llamadas de servicio web a través de HTTP, que mantiene una conexión abierta y espera hasta que se entrega la respuesta o expira el período de tiempo de espera. Sin embargo, la solicitud-respuesta también se puede implementar de forma asincrónica, con una respuesta que se devuelve en un momento posterior desconocido.

En programación web las request y las responses tiene una forma específica con secciones y atributos que deben estar presente y con determinados valores. Esto los especifica el protocolo HTTP o HTTPS con el cual se realizan casi todas las comunicaciones web

Arquitecturas multi-servidor

También puede haber varios servidores para ofrecer un servicio, estos pueden tener diferentes “direcciones” para accederlos y ciertas configuraciones para que si la disponibilidad de algunos de ellos se ve comprometida entonces el servicio lo ofrece alguno de los otros que se encuentra en correcto estado, esta tecnica se llama “**mirroring**” es bastante común en sitios web de descarga o streaming piratas para así evadir los controles ubicando los servidores “mirror” en diferentes países.

Otro otra forma de implementar la arquitectura es mediante la adquisición de capacidad de procesamiento de la nube mediante el uso de servicios de AWS, GCP, o Azure (también hay otros pero estos claramente son los más completos y por lejos acaparan la inmensa mayoría de la oferta). Estas nubes permiten disponibilizar el servicio que queremos sin que nos tengamos que hacer cargo de la infraestructura (el o los servidores físicos) ellos se encargan de asegurar un elevado ratio de disponibilidad a cambio del pago correspondiente. Esto se llama **procesamiento distribuido**

Los inconvenientes planteados anteriormente se ven enormemente mitigados por los servicios proveídos por las nubes mencionadas: las caídas se vuelven algo casi imposible de que suceda si contrata el plan adecuado ya que los recursos necesarios para brindar el servicio se encuentran siempre disponible y escalan cuando la demanda escala de esta manera los recursos del servidor ya no es algo de lo que nos tengamos que preocupar. La inversión de tiempo también se reduce considerablemente debido a las facilidades que las nubes nos dan así como también la disponibilidad de múltiples formas de capacitación y documentación que las mismas nos brindan.

Rest API

En su disertación original del año 2000, Roy T. Fielding (el creador del concepto) define REST como un "...estilo arquitectónico para sistemas distribuidos de hipermedia, describiendo los principios de ingeniería de software que guían REST y las constantes de interacción elegidas para retener esos principios...".

Estos principios son:

- Arquitectura cliente-servidor. Hace hincapié en la separación de responsabilidades y la portabilidad. Cuanto menos conoce el servidor sobre el cliente más desacoplada está su interacción y más fácil resulta el cambio de componentes. Por definición, REST es una arquitectura diseñada para funcionar con sistemas distribuidos centralizados, en oposición a los que no usan un servidor como nodo principal –por ejemplo, mediante una arquitectura entre iguales o P2P–.
- Ausencia de estado. El estado se guarda y mantiene en el cliente y no en el servidor. Es decir, las solicitudes deben proveer toda la información necesaria para poder realizarse en un servidor que no mantiene estados, por lo que no guarda contexto entre llamadas para un mismo cliente. Esto no significa que el servidor no pueda cambiar de contexto y facilitar esa transición al cliente. Para conseguir esto normalmente se usan redirecciones de peticiones; de forma que el cliente solo

realice una petición y el servidor la procese en un endpoint o recurso, y la redirija a otro para continuar su procesamiento allí. Esto ocurre de forma transparente para el cliente, y es habitual por ejemplo en casos de uso relacionados con el registro y posterior inicio de sesión automático: El servidor crea la cuenta, genera un token de sesión y, en vez de devolverlo al cliente para que este llame a la URI de login –lo que aumentaría los tiempos, pues cada petición es costosa para los clientes–, lo reenvía a dicho endpoint de inicio de sesión él mismo, devolviendo al cliente la respuesta final de la última petición realizada.

