# Introduzione alle applicazioni WEB

Oltre al Web Server esiste l’Application Server, che è il posto dove viene eseguita l'applicazione. Esso può:

* eseguire script preimpostati. Protocollo CGI che si occupa della comunicazione tra Web Server e Application Server.
* engine/container permette l’esecuzione di programmi (node.js). Si usa per programmare la logica delle applicazioni, facile da mantenere. CGI viene gestito dall’interprete per il linguaggio usato.

### SERVER SIDE - JAVA SERVLET

Servlet = oggetti Java che stanno nell’Application Server che fungono da programmi.

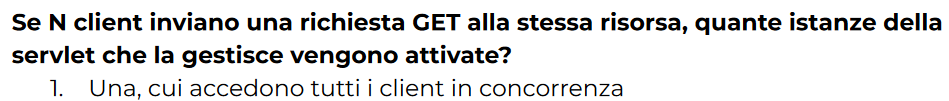
Le servlet implementano un’interfaccia rigida e ben nota.

I suoi metodi sono dei thread, e ad ogni richiesta dell’utente corrisponde un thread.

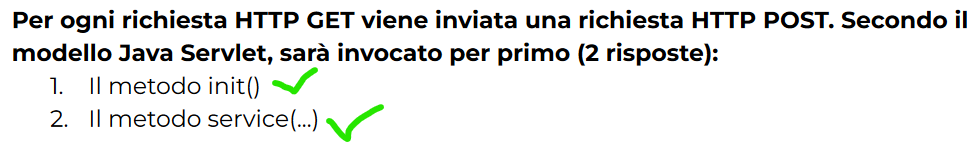
Le servlet mappano i principi del protocollo HTTP.

Ciclo di vita di una servlet:

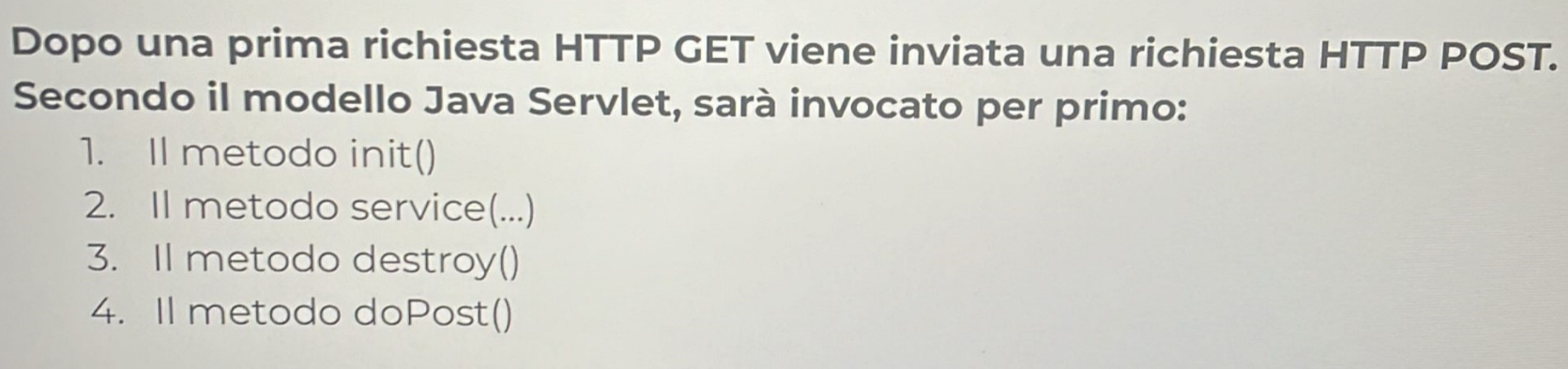
1. Una servlet viene creata dal container/engine, usando init()
   1. Quando viene effettuata la prima chiamata, la servlet viene condivisa da tutti client
   2. Ogni richiesta genera un Thread che esegue la doGet, doPost, etc…



1. Una servlet viene distrutta con destroy() all’occorrenza di uno dei 2 eventi:
   1. Quando non ci sono thread in esecuzione su quella servlet
   2. Quando è scaduto un timeout predefinito, ma bisogna verificare che service() non è ancora in esecuzione, perchè altrimenti bisogna aspettare







Annotazioni = nuove eccezioni java

### Java Server Pages

JSP = tecnologia/documento HTML con incorporato del codice Java. Eseguito sul server prima di inviare la pagina al browser del cliente. Separano la parte dinamica dei siti da HTML statico.

Le pagine sono costituite da tag tradizionali (HTML, XML) e da tag applicativi che controllano la generazione del contenuto.

**JavaBean**: Un JavaBean è una classe Java che deve avere un costruttore vuoto, metodi getter e setter e attributi privati.

