WEB SERVICE DI TIPO REST

Il Web è nato negli anni ’90 come una piattaforma per la condivisione di documenti testuali distribuiti su diverse macchine e tra loro interconnessi. Il tutto è nato e si è evoluto grazie alla standardizzazione di alcuni semplici concetti:

1. **URI** (**Uniform Resource Identifier**): è una sequenza di caratteri **che identifica univocamente una risorsa.**

   Esempi di **URI** sono :

   - un **URL (Uniform Resource Locator)** è un URI che per identificare la risorsa utilizza il mezzo con cui è possibile ottenerne una rappresentazione, ad esempio http://www.google.com indica la risorsa (una pagina web) una cui rappresentazione (il suo codice HTML) può essere ottenuta inviando una richiesta con protocollo http all’host che la ospita.
   - un **URN** (**Uniform Resource Name**) è un URI che identifica una risorsa attraverso un nome in un particolare dominio dei nomi (namespace) ma senza indicare in che modo possa essere ottenuta una rappresentazione di quella risorsa. Un esempio è il codice ISBN di un libro, esso individua il libro nel dominio dei nomi ISBN ma non specifica come ottenerne una copia. Ogni libro, caratterizzato da un titolo, autore, edizione, anno di pubblicazione, ha uno specifico codice ISBN che lo rende univoco. Una stessa risorsa libro può avere diverse forme (cartaceo, ebook) reperibili in modo diverso, ma tale risorsa sarà sempre identificata dallo stesso URN, ad esempio 8807900769. Un altro esempio di URN è l’IBAN di un conto corrente nello spazio di nomi degli IBAN.

2. **HTTP**: protocollo semplice e leggero per interagire (creare, richiedere, modificare, eliminare) una risorsa che si trova su un host di una rete informatica.

3. **HTML**: linguaggio per la rappresentazione di una risorsa.

Questa idea iniziale (condivisione di documenti testuali) si è evoluta nel corso degli anni non tanto nei concetti di base, quanto nel modo di utilizzarli.

Ad un certo punto ai documenti testuali si sono aggiunti contenuti multimediali, in seguito si è iniziato a produrre documenti generati dinamicamente e non pubblicati come pagine statiche, e infine il Web è uscito dalla visione esclusivamente ipertestuale per diventare una **piattaforma applicativa distribuita,** ossia un **contenitore di applicazioni software che possono collaborare fra di loro.**

Come si realizza questa piattaforma applicativa distribuita? Con il concetto di Web Service, nato intorno all’anno 2000. **Un Web Service è un sistema software progettato per supportare un’interoperabilità tra applicazioni, utilizzando le tecnologie e gli standard del Web.** Il meccanismo dei Web Service consente di far interagire in maniera **trasparente** applicazioni sviluppate con linguaggi di programmazione diversi, e che possono venire eseguite su sistemi operativi diversi.

Per uno sviluppatore, l'utilizzo dei web service nello sviluppo delle proprie applicazioni consente di avere a disposizione un grande numero di **funzionalità** già pronte, e di poter comporre la propria applicazione assemblando queste funzionalità, fornite dai web service, senza problemi di compatibilità.

Esempio di web service: il geocoding

Sul web sono disponibili una grande quantità di web service sia gratuiti che a pagamento. Generalmente l'accesso a questi web service viene consentito ad uno sviluppatore previa iscrizione al servizio. Una volta avvenuta l'iscrizione, lo sviluppatore riceve un codice chiamato **API key** che egli dovrà specificare ogni volta che accede alla risorsa. Spesso sono consentiti un numero massimo di accessi giornalieri o mensili gratuiti alle risorse, e quando tale numero di accessi viene superato, il web service richiede il pagamento di un canone per ulteriori accessi.

Un esempio di web service è il servizio di **geocoding** messo a disposizione dalla piattaforma **Google Maps** (qui si può trovare la documentazione: https://developers.google.com/maps/documentation/geocoding/intro)

Il geocoding è un servizio che consente di inviare al web service l'indirizzo stradale di un luogo, ed ottenere come risposta le coordinate geografiche (latitudine e longitudine) del luogo specificato. Il servizio inverso, vale a dire la determinazione di un indirizzo fornendo le coordinate geografiche, è chiamato **reverse geocoding**.

