

# Programmazione Web

---

---

## Programmazione Web

18/02/2019

Introduzione

Link utili

Linguaggi utilizzati

Lato client

Lato server

Strumenti utilizzati

Esercitazioni

Programma del corso

Struttura esame

Test Scritto 40%

Progetto 60%

Presentazione Progetto

Bibliografia

WEB Application: Schema Generale

Transito dell'informazione

Informazioni fondamentali in una Applicazione WEB

19/02/2018 LAB 01

GIT

Version Control Systems

Local Version Control Systems

Central Version Control Systems

Distributed Version Control Systems

Stati GIT

Creazione e Clonazione Repository

Creazione

Clonazione

Branches

Aggiunta Branch Nuovo

Cambio del Branch

Aggiunta file

Conferma versione

Modifica dell'ultimo messaggio

Modifica di più commit

Unione di due versioni

Ottenere nuove modifiche dal server

Controllare se ci sono nuove modifiche dal server

Inviare le modifiche al server

GitLab

Generare una chiave SSH

- Fornire key pubblica a GitLab
- Fornire le proprie informazioni a GitLab
  - Autenticarsi globalmente
  - Autenticarsi solo sulla Git Directory corrente

25/02/2019

- Introduzione ai Sistemi Distribuiti
  - Sistemi distribuiti: Definizione
  - Obbiettivi sistemi distribuiti
    - Condivisione delle risorse
    - Trasparenza della distribuzione
    - Apertura verso espansione dei servizi
    - Scalabilità
    - Fault Tolerance
    - Implicazioni dei sistemi distribuiti
    - Cluster di calcolo
    - Distributed Computing as an Utility

- Introduzione alle socket
  - Sockets: Definizione
  - Composizione socket
  - In JAVA:

26/02/2019 - Lab

- Disattivare proxy https in gitlab
- Versioni
- Package
- Creazione del progetto con maven

04/03/2019

- Visualizzazione pagine web
  - Storicamente
  - Come funziona?
  - Elemento HTML
  - Attributi HTML
  - Commenti HTML
  - Principalmente
  - Intestazioni
  - Tipi di carattere
    - Caratteri speciali
  - Liste
  - Elementi di formattazione
  - Meta Dati
  - Link interni(ed esterni)
  - Form
  - Script e Style
  - Standard da utilizzare
  - Validatore
  - HTML DOM
    - Utilizzo:

CSS

- Sintassi

Cascading

Tipi di elemento

Per gestire un elemento all'interno di un elemento

Gestire più di un elemento

Pseudo Selettori

Colori

Larghezze e percentuali

Composizione di un Box

Testo: Font

Testo: Spacing

Layout delle pagine

Mobile First Pages

05/03/2019-Laboratorio

11/03/2019

Web Architectures

Architetture multi-tier

Versione di base

Web Browser

Web Server

HTML

URI e URL

URL o URN?

Mailto URI(URL)

HTTP

Messaggio di request:

HTTP e HTTPS

Certificati HTTPS: Tipi

Validità

Web Arch: Estensioni

CGI

Application Server

Architetture multi-tier

Network e architetture distribuite

SOA

Web Services

Cloud Computing

12/03/2019- Laboratorio

25/03/2019

Cookies e Sessioni:

Session state information

Cookies

Servlet Cookies API

Ottenere i cookies

Aggiungere un cookie

Session Management

Accedere ai dati di sessione

Metodi HttpSession

Scadenza Sessioni

- Cookie di terze parti
  - Cookies: rischi
    - Soluzioni
- URL Rewriting: Servlet
- Request Dispatcher
  - Metodi
    - Passare i dati
- Session tracking remoto e locale
- Filtri
  - Schema di funzionamento
    - Filtro in ingresso
    - Filtro in uscita
  - Schema logico
  - Ipotesi di utilizzo
- 26/03/2019 - Laboratorio
- 01/04/2019
  - Listener
  - Database nelle App Web
    - Tipi di database
    - Database SQL
    - Database NoSQL
      - Schemi dinamici
      - Varietà dei dati
      - Cluster ad alta Availability
      - Open Source
      - Non dipendenza da SQL
      - Tipi di NoSQL
      - Key-Value data model
      - Column Oriented
      - Document Oriented
      - Graph Database
      - Problemi NoSQL
  - JDBC
    - Architettura
  - Microsoft CLI ODBC
  - Transaction
  - Schema classi Java
  - Step di uso per DB
  - Apache Derby
- 02/04/2019 - Laboratorio

---

---

**18/02/2019**

---

## Introduzione

#

L'obiettivo del corso è avere un contenuto Web **scalabile e accessibile da chiunque**.

### Link utili

- [Sito corso](#)
  - [Canale Telegram](#)
  - [Gruppo Telegram](#)
- 

## Linguaggi utilizzati

#

### Lato client

- JS
- HTML
- CSS
- Framework(Bootstrap)

### Lato server

- Database
  - Java Servlet
- 

## Strumenti utilizzati

#

- Netbeans/IntelliJ
- Tomcat/GlassFish (Server Side)
- Database Relazionale
  - Derby
  - PostgreSQL
  - MySQL
- GIT e *Version Control*

- Strumento **importante** per la gestione concorrente tra più persone di un progetto.
- 

## Esercitazioni

#

- Martedì 14:30-16:30 Dispari aula **b106**
- Venerdì 13:30-15:30 Pari aula **b106**

Uso di Linux: Consigliato, problemi di amministrazione su Windows.

---

## Programma del corso

#

- Cenni su reti

- Uso di socket, protocolli di comunicazione base come HTTP, etc etc

- Architetture software per App Web

- HTML, CSS, JS, JSON e XML ;
- Programmazione JAVA e cenni di Programmazione Concorrente.

- Pattern di progettazione del software

- Chi fa cosa all'interno della mia applicazione? Avremo *libertà di scelta* per ciò che serve nella nostra applicazione, **dovrà comunque seguire delle regole basi.**

## Struttura esame

#

### Test Scritto 40%

- Domande a crocette e alcune domande aperte.

- Uso di parziali: Aprile e Giugno, **accettato solo con  $\geq 18$** . Il voto **NON** deve necessariamente essere positivo per presentare il **progetto**.

### Progetto 60%

- Progetto di Gruppo ;

- Parte

- pubblica con
  1. Ricerca delle **informazioni**;
  2. **Autenticazione** dei *dispositivi/utenti* ;
  3. Possibilità di social/**feedback**.
- privata con
  1. **Amministrazione/Zona amministrativa**;
  2. Accesso **Gestore**.
- Generazione **automatica documenti**
  - Generazione di file
    1. **PDF**
    2. **Excel**
    3. **JSON/XML**

**Scaricabili** dall'utente.

- Avere informazioni **pseudo reali** presi dal web.

L'applicazione deve essere **simil-vendibile**, cioè deve avere una app completa che mi permette di essere venduta. Il progetto **varia in base alle persone che ci lavorano**.

N° Persone	Descrizione
2 a 4	<i>Normale richiesta .</i>
5 a 6	<i>Come 2 a 4 + <b>Internazionalità</b> del sito web stesso.</i>

Gruppi da 2 a 6 Persone .

### Presentazione Progetto

Valutazione in base a

1. **Compatibilità device;**
2. **presentazione effettuata;**
3. **struttura a lato server.**

La presentazione può essere fatta da **tutti**, ma se ci sono assenti devono **ripresentare successivamente** da soli.

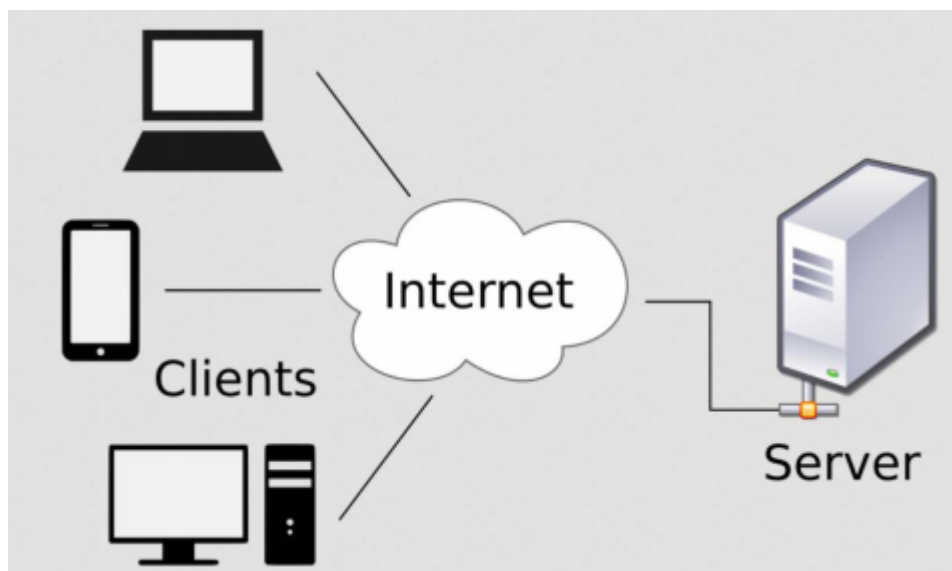
## Bibliografia

#

Troppo vasta, ci sono **tanti libri** e **tanti argomenti** estratti da essi.

## WEB Application: Schema Generale

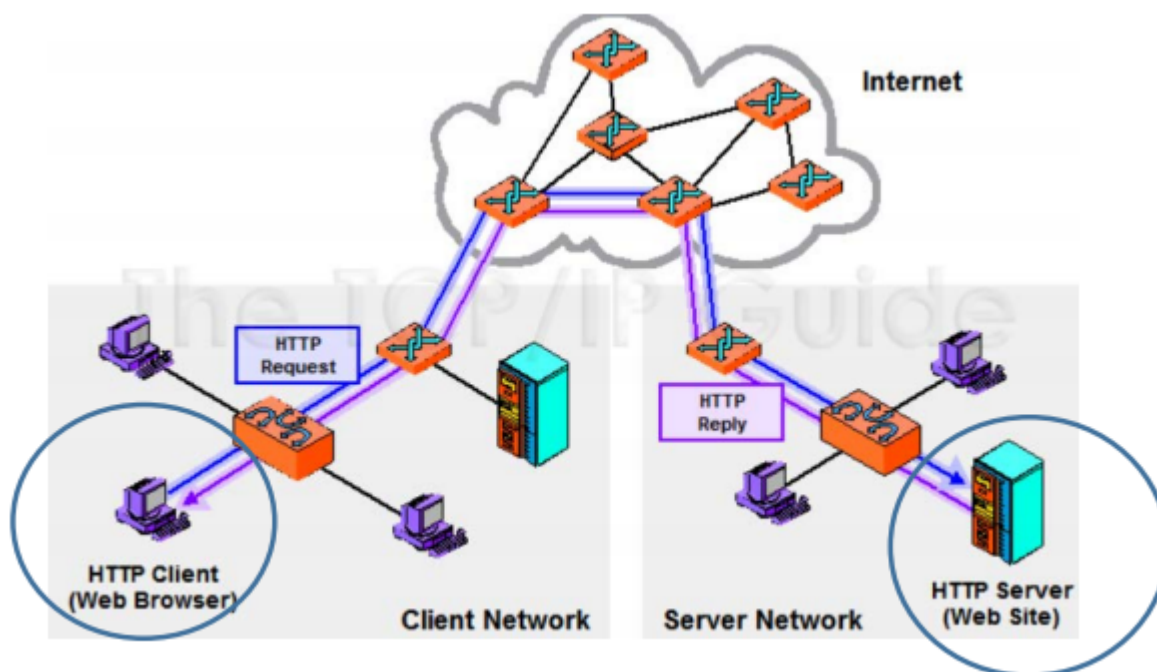
#



Bisogna fornire **sicurezza**, Servizi **real time** ed evitare **Leak di informazioni**. L'applicazione non deve creare problemi a **terzi**. Sul lato **Server** ci sono informazioni *personali* che non devono uscire dal server web stesso. Gli utenti tendono ad utilizzare **la stessa password per più servizi**.

Il furto di informazioni sensibili è un problema grave, forse il peggiore, per un'azienda.

## Transito dell'informazione



Informazione	Intercettabile?	Dati
Non Cifrata	Sì.	In chiaro, leggibili.
Cifrata	Sì.	Illeggibili.

L'informazione, **cifrata o meno**, può essere quasi sempre intercettata. Ovviamente, se cifrata, è meno probabile che venga **decifrata** e che i dati ottenuti vengano utilizzati in modo malevolo.

## Informazioni fondamentali in una Applicazione WEB

In un sito web devono esserci, **per legge**:

1. **Informativa sulla privacy** (Che deve rispettare GDPR) ;

Obbliga a chi fornisce il servizio di dire **come utilizza i dati** e **come vengono raccolti**.

Devo fornire anche informazioni sui **servizi terzi**. Ad esempio:

- Se utilizzo un Widget con **Google Maps**, nell'*informativa della Privacy* debbo inserire anche un **collegamento all'informativa sulla privacy di Google Maps**.

2. **Partita IVA**



Per vedere un quantitativo di informazioni utili in una pagina web **Inspect** → **Network** e ricarico la pagina. Grazie a questo strumento riesco a vedere tutti gli elementi presenti in questa pagina.

---

---

19/02/2018 LAB 01

---

## GIT

#

**Controllo di versione:** modo per gestire i cambiamenti in un sistema, non necessariamente legati ad un software.

Mentre un modo molto utilizzato è copiare le cartelle del nostro codice, questa è una **cattiva** gestione di versione.

Attraverso **GIT** invece ho un controllo migliore.

## Version Control Systems

#

Creati per avere un modo proprio e funzionale di gestire i cambiamenti.

### 1. Reversibilità:

Capacità **principale** di un **VCS**, ci permette di ritornare in qualsiasi punto delle nostre versioni salvate. Utile se effettuiamo cambiamenti che modificano drasticamente il funzionamento della nostra versione.

### 2. Concorrenza:

La concorrenza ci permette di avere **diverse persone che fanno cambiamenti allo stesso progetto**, facilitando il processo dell'integrazione dei cambiamenti nei pezzi di codici alle quali lavorano due o più developers. **Permette una gestione dei conflitti se più persone lavorano allo stesso file.**

### 3. Annotazione:

Possibilità di commentare ogni versione e modifica effettuata nel codice.

---

## Local Version Control Systems

#

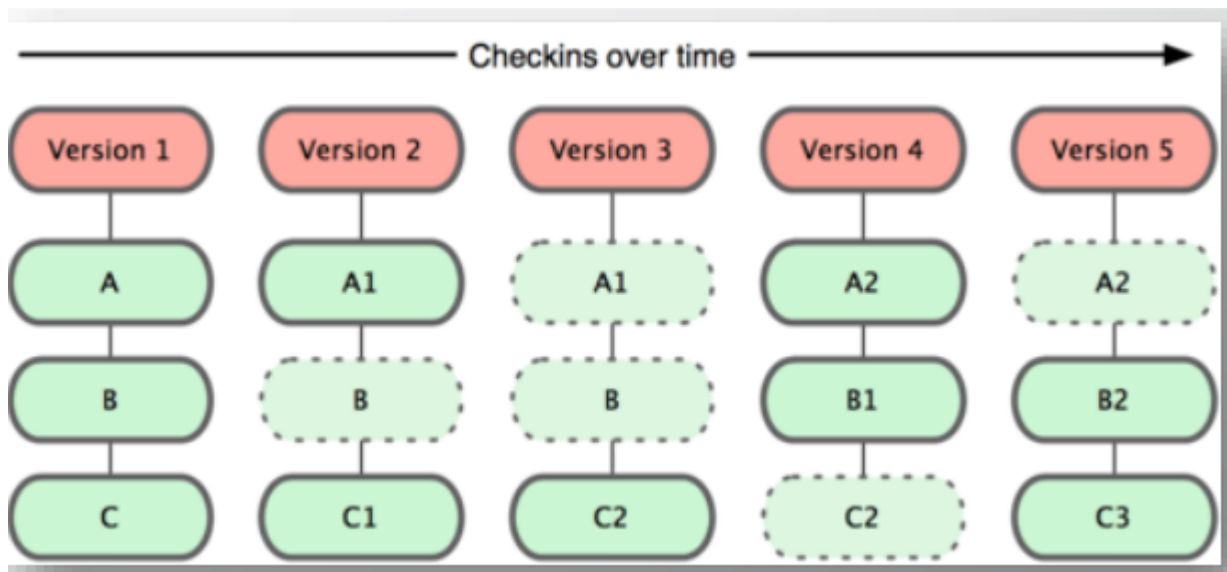
Versione "**cattiva**" della gestione delle versioni. **Copio le cartelle in base alla versione.**

---

## Central Version Control Systems

#

**SVN** o sistema a **Subversion** della *Oracle*, dove ho un server centrale e i computer locali fanno una copia del file sul quale lavorano, che poi **inviano al server**.



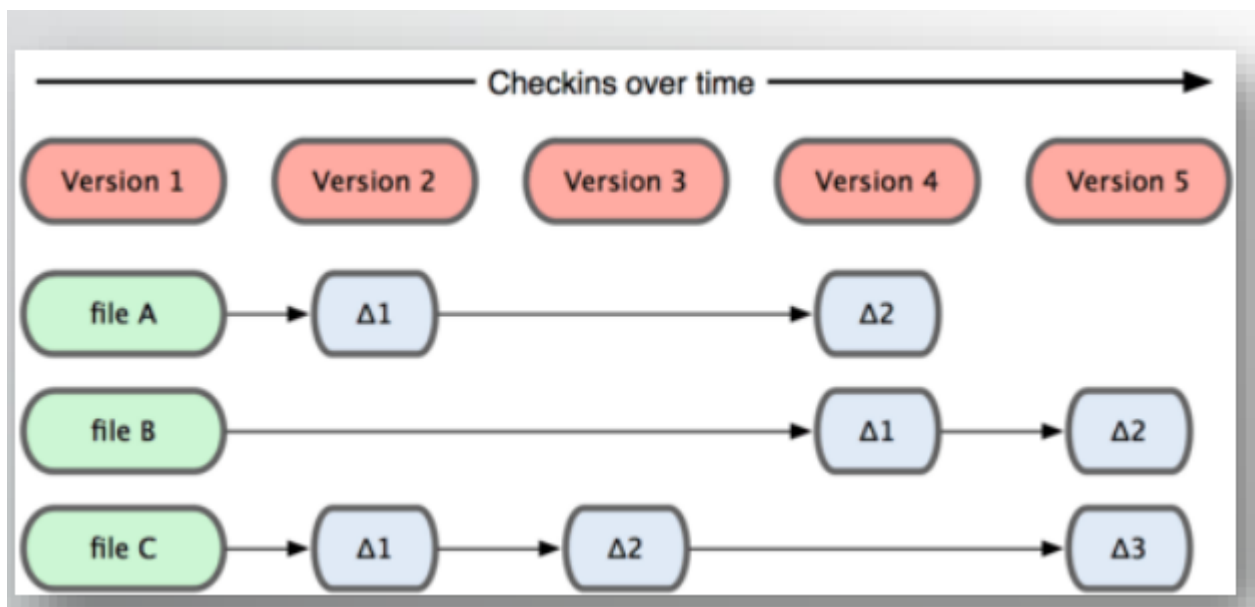
#### Dettagli

1. Il server farà il merge.
2. L'utente non ha accesso alla repository, ha accesso solo all'ultima versione.
3. L'utente può lavorarci solo se connesso al server.
4. È in rete, quindi tende ad essere più lento di un sistema distribuito.
5. Nel server effettua copie dei file modificati per ogni versione.

## Distributed Version Control Systems

#

GIT è un sistema distribuito, dove, al contrario del CVCS non ha bisogno di una continua connessione al server, perchè ha una copia identica della repository centrale e può gestire tutte le versioni senza essere collegato.