Pagine JSP = UI dinamica e visualizzazione

JavaBean = logica di business

Per mantenere lo stato in una comunicazione stateless come HTML si usano cookie (memorizzati dal client) e le sessioni (memorizzati dal container). Quest'ultime sono rappresentate da un hashmap (chiave-valore), che è gestita da una JSP.

Ciclo di vita di una JSP:

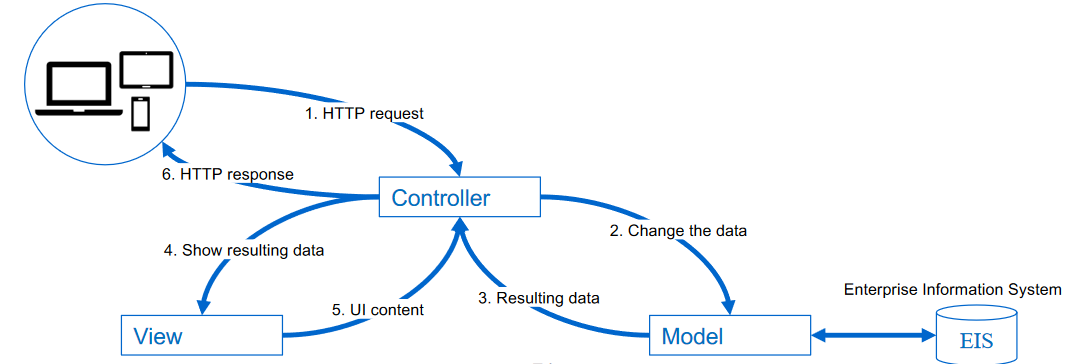


### IL PATTERN MVC – MODEL VIEW CONTROL

Esistono 2 tipi di architetture servlet: Quelle monolitiche e quelle MVC. MVC è un pattern di progettazione che separa le responsabilità in tre componenti principali:

* Model = oggetti trattati dalle applicazioni
* View = come la user interface funziona
* Controller = interfaccia tra i 2 componenti precedenti, in quanto stabilisce chi fa cosa

Separazione tra model e controller = integrazione tra dati e logica



**PRIMO MODO**

1. Il browser invia una richiesta per la pagina JSP
2. JSP accede a Java Bean e invoca la logica di business
3. Java Bean si connette al database e ottiene/salva i dati
4. La risposta, generata da JSP, viene inviata al browser

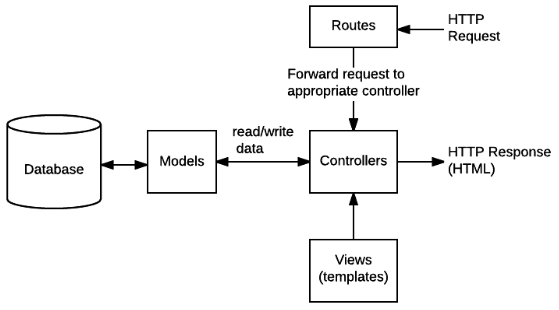
Ad ogni modo è complesso da mantenere e il passaggio del UI content al controller piuttosto che direttamente al client fa perdere tempo (si risolve con il forwarding).

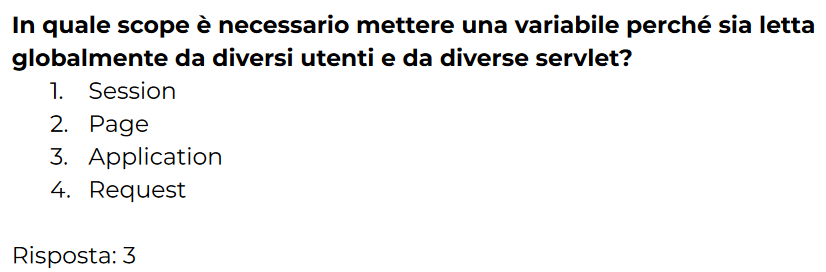
**Vantaggi:**

* separazione tra logica di business e logica di presentazione
* separazione tra logica di business e modello dei dati
* ogni componente ha una responsabilità ben definita

**Svantaggi:**

* aumento della complessità dovuta alla concorrenza, ricordiamo infatti che è un sistema distribuito
* inefficienza nel passaggio dei dati alla view, ovvero è presente un elemento in più tra cliente e controller





### SOA (Service-Oriented Architecture)

È un'architettura per la costruzione di applicazioni distribuite di grosse dimensioni. Riuso di funzionalità, evitando ripetizione del codice. *Per esempio la versione del software che il cliente ha diventa irrilevante.*

Svantaggi: il ciclo di vita del software è complicato, costoso, dependency hell, scarsa integrazione con soluzioni legacy (sistemi vecchi).

SOA è composto da servizi, identificati da URL, gestiti da WDSL, con formati ben noti come JSON o XML.

