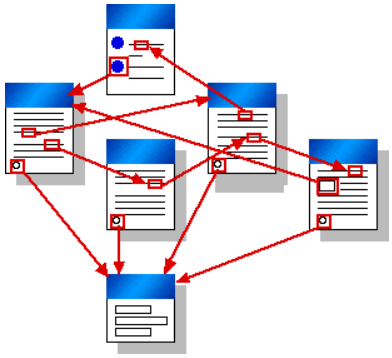
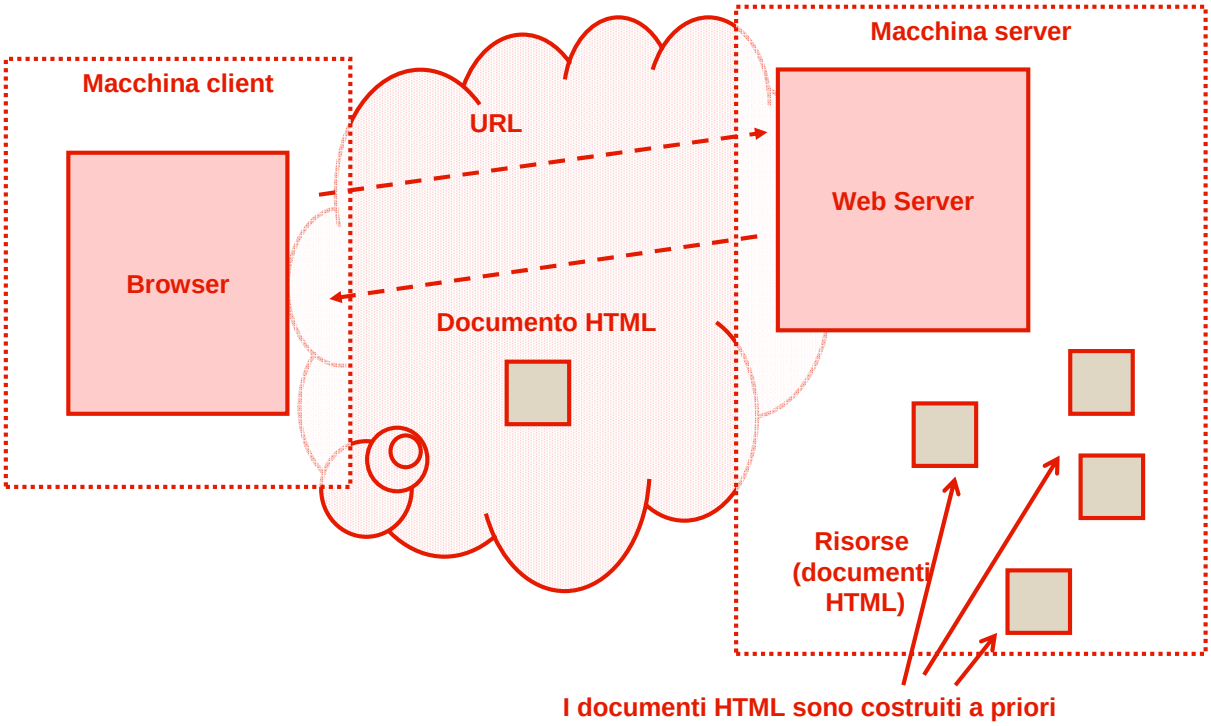
Evoluzione del modello
Il web dinamico

Modello statico

- Il modello che abbiamo analizzato fino ad ora, basato sul concetto di ipertesto distribuito, ha una natura essenzialmente statica
- Anche se l'utente può percorrere l'ipertesto in modi molto diversi l'insieme dei contenuti è prefissato:
  - Le pagine vengono preparate staticamente a priori
  - Non esistono pagine che vengono composte dinamicamente in base all'interazione con l'utente
- E' un modello semplice e molto potente ma presenta dei limiti

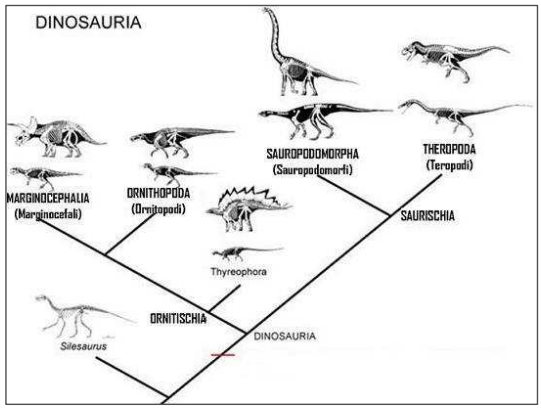


Modello del Web Statico



Limiti del modello statico

- Per capire quali sono i limiti del modello statico e come possono essere superati proviamo a ragionare su un semplice esempio
- Vogliamo costruire un'enciclopedia dei Dinosauri consultabile via web: **www.dino.it**
  - Possiamo creare una pagina HTML per ogni specie di dinosauro con testi e immagini
  - Possiamo poi creare una pagina iniziale che fa da indice basandoci sulla classificazione scientifica
  - Ogni voce è un link alla pagina che descrive un dinosauro