#### Dettagli

1. Dopo aver effettuato le modifiche, effettuo **pull request** o **push** diretto.
2. Lavora nei filesystem, non ha bisogno di una connessione al server per lavorare.
3. GIT salva solo i **DELTA** dei file, cioè le **modifiche**, quindi è molto più *leggero e veloce* .

## Stati GIT

Modifica i file nella directory, aggiungo uno snapshot dell'area attuale(area di **Staging**) ed effettuo un **COMMIT** nella git Repository.

**Working directory:** Directory Lavorativa, directory dove ho la mia repository.

**Staging Area:** Area dove ho effettuato le modifiche, ma non sono ancora confermate

**Git Directory:** Directory contenente i file di git. Con un **commit**, posso aggiornare questo descrittore e aggiornare anche la working directory .

## Creazione e Clonazione Repository

### Creazione

```
git init
```

*Comando che, nella cartella attuale, mi permette di **inizializzare** un file git per gestire le versioni nella directory attuale.*

### Clonazione

```
git clone https://link.xyz/miofile.git
```

*Mi permette di **clonare** la repository completa, con il file git completo con tutte le modifiche delle versioni salvate.*

## Branches

Git funziona con un sistema a "**rami**" dove possiamo avere dei **path di development indipendenti**.

Ogni branch **coesiste nella stessa directory**, ma hanno storia delle versioni **NON** in comune.

Questo mi permette di poter modificare un branch senza modificarne gli altri, potendo effettuare modifiche esclusivamente ad una versione/funzione del sistema.

**Master:** branch di default di git.

L'utilità è la possibilità di fare modifiche senza toccare la versione principale.

### Aggiunta Branch Nuovo

```
git branch NOMEDELBRANCHNUOVO
```

*esempio:*

```
git branch testing
```

Clono il branch *master* e lo chiamo *testing*

### Cambio del Branch

```
git checkout NOMEDELBRANCH
```

*esempio:*

```
git checkout testing
```

Imposto il mio **puntatore HEAD** su quale **BRANCH** spostarmi.

HEAD: Puntatore che punta al branch attuale dove mi trovo.

### Aggiunta file

```
git add FILEDAAGGIUNGERE
```

*esempio:*

```
git add file1.txt file2.jpg ../file3.mov
```

```
git add miofile.txt
```

```
git add *
```

Mi permette di **aggiungere** al commit(versione) attuale i file che voglio. Questi file saranno i file nuovi, i file che ho aggiunto o modificato.

### Conferma versione

```
git commit -m "MESSAGGIO"
```

*esempio:*

```
git commit -m "fix something, doing anything"
```

Ho **creato una versione**, lasciando un messaggio per chi leggerà.

### Modifica dell'ultimo messaggio

```
git commit --amend -m "messaggio modificato"
```

Utile se mi sono accorto di aver scritto qualche eresia nel messaggio dell'ultimo commit.

## Modifica di più commit

```
git rebase
```

Comando **avanzato**, [leggi la guida per saperne di più](#).

## Unione di due versioni

```
git merge BRANCH
```

Unisco la versione in **BRANCH** sul **branch attuale**.

Se voglio copiarle in **master**

```
git checkout master
```

```
git merge NOMEBRANCH
```

[Link utile](#)

## Ottenere nuove modifiche dal server

```
git pull
```

Vado ad ottenere dal server le **modifiche nel branch attuale** se ce ne sono .

## Controllare se ci sono nuove modifiche dal server

```
git fetch
```

```
git status
```

**FETCH** ci permette di ottenere solo la lista dei **delta** nel **branch attuale** (e non ottiene le modifiche salvandole in locale come **git pull** ).

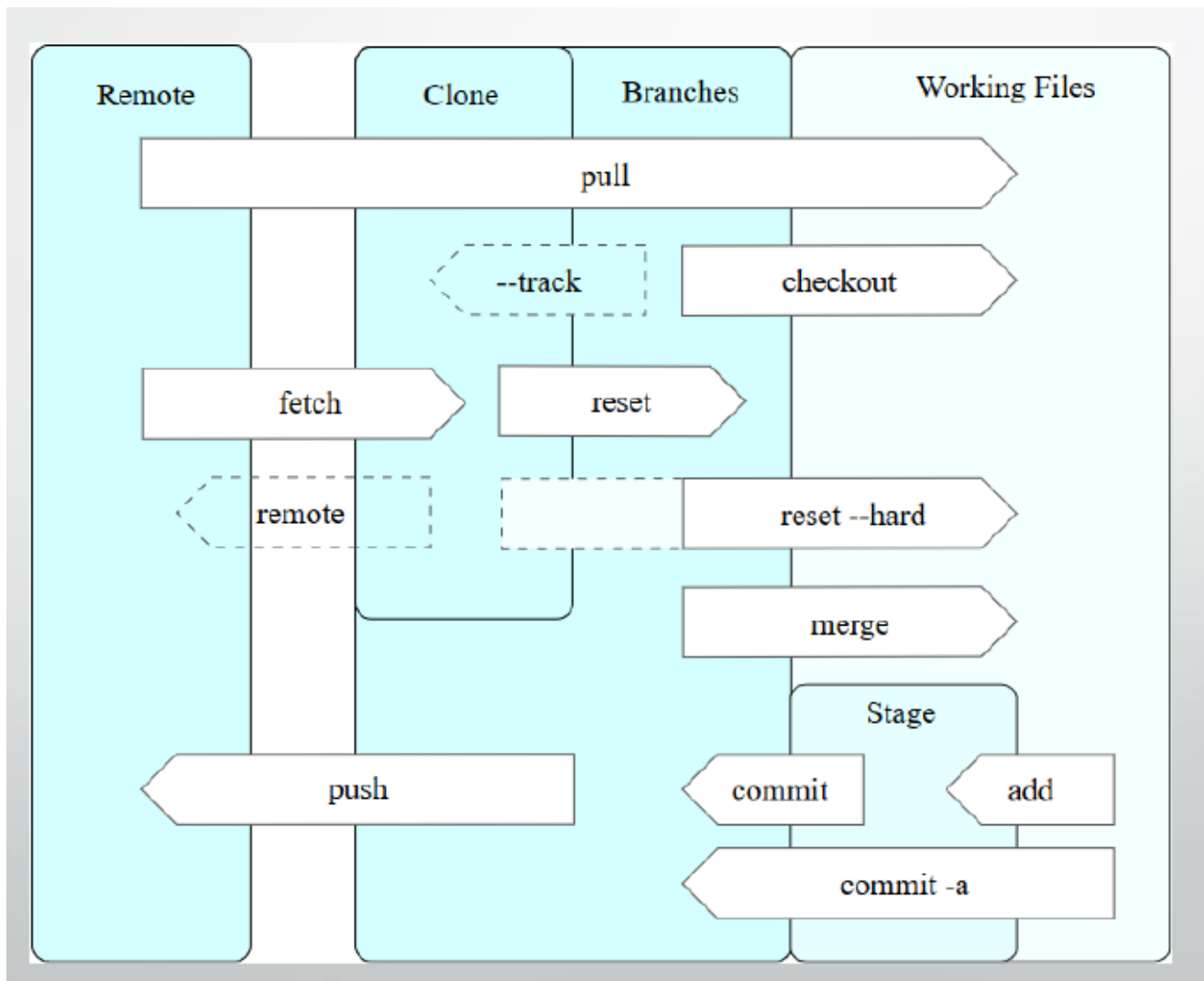
**STATUS** ci permette di controllare quanto è **avanti** o **indietro** la nostra versione con la versione sul server.

## Inviare le modifiche al server

```
git push
```

Invia al server le modifiche e i file modificati che differiscono (**delta**) dalla versione su server.

**ATTENZIONE**, questo comando funziona solo se siamo **avanti** di modifiche dalla versione sul server. Se la versione sul server è **aggiornata (più nuova della nostra)**, cioè ha modifiche che noi non abbiamo, **DEVI prima ottenere le modifiche e fixare eventuali conflitti**.



## GitLab

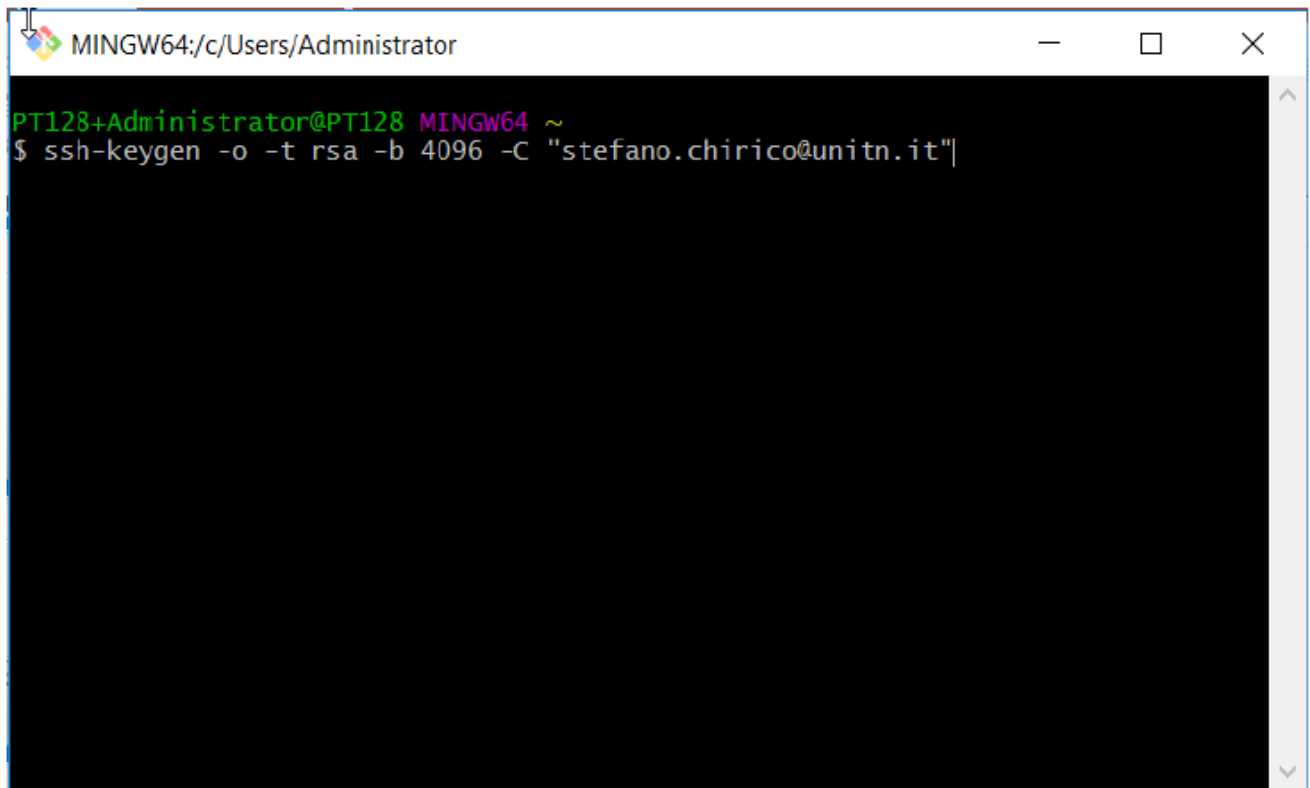
#

Tutorial fornito dal professore.

Per utilizzare gitlab dobbiamo utilizzare una chiave SSH.

### Generare una chiave SSH

Dopo aver scaricato GIT



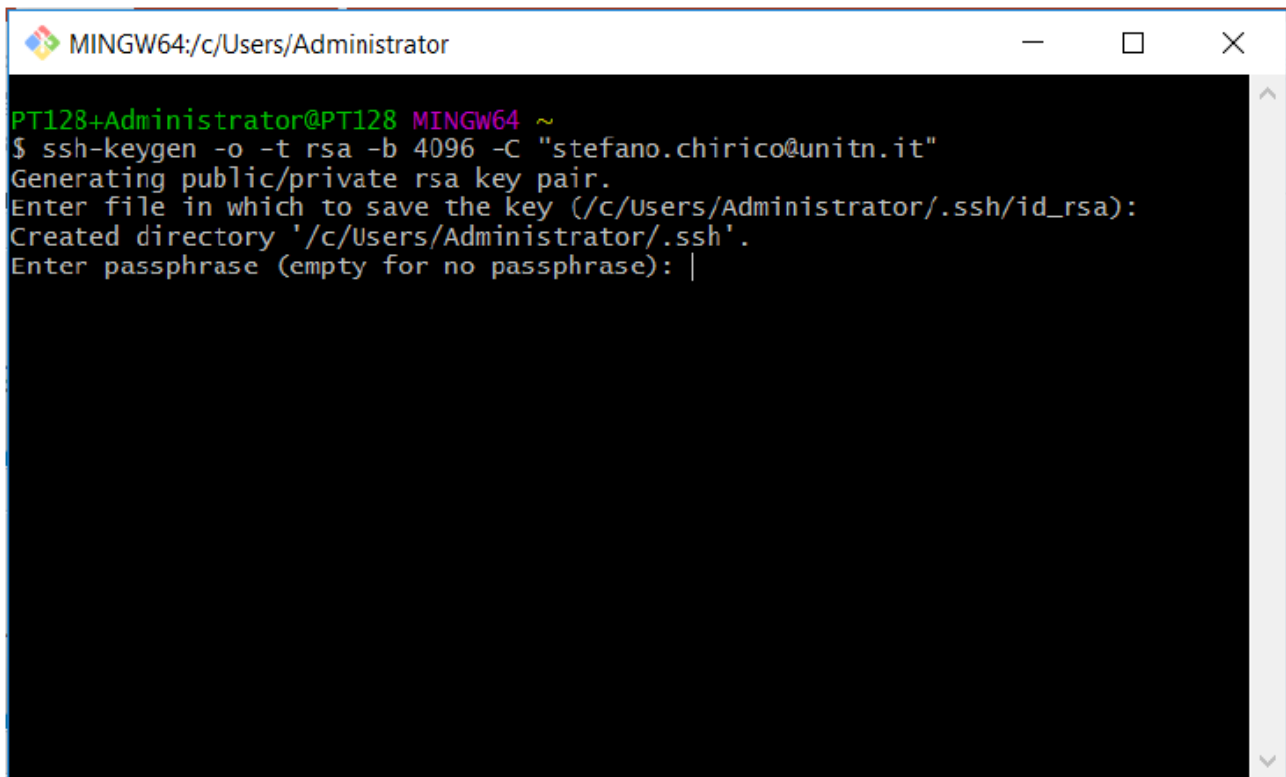
```
PT128+Administrator@PT128 MINGW64 ~  
$ ssh-keygen -o -t rsa -b 4096 -C "stefano.chirico@unitn.it"
```

```
ssh-keygen -o -t rsa -b 4096 -C "email@dominio.xyz"
```

Dove

- **-t** è il tipo di cifratura utilizzata (RSA);
- **-b** è il numero di bits con la quale verrà generata;
- **-C** è il commento che daremo per ricordarci a cosa è collegata.

Se non abbiamo un path con delle key ssh, ci chiederà dove metterle. **Lasciamo il percorso base.**



```
MINGW64:/c/Users/Administrator
PT128+Administrator@PT128 MINGW64 ~
$ ssh-keygen -o -t rsa -b 4096 -C "stefano.chirico@unitn.it"
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/Administrator/.ssh/id_rsa):
Created directory '/c/Users/Administrator/.ssh'.
Enter passphrase (empty for no passphrase): |
```

Ci chiede una password per proteggere la key. Per **NON** inserirla basterà dare un invio.

## Fornire key pubblica a GitLab

Dentro alla cartella di prima avremmo due file:

1. `id_rsa`
2. `id_rsa.pub`

noi dobbiamo **aprire la key pubblica e fornirla a gitlab**.

**ATTENZIONE** a fornire a gitlab la chiave **PUBBLICA** e **non** quella privata.

La chiave pubblica è quella con estensione `.pub`

Un modo per copiare la key è con il seguente comando effettuato **nella cartella dove abbiamo la key** `id_rsa.pub` :

```
cat id_rsa.pub | clip
```

dove

1. `cat` legge in uno stream i dati dal file `id_rsa.pub`
2. concatenando il pipeline con `clip` li copio sulla **clipboard**.

Copio su gitlab la key pubblica in *User Settings* → *SSH Keys* .



## SSH Keys

SSH keys allow you to establish a secure connection between your computer and GitLab.

### Add an SSH key

To add an SSH key you need to [generate one](#) or use an [existing key](#).

#### Key

Paste your public SSH key, which is usually contained in the file '~/.ssh/id\_rsa.pub' and begins with 'ssh-rsa'. Don't use your private SSH key.

1

```
/q2CP5+DcbmtQZT9l6V371XX/OVGbEx87A1udjzclP  
/ho3wNVVvJFvbqY8yFcVxB2sMgRGcJf0UUH6ThxkCzbP+fzgtg5et2CzXe0PuOic6GiMnTL5U0wvv  
AkE4/S2HLI2QyohzkMumfqpjMK7LWxKt  
/ntvh6oHZvs0PjUTfRjC6Hr9n3JhEDfhSt7dASLmRaWfxqfbFmzxu0xDyDY8swgBqY1heblLFSYdUU  
1KxqYPbYHLi++2CnqPgRnWjS21B6WLFVAUenZcWpJjPMPHYatQvidbMHBxAWXuQtaBn6D3VuBI  
W1N8QBLBrtDiPIBaqJuGWug62ALh2yllS7kcsDEeT8FCvRmV0nCxlpc5keQqVo+IcfjHv6UmfO4zZ  
QprZ/8TYkC62EtCgUyU/BTcOc2Ms3cHRXGT4mY+4NStZdGw14HcDux  
/ElAht72aZD7NcDpz4J3JywmJcNvBrIVq4Tw== stefano.chirico@unitn.it
```

#### Title

stefano.chirico@unitn.it

Name your individual key via a title

2

Add key

### Your SSH keys (0)

There are no SSH keys with access to your account.

Per altre informazioni riporto alle [Slide del professore](#).

## Fornire le proprie informazioni a GitLab

*Prima di clonare*, soprattutto se **ho creato una repository privata** (vedi slide del professore), essendo tale non è accessibile senza una effettiva autenticazione.

Al push, pull o clone ci verrà chiesta un autenticazione se la repository è privata.

Nonostante questo, per effettuare modifiche sul file git **dobbiamo dire chi siamo**. (altrimenti non sa chi ha modificato il file git).

### Autenticarsi globalmente

Posso impostare un **username globale**, cioè per ogni repository:

```
git config --global user.email "mia@email.xyz"
```

```
git config --global user.name "Nome Cognome"
```

### Autenticarsi solo sulla Git Directory corrente

Effettuabile solo in una cartella con un **git inizializzato**:

```
git config user.email "mia@email.xyz"
```

```
git config user.name "Nome Cognome"
```

---

25/02/2019

---

## Introduzione ai Sistemi Distribuiti

#

Sistema che funziona a livello globale è sicuramente distribuita.

*google, facebook etc etc*

Essendo arrivati al **limite fisico** dei transistor dei processori, la potenza di calcolo è data da una moltitudine di processori. Oltretutto posso avere più calcolatori che lavorano sullo stesso processo.

### Sistemi distribuiti: Definizione

1. **Da un punto di vista operativo** Un sistema distribuito è un sistema in cui i componenti hardware e/o software, situati in computer collegati in rete, comunicano e coordinano le loro azioni solo passando messaggi.
2. **Da un punto di vista dell'utilizzatore:** Un sistema distribuito è una raccolta di computer indipendenti che appare ai suoi utenti come un unico sistema coerente.

Da notare come l'utilizzatore ha **astrazione** e non vede i diversi server che ci sono dietro.

Il **sistema distribuito** è organizzato con un **middleware**, è esteso a più computer e offre a ciascuna app la stessa interfaccia(API)

Perchè faccio ciò? lo faccio per **distribuire il carico**.