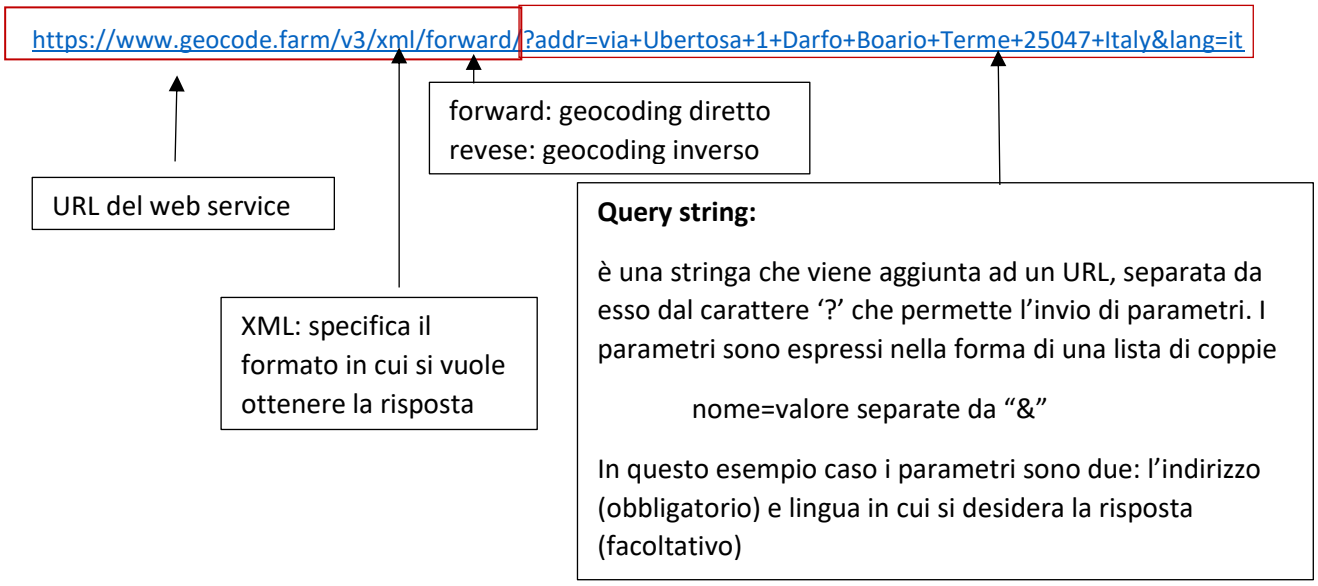
Generalmente i web service consentono un accesso gratuito ai propri servizi fino ad un numero massimo di richieste. Per ulteriori accessi è richiesta un'iscrizione e il pagamento di un canone, in seguito al quale viene fornita all'utilizzatore l'API Key. L' API Key va specificata ogni volta che viene inviata una richiesta al web service. Google Maps richiede di registrare una carta di credito per ottenere l' API Key.

Altri servizi di geocoding alternativi ma simili a quello di Google Maps, sono quelli forniti dai seguenti web service:

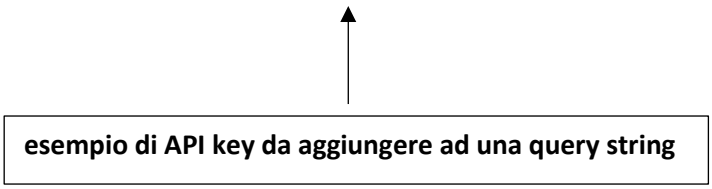
1. https://geocode.farm/ il quale consente 250 accessi gratuiti al giorno
2. https://nominatim.org/ che utilizza le mappe di OpenStreetMap, un progetto Open Source analogo a quello di Google Maps

Proviamo ad utilizzare il web service geocode.farm:

inviamo la richiesta al web service specificando, nella barra degli indirizzi di un browser, l'indirizzo "via Ubertosa 1 Darfo Boario Terme 25040 Italy". **In pratica questo corrisponde ad inviare una richiesta http di tipo GET**. I parametri della richiesta (ossia l'indirizzo) vanno specificati all'interno dell'URL del web service dopo il "?" in una lista denominata **query string** formata da **una serie di coppie** "**nome parameto=valore**" separate fra loro dal simbolo "**&**":



Nel caso in cui fosse necessaria la API key, essa andrebbe inserita come ulteriore parametro della query string, ad esempio : &003026bbc133714df1834b8638bb496e-8f4b3d9a-e931-478d-a994