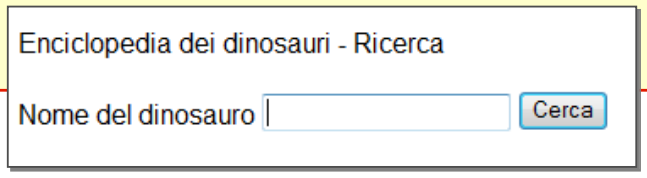
Limiti del modello statico

- Se vogliamo rendere più agevole l'accesso alle schede possiamo anche predisporre un'altra pagina con un indice analitico che riporta le specie di dinosauri in ordine alfabetico
- Tutto questo può essere realizzato facilmente con gli strumenti messi a disposizione dal web statico
- Il modello va però in crisi se proviamo ad aggiungere una funzionalità molto semplice: la ricerca per nome
- Quello che ci serve è una pagina con un form, costituito da un semplice campo di input e da un bottone, che ci consente di inserire il nome di un dinosauro e di accedere direttamente alla pagina che lo descrive

Ricerca

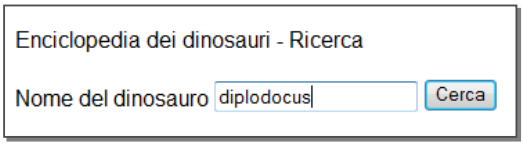
- Vediamo il codice della pagina HTML: abbiamo usato il metodo GET per semplicità

```
<html>
 <head>
  <title> Ricerca dinosauri </title>
 </head>
 <body>
   <p>Enciclopedia dei dinosauri - Ricerca</p>
   <form method="GET" action="http://www.dino.it/cerca">
     <p>Nome del dinosauro
     <input type="text" name="nomeTxt" size="20">
     <input type="submit" value="Cerca" name="cercaBtn">
     </p>
   </form>
 </body>
</html>
```



Eseguire la ricerca

- Se scriviamo il nome di un dinosauro e premiamo il bottone cerca si attiva una invocazione HTTP di tipo GET con un URL di questo tipo:
  `www.dino.it/cerca?nomeTxt=diplodocus&cercaBtn=Cerca`



- Il web server non è in grado di interpretare questa chiamata perché richiede una logica legata al particolare contesto
- E' quindi necessaria un'estensione specifica: un programma scritto appositamente per l'enciclopedia che:
  - Interpreta i parametri passati nel GET
  - Cerca nel file system la pagina diplodocus.html
  - La restituisce al Web server per l'invio al client

CGI

- La prima soluzione proposta per risolvere questo problema prende il nome di **Common Gateway Interface** (CGI)
- CGI è uno standard per interfacciare applicazioni esterne con un web server
- Le applicazioni che usano questo standard prendono il nome di **programmi CGI**
- Un programma CGI viene eseguito in tempo reale e fornisce perciò una **informazione dinamica**.
- Può essere scritto in qualunque linguaggio: ad esempio in C o in un linguaggio di script (tipicamente Perl)

Modello di interazione CGI

- Le operazioni si svolgono nel seguente ordine:
  - Il client, tramite HTTP, invia al server la richiesta di eseguire un programma CGI con alcuni parametri e dati in ingresso.
  - Il server, attraverso l'interfaccia CGI, chiama il programma passandogli i parametri e i dati inviati dal client.
  - Eseguite le operazioni necessarie, il programma CGI rimanda al server i dati elaborati (pagina HTML), sempre facendo uso dell'interfaccia CGI
  - Il server invia al client i dati elaborati dal programma CGI tramite il protocollo HTTP.

Comunicazione fra server e programma CGI

- I programmi CGI e il server comunicano in quattro modi:
  - **Variabili di ambiente** del sistema operativo
  - **Parametri sulla linea di comandi:** il programma CGI viene lanciato in una shell di sistema operativo e quindi può interpretare i parametri passati (usato soprattutto con il metodo GET)
  - **Standard Input** (usato soprattutto con il metodo POST)
  - **Standard Output:** per restituire al server la pagina HTML da inviare al client

Parametri: metodo GET

- Con il metodo GET il server passa il contenuto della form al programma CGI su un comando di linea di una shell del sistema operativo.
- Nel nostro esempio la URL era
  `www.dino.it/cerca?nomeTxt=diplodocus&cercaBtn=Cerca`
- Dove:
  - `www.dino.it` è l'indirizzo del server web
  - `cerca` è il nome del programma CGI
  - `nomeTxt=diplodocus&cercaBtn=Cerca` è la riga di comandi passata al programma cerca
- I comandi di linea hanno una lunghezza finita (massimo 256 caratteri in UNIX),
- La quantità di dati che possono essere inviati con il metodo GET è molto limitata.