*Più calcolatori gestiscono più richieste!* Evita un **attacco DDoS** (*Denial of Service* multiplo), mi permette di gestire meglio il traffico che arriva al pc.

Alla base uso il **round-robin** in **DNS**: Associa un nome ad un indirizzo IP, quando cerco con **nslookup** ottengo più indirizzi, poichè un singolo nome **dns** ha più server collegati.

Il **middleware** gestisce l'instradazione dei pacchetti sui vari server. Ovviamente dobbiamo avere una **larghezza di banda adeguata** e **interfacce di rete** che la supportano.

Approfondimento:

**VPS**: Virtual Private Server, virtualizzazione dentro una singola macchina di un OS a disposizione dall'utente. Molto **più lento** di un **barebone** perchè ho multitasking su  $x$  servizi.

Tre modi:

1. **DNS**: Uso il DNS per bilanciare il traffico;
2. **Hardware**;
3. **Software**: Uso di un middleware per gestire il traffico e indirizzarlo alle macchine.

Può essere usato in diverse modalità, ovviamente uso le più veloci per **real time** APP, mentre posso permettermi un rallentamento in **motori di ricerca** e simili.

Una **rete cellulare è un sistema distribuito**: le antenne sono sistemi distribuiti che forniscono il segnale a tutti.

### Obbiettivi sistemi distribuiti

1. Condivisione di risorse;
2. Trasparenza di distribuzione;
3. Apertura verso espansione dei servizi;
4. **Scalabilità**;
5. Fault Tolerance;
6. Etereogeneità di **HW** e **SW**.

### Condivisione delle risorse

L'idea è avere accesso remoto ad una risorsa, per ottimizzare avendo un controllo efficiente (stampanti, archivi...).

Ho diversi motivi per condividere, principalmente risparmio in **costi** (compro una stampante per piano, non faccio una stampante per ogni dipendente). Permette in oltre di gestire la concorrenza.

Si implementano modi a tessera, dove l'utente dopo aver inviato la stampa, quest'ultima inizia solo se l'utente passa una tessera sulla stampante stessa. Utile per loggare chi stampa cosa e per **privacy**.

Questo modello descrive come:

1. Le risorse sono rese disponibili;
2. Le risorse possono essere utilizzate;
3. Fornitore di servizi e utente interagiscono tra loro.

### Modelli di condivisione delle risorse:

1. Modello **Client-Server**:
  1. I processi server fanno da gestori delle risorse e offrono servizi;
  2. I client fanno richiesta al server che fornisce i servizi;
  3. Uso di **HTTP**.
2. Modello basato su **oggetti**:
  1. Uso di un sistema ad interfacce tra oggetti o simil oggetti

### Trasparenza della distribuzione

**Definizione:** Un sistema distribuito che è in grado di presentarsi ai suoi utenti ed applicazioni come se fosse solo un singolo sistema si definisce trasparente.

Transparency	Description
Access	Hide differences in data representation and how a resource is accessed
Location	Hide where a resource is located
Migration	Hide that a resource may move to another location
Relocation	Hide that a resource may be moved to another location while in use
Replication	Hide that a resource is replicated
Concurrency	Hide that a resource may be shared by several competitive users
Failure	Hide the failure and recovery of a resource

Ho diversi gradi di trasparenza, devo avere un bilanciamento tra prestazioni e alto grado di trasparenza.

I problemi che possono sorgere sono principalmente se succede qualcosa al datacenter e/o ho da effettuare un aggiornamento. Se ho un sistema bancario, non posso spegnere ed accendere il server! Allo stesso momento, devo **replicare i dati** per evitarne la perdita. Devo offrire **concorrenza** e **migrazione**, in uso o meno. **L'accesso**, indipendentemente da dove viene fatto, deve essere possibile. Devo avere una gestione delle **rottture** (comprare HW di qualità). Tutto ciò deve essere fatto **senza che l'utente lo sappia**.

#### Apertura verso espansione dei servizi

**Definizione:** Un sistema distribuito aperto è un sistema che offre servizi in base a regole standard che descrivono la sintassi e la semantica di tali servizi.

I servizi sono specificati tramite interfacce:

1. **Interface Definition Language (IDL):** acquisizione della sintassi (la semantica e' la parte difficile da specificare) ;
2. **Estensibilità:** un DS aperto può essere esteso e migliorato in modo incrementale, es aggiungendo o sostituendo componenti.

#### Scalabilità

**Definizione:** Un sistema è scalabile se rimane efficace quando vi è un aumento significativo della quantità di risorse (dati) e del numero di utenti.

**La scalabilità indica la capacità di un sistema di gestire un carico futuro crescente.**

Può essere:

1. Scalabile in dimensione;
2. Scalabile geograficamente;
3. Scalabile in modo amministrativo.

**Problematiche:**

### 1. Scalabilità in dimensione:

Soluzioni centralizzate creano problemi di banda e stabilità se cade il server principale.

Concept	Example
Centralized services	A single server for all users
Centralized data	A single on-line telephone book
Centralized algorithms	Doing routing based on complete information

### 2. Scalabilità Geografica

1. Comunicazione in LAN Sincrona
2. Comunicazione **inaffidabile** in WAN

### 3. Scalabilità Amministrativa:

1. Distribuzione su più domini, ho dei conflitti per l'utilizzo delle risorse.

### Metodologie:

#### 1. Distribuzione

1. Divido risorsa in più parti e la distribuisco su più server attraverso il sistema.

#### 2. Replica:

1. Replica di un servizio su più server dislocati;
2. Gestione della cache , locale e remota(**proxy server**);
  1. **DNS Caching**: Cerco il DNS, lo cache in locale dopo averlo ottenuto per le prossime richieste entro tot ore.
3. Aumenta disponibilità e bilanciamento di carico.

#### 3. Nascondere lentezze di comunicazione

Fare attenzione alla **Larghezza e ampiezza di banda**! La **Latenza** conta! Più aumenta la **latenza** e meno banda possiamo utilizzare. [Calcolo Throughput online.](#)

### Fault Tolerance

Tolleranza dei problemi agli **errori/rotture**.

#### 1. Rotture **Hardware**

#### 2. Database

1. Transaction
  1. Completa
  2. Uso di rollback se incompleta

#### 3. Ridondanza dell'informazione

1. Problema di avere informazioni memorizzate in più zone, ma uguali.

Spesso i cambi nel datacenter avvengono **ogni tre anni**, con controlli effettuati spesso.

### Implicazioni dei sistemi distribuiti

1. NON esiste **global clock**

1. Non posso sincronizzare tutte le macchine!
2. Non esiste un sistema di sincronizzazione globale!

## 2. Comunicazione

1. **Inaffidabile**;
2. **Non** protetta;
3. **Costosa**.

## Cluster di calcolo

Sistema di calcolo dove ho **più server** che lavorano ad uno stesso calcolo utilizzando lo stesso stema operativo.

Molto usati per

1. Problemi di ingegneria;
2. Previsione del meteo;
3. Struttura dell'universo;
4. ... e tutti i probemi **computazionalmente complessi**!

## Distributed Computing as an Utility

1. **PaaS**: Platform as a Service;
2. **SaaS**: Software as a Service;
3. **IaaS**: Infrastructure as a Service.

## Introduzione alle socket

#

Due possibilità di comunicazione:

### 1. TCP

1. Comunicazione affidabile;
2. Più lento per gli ACK;
3. Recupero dagli errori.

### 2. UDP

1. Non ho controllo errori;
2. Più efficiente;
3. Non si pone il problema di perdita dati.

TCP	UDP
Reliable, guaranteed	Unreliable. Instead, prompt delivery of packets.
Connection-oriented	Connectionless
Used in applications that require safety guarantee. (eg. file applications.)	Used in media applications. (eg. video or voice transmissions.)
Flow control, sequencing of packets, error-control.	No flow or sequence control, user must handle these manually.
Uses byte stream as unit of transfer. (stream sockets)	Uses datagrams as unit of transfer. (datagram sockets)
Allows to send multiple packets with a single ACK.	
Allows two-way data exchange, once the connection is established. (full-duplex)	Allows data to be transferred in one direction at once. (half-duplex)
e.g. Telnet uses stream sockets. (everything you write on one side appears exact in same order on the other side)	e.g. TFTP (trivial file transfer protocol) uses datagram sockets.

## Sockets: Definizione

**Definizione:** Una socket è un meccanismo che consente la comunicazione tra i processi, siano essi programmi in esecuzione sulla stessa macchina o diversi computer connessi in rete.

Più in particolare, le socket Internet forniscono un'interfaccia di programmazione allo stack del protocollo di rete gestito dal sistema operativo. Usando le API, un programma può inizializzare rapidamente un socket e inviare messaggi senza doversi preoccupare di problemi come il framing dei pacchetti o il controllo della trasmissione. Esistono diversi tipi di socket disponibili, ma siamo interessati solo ai socket Internet:

1. **Datagram Sockets (UDP)**
2. **Stream Sockets (TCP)**

Le differenze sono molteplici, ma principalmente per **tolleranza perdite**  $\neq 0$  si utilizza **TCP**.

Per applicazioni **VoIP** invece utilizzo di UDP, tollero la perdita di trasmissione( **ci\*\* c\*me va** si capisce comunque, mentre un trasferimento di un file deve essere impeccabile) e mi sarebbe **costosa** una ritrasmissione continua.

Mentre per i **Stream** devo **aprire e chiudere** la connessione, per un **Datagram** non ce n'è bisogno.

Un server sarà in **listen**, mentre un client invierà (TCP).

## Composizione socket

Una socket è rivolta in **host**→**DNS**→**Indirizzo IP + Porta**.

Utilizzo **protocollo IP**, con **Porta**, dove quest'ultima sarà  $> 1023$  e  $\leq 65535$  se ho applicazioni personalizzate, poichè le porte  $\leq 1023$  sono **porte conosciute e riservate**.

Debbono ovviamente **essere porte non già utilizzate nello stesso IP**.

*Esempio di porte standard*

<i>Port</i>	<i>Service Name, Alias</i>	<i>Description</i>
1	tcpmux	TCP port service multiplexer
7	echo	Echo server
9	discard	Like /dev/null
13	daytime	System's date/time
20	ftp-data	FTP data port
21	ftp	Main FTP connection
23	telnet	Telnet connection
25	smtp, mail	UNIX mail
37	time, timeserver	Time server
42	nameserver	Name resolution (DNS)
70	gopher	Text/menu information
79	finger	Current users
80	www, http	Web server

#### In JAVA:

```
import java.net.*;
```

1. Per socket **TCP**:
  1. **Socket** è il socket dalla parte del **client**;
  1. **connect** inizia una sessione **TCP**.
  2. **ServerSocket** è il socket dalla parte del **server**.
    1. **bind** dove collego il mio **IP** ad una **Porta** specifica;
    2. **listen** aspetta per **ricevere** la comunicazione.
2. Mentre per i socket **UDP**:
  1. **DatagramSocket** sia per **client** che per **server**.

Nota che i **byte-stream TCP** combina gli stream consecutivi. La **formattazione** deve essere fatta dall'app.

---

26/02/2019 - Lab

#

Uso di **OpenJDK**, cambia dall'originale solo la licenza.



What	What to do	Description	What to read
Programming Language	Download and install OpenJDK 11.0.2	OOP Programming Language we will be using	<ol style="list-style-type: none"> <li>1. Introducing to Java Programming</li> <li>2. Mkyong</li> <li>3. Benjamin Winterberg</li> </ol>
WEB Applications container	Download and Install Apache Tomcat 9.0.16	A Servlet and Java Server Pages container (an application server)	<ol style="list-style-type: none"> <li>1. Apache Tomcat 9 Tutorial for Beginners</li> </ol>
IDE	Download and install NetBeans IDE 10.0	An Open Source Integrated Development Environment	<ol style="list-style-type: none"> <li>1. Apache NetBeans Wiki</li> </ol>
Tools	Git 2.20 Apache Maven 3.3.9	Version Control System Dependencies Manager	<ol style="list-style-type: none"> <li>1. Pro Git book</li> <li>2. Apache Maven Project Documentation</li> </ol>

Dopo aver installato **Java**, come nelle slide, installare mettere la `JAVAHOME` sul `path` di sistema.

Dopo aver installato **Java**, installo **netbeans**.

Seguo le slide.

## Disattivare proxy https in gitlab

```
git config --global --unset https.proxy
```

## Versioni

$x.y.z$

se incremento qualcosa che è una nuova funzione, incremento  $x$ .

Per altre modifiche minori incremento  $y$  che non introduca nuova versione

Per modifiche bugfix e simili incremento  $z$ .

## Package

Dominio della azienda al contrario

```
it.unitn.disi.wp.labXX.nomeprogramma
```

## Creazione del progetto con maven

Crea progetto → Java Maven

## Visualizzazione pagine web

1. HTML: Linguaggio descrizione pagina (HyperTextMarkupLanguage)
  1. Sistema di markup, indica la struttura di esso.
  2. Statico, veloce, **ma poco versatile**.
2. CSS: Stile di visualizzazione pagina
3. Browser: Renderizzazione a schermo di pagina HTML in ascii

Da tenere conto i problemi di **disabilità visiva e funzionale**:

1. Evitiamo combinazioni con alto contrasto;
2. Daltonismo;
3. "vecchiaia".

Utilizzo di strumenti come JS con ingrandimento al click.

Un sito lento è poco usato.

**HTML5**: Tentativo di versatilità del documento html, con sistema di caching(?) supportato.

## Storicamente

I browser non rispettavano la definizione dei protocolli, perchè le direttive venivano date successivamente.

## Come funziona?

Utilizzo di **tag**: `<body></body>`

dove:

1. `<body>` è il tag di apertura;
2. `</body>` è il tag di chiusura.

Una pagina HTML deve essere scritta bene, oppure il browser interpreta come vuole.

## Elemento HTML

1. `<p>` è un tag
2. `<p>` *Contenuto* `</p>`
3. l'insieme dei due è un **elemento**.

## Attributi HTML

```
<a href="cose">Cose</a>
```

href è un attributo.

## Commenti HTML

```
<!-- COMMENTO -->
```

Un commento non viene tradotto né visualizzato dal browser.

## Principalmente

1. Non è **case sensitive**;
2. **Non** tutti i tag vengono chiusi una volta aperti;
  1. `<br>`
3. I TAG sconosciuti vengono ignorati;
4. `CLRF` vengono sostituiti con uno spazio;
- 5.

Diviso in

1. **HEAD**: Informazioni per definire la pagina. Informazioni non renderizzate.
2. **BODY**: Contenuto della pagina.

## Intestazioni

```
<h1>...</h1>
```

⋮

```
<h6>...</h6>
```

più alto è il numero, più è piccolo il titolo.

## Tipi di carattere

```
1 carattere normale
2 <b>carattere Bold</b>
3 <i>Carattere Italic</i>
4 <TT>Carattere Teletype</TT>
```

e varie combinazioni di essi.

## Caratteri speciali

< si scrive `&lt;`;

> si scrive `&gt;`;

& si scrive `&amp;`;

Devo rispettare le codifiche! è leggibile ovunque così.

## Liste

### 1. Ordinate

```
1 <ol>
2     <li>uno</li>
3     <li>due</li>
4 </ol>
```

### 2. Non ordinate

```
1 <ul>
2     <li>uno</li>
3     <li>due</li>
4 </ul>
```

### 3. Definite

```
1 <dl>
2 <dt> SGML <dd> Standard Generalized Markup Language
3 <dt> HTML<dd> Hypertext Markup Language
4 <dt> XML <dd> Extensible Markup Language
5 </dl>
```

Sto fornendo come "pallini", SGML,HTML e XML.

## Elementi di formattazione

```
<P><BR><BLOCKQUOTE><PRE><HR>
```

1. **<P>** : Paragrafo, tiene la formattazione.
2. **<BR>** Break, va a capo.
3. **<BLOCKQUOTE>** : Blocco quotato, simil citazione, indentato.
4. **<PRE>** : Contiene codice, formattazione forzata.
5. **<HR>** : Linea orizzontale.

## Meta Dati

```
<meta ...>
```

Inserite nello **HEAD**, ci permettono di indicizzare il tutto.

```
1 <meta Name="author" Content="Alessandro Borghese">
2 <meta Name=" keywords" Content="Cucina d'Osteria">
3 <base HREF="URL ">
```

Importante non **abusare di questa funzione** o il motore di ricerca indicizzerà male(o non lo farà proprio) il nostro sito.

## Link interni(ed esterni)

## Interni:

```
1 <a href="#altro">cose</a>
2 :
3 :
4 <a name="altro">Ecco le cose</a>
```

il primo è **cliccabile**, il secondo è un riferimento.

Mentre i link **esterni** sono come quelli visti prima.

```
1 <a href="URL">Link</a>
```

## Form

Fornisco all'utente la possibilità di **inviare informazioni al server**.

```
1 <FORM method="POST" action="/cgi-bin/elabora">
2 Scrivi il tuo nome
3 <Input type="text" size="25" maxlength="15" name="a">
4 <Input type="submit" value="spedisci">
5 <Input type="reset" value="annulla">
6 <Input type="radio" name="colore" value="rosso">Rosso
7 <Input type="radio" name="colore" value="argento" checked>Argento
8 Fai la tua scelta:
9 <Input type="checkbox" name="tipo" value="auto" checked>Auto
10 <Input type="checkbox" name="tipo" value="bus">Bus
11 <Input type="checkbox" name="tipo" value="camion">Camion
12 <Select name="colore">
13 <option>Rosso
14 <option selected>Argento
15 </select>
16
17 </FORM>
```

1. **Form** è la definizione del form stesso.
  1. **method** spiega come spedire i dati, cioè con uso di operazioni **CRUD**.
  2. **action** spiega dove spedirli;
  3. **enctype** spiega il **MIME** usato per spedire i dati;
2. **Input** è un campo di input
  1. **type** spiega che tipo di input è;
    1. **text** è testo;
    2. **submit** è il tasto che invia i dati;
    3. **reset** è il tasto che elimina tutti i dati inseriti nel form.
    4. **Radio** è una lista definita da **name** , dove avrà un solo output restituito(posso selezionare solo uno dei due). **Value** in questo caso è il valore che fornisco quando seleziono uno dei due.
      1. **checked** definisce quale è selezionato di base.

5. `checkbox` è una lista definita da `name` dove avrà uno o più output restituiti. (Posso selezionare più di uno). `Value` definisce quale di questi sono stati selezionati.
  1. `checked` definisce quale è selezionato di base.
6. `name` definisce l'oggetto/gli oggetti che stiamo settando.
  2. `value` è il testo mostrato all'utente sul `button` o il valore di ritorno.
3. `Select` crea un menù a tendina, con il relativo `name` di definizione.
4. `option` definisce una opzione del menù a tendina, con il relativo `value`.
  1. `selected` è l'opzione settata di base.

