

# TECNOLOGIE INFOMRATICHE PER IL WEB

Federico Mainetti Gambera

15 marzo 2020

## 0.1 LEZIONE 1 11/03/20

link <https://web.microsoftstream.com/video/58568b1d-5fc5-41c0-88f6-608e4b8f9f7a>

### 0.1.1 Calendario delle lezioni

<https://docs.google.com/spreadsheets/d/14ShTdkFCZ63MlyKP0LCSLPsPAeiu8D2sYVXSrkw9B6k/edit#gid=0>

### 0.1.2 Introduzione al corso

Non ci saranno prove in itinere per questo corso.

E' opportuno guardare i video relativi agli argomenti indicati prima di partecipare alla lezione.

Queste lezioni hanno lo scopo di aggiungere informazioni, chiarire dubbi e aiutarci a capire quali siano gli argomenti più importanti e i punti chiave.

I materiali video coprono interamente gli argomenti del corso.

### 0.1.3 Architetture

Il web è una piattaforma per sviluppo di applicazioni con un architettura molto particolare. Per architettura si intende l'insieme delle risorse (hardware, connettività, software di base, software applicativo). Le applicazioni web hanno una particolare conformazione della loro architettura che prevede tre livelli: Client, Middle-tier, Data-tier.

Le architetture sono cambiate molto negli anni. Se ne individuano tre grandi famiglie: Mainframe, Client-Server, Multi-tier (per esempio le applicazioni web).

Il compito di questo corso è quello di riuscire a programmare nel Middle-tier con qualche accenno al Client. In un'applicazione web il Client interagisce con il Middle-tier con un preciso protocollo, che non è il classico tcp-ip, ma il protocollo HTTP.

[Una lettura molto importante per il corso è il documento RequestforComment 1945, Tim Berners Lee (<https://www.w3.org/Protocols/rfc1945/rfc1945>), che rappresenta l'atto di fondazione del web.]

Il Client manda delle request al Server che invia delle responses. Le request del Client sono gestite tramite un applicazione detta User agent (browser).

Http è rivoluzionario per la sua semplicità: le richieste del Client e le risposte del server sono delle semplici stringhe.

Architettura delle applicazioni web: c'è un client (pc) con un user agent (browser) che emette richieste che vengono servite con delle risposte da un web server, detto anche HTTP server. L'architettura di cui però ci occupiamo è più complicata di così, perchè vede alle spalle del web server un application server (TomCat), che ha lo scopo di calcolare una risposta "personalizzata" secondo parametri e criteri precisi e di produrre una pagina web appropriata.

[Installare un'architettura completa sul pc che prevede tutti i livelli appena visti (guida nella cartella strumenti) (si può usare anche Intelij invece di Eclipse)]

Altri elementi dell'architettura sono il proxy e il gateway. Il proxy è un intermediario fra un client e un origin server, per definizione può comportarsi sia come server sia come client. Fa copie di risorse e le inoltra ai client e può essere anche utile per limitare gli accessi e gestire autorizzazioni. Il Gateway serve per far interagire un'applicazione web con applicazioni che non usano il protocollo HTTP, quindi il gateway traduce richieste HTTP per applicazioni che non lo comprendono, tipicamente usato per SQL.

### 0.1.4 HTTP

Come si identifica una risorsa? con l'URL (Uniform Resource Locator) che è una stringa formata in maniera molto semplice: prefisso (protocollo), indicazione della macchina fisica in cui è presente la risorsa, una eventuale porta in cui la macchina ascolta le richieste e un Path. Gli URL sono estremamente semplici, e utilizzano concetti già noti in precedenza.

HTTP request: è una banale stringa che contiene una request-line, degli header e un allegato (per gli upload) facoltativi. la request-line contiene tre informazioni: il metodo (funzione) della richiesta, l'URL, la versione del protocollo usata. I metodi di richiesta sono principalmente due, GET e POST; questi metodi possono essere visti come chiamate di funzione.