Parametri: metodo POST

- Usando POST non viene aggiunto nulla alla URL specificata da ACTION
- Quindi il comando di linea nella shell del sistema operativo contiene solo il nome del programma CGI.
- Nel nostro esempio la URL sarà semplicemente `www.dino.it/cerca` a cui corrisponde un comando di linea `cerca` senza alcun parametro
- I dati del form, contenuti nell'header HTTP, vengono inviati al programma CGI tramite lo **standard input**
- In questo modo si possono inviare dati lunghi a piacimento, senza i limiti di GET
- In C per accedere ai dati si apre aprire un file su `stdin` e si leggono i campi del post con `fgetc()`:

```
nome = fgetc(stdin); // nomeTxt=diplodocus
btn = fgetc(stdin); //  cercaBtn=Cerca
```

Variabili di ambiente

- Prima di chiamare il programma CGI il web server imposta alcune variabili di sistema corrispondenti agli header HTTP, ad esempio:
  - **REQUEST_METHOD:** metodo usato dalla form.
  - **QUERY_STRING:** parte di URL che segue il "?"
  - **REMOTE_HOST:** host che ha inviato la richiesta
  - **CONTENT_TYPE:** tipo MIME dell'informazione contenuta nel body della richiesta (nel POST)
  - **CONTENT_LENGTH:** lunghezza dei dati inviati
  - **HTTP_USER_AGENT** nome e la versione del browser usato dal client.
- Se si implementa la CGI in C, si può usare `getenv()`:

```
utente = getenv(REMOTE_HOST);
```


Output

- Il programma CGI elabora i dati in ingresso ed emette un output per il client in attesa di risposta.
- Per passare i dati al server il programma CGI usa `stdout`: in C si può usare la funzione `printf()`.
- Il server preleva i dati dallo standard output e li invia al client secondo il protocollo HTTP, ad esempio:

```
HTTP/1.0 200 OK
Date: Wednesday, 02-Feb-94 23:04:12 GMT
Server: NCSA/1.1
MIME-version: 1.0
Last-modified: Monday, 15-Nov-93 23:33:16 GMT
Content-type: text/html
Content-length: 2345

<HTML><HEAD><TITLE>
...
```

Parte generata dal web server

Parte generata dal programma CGI

Configurazione del server

- Se arriva la URL `www.dino.it/cgi-bin/cerca` il server deve rendersi conto che `cerca` non è un documento HTML ma un programma CGI.
- Perché ciò accada è necessario che:
  - I programmi CGI siano tutti in un'apposita directory
  - Nella configurazione del server sia specificato il path ove trovare i programmi CGI e l'identificatore che indica che è richiesta l'esecuzione di una applicazione.
- Di solito si sceglie come identificatore `/cgi-bin/`
- Tutto ciò che segue l'identificatore viene interpretato dal server come nome di programma da eseguire.
- Il server cerca il programma specificato nel path che è stato indicato nella configurazione.

CGI e dinosauri

- Un meccanismo come quello appena descritto consente di risolvere il problema della ricerca nell'esempio dell'enciclopedia
- Il browser usando il metodo GET invia il contenuto del campo di ricerca nella parte query dell'URL:
  `www.dino.it/cgi-bin/cerca?nomeTxt=diplodocus&cercaBtn=Cerca`
- Il web server usa l'interfaccia CGI per passare i parametri presenti nell'URL ad un programma denominato `cerca`
- `cerca` usa il valore del parametro `nomeTxt` per cercare le pagine che contengono il termine inserito nel campo di ricerca (`diplodocus`)
- Usa `stdout` per costruire una pagina con un elenco di link alle pagine che contengono il termine

Altri problemi

- La nostra enciclopedia ha anche un problema di manutenibilità
- Se vogliamo aggiungere un dinosauro dobbiamo infatti:
  - Creare una nuova pagina con una struttura molto simile alle altre
  - Aggiungere il link alla pagina nell'indice principale (quello basato sulla classificazione delle specie)
  - Aggiungere un link nell'indice alfabetico
- La pagina di ricerca invece non richiede alcuna modifica
- Se poi volessimo cambiare l'aspetto grafico della nostra enciclopedia dovremmo rifare una per una tutte le pagine

DynamicDino.it

- La soluzione più razionale è quella di separare gli aspetti di contenuto da quelli di presentazione
- Utilizziamo un database relazionale per memorizzare le informazioni relative ad ogni dinosauro
- Realizziamo alcuni programmi CGI che generano dinamicamente l'enciclopedia
  - `scheda`: crea una pagina con la scheda di un determinato dinosauro
  - `indice`: crea la pagina di indice per specie (tassonomia)
  - `alfabetico`: crea l'indice alfabetico
  - `cerca`: restituisce una pagina con i link alle schede che contengono il testo inserito dall'utente
- Tutti e 4 i programmi usano il DB per ricavare le informazioni utili per la costruzione della pagina

Struttura del database

- Per semplicità possiamo utilizzare una sola tabella con la struttura sotto riportata (non è normalizzata)
- Il campo testo contiene la pagina HTML
- Gli altri campi permettono di costruire agevolmente l'indice basato sulla tassonomia

Specie	Ordine	Famiglia	Genere	Testo
Dicraeosaurus hansemanni	Saurischia	Dicraeosauridae	Dicreaosaurus	<p>Il dicreosauro (gen. Dicraeosaurus) è un dinosauro erbivoro vissuto in Africa orientale nel Giurassico superiore (Kimmeridgiano, circa 150 milioni di anni fa).
…	…	…	…	…
…	…	…	…	…

