HTTP

**(richiesta)**

* GET /blogpost HTTP/1.1
* Host: [www.supsi.ch](http://www.supsi.ch)
* User-Agent : Mozilla
* Accept: text/html, \*/\*
* Accept-Language: en-us
* Accept-Charset: ISO-8869-1
* Connection: keep-alive

**(risposta)**

* HTTP/1.1 200 OK
* Date: Thu, 24 Jul…
* Server: Apache-Coyote/1.1
* Content-Length: 6866
* Content-Type: text/html
* <html>
* …
* </html>

**URL** <http://www.supsi.ch:80/blogpost?user=ciao#type>

**Status Code:**

* 100-199: Informational
* 200-299: Success
  + 200 OK
* 300-399 Redirection
  + 301 Moved Permanently, 303 See Other, 304 Not Modified
* 400-499 Client Error
  + 400 Bad request, 401 Unauthorized, 403 Forbidden, 404 Not Found
* 500-599 Server Error
  + 500 Internal Server Error

Web

**Middleware**

* Gestisce comunicazione tra component
* garantisce disaccoppiamento
* semplifica riconfigurazione
* sviluppo indipendente dei componenti

**2-Tier**

* Presentazione e logica nel client
* Middleware
* Dati ed accesso ai dati nel server

**3-Tier**

* Presentazione nel client
* MiddleWare
* Logica App
* MiddleWare
* Dati ed accesso ai dati nel server

**N-Tier**

* Come la 3-Tier ma modularizzare I contenuti (es. Mobile, Desktop, Tablet,…)

**Thin Client** caratterizzato dalla presenza di nessuna o un ristretto numero di [applicazioni](https://it.wikipedia.org/wiki/Applicazione_(informatica)) poiché il suo esercizio dipende strettamente da un [server](https://it.wikipedia.org/wiki/Server) centrale per lo svolgimento della maggior parte delle proprie funzioni

**Thick Client** Difficile aggiornamento, siccome le operazioni vengono svolte in locale.

**Smart Client** Applicazioni locali che interagiscono con il web

**Progressive Web App** Non richiedono install, si caricano rapidamente tramite memorizzazione nella cache (esempio App meteo)

**Tipi di pagine**

* Statiche 🡪 pagina che non cambia mai
* Dinamiche 🡪 il server crea le pagine in base alla richiesta
* Attive 🡪 tipo con uso di flash

**Html vs xhtml** Elementi xhtml devono rispettare la corretta implementazione (struttura rigida), mentre invece html no

ApplicationServer - Servlet

**ApplicationServer** 🡪 fornisce l’infrastruttura e le funzionalità, sviluppo ed esecuzione

**CGI** 🡪 permette di aggiornare dinamicamente pagine web tramite il server

**Tomcat**

* Catalina 🡪 è l’engine di Tomcat.
* Coyote 🡪 Connettore http (inoltra le richieste a Catalina)
* Jasper 🡪 Motore JSP, compila il codice che verrà gestito da Catalina

**Servlet** Classi che gestiscono richieste e risposte http tramite metodi (doPost, doGet,…)@WebServlet(value=”/hello”)

Public class Hello extends HttpServlet{

public void doGet(HttpServletRequest r, HttpServletResponse re) throws…{

}}

Deployment descriptor 🡪 Configurazione dell applicazione (web.xml)

**JSP**  è una pagina costituita da codice HTML con inserite sezioni di codice Java (speciali tag)

Rest

**RESt** è uno stile di architettura, non c’è specifica, non c’è standard

* I client richiedono le rappresentazioni
* Ogni richiesta del client deve contenere tutte le informazioni per comprenderle (server)
* Risorse sono nominate da URI
* Le risorse sono collegate mediante URI

**Vincoli**

* Interfaccia uniforme (tramite link e messaggi auto-descrittivi)
* Stateless tramite richiesta si identifica cosa ricevere
* Cacheable i clint possono memorizzare alcune risposte
* Client-server separati
* Sistema stratificato (proxy, load-balancer,..) tra i collegamenti

**Operazioni Crud** 🡪 Operazioni di insiemi standard (create = post, read = get,…)

Proprietà

* Safe 🡪 non alterano mai le risorse (tipo GET)
* Idempotent 🡪 Richieste che se ripetute non hanno effetti collaterali ( GET,PUT, DELETE)
* Cacheable 🡪 metodi che non dipendono da risposte (GET, HEAD)

Framework

**Central Dispatcher** 🡪 Smista le richieste tramite routing

**Routing** 🡪 Mappatura dell’indirizzo per una determinata risorsa

**Template Engine** 🡪 genera file basati su modelli

Maggior parte dei framework evitano attacchi di tipo SQL Injection, Cross-Site,… Inoltre sono integrate alcune librerie JS

**MVC** 🡪 Model (i dati), View (la rappresentazione), Controller (Comandi)

Spring MVC si occupa di mappare metodi e classi JAVA con url differenti, più differenti tipologie di presentazione al client, applicazioni internazionalizzate

@Entity  
public class BlogPost {  
 @Id  
 @GeneratedValue

}

@Repository

public interface BlogPostRepository extends JpaRepository<BlogPost, Integer> { }

@Service  
public class BlogPostService {  
 @Autowired  
 private BlogPostRepository br;

}

@RestController  
public class BlogPostController {  
 @Autowired  
 private BlogPostService bs;

}

**SOA** architettura che fornisce servizi attraverso interfacce pubbliche

**WOA** esposizione a servizi su rete internet

**Web Services** interfaccia che descrive un insieme di operazioni tramite messaggi standard

**SOAP** è un protocollo XML per scambiare informazioni, strutturata e tipizzata

Mediante **WSDL** può essere infatti descritta l'[interfaccia](https://it.wikipedia.org/wiki/Interfaccia_(informatica)) pubblica di un Web service (ovvero una descrizione basata su XML) che indica come interagire con un determinato servizio: un "documento" WSDL contiene infatti, relativamente al Web service descritto, informazioni su:

* **Cosa** (può essere utilizzato)
* **Come** (utilizzarlo)
* **Dove** (utilizzarlo)