La risposta ottenuta è un documento XML che contiene le coordinate ed ulteriori informazioni relative all'indirizzo specificato nella richiesta.

```
<geocoding_results>
  <LEGAL_COPYRIGHT>
    <copyright_notice>
      Copyright (c) 2018 Geocode.Farm - All Rights Reserved.
    </copyright_notice>
    <copyright_logo>https://www.geocode.farm/images/logo.png</copyright_logo>
    <terms_of_service>
      https://www.geocode.farm/policies/terms-of-service/
    </terms_of_service>
    <privacy_policy>https://www.geocode.farm/policies/privacy-policy/</privacy_policy>
  </LEGAL_COPYRIGHT>
  <STATUS>
    <access>FREE_USER, ACCESS_GRANTED</access>
    <status>SUCCESS</status>
    <address_provided>via Ubertosa 1 Darfo Boario Terme 25040 Italy</address_provided>
    <result_count>1</result_count>
  </STATUS>
  <ACCOUNT>
    <ip_address/>
    <distribution_license>NONE, UNLICENSED</distribution_license>
    <usage_limit>250</usage_limit>
    <used_today>9</used_today>
    <used_total>9</used_total>
    <first_used>29 Dec 2018</first_used>
  </ACCOUNT>

  <RESULTS>
    <result>
      <result_number>1</result_number>
      <formatted_address>
        Via Ubertosa, 25047 Darfo Boario Terme BS, Italy, Italy
      </formatted_address>
      <accuracy>UNKNOWN_ACCURACY</accuracy>
      <ADDRESS>
        <street_name>Via Ubertosa</street_name>
        <locality>Darfo Boario Terme</locality>
        <admin_2>BS</admin_2>
        <admin_1>Lomb.</admin_1>
        <postal_code>25047</postal_code>
        <country>Italy</country>
      </ADDRESS>
      <LOCATION_DETAILS>
        <elevation>UNAVAILABLE</elevation>
        <timezone_long>UNAVAILABLE</timezone_long>
        <timezone_short>Europe/Rome</timezone_short>
      </LOCATION_DETAILS>
      <COORDINATES>
        <latitude>45.8817517238067</latitude>
        <longitude>10.1761667368550</longitude>
      </COORDINATES>
      <BOUNDARIES>
        <northeast_latitude>45.8807517008691</northeast_latitude>
        <northeast_longitude>10.1741666993356</northeast_longitude>
        <southwest_latitude>45.8837517524787</southwest_latitude>
        <southwest_longitude>10.1781667622502</southwest_longitude>
      </BOUNDARIES>
    </result>
  </RESULTS>
  <STATISTICS>
    <https_ssl>ENABLED, SECURE</https_ssl>
    <time_taken>0.34373307228088</time_taken>
  </STATISTICS>
</geocoding_results>
```


Il risultato ottenuto è un documento XML poichè nell'URL della richiesta è stato specificato tale formato per la risposta.

Le risposte dei web service possono essere fornite in diversi formati (specificando all'interno della richiesta il formato desiderato). Un formato di risposta molto utilizzato è, oltre a XML, il formato **JSON** (Java Script Object Notation). Questo formato è molto utilizzato per la trasmissione di dati da

web service poiché utilizza una sintassi più compatta rispetto a XML (spesso considerato eccessivamente verboso, ossia sovrabbondante, prolisso). Pur essendo un formato immediatamente utilizzabile nel contesto del linguaggio Javascript, JSON è ormai considerato un formato indipendente da quel linguaggio, infatti anche per JAVA, ad esempio, esistono apposite librerie per il parsing di documenti JSON.

Il formato dell'URL di richiesta può variare leggermente a seconda del web service al quale è rivolto, comunque richiede il rispetto delle regole sintattiche di qualunque URL :

- non si possono utilizzare gli spazi bianchi, al loro posto si utilizza "+"
- i caratteri non permessi negli URL sono sostituiti dai relativi codici ASCII espressi in esadecimale preceduti dal simbolo "%". Ecco i principali:

TABELLA 1

!	"	#	$	%	&	'	(	)	*	+	,	/	:
%21	%22	%23	%24	%25	%26	%27	%28	%29	%2A	%2B	%2C	%2F	%3A
;	<	=	>	?	@	[	\	]	^	`	{	|	}
%3B	%3C	%3D	%3E	%3F	%40	%5B	%5C	%5D	%5E	%60	%7B	%7C	%7D

Il processo che trasforma una stringa in una stringa codificata come URL viene detto **URL-encoding**, vi sono appositi metodi e funzioni, nei diversi linguaggi di programmazione per realizzare l'URL-Encoding.