Vantaggi della nuova soluzione

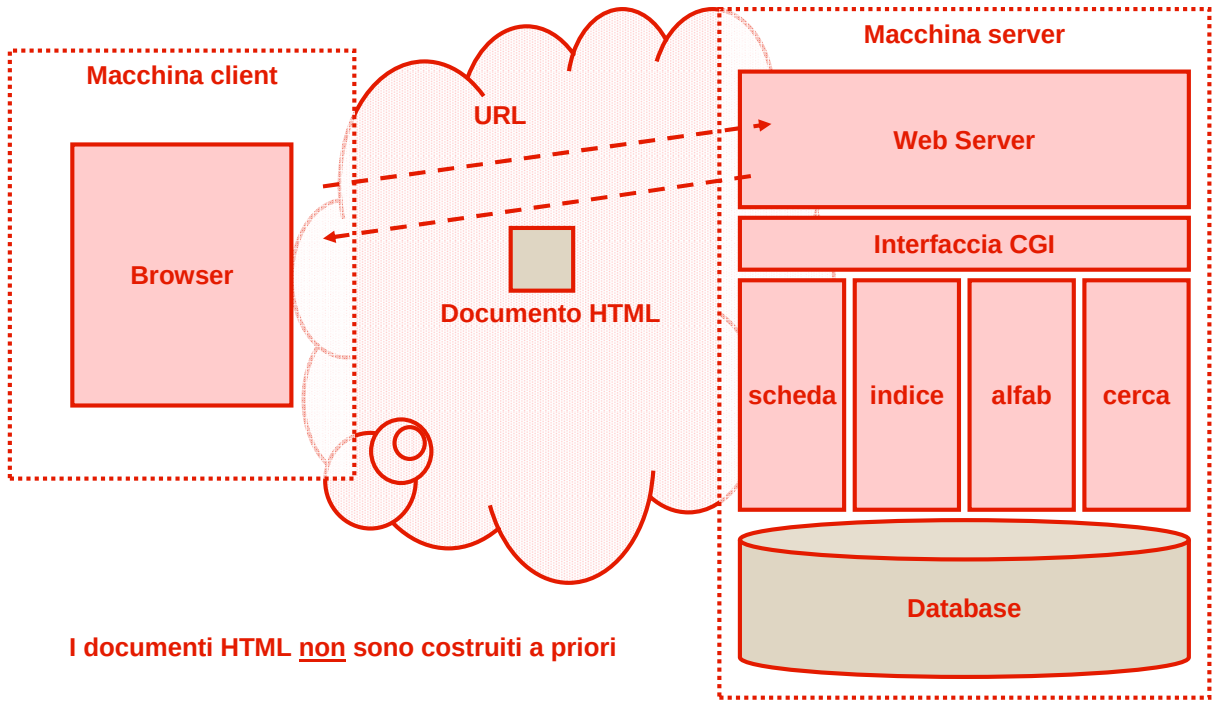
- In questo modo la gestione dell'enciclopedia è sicuramente più semplice:
  - Basta inserire un record nel database per aggiungere una nuova specie
  - L'indice tassonomico e quello alfabetico si aggiornano automaticamente
- E' anche possibile cambiare agevolmente il layout di tutte le pagine
  - E' infatti possibile utilizzare una pagina HTML di base (template) con tutte le parti fisse
  - Il programma scheda si limita a caricare il template e a inserire le parti variabili
- Per cambiare l'aspetto grafico di tutte le schede è sufficiente agire una sola volta sul template

Un nuovo modello

- Il modello è cambiato in modo significativo
- Abbiamo aggiunto una forma di elaborazione sul lato server
- L'insieme delle CGI che gestiscono l'enciclopedia costituisce un'applicazione distribuita
- Ogni CGI può essere vista come una procedura remota invocata tramite HTTP

URL / CGI	Procedura
`/scheda?specie=Diplodocus`	`scheda("Diplodocus");`
`/indice`	`indice();`
`/alfabetico`	`alfabetico();`
`/cerca?testo=diplod`	`cerca("diplod");`

Modello del Web dinamico



Tutto a posto?

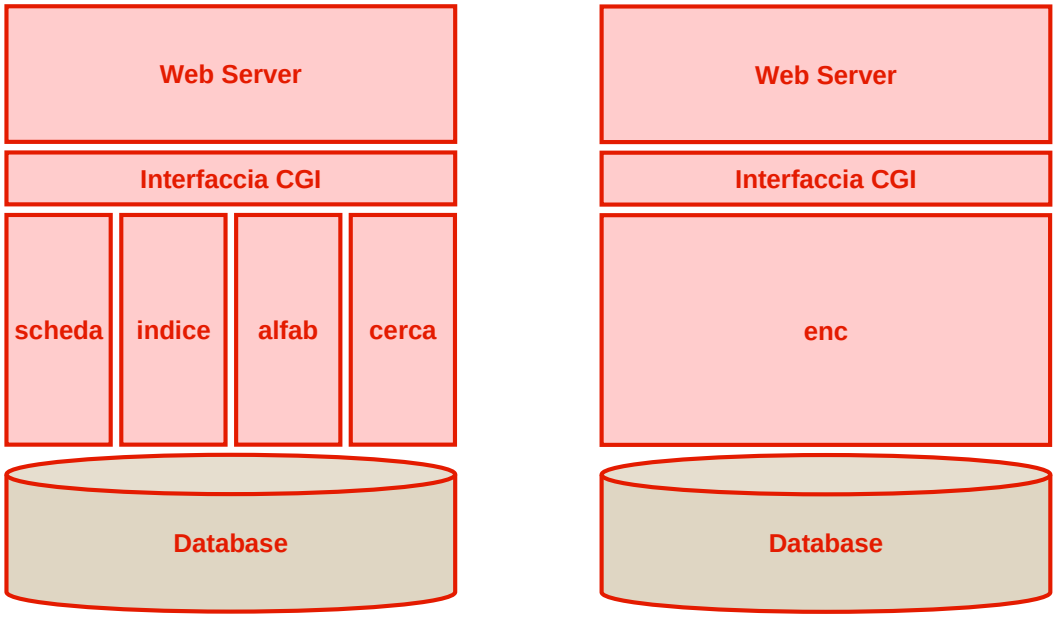
- L'architettura che abbiamo appena visto presenta numerosi vantaggi ma soffre anche di diversi problemi
  - Le CGI, soprattutto se scritte in C, possono essere poco robuste (soggette a errori bloccanti)
  - Abbiamo scarse garanzie sulla sicurezza
  - Ci sono problemi di prestazioni: ogni volta che viene invocata una CGI si crea un processo che viene distrutto alla fine dell'elaborazione
  - Ogni programma CGI deve reimplementare tutta una serie di parti comuni: accesso al DB, logica di interpretazione delle richieste HTTP e di costruzione delle risposte, gestione dello stato ecc.
- Esistono varianti di CGI che risolvono il problema delle prestazioni (FastCGI)