L'output sarà `nome=valore`

## Script e Style

Posso avere esecuzione di codice **JS** sul browser, con `<script>...</script>`

mentre posso avere lo stile della pagina con `<style>...</style>` senza necessariamente utilizzare un file CSS. (Va incluso nell' `<HEAD></HEAD>` )

```
1 <HTML>
2   <HEAD>
3     <STYLE TYPE="text/css">
4       h1 {color:red}
5       p {font-size:18}
6     </STYLE>
7   </HEAD>
8 </HTML>
```

Lo stile va per **livelli**, cioè viene eseguito con priorità quello dei tag:

```
<p style="...">
```

ha **meno** priorità quello di pagina:

```
<style>...</style>
```

e ha ancora **meno** priorità quello esterno (in un file CSS) collegabile inserendo nell' `<HEAD></HEAD>` il seguente :

```
<link rel="stylesheet" type="text/css" href="PATH/AL/CSS.css" >
```

*Script di esempio da inserire*

```
1 <script type="text/javascript" language="javascript">
2   <!--
3   function ciao() {
4     alert("hello world")
5   }
6   //-->
7 </script>
```

*È buona norma inserire il tag script in **fondo pagina**, ciò permette di caricare tutta la pagina prima che si carichino li script, che di norma sono più pesanti dei tag HTML statici .*

## Standard da utilizzare

```
1 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
```

Spiega al browser quale standard utilizzare. **Non è un tag.**

## Validatore

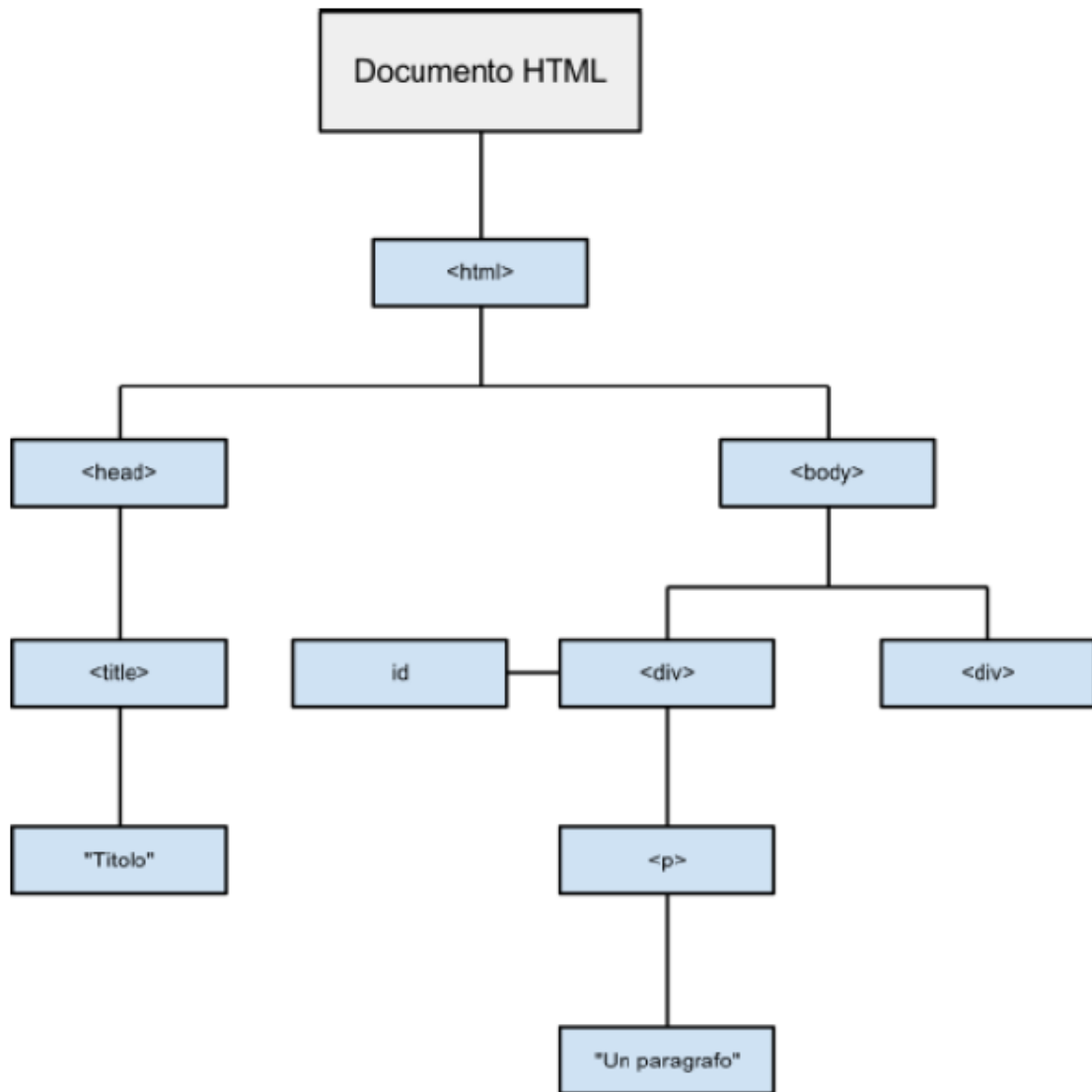
Controlla che il codice sia corretto. [Qui](#).

## HTML DOM

Dom è un Document Object Model.

**Document Object Model:** Rappresento un documento HTML come oggetti, annidati uno dentro l'altro, come i TAG HTML.

*Struttura di esempio :*



Ogni oggetto(quindi tag) in DOM ha:

1. **Proprietà:** Caratteristica, ad esempio, nome.

1. `n.innerHTML` - restituisce tutto ciò che è contenuto in n;
2. `n.nodeName` - restituisce il nome di n;
3. `n.nodeValue` - restituisce il valore id n;
4. `n.parentNode` - restituisce il nodo padre di n;
5. `n.childNodes` - restituisce l'elenco di nodi figli diretti di n;
6. `n.attributes` - restituisce l'elenco degli attributi di n.

2. **Metodi:** Meccanismo per modificare/interagire una proprietà del nodo stesso.

1. `n.getElementById(id)` - restituisce l'elemento con uno specifico id;
2. `n.getElementsByTagName(name)` - restituisce l'elenco di tutti gli elementi del tipo indicato tra parentesi e contenuti nel nodo n;
3. `n.appendChild(node)` - inserisce il figlio indicato tra parentesi come figlio di n;
4. `n.removeChild(node)` - elimina il nodo figlio di n indicato tra parentesi.



## Utilizzo:

```
1 document.getElementById("primo").innerHTML="<h3>Cambio contenuto</h3>";
```

Cambio contenuto dell'elemento con ID "primo" e lo metto con ciò scritto dentro a innerHTML.

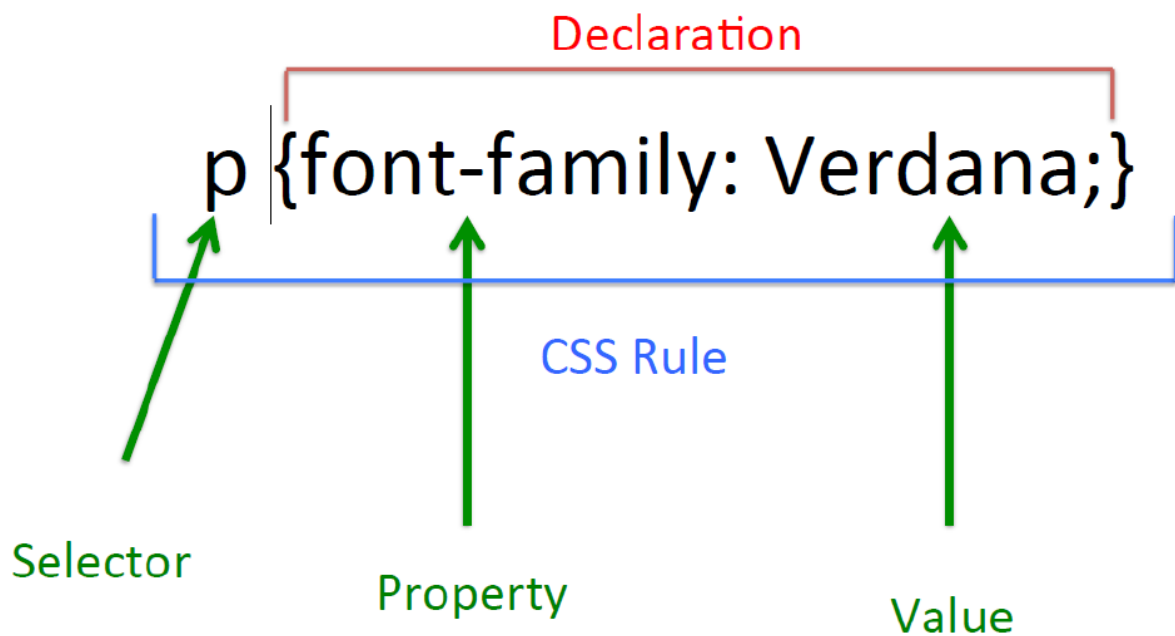
## CSS

#

CSS sta per Cascading Style Sheet.

Descrivo lo stile della pagina da utilizzare.

## Sintassi



## Cascading

Cascading indica il modo in cui vengono risolti i conflitti in caso di conflitti:

1. Inline style (**highest** priority);
2. Internal style sheet (second priority);
3. External style sheet (third priority);
4. Web browser default (only if not defined elsewhere).

Se più stili in conflitto sono definiti nel foglio samestyle, verrà applicato solo quello finale.

```
1 p {  
2   color: red;  
3   font-style: italic;  
4   text-align: center;  
5 }
```

Gli spazi sono **ignorati**, come gli acapo.

## Tipi di elemento

### 1. Selector:

```
1  p{
2      color:yellow;
3  }
```

Modifico **tutti** i tag che hanno quel tag specifico. (in questo caso `<p>` )

### 2. ID:

```
1  #primo{
2      color:yellow;
3  }
```

Modifico **tutti** i tag che hanno quell'ID. (in questo caso `<taggenerico id="primo">` )

### 3. Class:

```
1  .nomeclasse{
2      color:yellow;
3  }
```

Modifico **tutti** i tag che hanno quella classe.

(in questo caso `<taggenerico class="nomeclasse">` )

È possibile specificare una classe che funziona solo per il determinato tag:

```
1  h1.nomeclasse{
2      color:yellow;
3  }
```

funzionerà solo in `<h1 class="nomeclasse"></h1>`

Mentre

```
1  .nomeclasse h1 {
2      color:yellow;
3  }
```

funzionerà solo nei tag `h1` dentro ad un tag con classe `nomeclasse`

```
1  <div class="nomeclasse">
2      <h1>
3          Ciao!
4      </h1>
5  </div>
```

Per gestire un elemento all'interno di un elemento

```
1  div h1{
2      color: yellow;
3  }
```

Come nell'esempio delle classi, il funzionamento è lo stesso.

**h1 dentro div.**

```
1  <div>
2      <h1>
3          Ciao!
4      </h1>
5  </div>
```

### Gestire più di un elemento

```
1  h2, h1 {
2      color: yellow;
3  }
```

### Pseudo Selettori

1. **hover** : se ci passa sopra con il mouse

```
1  h1: hover {
2      color: yellow;
3  }
```

2. **link** : se è un link

```
1  a: link {
2      color: yellow;
3  }
```

3. **visited** : se è un link ed è già stato visitato

```
1  a: visited {
2      color: yellow;
3  }
```

[Lista completa](#)

### Colori

sono definiti in diversi modi:

1. per Nome: `color: yellow;`
2. per Valore Esadecimale: `color: #FF0000 ;`
3. per valore RGB(RedGreenBlue): `color: rgb(255, 0, 0);`
4. per valore HSL: `color: hsl(0, 100%, 100%);`

Ricordiamo che per nome sono molto limitati:

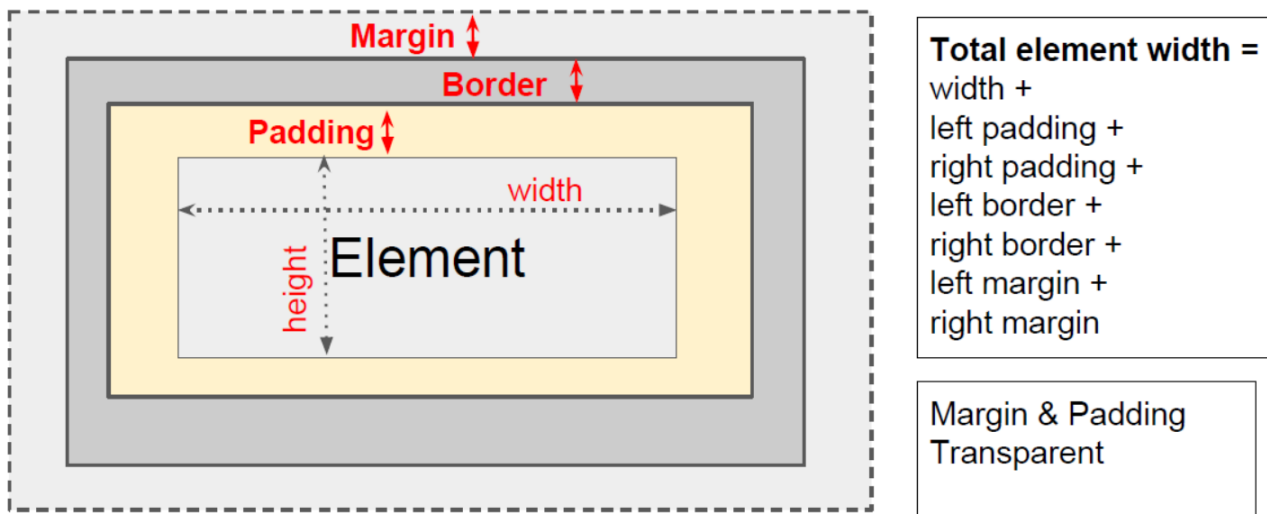
aqua, black, blue, fuchsia, gray, grey, green, lime, maroon, navy, olive, purple, red, silver, teal, white, yellow.

`color` per il colore dell'oggetto/font, mentre `background-color` per il colore di background.

## Larghezze e percentuali

1. `px` unità per il pixel;
2. `em` grandezza del font attuale ( `2em` è due volte la grandezza del font);
3. `pt` unità per i punti, usata spesso nei media;
4. `%` sono percentuali ( `width: 80%` )
5. Altre unità includono `cm` , `mm` , `in` (inches).

## Composizione di un Box



## Testo: Font

I font non sono tutti considerati **sicuri**. Quelli considerati tale, sono di base a tutti i browser.

Si specificano con `font-family` .

```
1  p{  
2    font-family: Arial, Helvetica, Serif;  
3  }
```

Cercherà *Arial* , se non lo trova, mette *Helvetica* , se non lo trova mette *Serif* . Se non lo trova mette il font di base.

Altri attributi:

1. `font-size` : setta l'altezza del font (remember units such as px, em, pt);
2. `font-weight` : indica la tipologia di variante del font (grassetto, etc...)(bold, light, normal,...);
3. `font-style` : sIndica se il testo e' scritto in corsivo (italic, normal);
4. `text-decoration` : indica se deve essere sottolineato o barrato (underline, overline, line-through, none);
5. `text-transform` : cambia il capitalizzazione (capitalize, uppercase, lowercase, none).

## Testo: Spacing

**letter-spacing** : imposta la spaziatura tra le lettere. Il valore può essere length o normal; **word-spacing**: imposta la spaziatura tra le parole. Il valore può essere length o normal; **line-height**: imposta l'altezza delle linee in un elemento, ad esempio un paragrafo, senza regola la dimensione del carattere. Può essere un numero (che specifica un multiplo della dimensione del carattere, quindi "2" sarà due volte la dimensione del carattere, ad esempio), una lunghezza, una percentuale o normal; **text-indent**: indenterà la prima riga di un paragrafo, a una data lunghezza o percentuale; **text-align**: allinea il testo all'interno di un elemento a sinistra, a destra, al centro o giustificato.

[Link generale](#)

## Layout delle pagine

#

### Mobile First Pages

Pagine che sono basate su **framework** per funzionare correttamente.

1. Comode, veloci e adattive;
2. Difficili da programmare/gestire.

Framework css molto utilizzato: **Bootstrap**.

---

## 05/03/2019-Laboratorio

Seguito le slide pari pari.

**Firefox** e **Chrome** mi mostrano la struttura del **DOM** modificata al momento dai vari script.

**.ready()** è il codice che si esegue a fine caricamento.

---

## 11/03/2019

### Web Architectures

#

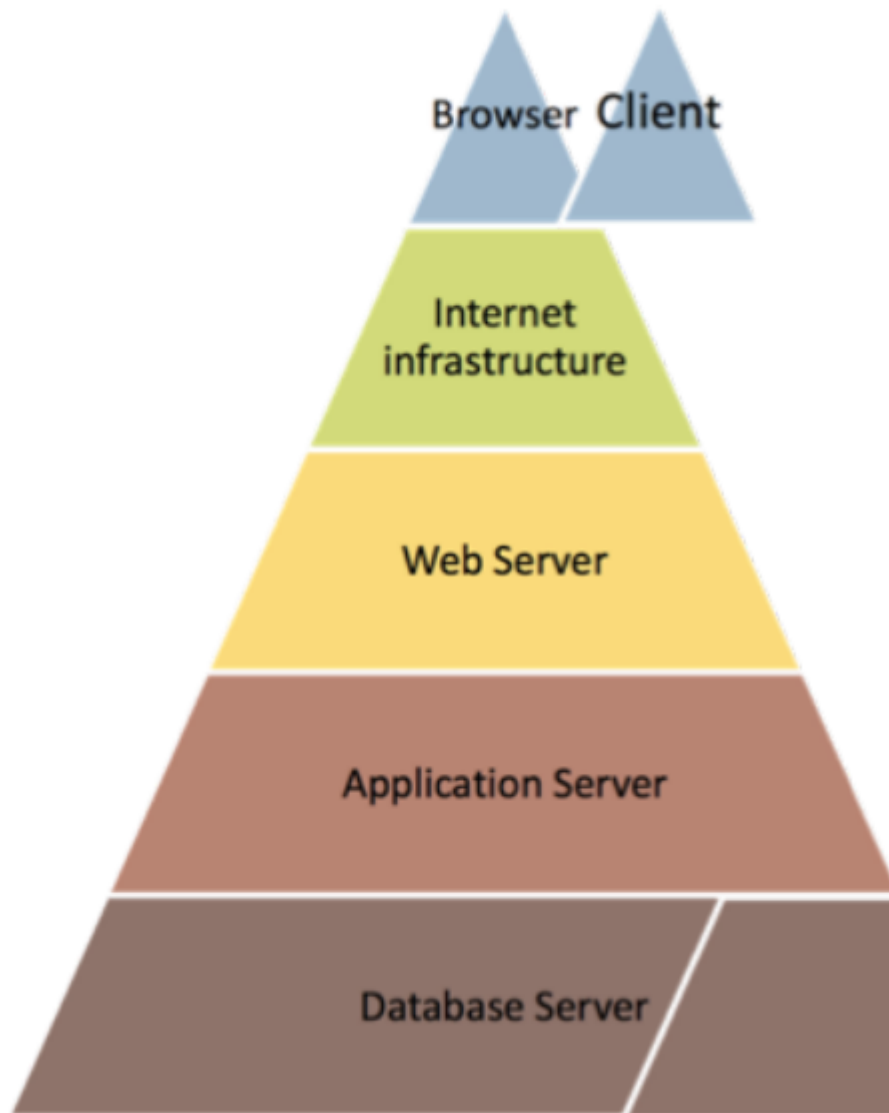
Tim Berners-Lee 1989 → www

Ha sviluppato *HTML*, *URL*, *HTTP* ;

Recupero informazioni da diverse fonti, anche le **NewsGroup**.

### Architetture multi-tier

#



Architetture a più tier per ottenere informazioni:

- Ho una rete;
- Ogni livello ha diverse funzionalità;
- Il frontman delle richieste è il **WebServer**. Il WebServer decide dove instradare il tutto.
  - Sotto di esso ci sono gli AppServer/DBserver

La più basilare è composta da client e server.

### Versione di base

Ha due livelli, come detto prima.