- Habilitación y uso de la caché. Todas las solicitudes deben declarar si son o no cacheables, una forma estándar de hacerlo es con los encabezados cache-control de HTTP. De esta forma, se pueden enviar respuestas cacheadas desde cualquier punto de la red sin necesidad de que la petición llegue al servidor. Pese a que en el caso de una API dinámica esto puede parecer no tener sentido, puede ahorrar costes en casos de datos inmutables, como definiciones.
- Sistema por capas. Tiene relación con la separación de responsabilidades anteriormente mencionada, y establece que un cliente debe conocer únicamente la capa a la que le está hablando. Es decir, no debe tener en cuenta aspectos concretos, como particularidades de la base de datos usada; o abstracciones como cachés, proxies o balanceadores de carga implicados. Si se necesita seguridad, esta se debe añadir encima de los servicios Web, permitiendo que la lógica y seguridad permanezcan separadas.
- Interfaz uniforme.
    - Identificación de recursos en las peticiones. Como se señala en la descripción de esta página, las solicitudes identifican a recursos individuales. Sin embargo, REST recalca que los recursos están conceptualmente separados de las representaciones que son devueltas por el servidor (HTML, XML, JSON...). El tipo de formato se puede especificar en las cabeceras HTTP y, mediante negociación de contenido, servidor y cliente pueden ponerse de acuerdo en la respuesta que mandará el primero y que espera el segundo.
    - Manipulación de recursos a través de representaciones. La especificación de REST intenta economizar peticiones en todo lo que sea posible. Por ello, cuando un cliente posee la representación de un recurso, incluyendo cualquier metadato adjunto, tiene suficiente información para modificar o eliminar el estado del recurso. Es decir, mediante las herramientas que REST promueve (uso de API autodocumentada, descriptiva, verbos HTTP...), el cliente puede saber predecir el resultado esperado de hacer cualquier operación sobre un recurso recibido primeramente con un GET.
    - Mensajes autodescriptivos. La idea de un mensaje autodescriptivo es contener toda la información que el cliente necesita para entenderlo. Por lo tanto, no debería existir información adicional en una documentación separada o en otro mensaje.
    - Hipermedia como motor del estado de la aplicación (HATEOAS por sus siglas en inglés, Hypermedia As The Engine of Application State). Una vez se ha accedido a la URI inicial de la aplicación, un cliente REST debería ser capaz de usar los enlaces provistos dinámicamente por parte del servidor para descubrir todos los recursos disponibles que necesita. Según continúa el proceso, el servidor responde con texto que incluye enlaces a otros recursos

que están actualmente disponibles. Esto elimina la necesidad de que el cliente tenga información escrita en el código (hard-codeada) con respecto a la estructura o referencias dinámicas a la aplicación

Niveles por los que pasa una API para ser REST Full API

los niveles por los que pasa una especificación de API REST desde que es creada hasta que se perfecciona adquiriendo controles hipermedia. Estos niveles son:

- Nivel 0. Los servicios cuentan con una sola URI que acepta todo el rango de operaciones admitidas por el servicio, con unos recursos poco definidos. No se considera una API RESTful.
- Nivel 1. Introduce recursos y permite hacer peticiones a URIs individuales para acciones separadas en lugar de exponer un punto de acceso universal. Los recursos siguen estando generalizados pero es posible identificar un ámbito algo más concreto. El primer nivel sigue sin ser RESTful, pero está más orientado a adquirir la capacidad de serlo.
- Nivel 2. El sistema empieza a hacer uso de los verbos HTTP. Esto permite mayor especialización y generalmente conlleva la división de los recursos en dos: Uno para obtener únicamente datos (GET) y otro para modificarlos (POST), aunque un grado mayor de granularidad también es posible. Una de las desventajas de proveer un sistema distribuido con más de una petición GET y POST por recurso puede ser el aumento de complejidad del sistema, a pesar de que el consumo de datos por clientes de la API se simplifica en gran medida.
- Nivel 3. El último nivel introduce la representación hipermedia. Esta representación se realiza mediante elementos incrustados en los mensajes de respuesta de los recursos, que permiten al cliente que envía la petición establecer una relación entre entidades de datos individuales. Por ejemplo, una petición GET a un sistema de reservas de un hotel, podría devolver el número de habitaciones disponibles junto con los enlaces que permiten reservar habitaciones específicas.

Navegadores, Motores de búsqueda y Postman

Existen muchas posibilidades de que el cliente se pueda comunicar con el servidor. Un celular, una tablet, una PC e incluso dispositivos más limitados como un chromecast o el android auto de un automóvil, pueden comunicarse con uno o varios servidores mediante diferentes aplicaciones como spotify para reproducir música o AccuWeather para recibir notificaciones del clima.

Además de estas aplicaciones están los navegadores o browsers que nos permiten navegar por la internet de manera mucho menos ilimitada que una aplicación específica, los navegadores son la pieza clave del cliente en las aplicaciones web. En esencia un navegador no es más que un “intérprete” de HTML CSS y Javascript el cual interpreta el código fuente de los sitios web con los que interactúa y realiza las solicitudes al servidor especificadas en el código y procesa las respuestas que recibe del servidor. Un navegador es la herramienta única y necesaria para acceder a cualquier sitio web de la internet.

Los motores de búsqueda (Google, Bing, The Pirate Bay) son servicios web que nos proveen de las direcciones de sitios web mediante la búsqueda en enorme base de datos de sitios webs con el uso de algoritmos que intentan brindarnos las mejores aproximaciones de lo que buscamos mediante coincidencias en el contenido de los sitios y nuestros parametros de busqueda. Los navegadores (Chrome, Mozilla, Firefox, Safari, Thor) son aplicaciones que pueden usar estos buscadores para acceder a sitios webs, esencialmente lo único que necesitan los navegadores para acceder a un sitio web es su dirección (dominio o ip) y conexión a internet

Postman es una aplicación que nos permite realizar solicitudes “custom” a servidores, es decir no tiene que interpretar código fuente para realizar una solicitud como lo hace un navegador con los sitios web, sino que podemos armar la solicitud como queramos. El uso de esta herramienta está limitado a probar el funcionamiento de un servidor, esto es muy útil en la programación o en el testing