Una sola applicazione

- Per ovviare all'ultimo punto si potrebbe realizzare una sola CGI (enc) che implementa tutte e quattro le funzionalità (vedi schema sottostante)
- In questo modo però si ha un'applicazione monolitica e si perdono i vantaggi della modularità
- Gli altri problemi rimangono invariati

URL / CGI	Procedura
`/enc?azione=scheda&specie=Diplodocus`	`scheda("Diplodocus");`
`/enc?azione=indice`	`indice();`
`/enc?azione=alfabetico`	`alfabetico();`
`/enc?azione=cerca&testo=diplod`	`cerca("diplod");`

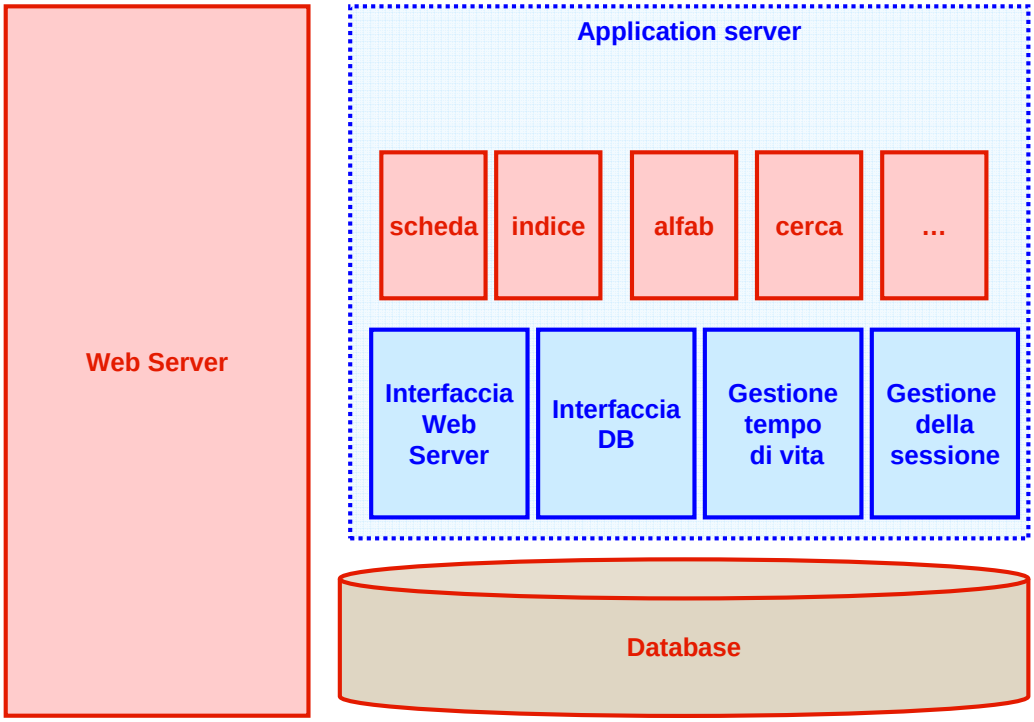
Le due soluzioni a confronto



Application server

- La soluzione migliore è quella di realizzare un contenitore in cui far "vivere" le funzioni server-side
- Il contenitore si preoccupa di fornire i servizi di cui le applicazioni hanno bisogno:
  - Interfacciamento con il Web Server
  - Interfacciamento con il database
  - Gestione della sicurezza
  - Gestione del tempo di vita (attivazione on-demand delle funzioni)
- Si ha così una soluzione modulare in cui le funzionalità ripetitive vengono portate a fattor comune
- Un ambiente di questo tipo prende il nome di **application server**

Architettura basata su application server



Application server e tecnologie server side

- Le due tecnologie più diffuse nell'ambito degli application server sono 2:
  - .NET di Microsoft
  - Java J2EE
- Ci sono altre tecnologie interessanti:
  - Ad esempio quelle basate su python (zope…)
- Altre soluzioni hanno una struttura più semplice e non sono application server a tutti gli effetti (si parla di moduli di estensione del web server):
  - PHP (molto diffuso e di semplice utilizzo)
  - Le vecchie tecnologie ISAPI e ASP di Microsoft
  - Quelle basate su ruby (ruby on rails)