Web API e programmable web

Con il termine **web API** (Application Programmable Interface) si indica **l'interfaccia** che consente di interagire con un web service, ossia lo strumento che un web service mette a disposizione di un utilizzatore affinchè questi ne possa utilizzare i servizi.

Una definizione chiara di interfaccia informatica è la seguente "*un contratto fra il fornitore di un servizio ed il fruitore del servizio nel quale si stabiliscono le modalità per la fruizione del sevizio*".

Il vantaggio delle webAPI consiste nella semplicità d'uso per chi le utilizza e nella sicurezza per il web service, infatti il web service può mettere a disposizione le proprie risorse verificando che chi vi accede rispetti determinati criteri di accesso e, soprattutto, senza che sia noto all'utilizzatore come e dove sono memorizzate le risorse. Non c'è un accesso diretto al database sa chi chiede le risorse.

In particolare le web API definiscono:

- **le specifiche di composizione degli URL che permettono di accedere ad una risorsa di un web service, generalmente contengono:**

- **i parametri associati alle richieste**
- **il formato delle risposte.**

Qui si possono trovare, ad esempio, le specifiche per le webAPI di geocode.farm: https://geocode.farm/geocoding/free-api-documentation/

L'insieme dei web service e dei loro servizi disponibili in rete viene denominato **programmable web**.

Il programmable web è dunque l'insieme dei servizi forniti dai web service. I web service mettono a disposizione i servizi attraverso le proprie webAPI.

A disposizione di chi?

A disposizione di sviluppatori che possono realizzare siti web, applicazioni desktop, APP, oppure altri web service, utilizzando i servizi forniti dai diversi web service. Questa tecnica di realizzazione di siti web, applicazioni e web service, viene indicata con il termine di **mashup** (mescolamento). Un esempio di questa tecnica è l'utilizzo, all'interno di un sito, delle mappe stradali fornite come servizio dal web service di Google Maps o da OpenStreetMap.

Ad esempio noi potremmo creare un web service o un sito web che, dati due indirizzi, calcola la distanza geometrica fra i due punti, e potremmo, per fare questo, realizzare il codice che svolge i calcoli (in Java, o in PHP,…) a partire dai dati ottenuti invocando le risorse di geocode.farm.

Esempio di mashup: la mappa del sito **https://www.casa.it/ .**

Altri esempi spiegati su questo video: https://www.youtube.com/watch?v=jHP-QJ-uSjw .

WEB SERVICE DI TIPO REST

La tipologia di web service che studieremo noi è indicata con il termine **REST** (**REpresentational State Transfer**).

Un'altra tipologia di web service esistente è quella dei web service **SOAP** (Simple Object Access Protocol), ma mentre SOAP è un vero e proprio protocollo, che si basa sullo scambio di messaggi fra un client e un server per l'invocazione di servizi remoti, e che stabilisce delle regole anche sintattiche relative ai messaggi che vengono scambiati, REST **non è un protocollo** ma uno **stile architetturale**. Uno stile architetturale stabilisce semplicemente dei **princìpi** che devono essere rispettati dall'architettura software, non "il come" tali princìpi devono essere rispettati. Lo stile architetturale REST è stato ideato nel 2000 da **Roy Fielding**.

Uno **stile architetturale** dunque è una definizione che si riferisce ad un livello di astrazione superiore rispetto ad un protocollo. Ad esempio un protocollo potrebbe stabilire: "la risposta dei dati deve essere in formato XML". Uno stile architetturale invece potrebbe stabilire il principio che "i client possono contattare i server ma non il contrario". (Stabilisce cosa devono fare gli elementi dell'architettura ma NON COME lo devono fare..)

I princìpi fondamentali alla base dello stile architetturale REST sono quelli di seguito elencati. Tali princìpi non sono necessariamente legati al web, sono principi astratti di cui però il World Wide Web inteso come piattaforma distribuita di elaborazione, risulta essere una applicazione concreta.