Usiamo WSDL per la composizione di servizi di terze parti.

SOA è stateless = servizi indipendenti dagli altri servizi.

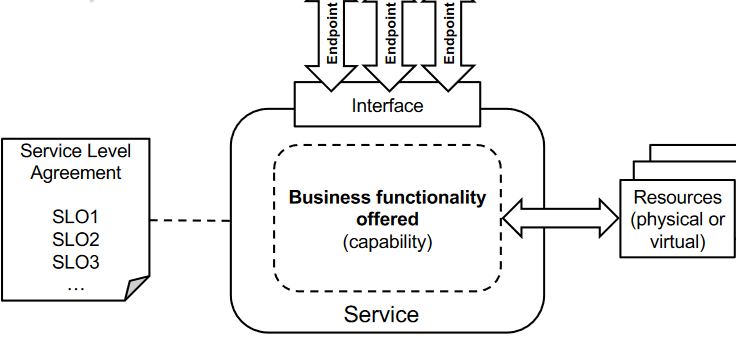
API Gateway veicola semplicemente le chiamate alle API.

L'accoppiamento dunque è debole, se fosse forte non sarei in grado di integrare così tanti servizi in modo così dinamico.

Servizio WEB = applicazione software che è identificato da URI, interfacce/API che possono essere descritte/definite/scoperte e supporta interazioni dirette con altri protocolli.

SLA è un contratto LEGALE tra provider e user che definisce:

* caratteristiche non funzionali (come, NON cosa)
* SLO (descrizione del servizio, quanto costa, tempi di risposta) di un servizio.



SOA usa la composizione di servizi. Ci sono 2 tipi di composizione:

* **orchestrazione** = un servizio è più importante di altri, di solito è una specie di controller. deve anche aspettare che altri micro-servizi facciano il lavoro per lui
* **coreografia** = ogni servizio sa quello che deve fare, dunque non c’è un controller o servizio principale. Ogni servizio non sa niente di altri servizi, sa solo che deve invocarli.

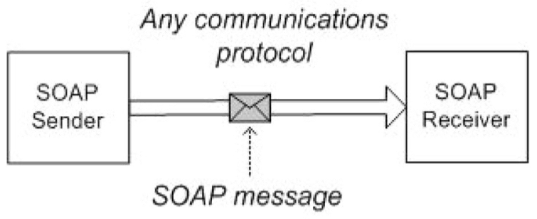
I **Business Processes** sono un workflow di azioni eseguiti per ottenere un risultato (servizio o prodotto). Sono creati dalla composizione di servizi e sono privi di orchestrazione .

**ESB (enterprise service bus)** è un sistema di comunicazione tra i servizi di SOA. Implementano un approccio coreografico e publish/subscribe.

Service requestor = Interagiscono con i Service Provider e trovano le agenzie.

### SOAP

Protocollo basato su XML per il trasporto dei messaggi e accesso ad oggetti.



### WSDL

Linguaggio di descrizione dei servizi WEB basato su XML. ESTREMAMENTE preciso, contiene tantissime informazioni.

* Informazioni sull'interfaccia
* Dichiarazioni di tipi di dati per tutti i messaggi
* Informazioni di binding sul protocollo di trasporto (HTTP, SMTP, UDP)
* Informazioni sull'indirizzo per la localizzazione del servizio (URI)

La complessità elevata ha ucciso SOAP e WSDL rendendoli troppo difficili da mantenere