Altri aspetti: lo stato

- L'enciclopedia dei dinosauri è un'applicazione **stateless**
- Il server e i programmi CGI non hanno necessità di tener traccia delle chiamate precedenti
- L'interazione tra un Client e un Server può essere infatti di due tipi:
  - **Stateful**: esiste lo stato dell'interazione e quindi l'n-esimo messaggio può essere messo in relazione con gli n-1 precedenti.
  - **Stateless**: non si tiene traccia dello stato, ogni messaggio è indipendente dagli altri

Interazione stateless

- Un'interazione stateless è possibile solo se il protocollo applicativo è progettato con **operazioni idempotenti**.
- Operazioni idempotenti producono sempre lo stesso risultato, per esempio, un Server fornisce sempre la stessa risposta a un messaggio M indipendentemente dal numero di messaggi M ricevuti dal Server stesso.
- Nel nostro caso tutte e 4 le operazioni gestire dall'enciclopedia (`indice`, `alfabetico`, `scheda` e `cerca`) sono idempotenti.

Interazione stateful

- Non tutte le applicazioni possono fare a meno dello stato
- Ad esempio se è prevista un'autenticazione diventa necessario tener traccia fra una chiamata e l'altra del fatto che l'utente si è autenticato e quindi nasce l'esigenza di avere uno stato
- Se estendiamo la nostra applicazione per consentire ad utente autorizzato di modificare le schede dei dinosauri via web anche la nostra applicazione cessa di essere stateless

Diversi tipi di stato

- Parlando di applicazioni web è possibile classificare lo stato in modo più preciso:
  - **Stato di esecuzione** (insieme dei dati parziali per una elaborazione): rappresenta un avanzamento in una esecuzione, per sua natura è uno stato volatile normalmente mantenuto in memoria come stato di uno o più oggetti
  - **Stato di sessione** (insieme dei dati che caratterizzano una interazione con uno specifico utente): la sessione viene gestita di solito in modo unificato attraverso l'uso di istanze di oggetti specifici
  - **Stato informativo persistente** (ad esempio gli ordini inseriti da un sistema di eCommerce): viene normalmente mantenuto in una struttura persistente come un database

Il concetto di sessione

- La **sessione** rappresenta lo stato associato ad una sequenza di pagine visualizzate da un utente:
- Contiene tutte le informazioni necessarie durante l'esecuzione;
  - Informazioni di sistema: IP di provenienza, lista delle pagine visualizzate…
  - Informazioni di natura applicativa: nome e cognome, username, quanti e quali prodotti ha inserito nel carrello per un acquisto…
- Lo **scope di sessione** è dato da:
  - **Tempo di vita** della interazione utente (lifespan)
  - **Accessibilità**: la richiesta corrente e tutte le richieste successive provenienti dallo stesso processo del browser.

Il concetto di conversazione

- La **conversazione** rappresenta una sequenza di pagine di senso compiuto (ad esempio l'insieme delle pagine necessarie per comperare un prodotto)
- È univocamente definita dall'insieme delle pagine che la compongono e dall'insieme delle interfacce di input/output per la comunicazione tra le pagine (**flusso della conversazione**)

Esempio di conversazione: acquisto online

1. L'utente inserisce username e password: **inizio della conversazione**
   - Il server riceve i dati e li verifica con i dati presenti nel DB dei registrati: viene **creata la sessione**.
2. L'utente sfoglia il catalogo alla ricerca di un prodotto
   - Il server lo riconosce attraverso i dati di sessione
3. L'utente trova il prodotto e lo mette nel carrello:
   - La sessione viene aggiornata con le informazioni del prodotto
4. L'utente compila i dati di consegna
5. L'utente provvede al pagamento, **fine della conversazione** di acquisto
   - L'ordine viene salvato nel DB (stato persistente)
   - La sessione è ancora attiva e l'utente può fare un altro acquisto o uscire dal sito