I Web service che rispettano IN MANIERA INTEGRALE i seguenti princìpi individuati da Fielding sono detti **RESTful,** possono esistere web service REST che implementano solo parzialmente questi princìpi, in tal caso tali web service **non sono RESTful.**

I PRINCIPI REST

1. **Client Server**
   Separazione dei ruoli degli elementi dell'architettura fra un elemento che richiede le risorse (client) e un elemento che fornisce le risorse (client).

2. **Caching**
   Per ridurre l'utilizzo della rete, nel percorso fra client e server è opportuno che siano presenti dei sistemi di caching che possono memorizzare le rappresentazioni delle risorse in modo che non sia necessario che un client debba richiedere risorse già precedentemente richieste.

3. **Interfaccia uniforme**
   Il principio più importante per l'architettura REST è l'utilizzo di **un'interfaccia uniforme.** Con questo termine si indicano in realtà 4 caratteristiche:
   - **Risorse identificate univocamente**
   - **Rappresentazione delle risorse**
   - **Messaggi autodescrittivi**

- **HATEOAS (Hypermedia As The Engine Of Application State)**

Descriviamo queste quattro caratteristiche.

Risorse identificate univocamente

I web service di tipo REST si basano sul concetto di risorsa, a differenza dei web service di tipo SOAP il cui elemento fondamentale è la remote call, ossia l'invocazione di una funzione o procedura remota.
Cosa è una **risorsa**? **Un qualsiasi elemento informatico che può essere oggetto di elaborazione**. Ad esempio: un libro, un articolo di giornale, un prodotto, un cliente, una mappa, un sensore, delle coordinate. Facendo un parallelismo con la programmazione ad oggetti si potrebbe assimilare una risorsa ad una istanza di una classe. Il principio dice che ogni risorsa deve essere identificata in modo univoco. Poiché il WEB possiede già un identificatore univoco per ciascuna risorsa esistente, l'URL, per i WEB service di tipo REST si è deciso di usare tale metodo di identificazione.

Esempi di identificativi di risorse in un ipotetico web service che fornisce risorse relative ad una attività commerciale:

- http://www.myapp.com/clienti/1234
- http://www.myapp.com/ordini/2022/98765
- http://www.myapp.com/prodotti/7654
- http://www.myapp.com/ordini/2011
- http://www.myapp.com/prodotti?colore=rosso

Questi URL ci permettono di "intuire" il tipo di risorsa identificata, ossia

- il cliente con id 1234
- l'ordine 98765 del 2022
- il prodotto con id 7654
- tutti gli ordini del 2011
- tutti i prodotti di colore rosso

comunque, in generale, per un web service, un URL è soltanto una stringa che identifica una risorsa. Un cliente potrebbe essere identificato anche dal seguente URL con valori meno "intuitivi" rispetto al precedente: *http://www.myapp.com/tgw34/2099ww*

Messaggi autodescrittivi:

I messaggi restituiti a un client devono contenere le informazioni necessarie per descrivere come il client deve elaborare l'informazione. Ad esempio se il messaggio inviato contiene del testo (json, cml, txt…) il messaggio deve specificare quale codifica di caratteri è stata

utilizzata per il testo (ASCII o altro), oppure se il messaggio contiene informazioni diverse da quelle testuali deve essere specificata nel messaggio la codifica che consente di interpretare i byte ricevuti (può essere utilizzato lo standard MIME, che è uno standard che permette di codificare immagini, dati binari, file compressi ecc.).

Rappresentazione delle risorse.
Abbiamo visto come sono identificate le risorse di un web service ma come vengono inviate al client? Il web service **non invia al client direttamente un record del proprio database ma una sua rappresentazione** in un uno specifico formato (XML, JSON, o anche HTML, o JPEG se è un'immagine). Spesso i web service hanno la possibilità di inviare una risorsa in diversi formati. In tal caso il client può richiedere che la risposta abbia uno specifico formato indicandolo nell'attributo *Accept* della richiesta http GET

```
GET /clienti/1234
HTTP/1.1
Host: www.myapp.com
Accept: application/vnd.myapp.cliente+xml
```

La rappresentazione della risorsa viene inviata dal web server al client nel body della risposta

HATEOAS (Hypermedia As The Engine Of Application State: Hypermedia come motore dello stato della applicazione).