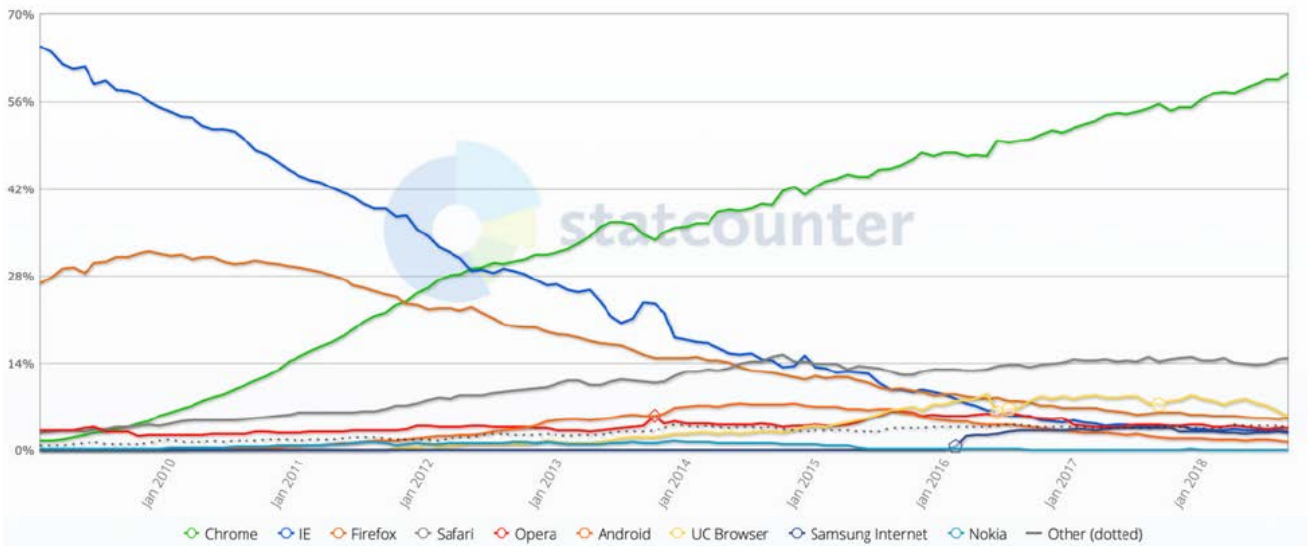
Il browser fa richiesta, ottiene la pagina dal server e mostra a schermo.

### Web Browser

Programma con il quale porto informazioni, graficamente e non, all'utente.

## Browser Market Share Worldwide

Jan 2009 - Sept 2018



## Web Server

Usa il protocollo HTTP, fornisce al client ciò che gli viene richiesto se disponibile e se permesso.

I più usati sono **Apache e Nginx**.

Entrambi sono multithread. Se crasha una istanza, il webserver rimane online!

**Lighthttpd** si utilizza nei *server Embedded*.

## HTML

Non è un linguaggio di programmazione!

## URI e URL

Uniform Resource Identifier (URI) e Uniform Resource Locator (URL) **non sono sinonimi**.

- 1 Un URI può essere classificato come un localizzatore, un nome o entrambi. Il termine "Uniform Resource Locator" (URL) si riferisce al sottoinsieme di URI che, oltre a identificare una risorsa, forniscono un mezzo per localizzare la risorsa descrivendo il suo meccanismo di accesso primario (ad esempio, la sua "posizione" di rete). Il termine "Uniform Resource Name" (URN) è stato usato storicamente per riferirsi a entrambi gli URI sotto lo schema "urna" [RFC2141], che sono tenuti a rimanere globalmente unici e persistenti anche quando la risorsa cessa di esistere o diventa non disponibile, e a qualsiasi altro URI con le proprietà di un nome.

Tutti gli URL sono URI (Falso se cambio interpretazione), ma non tutti gli URI sono URL perché ho anche gli **URN**.

- Un **Uniform Resource Locator** o **URL**

è un insieme di caratteri che identifica in modo univoco una risorsa nel Web.

È identificato da

```
protocollo://[username[:password]@]host[:porta]</percorso>[?querystring][#fragment]
```

dove la parte obbligatoria è `protocollo://host` .

Un URL senza protocollo non è un URL ma un semplice URI.

Esempio URL

```
http://www.google.com/cose/index.html
```

ho un **protocollo**, un **web domain**, una o più **folder** ed eventualmente un **file HTML**.

**ATTENZIONE**

`www.google.com/cose/index.html` non è un **URL**, ma un semplice **URI**.

Questo perchè è **provvisto di protocollo**, quindi potrebbe essere `ftp://...` e non solo .

- Un **Uniform Resource Name** o **URN**

è un **URI** che identifica una **risorsa all'interno** di un **namespace**, ma, a differenza del **URL**, **non permette l'identificazione della locazione** della risorsa stessa ( *what invece di where* ).

È identificato da `<URN> ::= "urn:" <NID> ":" <NSS>`

cioè, più semplicemente `urn:<NID>:<NSS>`

Dove

- **NID** è *Namespace Identifier* , identifica sintatticamente NSS.
- **NSS** è la *Namespace Specific String* , cioè la stringa che identifica la risorsa in modo **univoco, nel tempo, anche in lunghi periodi e anche se la risorsa viene eliminata, ma non dove è situata**.

Esempio URN

```
urn:isbn:0451450523
```

Un esempio di URN è il codice **ISBN**: questi identifica univocamente un libro, ma **non ci dà alcuna informazione sulla locazione dello stesso**.

**URL o URN?**

Vari esempi forniti dal professore.



```
1 URL http://www.pierobon.org/iis/review1.htm
2 URN urn:isbn:978-88-96297-26-1
3 URI http://www.pierobon.org/iis/review1.htm.html#one
4 URL: ftp://ftp.is.co.za/rfc/rfc1808.txt
5 URN: urn:urn-7:3gpp-service.ims.icsi.mcptt
6 URL: http://www.ietf.org/rfc/rfc2396.txt
7 URL: ldap://[2001:db8::7]/c=GB?objectClass?one (indirizzo IPv6!)
8 URL: news:comp.infosystems.www.servers.unix
9 URL: telnet://192.0.2.16:80/
10 URN (not URL): urn:oasis:names:specification:docbook:dtd:xml:4.1.2
11 URN: urn:uuid:6e8bc430-9c3a-11d9-9669-0800200c9a66 (un uuid tipo 1)
12 URN: urn:lex:eu:council:directive:2010-03-09;2010-19-UE (Direttiva europea)
13 URL: tel:1-800-555-5555
```

## Mailto URI(URL)

`mailto:randomuser@rizzi.xyz`

Invio una mail cliccando sul link(si tratta di un **URL**). Ci sono vari parametri.

---

Q: Perché è un URL?

A: Perché è composto da

- **protocollo**= `mailto` ;
- il valore (facoltativo su un URL generico, ma obbligatorio per inviare una mail) **username**= `randomuser` e
- **@ dominio**= `rizzi.xyz`

---

**NON** usare nelle pagine web, a meno di sistemi **anti web crawler** (o l'email viene presa dai robot e inserita in sistemi di antispam).

*Esempio*

```
1 <a href="mailto:mario.rossi@unitn.it?subject=Superamento%20esame&
  amp;cc=programmazioneweb%40gmail.com&amp;body=Buongiorno
  %2C%20Lei%20ha%20uperto%20l'esame">mario.rossi@unitn.it</a>
```

## HTTP

Protocollo request response standard di client e server. Il protocollo inizia dal client, va verso il server e ho un valore di ritorno.

**Messaggio di request:**

- Request Line
- Headers

- Empty Line
- Body

HTTP mette a disposizione otto metodi:

1. HEAD
2. **GET**

`www.google.com/q="cose"&parametrodue="cosedue"`

`q` e `parametrodue` sono due parametri, con valori rispettivamente `cose` e `cosedue`. Separati da un `&`.

3. **POST**

Parametri passati direttamente **nell'header**.

4. PUT
5. DELETE
6. TRACE
7. OPTIONS
8. CONNECT

Nome	Tipologia
GET	Safe
HEAD	Safe
OPTIONS	Safe
TRACE	Safe
POST	Unsafe
PUT	Unsafe
DELETE	Unsafe

I metodi definiti **safe** sono metodi che ottengono informazioni senza modificarle.

I metodi definiti **unsafe** sono metodi che modificano i dati, quindi non "sicuri".

Nei messaggi di response ottengo uno **status code**: Un codice che mi fa capire se la transazione è andata a buon fine( `200` ) o meno ( `404` ). Ne ho di diversi tipi:

1. `2XX` Sono codici di successo;
2. `3XX` Sono codici di reindirizzamento;
3. `4XX` Sono codici di mancanza risorsa/impedimenti vari;
4. `5XX` Sono codici di errore del server.

HTTP è **Stateless**.

È versatile, ma ad ogni richiesta il server deve inviare l'informazione! Per mantenere delle informazioni utilizzo diverse tecniche:

- Cookie
- Sessioni
- Variabili nascoste(Form)
- Parametri nell'URL (Get)

```
1 GET / HTTP/1.1
2 Host: www.unitn.it
```

è un esempio di richiesta.

Il browser può CACHEARE la richiesta!

- Ha una scadenza;
- Utile perchè controlla solo se la pagina è stata aggiornata, altrimenti ti mostra la pagina salvata in cache.

```
Date: Mon, 11 Mar 2019 03:49:13 GMT
Server: Apache
X-Drupal-Cache: HIT
Content-Language: it
X-Frame-Options: SAMEORIGIN
X-UA-Compatible: IE=edge
X-Generator: Drupal 7 (https://www.drupal.org)
Link: <https://www.unitn.it/>; rel="canonical",<https://www.unitn.it/>; rel="shortlink",</>; rel="hreflang_xdefault",<it>; rel="hreflang_it",<en>; rel="hreflang_en",<de>; rel="hreflang_de",<fr>; rel="hreflang_fr",<zh-hans>; rel="hreflang_zh-hans"
Cache-Control: public, max-age=0
Expires: Sun, 19 Nov 1978 05:00:00 GMT
Vary: Cookie,Accept-Encoding
Content-Encoding: gzip
Etag: "1552273737-1"
Last-Modified: Mon, 11 Mar 2019 03:08:57 GMT
Keep-Alive: timeout=5, max=500
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html; charset=utf-8
```

Noto che quasi tutti i parametri sono capibili. **Content-type** mi dice che codifica uso dopo.

## HTTP e HTTPS

HTTP è la versione **non sicura**, gira sulla porta **80**, mentre https nella porta **443**

HTTPS cifra i dati nella connessione, rendendoli difficili da **sniffare**.

L'esistenza dell'HTTPS è possibile solo attraverso i **certificati**.

I certificati **certificano** che il sito sia quello e che non sia un falso.

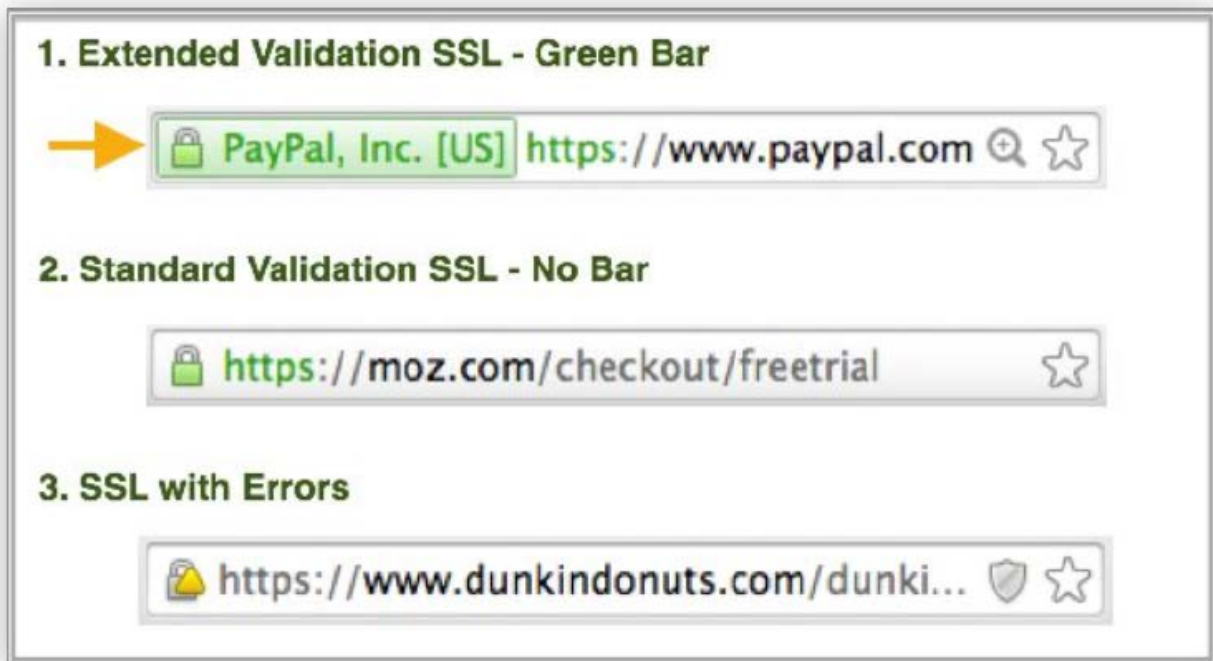
Una **certification authority** firma il certificato approvando i dati.

Un certificato vale per l'URL scritto sul tale. Posso avere i **certificati asterisco**:

`*.unitn.it`

vale per `unitn.it` e sottodomini!

## Certificati HTTPS: Tipi



### 1. Grigio: Self-Certified

Non ho una certification authority che l'ha firmata, non sono affidabile


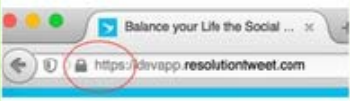

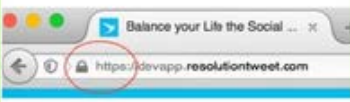


### 2. Verde non esteso:

Certificato ma senza dettagli.

### 3. Verde Esteso:

Certificato con dettagli.

## Validità

FEATURE	Displays	Info Displayed	Validity
 <b>Domain</b>	 Browser Padlock	Domain name	1 to 3 years
 <b>Organization</b>	 Browser Padlock	Domain name Organization name	1 to 3 years
 <b>Extended</b>	 Green Address Bar	Domain name Organization name Organization address	1 to 2 years

## Web Arch: Estensioni

#

### CGI

Estende l'architettura a 3 tiers aggiungendo al web server un servizio/ server che funziona dietro al web server.

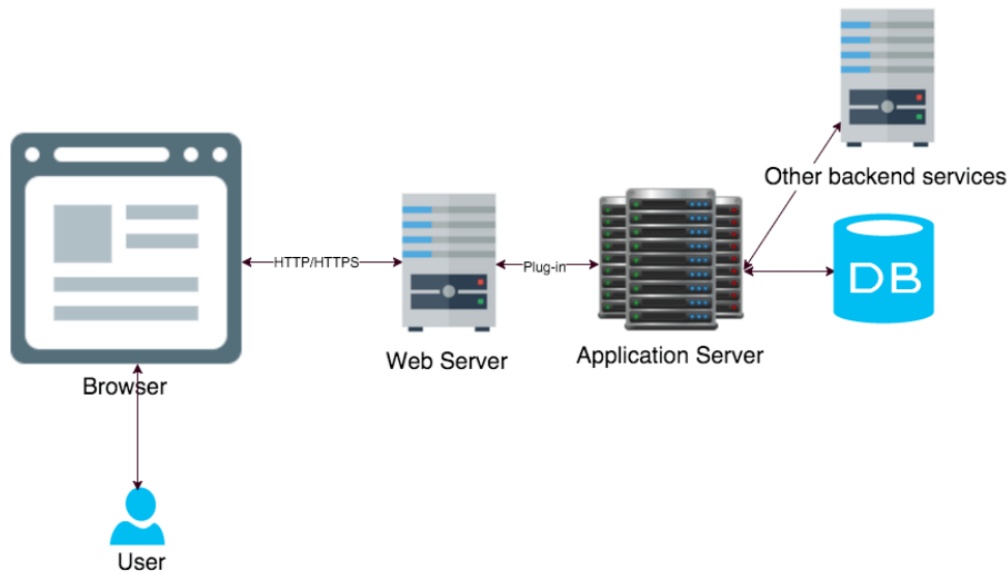
Ho un funzionamento gerarchico!

È svantaggioso per lo spawn continuo di processi.

Provocando tempi lunghi e possibili blocchi!

### Application Server

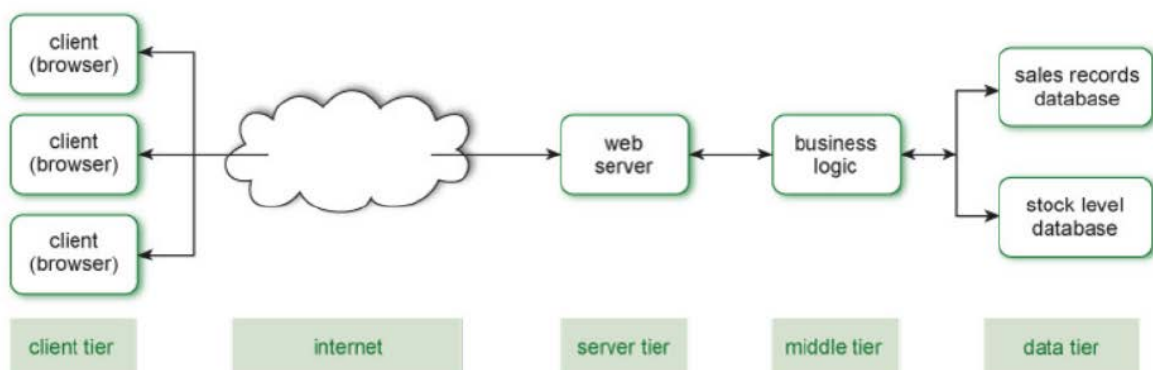
Aggiungo uno strato **applicativo** che esegue una determinata funzione. Questo server aspetta un input, lo esegue in base alla richiesta e lo rifornisce al webserver.



Per esempio **tomcat** e **glassfish** sono application server.

## Architetture multi-tier

#



Dei servizi specializzati fanno una **determinata funzione**. Servizi già fatti, già pronti e più sicuri!

Ho il vantaggio che posso modificare più facilmente ogni livello per la scalabilità.

## Network e architetture distribuite

#

Siamo passati da applicazioni monolitiche a client-server e quindi a N-tier, l'applicazione è stata suddivisa in un numero sempre maggiore di parti.

Questa tendenza è stata estesa in un approccio moderno chiamato architettura orientata ai servizi (SOA). SOA si basa sull'idea di scomporre un'applicazione in un insieme di attività molto più piccole che possono essere eseguite da piccoli "componenti software" indipendenti, ognuno dei quali esegue un'attività discreta chiamata comunemente servizio.

### SOA

Service-oriented Architecture:

I componenti software forniscono servizi ad altri componenti tramite un protocollo di comunicazione, in genere su una rete. La parte che offre il servizio è conosciuta come un fornitore di servizi (un server) e la parte che invoca il servizio è un consumatore di servizi (un client).

Ogni servizio è (processo aziendale)

1. **Ben definito:** un componente software con un'interfaccia e un risultato chiaramente definiti
2. **Out of the box:** l'implementazione del servizio è completa e indipendente da qualsiasi prodotto, fornitore o tecnologia
3. **Usò di BlackBox:** l'implementazione del servizio è nascosta (incapsulata) dal consumatore del servizio stesso.

Dei servizi potrebbero essere:

- Conversione di valuta;
- Controllo credito cliente;
- Fornitura dati metereologici;
- Archiviazione dati.

Il SOA è basato su dei principi:

- **Interoperabilità:** Servizi implementati in diversi sistemi, zone o business domains, usando diverse tecnologie, dai linguaggi e piattaforme, che devono lavorare assieme per permettere a diversi clienti e provider di comunicare. Perchè avvenga con successo, devi fare in modo che si utilizzino dei protocolli standard tra i vari dispositivi.
- **Posizione:** Gli utenti devono usufruire del sistema senza necessariamente sapere dove il servizio è posizionato geograficamente.
- **Reperibilità:** Il consumatore deve avere accesso ai servizi rilevanti, cioè deve avere accesso ad appropriati metadata per il servizio.