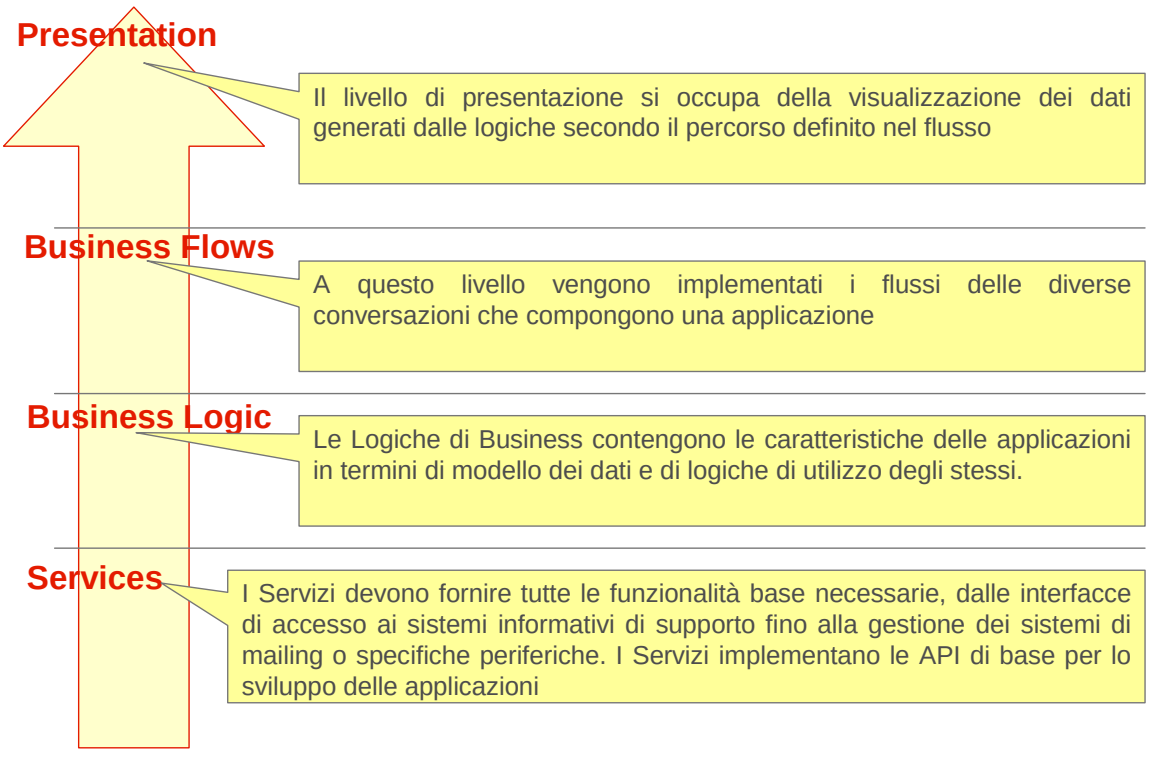
Tecniche per gestire lo stato

- Lo stato della conversazione deve presentare i seguenti requisiti:
  - Deve essere condiviso dal Client e dal Server
  - È associato ad una o più conversazioni effettuate da un singolo utente
  - Ogni utente possiede il suo singolo stato
- Ci sono due tecniche di base per gestire lo stato:
  - Utilizzo della struttura dei cookie
  - Gestione di uno stato sul server per ogni utente collegato (sessione)
- La gestione della sessione è uno dei servizi messi a disposizione da un **application server**

Applicazioni multilivello

- Sviluppare applicazioni secondo una logica ad oggetti e/o a componenti significa scomporre l'applicazione in blocchi, servizi e funzioni.
- È molto utile separare logicamente le funzioni necessarie in una struttura multilivello (**multitier**) al fine di fornire astrazioni via via più complesse e potenti a partire dalle funzionalità più elementari.
- Nel tempo si è affermata una classificazione indipendente dalla implementazione tecnologica, basata su una struttura a 4 livelli principali.
- Questa struttura non fornisce dettagli implementativi, non specifica quali moduli debbano essere implementati client side o server side, né nessuna altra specifica tecnica: è una architettura essenzialmente logico funzionale

Schema di struttura multilivello



Servizi

- I servizi realizzano le funzioni di base per lo sviluppo di applicazioni:
- Accesso e gestione delle sorgenti di dati:
  - Database locali
  - Sistemi informativi remoti
  - Sistemi di messaging and queuing
  - Legacy Systems (termine generico che identifica applicazioni esterne per la gestione aziendale – OS390, AS400, UNIX systems, ecc…)
- Accesso e gestione risorse:
  - Stampanti
  - Sistemi fax, sms, e-mail
  - Dispositivi specifici, macchine automatiche…

Business logic

- E' l'insieme di tutte le funzioni offerte dall'applicazione
- Si appoggia sui servizi per implementare i diversi algoritmi di risoluzione e provvedere alla generazione dei dati di output.
- Esempi di moduli di business logic possono essere:
  - Gestione delle liste di utenti
  - Gestione cataloghi online
- A questo livello:
  - non è significativo quali sono le sorgenti di dati (gestite dal livello dei servizi)
  - non è significativo come arrivano le richieste di esecuzione dei servizi e come vengono gestiti i risultati ai livelli superiori
- La business logic presenta un elevato grado di riuso

Business flow

- Una conversazione è realizzata da un insieme di pagine collegate in un flusso di successive chiamate.
- Il **business flow** raccoglie l'insieme delle chiamate necessarie per realizzare una conversazione
- Ogni chiamata deve:
  - Caricare i parametri in ingresso,
  - Chiamare le funzioni di business logic necessarie per effettuare l'elaborazione
  - Generare l'output che dovrà essere visualizzato.
- Un flusso identifica quindi univocamente una conversazione,
- L'astrazione fornita dal livello della business logic permette di definire l'esecuzione delle singole pagine in modo indipendente dalla struttura dei dati e degli algoritmi sottostanti

Presentazione

- Il business flow è in grado di fornire i dati di output necessari;
- Il **livello di presentazione** ha il compito di interpretare questi dati e generare l'interfaccia grafica per la visualizzazione dei contenuti (rendering).
- Questi due livelli sono concettualmente divisi poiché la generazione dei dati è logicamente separata dalla sua rappresentazione e formattazione.
- Questo permette di avere diverse tipi di rappresentazione degli stessi dati, per esempio una rappresentazione in italiano ed una in inglese o una in HTML ed una in XML.