HATEOAS significa che per consentire l'evoluzione di una applicazione, il web service mette a disposizione come unico strumento un insieme di collegamenti ipertestuali. Tali collegamenti devono quindi essere forniti dal web service all'interno della risorsa richiesta dal client.

Si ricorda che lo stato di un'applicazione lo possiamo definire, in pima approssimazione, "l'insieme dei valori assunti da tutte le sue variabili in un preciso istante".

Spieghiamo con un esempio.

Supponiamo che il client riceva da un web service una risorsa che rappresenta un ordine di un'azienda.

L'applicazione client potrà svolgere un parsing del documento ricevuto, estrarre l'identificativo dell'ordine e assegnare il valore dell'identificativo ad una variabile. Il client ora si trova nello stato 1 in cui: "variabile $ordine=12345678"

Per consentire al client di ottenere le risorse relative ai clienti o ai prodotti collegati a tale ordine, (quindi di modificare il proprio stato) è necessario che la risorsa *ordine* contenga i link ipertestuali per l'accesso alla risorsa *prodotto* o alla risorsa *cliente*, come nel seguente esempio:

Risorsa Ordine rappresentata in XML

```xml
<ordine>
  <numero>12345678</numero>
  <data>01/07/2011</data>
  <cliente rif="http://www.myapp.com/clienti/1234" />
  <articoli>
    <articolo rif="http://www.myapp.com/prodotti/98765" />
    <articolo rif="http://www.myapp.com/prodotti/43210" />
  </articoli>
</ordine>
```

In questo modo lo stato del client può evolvere grazie a nuove risorse che il client può ottenere grazie ai collegamenti. (*stato2*: variabile $ordine=12345678, variabile $id_prodotto=43210)

Il vantaggio dell'HATEOAS è il grande **disaccoppiamento** fra client e server, ossia il fatto che client e server possano modificarsi in maniera indipendente l'uno dall'altro. Per esempio, se in seguito ad una modifica un web service non fornisse più direttamente alcune risorse, esso potrebbe continuare a fornire i propri servizi indicando all'interno delle proprie risorse altri link identificanti, su altri web service, le risorse che esso ha smesso di fornire. Il tutto senza che siano necessarie modifiche software al client.

4. **Stateless** (senza stato).
   Il web service non memorizza lo stato dell'interazione con il client, pertanto ogni richiesta del client è come se fosse la prima richiesta, non è legata alle richieste precedenti, e quindi deve contenere tutte le informazioni necessarie affinchè il server la possa comprendere e servire. Questa è una caratteristica propria del protocollo HTTP. Attenzione però! Il fatto che il web service non mantenga lo stato non significa che l'intera applicazione client server non abbia lo stato, significa che il compito di mantenere lo stato è del client e non del server. Ad esempio nel caso presentato nell'esempio precedente sul HATEOAS, il client può mantenere lo stato memorizzando in opportune variabili il numero e la data dell'ordine, gli URL che gli consentono di accedere al cliente o ai prodotti.

   Il vantaggio dello stateless nei web service è la scalabilità. La **scalabilità** di un servizio informatico è la capacità di adeguare le proprie risorse alla variazione della quantità delle richieste. Un server senza stato è molto più scalabile. Esempio: arrivano troppe richieste al server e lui non le sopporta? Basta realizzare più istanze dello stesso servizio su più server e bilanciare il carico distribuendo le richieste su più server (questa operazione può essere svolta automaticamente da un dispositivi chiamato load balancer). Se fosse il server a

mantenere lo stato dell'interazione con il client, il servizio non potrebbe essere replicato rapidamente su un qualsiasi altro server.

ARCHITETTURA REST E METODI HTTP

Come già detto, REST non è un protocollo quindi i suoi principi non specificano come realizzare tale architettura. Teoricamente si potrebbe realizzare un'architettura REST con qualsiasi protocollo (FTP, SMPTP…) ma poichè il protocollo http fornisce già molti strumenti per lo scambio di dati sul web, i web service di tipo REST vengono realizzati praticamente sempre utilizzando tale protocollo.

Come detto, i web service REST si basano sul concetto di risorsa e abbiamo già spiegato ***come*** vengono identificate le risorse (con gli URL). Ma quali sono le operazioni che si possono svolgere su una risorsa? Sono quelle identificate dall'acronimo **CRUD (Create, Read, Update, Delete), e poiché** web service interagiscono con le applicazioni attraverso le tecnologie del web, ci dobbiamo ricordare che il protocollo http prevede dei metodi che svolgono esattamente queste operazioni.