Esiste un nuovo approccio -> REST

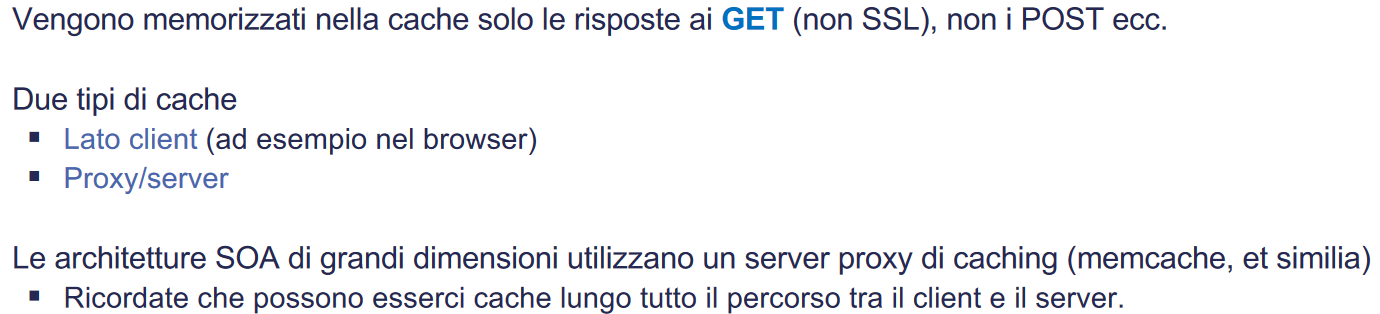
# REST

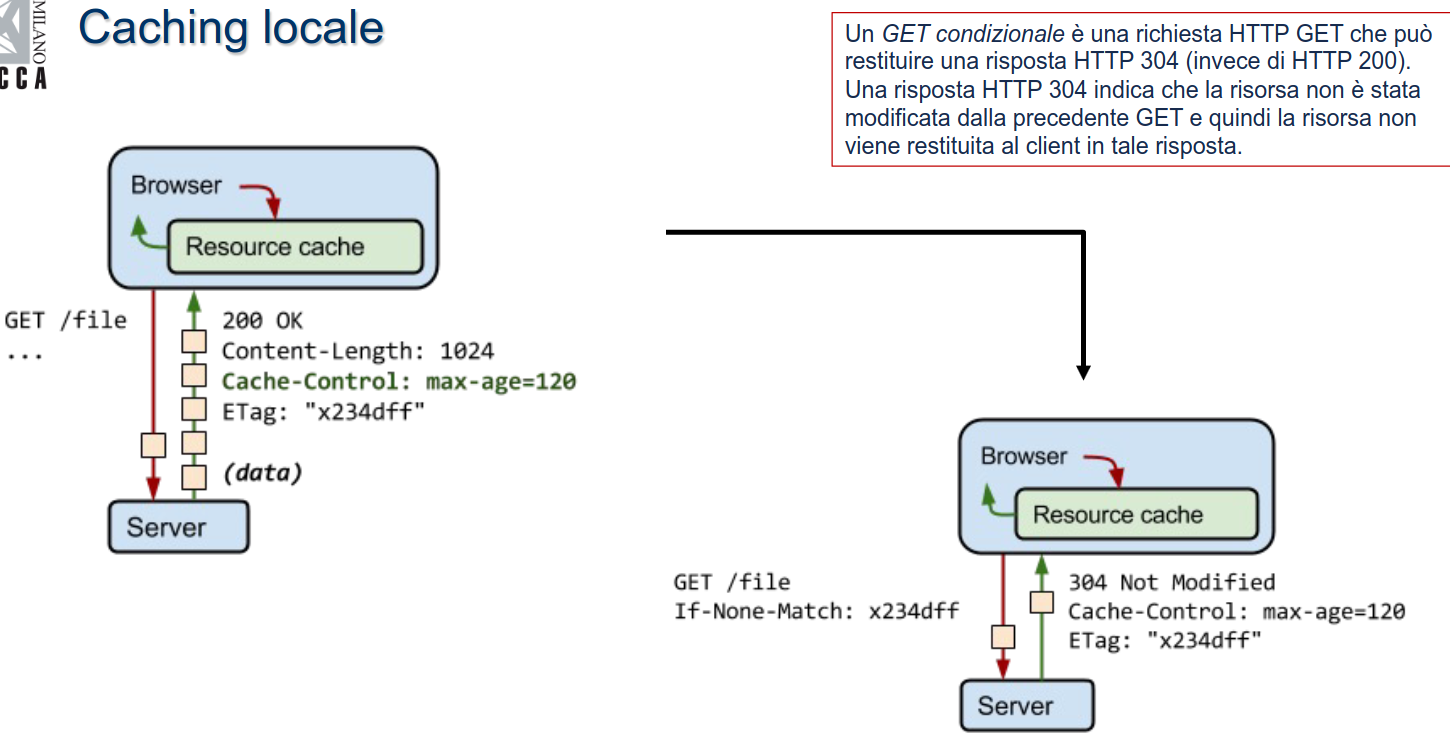
REST è una serie di regole (o consigli) per la scrittura di un servizio. L’idea è tornare alle origini e sfruttare HTTP in modo nativo per il trasporto dei msg. Non permette di compilare ed è meno potente ma proprio per questo è più semplice.

Le risorse sono centrali e identificate da un URI univoco.

Esiste un'interfaccia omogenea molto semplice dove c’è: Verbo + Identificatore URI.

Il formato dei messaggi è dato da XML e JSON. I msg sono stateless e autodescrittivi (proprietà di cacheability: le risorse si salvano in cache, riducendo la latenza).



*max-age=120 indica che la risorsa va tenuta in cache per 120 secondi. Avere una cache locale permette alla rete di ridurre il flusso di messaggi.*

Le parti cacheabili di REST sono risorse, GET e alcuni Header come -> cache control max age

Inoltre mentre SOAP/WSDL usano API standardizzate, REST usa HTTP (GET, DELETE…) e JSON (per rappresentare le risorse) per manipolare le risorse.

annotazioni = separazioni interfaccia e implementazione