Questo servizio è spesso effettuato attraverso dei registri.

Distinguiamo tra:

- **Run-Time Discovery:**(Dinamico) intendiamo che il software in esecuzione, eseguirà alcune configurazioni inviando dei probes sulla rete (o le query a un repository centrale se si vuole comunque pensare in questo modo) per ottenere l'indirizzo IP dei servizi remoti.
- **Design-Time Discovery:** Al tempo di design si cercano servizi rilevanti per il contesto.
- **Accoppiamento e Incapsulamento liberi:** Le interfacce dovrebbero essere basate su standard di comunicazione più che proprietari e dovrebbero dare meno possibile informazioni sulla propria implementazione, effettuando incapsulazione.
- **Astrazione:** Sia l'implementazione incapsulata, sia la tecnologia implementativa e la posizione fisica dei servizi dovrebbero essere completamente invisibili ai consumatori, i quali dovrebbero essere dipendenti solamente dalle interfacce pubbliche.
- **Autonomia:** Più il servizio è autonomo, più controllo avrà verso le proprie implementazioni e sul run-time. Ciò implica maggiore flessibilità e potenzialità per evoluzione. Sia le implementazioni che il run-time possono essere modificate senza dare problemi ai clienti.
- **Statelessness:** I protocolli utilizzati sono stateless, cioè non mantengono lo stato, migliorando la scalabilità e l'uptime( *meno possibilità di crash(?)* ). Nonostante questo, i servizi dovrebbero rimanere stateless fino a quando è permesso loro fare ciò che devono fare senza stato.

- **Interfacce e contratti standardizzati:** I servizi devono essere descritti in modo tecnico e standard. La qualità del servizio fornito, il livello di service agreement, response time availability... e costi! Metadata aggiuntivi potrebbero coprire un range di informazioni, per esempio includere un rating della soddisfazione utente, per development futuri.
- **Riusabilità:** I servizi devono essere progettati con l'idea di riutilizzarli. In generale, i service of terms dovrebbero includere una clausola per permettere il riuso in altri progetti.
- **Riconfigurazione dinamica:** I sistemi service-based dovrebbero essere configurabili e riconfigurabili dinamicamente permettendo l'incorporare di nuovi servizi, ma mantenendo coerenza e integrità.

## Web Services

Evoluzione SOA, si utilizzano **servizi Web**, dove descrive come un client accede su un server su internet, usando i protocolli noti.

Il protocollo noto è l'uso del protocollo **HTTP REST**.

Altri modi:

- XML (eXtensible Markup Language);
- SOAP (Simple Object Access Protocol);
- REST (Representational State Transfer);
- JSON (JavaScript Object Notation) basati.

I servizi web basati sui principi di progettazione SOA garantiranno che le applicazioni web scritte in vari linguaggi di programmazione possano essere eseguite su varie piattaforme e possano utilizzare i servizi Web per scambiare i loro dati su reti di computer in modo simile a quanto avviene nella programmazione concorrente su un singolo host.

## Cloud Computing

Evoluzione SOA, utilizzo della cosiddetta tecnologia **CLOUD**.

Definizione NIST:

- Il cloud computing è un modello per abilitare l'accesso alla rete onnipresente, conveniente e on-demand a un pool condiviso di risorse di calcolo configurabili (ad esempio reti, server, storage, applicazioni e servizi) che possono essere rapidamente fornite e rilasciate con un minimo sforzo o servizio di gestione interazione con il fornitore.
- Il cloud computing è la pratica di fornire servizi di elaborazione - server, storage, database, networking, software, analisi e altro - on-demand su Internet. È un mezzo per fornire ai servizi informatici un'utilità per i consumatori allo stesso modo di altri servizi come il gas e l'elettricità. Le aziende che offrono questi servizi informatici in genere fanno pagare per i servizi di cloud computing basati sull'utilizzo.

Si potrebbe considerare che, mentre SOA fornisce servizi su una rete, il cloud estende il principio ad altre risorse come la potenza di calcolo ed i dischi e non si limita alla fornitura di servizi. Ci sono, ovviamente, molti dettagli impliciti: In che modo le risorse vengono addebitate, rese sicure, "pulite" dopo l'uso, ecc. Sono solo alcuni degli aspetti che devono essere considerati.

I servizi possono essere **facilmente forniti a poco sforzo!**

Caratteristiche principali:



- **On demand web service:** Un consumatore può automaticamente e unilateralmente ricevere capacità di computing come server time e network storage quando servono, senza richiedere un intervento umano.
- **Broad network access:** Le capacità di computing sono disponibili per tutto il network e accessibili attraverso meccanismi standard che promuovono l'uso da piattaforme clienti eterogenee (tablet, laptop, workstation e telefoni).
- **Resource pooling:** Il provider delle risorse fornisce a più consumatori in contemporanea utilizzando un modello a accesso condiviso, con differenti risorse fisiche e virtuali, dinamicamente riassegnate in base alla richiesta dell'utente. Abbiamo un sistema di astrazione che ci permette di evitare che l'utente abbia controllo o conoscenza su dove e come le resources sono distribuite, ma potrebbero ottenere informazioni sul luogo (datacenter, stato, città..).
- **Rapid Elasticity:** Capacità di adattarsi a forti richieste dei clienti, magari improvvise, con richieste di load molto più forti o molto più deboli del richiesto, in modo che sembrino illimitate e scalabili.
- **Measure service:** L'uso della risorsa dovrebbe essere misurato, monitorato, controllato e segnalato, permettendo trasparenza sia per il provider che per il cliente stesso, utilizzatore del servizio.

Divise in:

- **SaaS:** Software as a Service  
Fornisce applicazioni per utenti finali, fornite via web (dropbox, gdrive, gmail..);
  - **PaaS:** Platform as a Service  
Fornisce tools e servizi per sviluppare e fare deployment dell'app (Heroku);
  - **IaaS:** Infrastructure as a Service  
Fornisce server, storage, network e virtual machines da utilizzare (VPS Hosting).
- 
- 

## 12/03/2019- Laboratorio

---

Seguo slide

## 25/03/2019

---

### Cookies e Sessioni:

#

Ci permettono, nel protocollo stateless, di mantenere dei dati (come login).

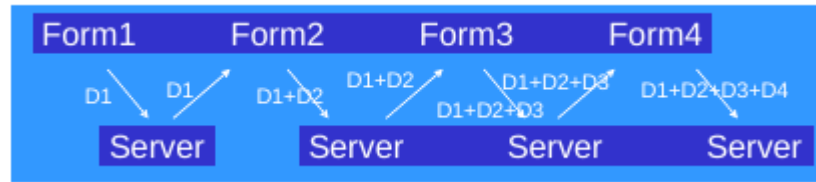
Ci sono due versioni di HTML

- 1.0 apre una connessione per ogni richiesta;
- 1.1 apre una connessione e la mantiene aperta per più richieste.

Attraverso i cookie e sessioni vedo quali dati avevo già inserito e quali no. Crea quasi uno stato, in un protocollo stateless, mantenendo un accesso cosicché l'utente non debba reinserirlo subito.

## Session state information

Precedentemente, per mantenere lo stato, rimandavo al cliente i dati del form indietro nel form successivo, in maniera nascosta, in modo che l'ultimo form aveva tutti i dati necessari.



Approccio **pesante!**

Ora, invece abbiamo l'utilizzo di

- **COOKIES:** Salvati sul client, stato del client;
- **SESSIONI:** Salvati sul server, stato del server.

utilizzando le Servlet, mantengo lo stato nei seguenti modi:

- Includo parametri GET nell'URL;
- **URL Rewriting:** Riscrivo i parametri man mano che l'utente va avanti.
- Field Form nascosti;

`<INPUT Type="hidden">` mi permette di nascondere i dati.

Poco sicuro, cache dei motori di ricerca.

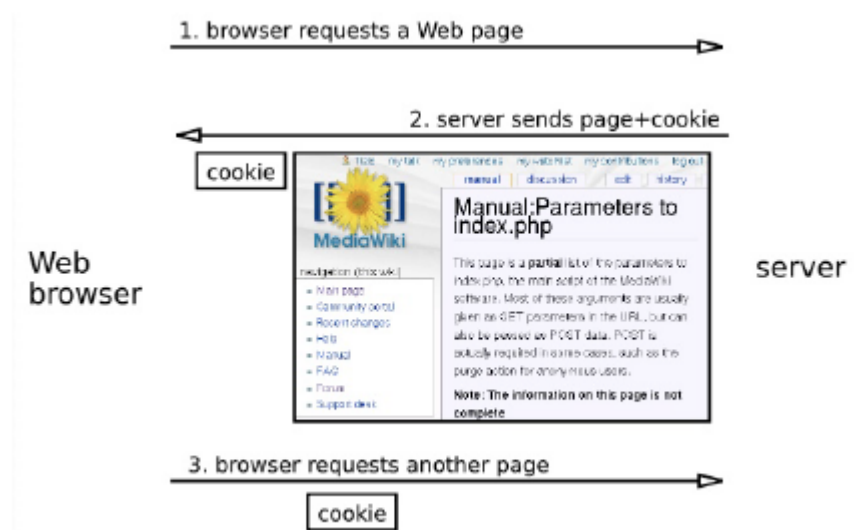
- Cookies con le Servlet API session tracking tools.

Creati dal web browser, due tipi:

- Con **scadenza:** Hanno una data di scadenza oltre la quale non sono più validi.
- Senza scadenza: Alla chiusura del browser vengono eliminati.

Passano **nell'HEADER** delle richieste, vengono creati dal server.

Al cookie rimane il PATH e il TOKEN inviato dal server dopo che il browser ha fatto la richiesta.



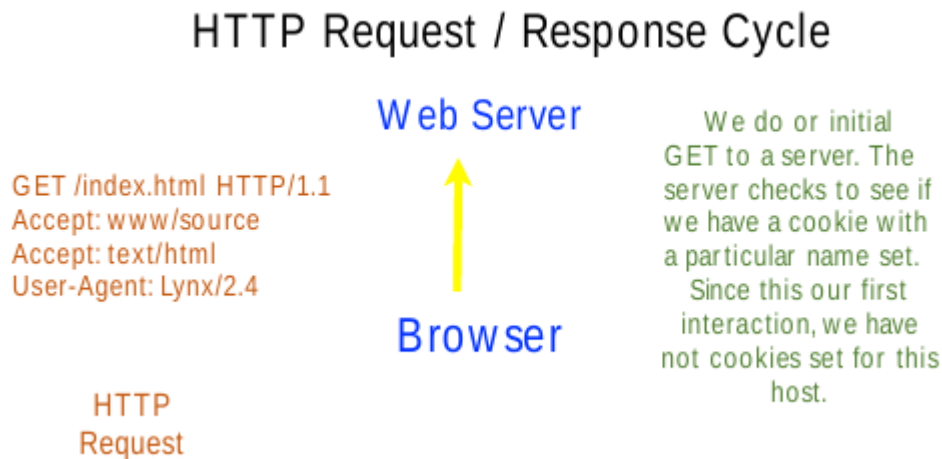
**session:** Serie di interazioni tra client e web server

Tutto parte scambiandosi un **token**, che è un **id di sessione**.

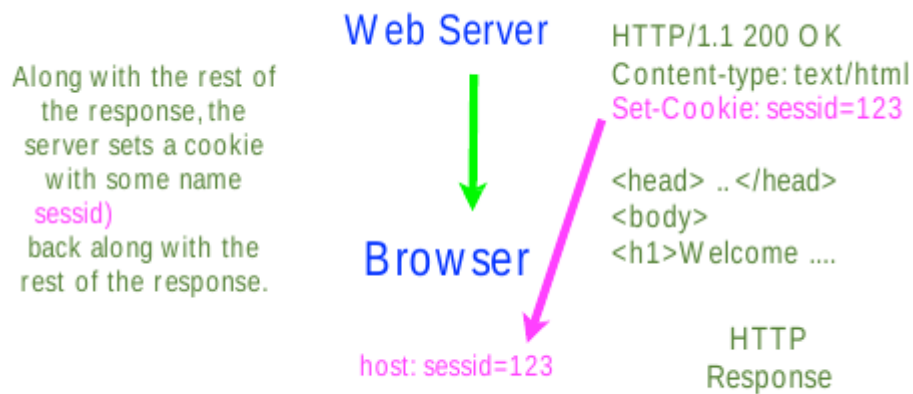
## Cookies

#

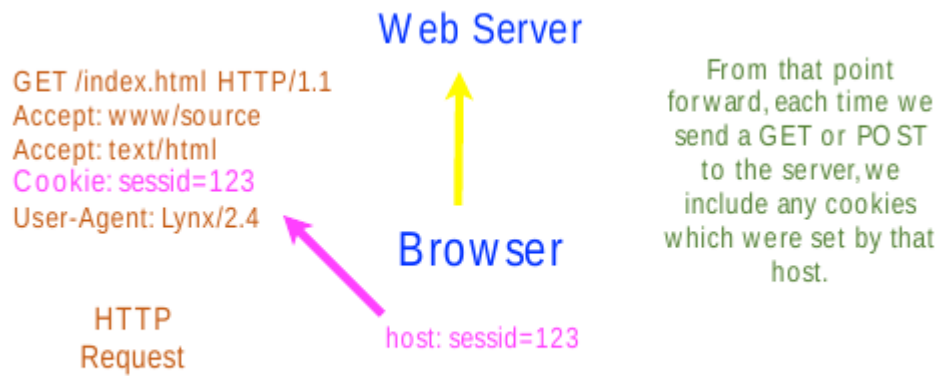
In andata:



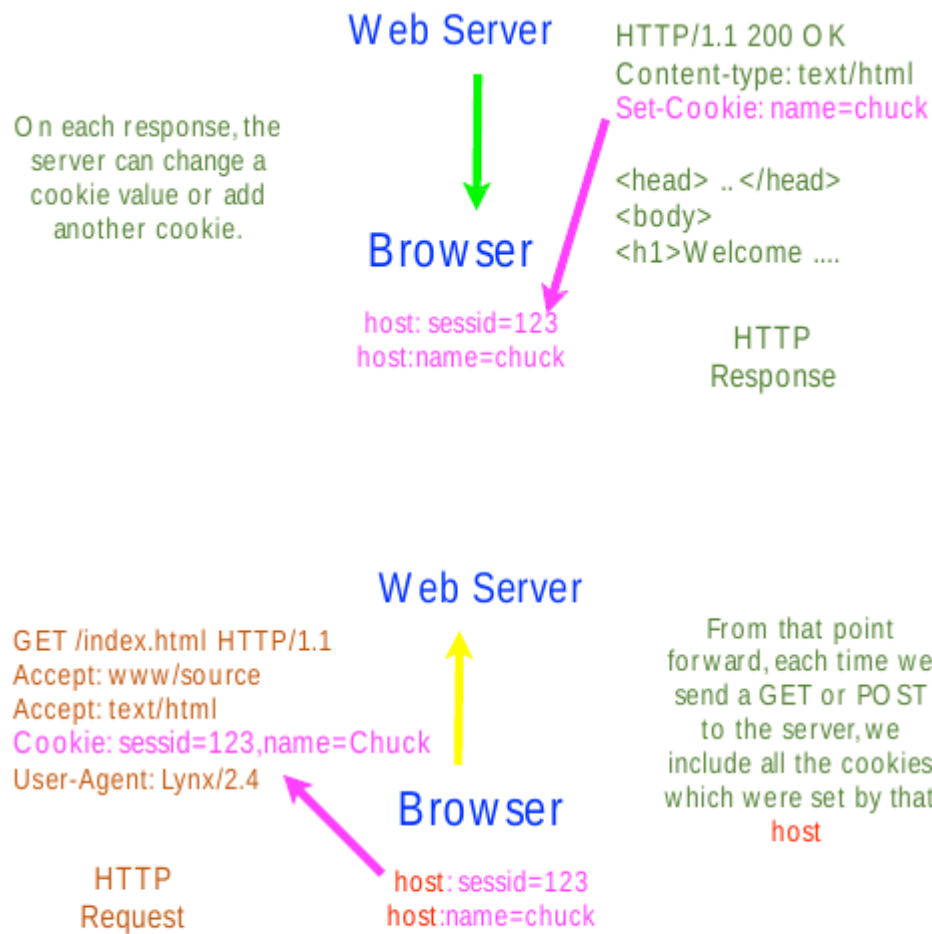
In ritorno:



Io client lo memorizzo:



Ora ne può settare un altro:



I cookie possono tracciare ciò che faccio e dove vado! Attraverso IFRAME o richieste multiple, posso salvare cookie particolari e ottenere una specie di storico di siti visitati, **perdendo privacy**.

## Servlet Cookies API

`javax.servlet.http.Cookie`

- Getter:

```
getName(),getValue(),getPath(),getDomain(),getMaxAge(),getSecure()...
```

- Setter

```
setValue(),setPath(),setDomain(),setMaxAge()...
```

Ottenere i cookies

```
Cookie[] HttpServletRequest.getCookies()
```

Aggiungere un cookie

```
HttpServletResponse.addCookie(Cookie cookie)
```

*WelcomeBack.java*

```
public class WelcomeBack extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        String user = req.getParameter("username");
        if (user == null) { // Find the "username" cookie
            Cookie[] cookies = req.getCookies();
            for (int i = 0; cookies != null && i < cookies.length; ++i) {
                if (cookies[i].getName().equals("username"))
                    user = cookies[i].getValue();
            }
        } else res.addCookie(new Cookie("username", user));
    }
}
```

Look for "username" parameter in the query

Look for "username" cookie

If "user" parameter was sent, create a "username" cookie

29

```
if (user == null) // No parameter and no cookie
    res.sendRedirect("getname.html");

res.setContentType("text/html");
PrintWriter out = res.getWriter();
out.println("<html><body><h1>Welcome Back " + user
    + "</h1></body></html>");
}
}
```

Delay setting the **ContentType** until clear that a content should be sent

*WelcomeBack.java*

## Session Management

#

La sessione si gestisce lato server:

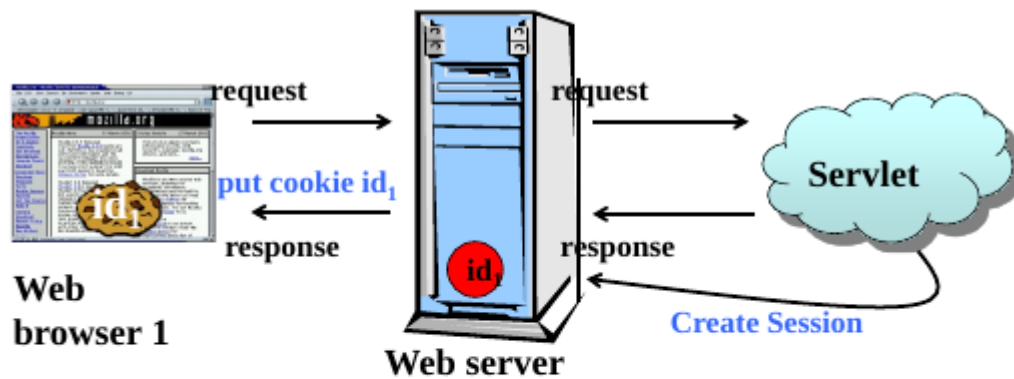
Una sessione cattura la nozione di continua interazione tra server e client.