Il protocollo **http** fornisce infatti dei metodi che specificano l'intenzione di svolgere queste quattro operazioni (**POST, GET, PUT, DELETE**), in un web service REST dunque il client per interagire con una risorsa (svolgere le operazioni CRUD) può utilizzare i metodi del protocollo **http rispettandone la semantica**! (cioè il significato)

Quindi per **leggere** (ottenere) una risorsa il client invia un richiesta http **GET**
per **scrivere** nuova risorsa il client invia un richiesta http **POST**,
per **modificare** una risorsa il client invia un richiesta http **PUT**,
per **eliminare** una risorsa il client invia un richiesta http **DELETE**

Metodo HTTP	Operazione CRUD	Descrizione
POST	Create	Crea una nuova risorsa
GET	Read	Ottiene una risorsa esistente
PUT	Update	Aggiorna una risorsa o modifica lo stato
DELETE	Delete	Elimina una risorsa

Si osservi che quando si digita un URL nella barra degli indirizzi di un browser stiamo implicitamente inviando una richiesta con il metodo http GET per ottenere la risorsa indicata (ad esempio il codice html di una pagina web).

Spesso però i metodi http NON SONO utilizzati rispettandone la semantica, ad esempio per eliminare una risorsa "cliente" l'applicazione client potrebbe inviare al web service una richiesta http GET specificando l'id_cliente del cliente da eliminare.

```
http://www.myapp.com/deleteCustomer?name=Rossi
```

Questo è l'utilizzo del metodo http GET per eliminare una risorsa anziché per ottenerla. In questo modo non stiamo sfruttando appieno le potenzialità del protocollo http perché con lo scopo di eliminare una risorsa, stiamo inviando un metodo http (GET) che serve per leggere una risorsa, quando invece http già espone il metodo DELETE per svolgere l'eliminazione.

Questo approccio non è conforme ai principi REST perché una stessa risorsa (il cliente Rossi) può essere invocata in modi diversi in base all'operazione che si vuole svolgere (deleteCustomer, UpdateCostumer,..), vìolando il principio di **identificazione univoca** della risorsa **dell'interfaccia uniforme**. Nei WEB service di tipo REST gli URL non dovrebbero mai contenere verbi perché il "cosa fare" è già specificato nel metodo utilizzato, l'URL deve solamente individuare la risorsa.

Nell'interfaccia uniforme il "cosa fare" è identificato dal metodo http utilizzato, l'oggetto dell'azione (la risorsa) è identificata dall'URL.

REST e RESTFull

Vi sono web service che utilizzano il protocollo HTTP ma non rispettano tutti i principi precedentemente indicati, tali web service sono indicati con il termine **REST** mentre quelli che implementano completamente i principi dello stile architetturale REST sono detti **RESTful**.

SOAP e REST

Spesso i web service vengono distinti fra SOAP (Simple Object Access Protocol) e REST. In realtà i due acronimi indicano cose diverse poiché SOAP è un vero e proprio protocollo (che usa HTTP semplicemente come protocollo di trasporto) mentre REST, come visto, è uno stile architetturale.

La differenza principale fra l'utilizzo di web service SOAP e web service REST è il grado di accoppiamento fra client e server. Come detto REST garantisce un elevato grado di **disaccoppiamento**, mentre SOAP prevede un elevato grado di **accoppiamento**. Il concetto principale di SOAP non è l'esposizione di "risorse" individuate da un URL ma l'esposizione di "metodi" (come nella programmazione ad oggetti) la cui esecuzione viene eseguita dal web service su richiesta del client (le richieste sono chiamate **RPC**= **Remote Procedure Call**). Come accade nella programmazione ad oggetti in cui per invocare un metodo è necessario "passare" alla richiesta opportuni parametri in maniera corretta, così per interagire con un web service SOAP, il client deve inviare le richieste al web service nel formato corretto specificato dal web service. Quindi c'è un

contratto rigido tra il client e il server. Questo comporta che, se uno specifico web service SOAP dovesse modificare la propria interfaccia, il client dovrebbe modificare il proprio codice. L'interfaccia uniforme dei web service di tipo REST e l'utilizzo dei metodi http rispettandone la semantica, invece, riducono questa interdipendenza fra client e server.