HTTP response: anche questa è una banale stringa, che, ancora più semplicemente contiene un codice di stato (es. 404 file not found), degli header facoltativi, e un allegato.

La conseguenza di un protocollo così semplice è che con un solo client si può interagire con tutti i backend. La complessità si sposta però nell'application server.

Gli header sono informazioni opzionali a cura del browser, trasmesse come fossero parametri che aggiungono informazioni alla request o alla response.

### **0.1.5 CGI, Common Gateway Interface**

Tecnologia ormai morta e superata.

CGI è una convenzione che standardizza un certo numero di variabili d'ambiente (condivisibili fra più processi) grazie alle quali il web server può smontare una richiesta HTTP e salvarla in una zona di memoria alla quale un processo di un applicazione esterna può accedere.

Per invocare un applicazione tramite CGI, l'URL della richiesta HTTP deve presentare un path che porta a un applicazione eseguibile. Una volta ricevuta una richiesta HTTP che rimanda a un eseguibile, il web server smonta la richiesta, ne preleva le informazioni (parametri) e le salva nelle variabili d'ambiente, successivamente fa eseguire all'application server il programma indicato nel path dell'URL. L'application server preleva le informazioni dalle variabili d'ambiente e costruisce un file HTML che poi verrà mandato come risposta al client.

CGI ha due grandi difetti, il primo riguarda la sicurezza, in quanto i file eseguibili erano direttamente accessibili, il secondo è che siccome i processi dell'application server muoiono dopo ogni esecuzione, non c'è modo di creare un sistema che mantiene una sessione attiva. Inoltre le prestazioni di CGI sono molto basse, per esempio, siccome i processi muoiono continuamente, c'è un continuo bisogno di stabilire connessioni coi database, che è un operazione onerosa.

## 0.2 LEZIONE 2 12/03/20

link <https://web.microsoftstream.com/video/58568b1d-5fc5-41c0-88f6-608e4b8f9f7a>

### Servlet

Nasce l'esigenza di generare contenuti dinamici per le applicazioni web, inizialmente, infatti, HTTP era concepito come protocollo per scambio di documenti.

Il gateway è un elemento imprescindibile che aumenta le capacità di un'applicazione web. Abbiamo visto che la versione arcaica di gateway era il CGI, che rappresenta il modo più semplice e che usa le variabili d'ambiente per adempiere al suo compito. Abbiamo però visto che CGI ha diversi problemi, perciò è necessario trovare una nuova soluzione.

Vediamo ora il meccanismo più popolare (almeno fino a qualche anno fa) per realizzare i requisiti di produzione dinamica di contenuti. HTTP è nato per essere semplice, ha solo due metodi, GET e POST, non prevede l'identificazione di un client, tutte le richieste sono uguali. Per un Web Server la nozione di sessione non esiste. I metodi GET e POST sono richieste parametriche, cioè alle quali si possono aggiungere informazioni sotto forma di parametri stringa. Il metodo GET i parametri li mette nella query-string, il metodo POST li mette nel body. HTML ha un costrutto "form" che serve al browser per costruire un richiesta dove i parametri sono complicati, per esempio richieste dove un parametro è un file. Il metodo POST è quindi spesso usato con il costrutto HTML form.

Java Servlet è un modo nuovo, rispetto a CGI, di strutturare l'applicazione che riceve la request e formula la response.

Al programmatore di un Servlet si chiede di programmare una classe, egli dovrà lavorare all'interno di un Framework, cioè una soluzione parziale a una serie di problemi con caratteristiche comuni. Il framework è uno schema, una soluzione parziale che omette la parte variabile di una serie di applicazioni con qualcosa in comune. Java Servlet è appunto un framework, tutte le applicazioni web hanno in comune tutto il protocollo HTTP. Dal framework ci aspettiamo quindi di non dover programmare la gestione delle request e delle response.