La session Management dovrebbe essere **efficiente**:

*Il client non dovrebbe inviare lo shopping cart completo ogni volta che un prodotto è stato aggiunto!*

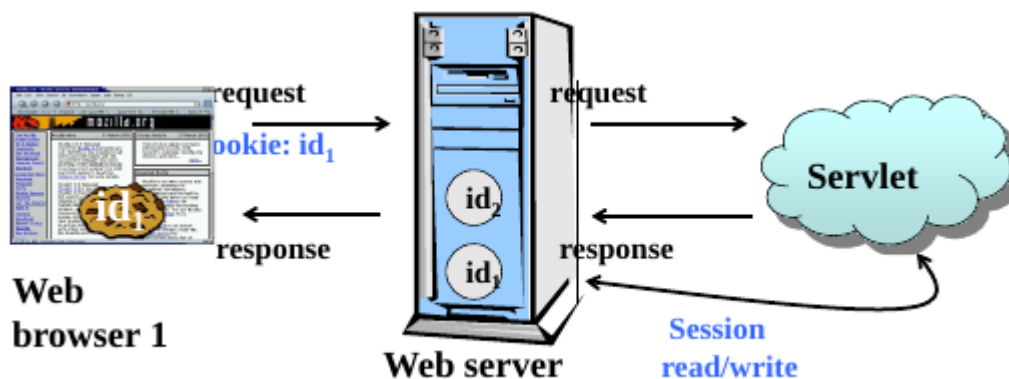
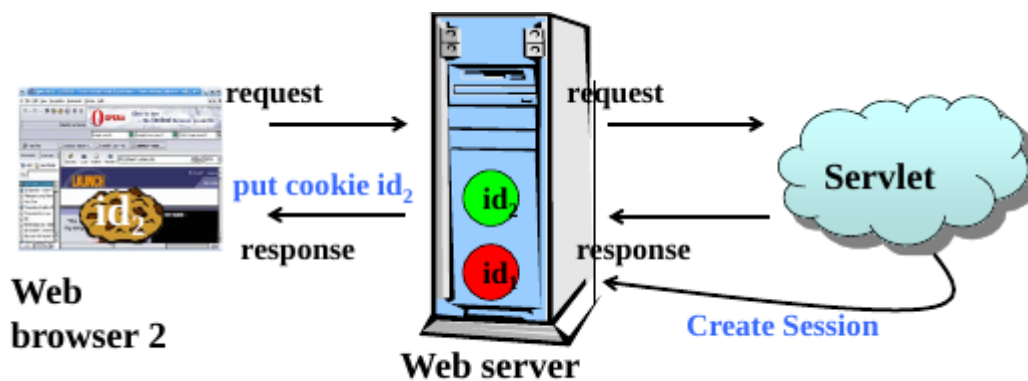
Due meccanismi utilizzabili:

- Session Cookies
- URL Rewriting.

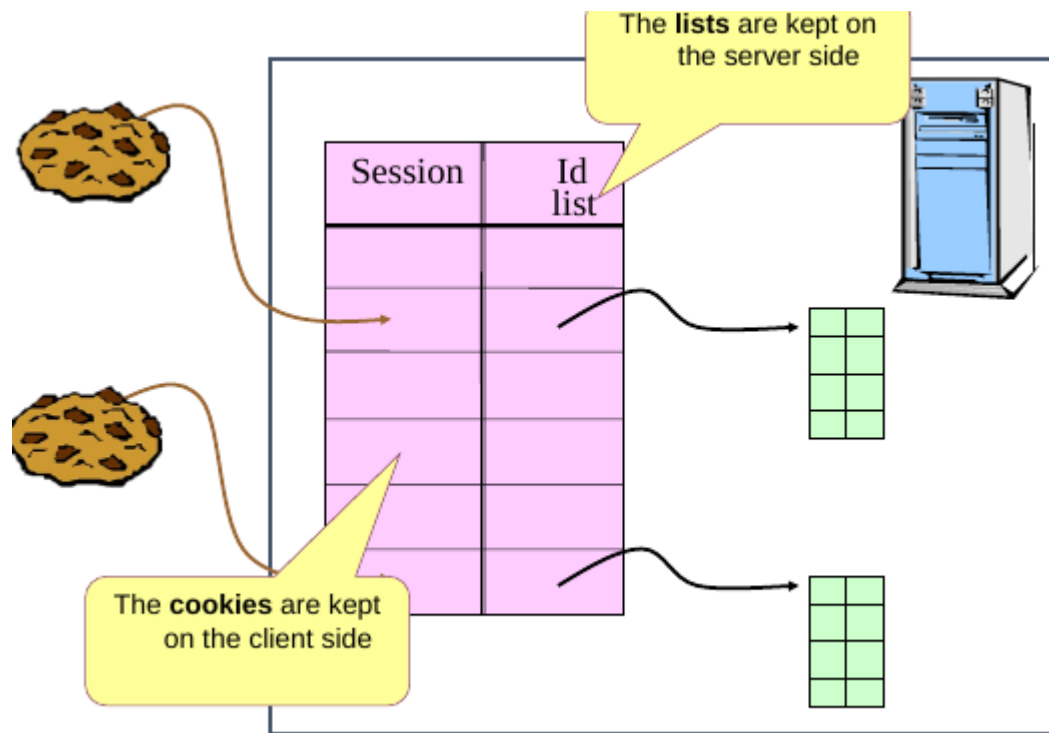


The cookie contains the **session-ID**: a unique (sometimes random) identifier assigned by the Servlet container. This is an example for something called an opaque identifier (coat-checks, typically implemented in C with void\* are another very different example worth knowing). Why is it a good strategy to "store" the session data using an opaque identifier on the client?

33



Le liste di sessioni (ID list) è disponibile solo sul server, mentre il cookies sono settate nel client side.



Questo però mi permette, avendo i cookie di qualcun altro, di entrare nella sua sessione!

Sì, però possiamo fare in modo che le altre pagine web non ottengano il nostro cookie.

### Accedere ai dati di sessione

`HttpServletRequest.getSession()` accede all'object session. Se non ce n'è una viene creata.

per non crearla se non esistente `HttpServletRequest.getSession(false)` .

Le sessioni di solito scadono dopo x minuti arbitrari, generalmente 30.

### Metodi HttpSession

I dati di sessione sono dentro una hash-table:

- `setAttribute(String name, Object value)`
- `Object getAttribute(String name)`

Altri metodi abbastanza intuitivi:

- `getId()`
- `removeAttribute()`
- `Enumeration getAttributeName()`
- `isNew()`
- `getCreationTime()`
- `getLastAccessedTime()`

*Curiosità: Con i processori AMD posso avere un terabyte di ram per processore, mentre per gli Intel fino a 1.4 TB di ram.*

## Scadenza Sessioni

Ho un **max-age** che mi setta il massimo che può rimanere attiva la sessione.

Serverside `session.invalidate()` dopo il timeout di sessione.

```
1 <web-app>
2   <session-config>
3     <session-timeout>10</session-timeout>
4   </session-config>
5 </web-app>
```

*Dopo 10 minuti, la sessione scade.*

Settabile e gettabile con:

```
session.setMaxInactiveInterval()
```

```
session.getMaxInactiveInterval()
```

## Cookie di terze parti

Cookie che NON centrano con il sito visualizzato.

Ogni cookie è BLOCCABILE.

## Cookies: rischi

I cookies possono creare rischi alla sicurezza, perchè ogni server può mettere i cookie **non solo propri** (cookie di terze parti) e può leggere **cookies di ogni sito memorizzato**.

Cosa succede se ho un computer condiviso?

- Cookie di più utenti nella stessa macchina!

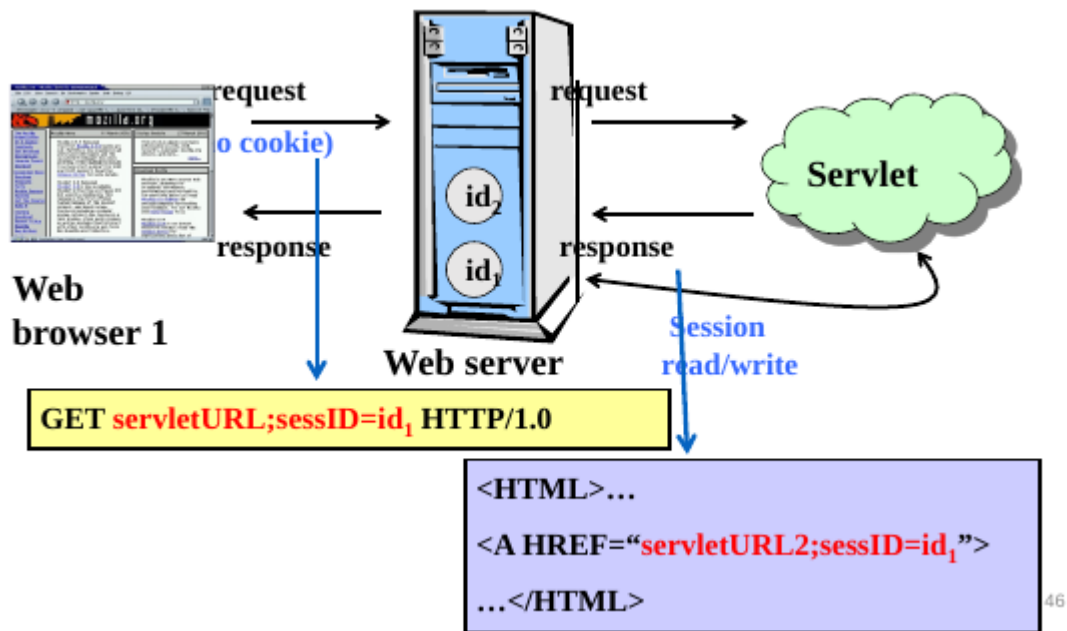
## Soluzioni

- **Uso session cookies**, cioè cookies che funzionano SOLO in una determinata sessione.
- **Uso URL Rewriting**: L'URL in GET ha un valore `sessID=ID` dove salva l'ID di sessione.

Ci permette l'utilizzo delle sessioni con i cookie disabilitati!

Il contatore del timeout session viene resettato ogni richiesta, se non faccio una richiesta entro tot minuti, allora perdo la sessione(timeout).





## URL Rewriting: Servlet

- `String HttpServletResponse.encodeURL(String url)` Hyperlink HTML
- `String HttpServletResponse.encodeRedirectURL(String url)` HTTP Redirections

Questo mi permette di vedere se la sessione ha senso venga codificata nell'URL, cioè se ho un session cookie, l'URL rimane **senza modifiche**, mentre se non trova un cookie, lo mette.

Qualche server implementa entrambi in modo identico.

## Request Dispatcher

#

Come chiamare dal server un'altra risorsa del server.

`getServletContext().getRequestDispatcher("x")` , richiedo la risorsa `x` .

`RequestDispatcher` è un object usato per inviare una richiesta a qualsiasi altra risorsa del server.

La risorsa può essere dinamica o statica.

## Metodi

- `void forward(ServletRequest request, ServletResponse response)` inoltra la richiesta da un servlet ad un'altra richiesta. È importante non scrivere il contenuto della richiesta se faccio ciò.
- `void include(ServletRequest request, ServletResponse response)` Include il contenuto in una risorsa in risposta della servlet corrente(e forse dovrò scrivere il contenuto prima o poi)

## Passare i dati

Ho tre modi diversi:

1. `request.setAttribute("key", value);` Dati utilizzabili solo per questa richiesta.
2. `session.setAttribute("key", value);` Dati utilizzabili per più richieste, però nella stessa sessione.
3. `context.setAttribute("key", value);` Dati utilizzabili in futuro per ogni client.

## Session tracking remoto e locale

#

Feature	Cookies	localStorage	sessionStorage
Maximum data size –	4 kB	5 MB	5 MB
Blockable by users –	yes	yes	yes
Auto-expire option –	yes	no	yes
Supported data types –	string only	string only	string only
Browser support –	very high	very high	very high
Accessible server-side –	yes	no	no
Data transferred on every HTTP request –	yes	no	no
Editable by users –	yes	yes	yes
Supported on SSL –	yes	n/a	n/a
Can be accessed on –	server-side & client-side	client-side only	client-side only
Clearing / deleting –	Java, PHP, JS & automatic	JS only	JS & automatic
Lifetime –	as specified	till deleted	till tab is closed
Secure data storage –	no	no	no

1. **Local storage:** Storage locale interagibile solo con HTML5 e JS.
2. **Session Storage:** Disponibile per la Browser session e la session storage è eliminata quando la sessione viene chiusa o il browser viene chiuso. Disponibile solo per browser con HTML5 come il local storage.
3. I **cookies** sono accessibili ovunque.

I cookie ti consentono di memorizzare le stringhe. Sessione e Archiviazione locale consentono di memorizzare le primitive JavaScript (tipo di dati), ma non gli oggetti o gli array. Storage di sessione consente di archiviare qualsiasi tipo di oggetti supportati tramite il linguaggio di programmazione o il framework di Server Side.

Archiviazione locale e archiviazione sessione (web storage) sono nuove API e sono quasi gli stessi (sia in API che in capacità) con l'unico singola eccezione di persistenza. L'archiviazione della sessione è disponibile solo per durata della sessione del browser (apertura e chiusura) e viene cancellata quando la scheda o la finestra è chiusa.

Tutti e tre i metodi sono utilizzati per salvare i dati nella zona clientside, hanno la propria **capacità** e la propria **expiration date**.

**Local storage e Session storage** sono i migliori per dati non protetti (dati con informazioni normali come nome, genere), ad esempio, punteggi in quiz e giochi.

I **cookies** sono migliori per l'autenticazione e il mantenimento della sessione

I cookie consentono solo di memorizzare solo **4 KB di dati** mentre lo storage Web (sia locale che di sessione) fornisce circa **10 MB di spazio per i dati da memorizzare**.

Ricordiamo che essendo **stateless**, alla chiusura del browser senza queste funzioni perderemmo i dati.

I dati di sessione sono controllati dall'applicazione **server side** in modo assoluto.

Lo svantaggio è che il server deve dare potenza e memoria per gestire questi dati, tra il quale inviare i dati quando il client lo richiede ad ogni richiesta. Il server deve sopportare anche l'eliminazione dei dati da parte del client.

localStorage, sessionStorage e cookie sono tutti soggetti a regole "**same-origin**", il che significa che i **browser** dovrebbero impedire l'accesso ai dati tranne il dominio che imposta le informazioni con cui iniziare.

<b>LocalStorage</b>	5MB/10MB storage It's not session based, need to be deleted via JS or manually Client side reading only Less older browsers support
<b>SessionStorage</b>	5MB storage It's session based and working per window or tab Client side reading only Less older browsers support
<b>Cookie</b>	4KB storage Expiry depends on the setting and working per window or tab Server and client side reading More older browsers support

## LocalStorage

### Pros:

1. Web storage can be viewed simplistically as an improvement on cookies, providing much greater storage capacity. If you look at the Mozilla source code we can see that **5120KB (5MB)** which equals **2.5 Million chars** on Chrome) is the default storage size for an entire domain. This gives you considerably more space to work with than a typical 4KB cookie.
2. The data is not sent back to the server for every HTTP request (HTML, images, JavaScript, CSS, etc) - reducing the amount of traffic between client and server.
3. The data stored in localStorage persists until explicitly deleted. Changes made are saved and available for all current and future visits to the site.

### Cons:

1. It works on [same-origin policy](#). So, data stored will only be available on the same origin.

## SessionStorage

### Pros:

1. It is similar to `localStorage`.
2. The data is not persistent i.e. data is only available per window (or tab in browsers like Chrome and Firefox). Data is only available during the page session. Changes made are saved and available for the current page, as well as future visits to the site on the same window. Once the window is closed, the storage is deleted.

### Cons:

1. The data is available only inside the window/tab in which it was set.
2. Like `localStorage`, it works on [same-origin policy](#). So, data stored will only be available on the same origin.

# Cookies

## Pros:

1. Compared to others, there's nothing AFAIK.

## Cons:

1. The 4K limit is for the entire cookie, including name, value, expiry date etc. To support most browsers, keep the name under 4000 bytes, and the overall cookie size under 4093 bytes.
2. The data is sent back to the server for every HTTP request (HTML, images, JavaScript, CSS, etc) - increasing the amount of traffic between client and server.

Typically, the following are allowed:

- 300 cookies in total
- 4096 bytes per cookie
- 20 cookies per domain
- 81920 bytes per domain (Given 20 cookies of max size 4096 = 81920 bytes.)

I limiti sono consigliati, MOLTE volte i browser mettono limiti propri e cambia da desktop a mobile.

## Desktop

	Chrome	Firefox	Safari	Safari	IE	IE
	40	34	6, 7	8	9	10, 11
Application Cache	up to quota	500MB, Unlimited	Unlimited?	Unlimited?		100MB?
FileSystem	up to quota					
IndexedDB	up to quota	50MB, Unlimited		up to quota?		10MB, 250MB (~999MB)
WebSQL	up to quota		5MB, 10MB, 50MB, 100MB, 500MB, 600MB, 700MB...	5MB, 10MB, 50MB, 100MB, 500MB, 600MB, 700MB...		
LocalStorage	10MB	10MB	5MB	5MB	10MB	10MB
SessionStorage	10MB		Unlimited	Unlimited	10MB	10MB

## Mobile

	Chrome	Android Browser	Firefox	Safari	Safari	iOS WebView	iOS WebView
	40	4.3	34	6, 7	8	6, 7	8
Application Cache	up to quota	Unlimited?	5MB, Unlimited	300MB?	300MB?	100MB?	100MB?
FileSystem	up to quota						
IndexedDB	up to quota		5MB, Unlimited		up to quota?		up to quota?
WebSQL	up to quota	200MB~		5MB, 10MB, 25MB, 50MB	5MB, 10MB, 25MB, 50MB	50MB	50MB
LocalStorage	10MB	2MB	10MB	5MB	5MB	5MB	5MB
SessionStorage	10MB	Unlimited		5MB	Unlimited	5MB	Unlimited

## Filtri

#

Un filtro è un oggetto che trasforma una request o modifica una response.

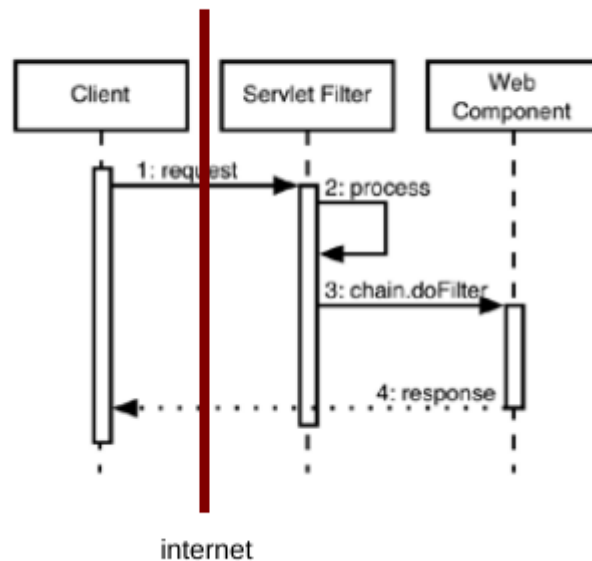
Ci sono preprocessori che gestiscono il tutto prima che raggiunga la servlet e postprocessori che modificano la response appena uscito dalla **servlet**.

### Schema di funzionamento

#### Filtro in ingresso

- Il client invia una richiesta al server e il filtro la intercetta
- Il filtro preprocessa la richiesta, raccogliendo eventuali informazioni
- Il filtro richiama il metodo `chain.doFilter` (lo vedremo meglio dopo) per invocare la prossima servlet o il webcomponent
- Il webcomponent invocato genera la risposta.

