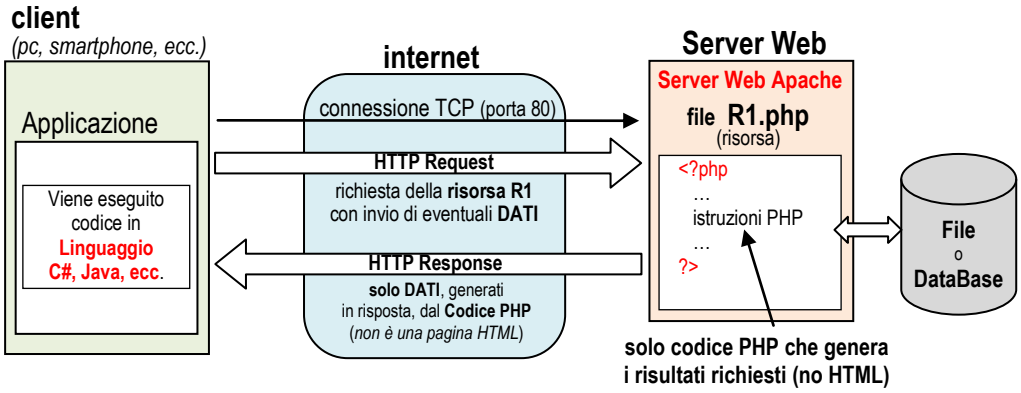
Protocollo HTTP: Applicazioni che Inviano/Ricevono Dati da un Server Web

Applicazioni e Server Web

Spesso, a scambiare dati con un Server Web, **non è un Browser**, bensì **un'Applicazione**.
Il client potrebbe essere:

- un **PC con Windows** e un'applicazione creata con **Visual Studio** in **linguaggio C#**
- uno **Smartphone Android** e un'app creata con **Android Studio** in **linguaggio Java**
- un **Dispositivo Integrato con Linux** e un'app creata con **NetBeans o Eclipse** in linguaggi quali **Java, C/C++, ecc.**



L'Applicazione effettua la Request su una risorsa, allo scopo di **ottenere dei Dati dal Server** (lo scopo non è quello di ricevere una *pagina web in HTML*, come nel caso del *browser*).

L'Applicazione può **inviare dati al Server** con i **metodi GET e POST** già visti.

Il Server Web può mettere a disposizione più "risorse" (es. *R1.php, R2.php*, ecc.) in base alle diverse funzioni da offrire all'applicazione.

Il Server Web, per soddisfare le Request, può accedere a **files (o database) "lato server"** da cui *recuperare le informazioni richieste* o in cui *memorizzare permanentemente i dati ricevuti*.

come Creare una Request
(App in C#)

Per **Creare una Request in C#**, si crea un'oggetto di classe **WebRequest**, utilizzando il suo **metodo statico Create**:

```
string MioURL = "http://itis.it/pg.html";
WebRequest Req = WebRequest.Create ( MioURL )
```

Il parametro **MioURL** deve essere una stringa contenente **l'intero URL** (inclusi il protocollo e gli eventuali parametri, in caso di *GET con parametri*).

N.B.: C# recupera automaticamente dall'URL il *protocollo* da usare (**http**), il nome di *dominio* (**itis.it**) e il nome della *risorsa* (**pg.html**) da usare nell'invio della request.

come Impostare una Request
(App in C#)

La **proprietà Method** (*stringa*) permette di scegliere il tipo di Request (GET, POST, ecc.)

```
Req.Method ( "POST" )
```
... oppure **"GET"**

In caso di POST, le **proprietà ContentLength** (*long*) e **ContentType** (*string*) consentono di specificare la **lunghezza** in byte dei dati da inviare e il loro **tipo**.

```
Req.ContentLength = 13;
Req.ContentType = "application/x-www-form-urlencoded";
```

come Preparare e Inviare Dati in una Request di tipo POST *(App in C#)*	I dati si inviano, al Server Web, utilizzando un oggetto di **classe Stream** (*flusso*) fornito dall'oggetto **Req** con il suo metodo **GetRequestStream**:
	`Stream FlussoReq = Req.GetRequestStream ( );`
	Sul flusso è possibile scrivere solo **sequenze di byte**, per cui è necessario definire un **Vettore di Byte** e convertire in esso i dati che si desidera inviare:
	`string MieiDati = "Base=7&Altezza=4";` *… dati nel formato form*
	`byte[ ] VetDati = Encoding.ASCII.GetBytes( MieiDati );` *… da stringa a vet.byte*
	Per **scrivere i dati** nel corpo della request, si usa il metodo **Write** del flusso:
	`FlussoReq.Write ( VetDati, 0, VetDati.Length );`
	A partire dal byte in posiz. **0**, saranno scritti *tanti byte quanto è lungo il vettore*, quindi **tutti**.
	Infine è necessario **chiudere il flusso** con il suo metodo **Close**:
	`FlussoReq.Close ( );`

come Recuperare la Response proveniente dal Server Web *(App in C#)*	Per **Recuperare la Response**, si utilizza un'oggetto di classe **WebResponse**, fornito dall'oggetto **Req** con il suo metodo **GetResponse**:
	`WebResponse Resp = Req.GetResponse ( );`
	Per **accedere ai dati** della response, si utilizza un altro oggetto di **classe Stream** fornito, questa volta, dall'oggetto **Resp** con il suo metodo **GetResponseStream**:
	`Stream FlussoResp = Resp.GetResponseStream ( );`
	Per sapere **quanti byte sono arrivati** nella response, si usa la sua proprietà **ContentLength**:
	`int N = Resp.ContentLength ;`
	Questo ci consente di definire un **Vettore di Byte** della giusta dimensione (**N**) e di **leggere in esso i dati della response** con il metodo **Read** del flusso:
	`byte[ ] VetRis = new byte[ N ];` *… vettore di N bytes*
	`FlussoResp.Read (VetRis, 0, N);` *… legge N bytes*
	Vengono letti, a partire dal primo (posiz. **0**) un numero pari a *N bytes*, cioè **tutti**.
	Infine si *converte da vettore di byte a stringa*:
	`string Risposta = Encoding.ASCII.GetString ( VetRis );`
	Nella stringa **Risposta** è presente l'intero testo ricevuto dal Server Web.
	Infine è necessario **chiudere il flusso** con il suo metodo **Close**:
	`FlussoResp.Close ( );`

il Formato dei Dati e la Serializzazione (JSON, XML, ecc.)	Il formato denominato **application/x-www-form-urlencoded** è utilizzato nell'invio di dati al Server Web, sia in caso di GET (parametri nell'URL) sia di POST (parametri nel Corpo), **quando i dati stessi possono essere posti facilmente nella forma *nome=valore***. L'invio e la ricezione di **strutture dati complesse** costringe a definire un **proprio formato** (con un appesantimento del codice) o a ricorrere a tecniche di **serializzazione** dei dati. La **Serializzazione** (e **Deserializzazione**) è la trasformazione dei **dati contenuti in un oggetto** (*stato dell'oggetto*) in una **sequenza di byte** (e viceversa). Molti linguaggi offrono **propri strumenti di serializzazione**. In alternativa, è possibile ricorrere a formati standard quali **JSON** o **XML** che sono, fra l'altro, formati testuali e quindi "leggibili".