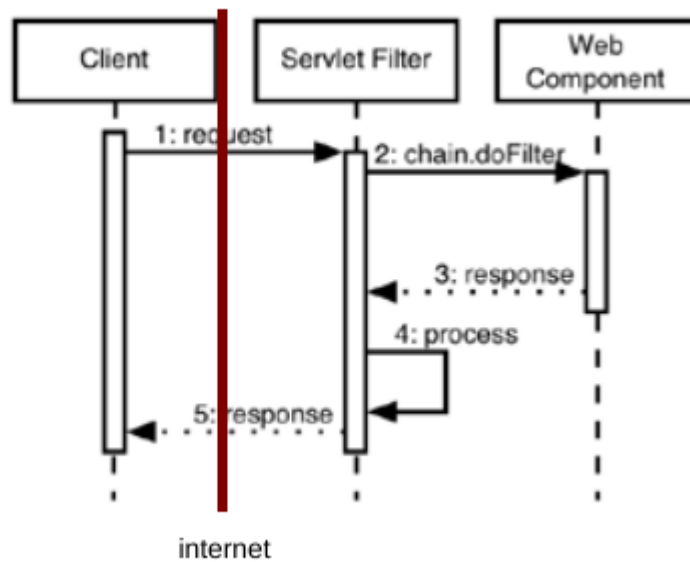
### Filtro ingresso



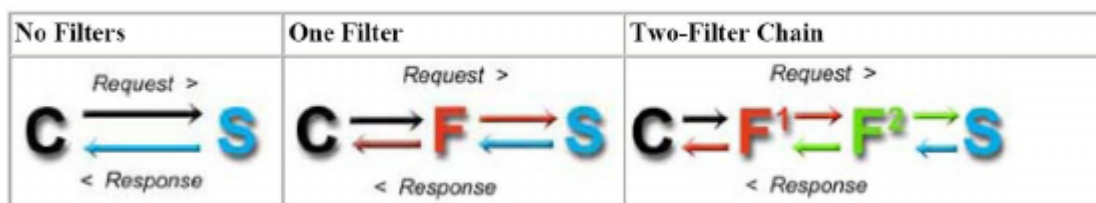
#### Filtro in uscita

- Il client invia una richiesta al server e il filtro la intercetta.
- Il filtro richiama il metodo `chain.doFilter`.
- Il web component corrispondente risponde, generando la risposta.
- La risposta viene intercetta dal filtro e viene processata.
- La risposta eventualmente modificata viene restituita al client.

# Filtro uscita



## Schema logico



## Ipotesi di utilizzo

- In particolare l'applicazione pratica che noi vedremo sarà un esempio di filtro di autenticazione, ossia il nostro filtro fornirà un forma di protezione da tentativi di accesso non autorizzato a certe risorse sul webserver.
- Altri esempi di possibile filtro: una filtro che effettua del logging con informazioni riguardo al client oppure ai tempi di elaborazione delle pagine oppure un filtro di compressione dati.

26/03/2019 - Laboratorio

Seguo slide

01/04/2019

Ci servirà solo la prima servletlistener

## Listener

#

Un listener è un **event handler** che il server invoca quando un determinato evento avviene.

In termini di design: **Observer pattern**.

Un osservatore(in questo caso il listener) è avvisato di quando l'evento avviene nel server.

Usi tipici:

- DB management;

Se non uso un listener, ogni utente dovrà connettersi al DB! Questo crea un rallentamento da parte dell'utente e da parte del server!

**Connection Pool:** *Istanza N connessioni al DB(Facendo partire N thread), ogni volta che arriva una richiesta si connette con un thread al Servizio DB. Ci riconnette al DB in qualsiasi situazione, incluso il **timeout**(Session Timeout dal server SQL).*

- Dependencies Management;
- Monitoring.

Poichè dopo un po' il sito "scade", utilizzo un JS per tenere viva la connessione, rinnovando il cookie. Il servizio **keep-alive** prende un sacco di ram server-side e client-side.

Tipi di listener:

- **ServletContextListener** web app inizializzata/ spegnimento
- **ServletRequestListener** Request handler start/stop
- **HTTPSessionListener** session creata/invalidata
- **ServletContextAttributeListener** context attribute aggiunto,rimosso, replicato
- **HttpSessionAttributeListener** session attribute aggiunto, rimosso, replicato

Per usare un listener devo implementare l'appropriata interfaccia e registri nel **deployment descriptor**.

## Database nelle App Web

#

### Tipi di database

- **SQL**: Database Relazionali
- **NoSQL**: Not Only SQL

I dati variano da semplici messaggi di testo a file video ad alta risoluzione. Il tradizionale RDBMS non è in grado di far fronte alla velocità, al volume e alla varietà dei dati richiesti da questa nuova era. Sono dunque nati i **NoSQL** (Che sono pure open source).

## Database SQL

Nel corso di DB.

MariaDB:

- Se crasha MariaDB: Devo ricontrollare gli indici! (Non sono operazioni da poco.)
- Posso suddividere le tabelle grandi in tabelle più piccole uguali per avere accesso più veloce.
- Subquery aggiunte dopo.

PostgreSQL:

- Se crasha PostgreSQL: Risale con gli indici giusti.
- Possibilità di subquery e vettori.

MSSQL:

- Gratis per poche grandezze;
- MONOTHRD, grande problema.

SQLite:

- Comodo, versione più leggera di SQL, ha meno comandi **ALTER**, salvato in un singolo file;
- Non ho controllo del tipo;
- Passa comunque per un accesso ad un file(più lento).

*La gente utilizza i software con licenza perchè ho meno responsabilità, perchè pago una licenza, dunque se succede un problema non sono pieno di problemi perchè la patch di sicurezza da applicare al software non è a carico mio come un OpenSource.*

Da sapere la differenza del **timestamp** e **data**. Attenti al **fuso orario**.

Inserire sempre il timestamp nelle tabelle! Stare attenti al server che l'orario del server sia giusto. Oltretutto, devo stare attento alla codifica del DB stesso.

## Database NoSQL

Le caratteristiche di base dei database NoSQL sono:

- **Senza uno schema**;
- **Distribuiti** e **scalabili** orizzontalmente;
- **Hardware commodity**: È un termine per dispositivi a prezzi accessibili che sono generalmente compatibili con altri dispositivi di questo tipo. In un processo chiamato commodity computing o commodity computing computing, questi dispositivi sono spesso collegati in rete per fornire maggiore potenza di elaborazione quando coloro che li possiedono non possono permettersi di acquistare più elaborati supercomputer, o vogliono massimizzare i risparmi nella progettazione IT.

I database NoSQL offrono tante funzioni per risolvere i problemi che il DB SQL Relazionale aveva per archiviare i dati non strutturati (il cosiddetto **blob**).

### Schemi dinamici

I database NoSQL consentono allo schema di essere flessibile, con nuove colonne aggiungibili in qualsiasi momento.



Le richè possono o non possono avere valori per colonne e non ho applicazione rigida per i tipi delle colonne.

Ho più flessibilità ai cambiamenti!

### **Varietà dei dati**

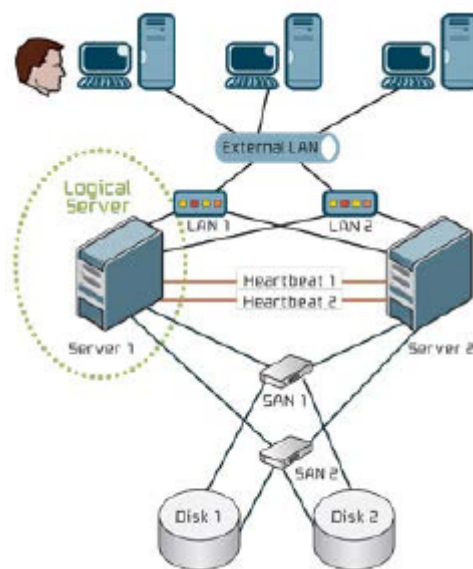
I database NoSQL supportano qualsiasi dato.

Supportano dati **strutturati**, **semi-strutturati** e **non strutturati** da memorizzare. Supporta log, file di immagini, video, grafici, jpegs, JSON, XML da memorizzare e utilizzare così come sono senza alcuna pre-elaborazione. Riducono quindi la necessità di **ETL** (abbreviazione di Extract - Transform - Load).

### **Cluster ad alta Availability**

I database NoSQL supportano storage distribuito!

Ho anche scalabilità orizzontale.



Attenzione, **non è LOAD BALANCING**, se un server va offline, l'altro, con arp, fa finta di essere il server primario cambiando il proprio mac address.

### **Open Source**

I DB sono OpenSource! Posso inoltre modificare il codice del DBMS NoSQL, senza problemi sulla licenza indipendentemente dall'uso.

### **Non dipendenza da SQL**

I database SQL non dipendono solo da SQL per recuperare i dati, anzi permettono un accesso **CRUD**:

- Create
- Read
- Update
- Delete

Anche dette **API**.

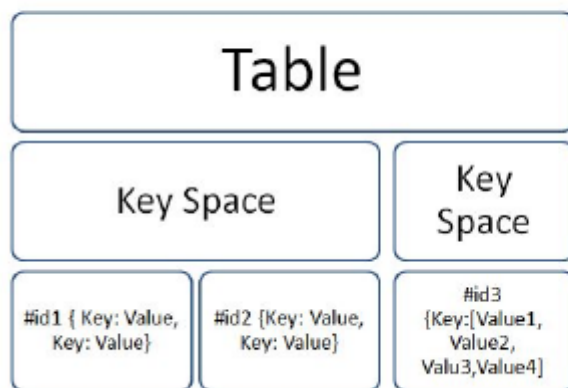
Permettono anche accesso **DML** (Data manipulation language).

### **Tipi di NoSQL**

- Esistono quattro tipi di basi di dati NoSQL.
  - database di valori-chiave (key value)
  - database orientato alle colonne (column oriented)
  - database orientati ai documenti (document oriented)
  - database Graph.
- Non ho :
  - Join
  - Non ho indici modificabili (li gestisce il DBMS Automaticamente)
- Posso avere più Database dentro al database (Inception)
- Questa gerarchia è comune a tutti i database NoSQL, ma le terminologie potrebbero variare.

### Key-Value data model

- La tabella contiene molti spazi chiave e ogni spazio chiave **può avere molti identificatori** per memorizzare coppie di valori chiave.
- Lo spazio-chiave è simile alla colonna nel tipico **RDBMS** e il gruppo di identificatori presentato sotto lo spazio-chiave può essere considerato come righe.
- È adatto per la costruzione di applicazioni semplici, non complesse e disponibili. Poiché la maggior parte dei database di valori chiave supportano nell'archiviazione della memoria, possono essere utilizzati per creare un meccanismo di **cache**.



- DynamoDB
- redis

### Column Oriented

- Database basati su colonne sono sviluppati sulla base del whitepaper di Big Table pubblicato da Google. Ciò richiede un approccio diverso rispetto al tradizionale RDBMS, in cui supporta l'**aggiunta di un numero sempre maggiore di colonne e una tabella più ampia**.
- Dal momento che la tabella sarà molto ampia, supporta il raggruppamento della colonna con un nome di famiglia, denominandola "Famiglia di colonne" o "**Super colonna**". La famiglia di colonne può anche essere facoltativa in alcune basi di dati della colonna.
- Secondo la filosofia comune dei database NoSQL, i valori delle colonne possono essere distribuiti in modo spartano. (Utile per avere integrazione di più sistemi di più linguaggi diversi!)
- La tabella contiene famiglie di colonne ( *facoltativo* ). Ogni famiglia di colonne contiene molte colonne. I valori per le colonne potrebbero essere scarsamente distribuiti con coppie chiave-valore. **I database orientati alle colonne**

sono alternativi ai database di Data warehousing tipici e sono adatti per il tipo di applicazione **OLAP**.

- Apache Cassandra

Table			
Column Family 1		Column Family 2	Column Family 3
Column 1	Column 2	Column 3	Column 4
#1 {Key: Value, Key: Value}	#1 {Key: Value, Key: Value}		#1 {Key: Value, Key: Value}
#2 {Key: Value, Key: Value}	#2 {Key: Value, Key: Value}	#2 {Key: Value, Key: Value}	#2 {Key: Value, Key: Value}

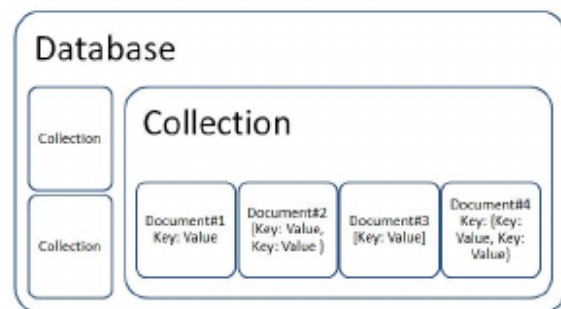
- HBase

### Document Oriented

- I database orientati ai documenti offrono supporto per la memorizzazione di **dati semi-strutturati**. Può essere **JSON**, XML, YAML o persino un documento di Word. L'unità di dati è chiamata documento (simile a una riga in RDBMS). La tabella contiene documenti ed è chiamata **raccolta(collection)**.
- Il database contiene molte raccolte.
- Una raccolta contiene molti documenti.
  - Ogni documento può contenere un documento **JSON** o un documento XML o YAML o anche un documento Word.
- I database di documenti sono adatti per applicazioni basate su Web e applicazioni che espongono servizi **RESTful**.

- MongoDB

- Couchbase



### Graph Database

Database gestito a grafo: Contiene **nodi e vertici**, I database di grafici ci consentono di memorizzare ed eseguire operazioni di manipolazione dei dati su nodi, relazioni e attributi di nodi e relazioni.

*I database di grafi funzionano meglio quando i grafi sono grafi **diretti**, cioè quando vi sono relazioni tra grafi.*

Ogni grafo è simile ad una tabella, ho :

- Nodo;
- Proprietà di esso;

- Relazioni;
- Proprietà di esse.

Adatti per i social media.

- Neo4j
- OrientDB
- HyperGraphDB
- GraphBase
- InfiniteGraph

Graph			
Node	Labels / Properties	Relations	Relation Properties
Node Id A	{Key: Value, Key: Value, Key Value}	Node Id B	{Key: Value, Key: Value}
Node Id A	{Key: Value, Key: Value}	Node Id C	{Key: Value}
Node Id B	{Key: Value}	Node Id A	{Key: Value, Key: Value}

## Problemi NoSQL

- **no ACID transactions:**

La maggior parte dei database NoSQL non supporta le transazioni ACID. Per esempio: ~~MongoDB~~ (A quanto pare le supporta nelle ultime versioni ), CouchBase, Cassandra.

- **Atomicità:** Le transaction devono essere atomiche in natura. (All or nothing rule). Se una parte della transaction non va, devo effettuare un **rollback** e annullare tutta la transazione.
- **Consistenza:** Il Database deve assicurare che solo i dati **validi** possono essere salvati. Questo dipende dal NoSQL che sto usando.
- **Isolazione:** Il database permette multiple transazioni in parallelo e le tiene isolate.
- **Durabilità:** Transazioni persistenti del DB.

- **API:**

Alcuni NoSQL supportano solo API e non SQL.

- **Lee-Way of CAP Theorem:**

Consistenza, Disponibilità e Partizionamento.

- **Consistenza:** Tutti i nodi devono essere dello stesso dato nello stesso tempo in un sistema distribuito.
- **Disponibilità(Availability):** Deve essere in un High Available System, cioè sempre disponibile.
- **Partizionamento:** Deve essere e rimanere operabile nonostante partitioning arbitrario creato da problemi di network.

Non sono rispettati tutti e tre, ne supportano generalmente solo due.

- **No JOIN:**

Non tutti le supportano.

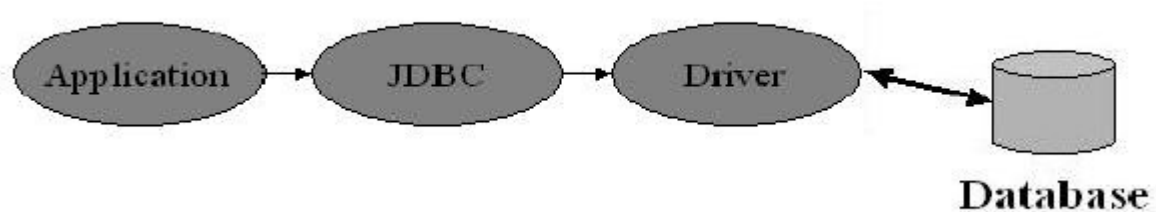
## JDBC

Insieme di classi JAVA (Java Database Connectivity) in stile **ODBC** per interfacciarsi ai SQL databases.

Provide an object interface to relational database Based on the X/Open SQL CLI (Call Level Interface) X/Open C.

Standardizza l'accesso al DB attraverso JAVA.

Richiede che i drivers accettino una chiamata CLI e trasforma in chiamate native per l'accesso al db. CLI fornisce un driver manager che conversa con i driver attraverso un Service Provider Interface (SPI).



## Architettura

Ti permette di scrivere DBMS-Indipendente.

Fornisce due interfacce:

- Interfaccia Applicativa
- Interfaccia Driver

Uso il driver Manager per caricare automaticamente il giusto driver JDBC e parlare al database corretto(MariaDB,Postgree...)

```
Jdbc:<subprotocol>:<domain name> example jdbc:odbc://www.bob.com/MyJavaDB
```

## Microsoft CLI ODBC

*Microsoft's ODBC Windows API standard for SQL is an extended version of the SAG CLI .*

### Controindicazioni

- specification è controllata da Microsoft
- ODBC drivers sono difficili da creare e mantenere.
- Crea molto overhead.
- Mai veloci come le API Native.(10%)

## Transaction

Nelle transaction sono principalmente in autocommit mode.

Stored Procedure: supporto alle chiamate.

## Schema classi Java

- JDBC Core Interfaces
  - Interfacce e classi i quali ogni driver JDBC driver deve implementare

### Driver:

- **DriverManager:** Gestisce i driver e le dispense di oggetti connection .
- **Driver:** Quando un driver è caricato, crea una istanza di se stesso e la registra nel drivermanager.

- **DriverPropertyInfo**: Fornisce variabili pubbliche che ti lasciano dinamicamente trovare le proprietà che quella connessione richiede.
- **Connection**: Interfacce che forniscono un contesto per eseguire SQL Statements e processare i loro risultati
- **Statement**: Interfacce che
  - **Statement**: Esegue un SQL statement statico e ottieni i suoi risultati.
  - **PreparedStatement**: Invia comandi precompilati in SQL, serve come container di query che eseguiamo più volte. Più veloce e più sicuro(anti SQL Inject)
  - **CallableStatement**: Invia la procedura salvata usando la sintassi JDBC.
- **ResultSet**: Interfacce di due tipi
  - **ResultSet**: Fornisce i metodi per accedere al risultato di una query
  - **ResultSetMetaData**: Fornisce metadata sul numero di colonne, tipi, proprietà di un resultSet.
- Java language extensions
  - Java language extensions per SQL.
- Java utility extensions
  - estensione of java.util.Date
  - **Utility**: Classi che
    - **Date** rappresenta SQL Date
    - **Time** rappresenta SQL Time
    - **Timestamp** rappresenta SQL Timestamp
  - **DatabaseMetaData**: Interfacce che forniscono informazioni generali o di loro stesse sui metadati che puoi ottenere a runtime.
- SQL metadata interfaces
  - dynamically discover what SQL databases and JDBC drivers can do

## Step di uso per DB

1. Caricare il driver JDBC Appropriato;
2. Richiedi connessione al DB;
3. Invia la query SQL al DB;
4. Processa il risultato.

## Apache Derby

Integrato con JAVA EE 6

Due modalità:

- **SERVER** multiutente
- **EMBEDDED** monoutente.

Java DB Database (e' possibile scegliere tre modalita' diverse): Funzionamento come:

- **In memory DB** (interamente in **memoria**, **monoutente**)
- **Server DB** (server in ascolto su **porta**, **multiutente**)
  - Tre diverse modalita' noi sceglieremo quella normale
    - Le altre due si riferiscono a connessioni che possono transitare come finto protocollo http per attraversare firewall

- **Local DB** (server autoincluso nella VS applicazione: attenzione diventa **MonoUtente**)
- 
- 

## 02/04/2019 - Laboratorio

---

Seguo slide JDBC.