Strutture semplificate

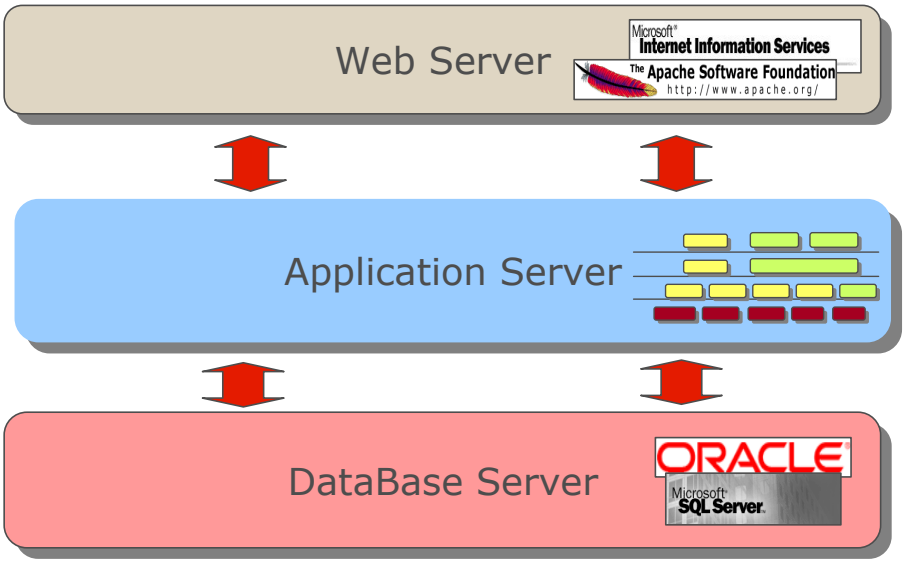
- Non tutte le tecnologie permettono di rispettare questa suddivisione,
- In molti casi i sistemi vengono realizzati a 2 o 3 livelli.
- Queste semplificazioni portano in certi casi a miglioramenti nelle performance e nella rapidità di sviluppo, ma comportano un netta riduzione della leggibilità e della manutenibilità
- Come sempre il compromesso viene deciso in base al contesto, non esiste la soluzione ideale ad ogni situazione.
- Esempio di semplificazione a 2 livelli:

Presentation **Business Flows**

Business Logic **Services**

Architetture dei sistemi web

- La struttura rappresenta rappresenta uno dei tre elementi di un sistema web: l'application server
- Ne esistono almeno altri due: il web server e il database server



Distribuzione dei servizi

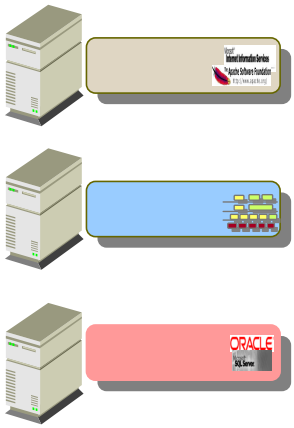
- La struttura a 3 livelli rispecchia i 3 principali servizi che realizzano un sistema Web.
- Questi 3 servizi possono risiedere sullo stesso HW oppure essere divisi su macchine separate:
- Si parla in questo caso di **distribuzione verticale** dell'architettura
  - E' molto efficiente perché non necessita di nessun accorgimento specifico
  - Viene realizzata essenzialmente per motivi di performance soprattutto quando si dividono il livello applicativo da quello database.
- Non prevede replicazione, non è quindi utile per risolvere problemi di fault tolerance.

Distribuzione dei servizi - 2

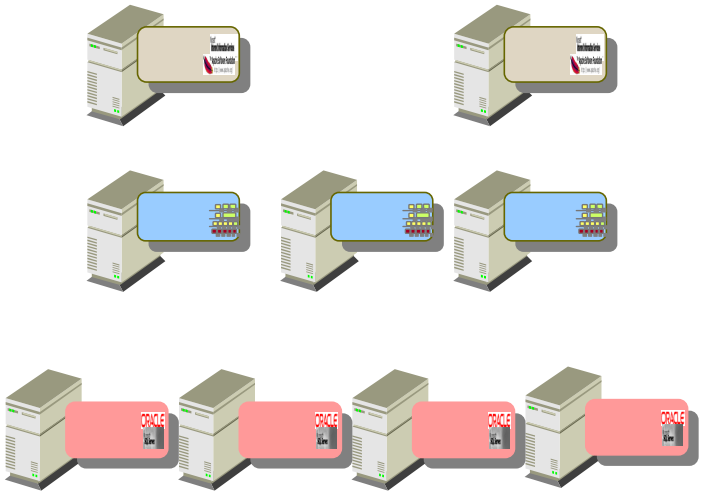
- Orizzontalmente ad ogni livello è possibile replicare il servizio su diverse macchine
- Si parla in questo caso di **distribuzione orizzontale**,
  - necessità di importanti accorgimenti strettamente dipendenti dalla tecnologia d'uso.
- Essendo una distribuzione per replicazione è possibile implementare politiche per la gestione della fault tolerance

Distribuzione verticale e orizzontale

Distribuzione Verticale



Distribuzione Orizzontale



Replicazione: database

- Il database server è un server stateful
- La replicazione è molto delicata perché deve mantenere il principio di atomicità delle transazioni.
- I database commerciali, come Oracle e Microsoft SQL Server prevedono delle configurazioni di **clustering** in grado di gestire in modo trasparente un numero variabile di CPU e macchine distinte.
- Sistemi RAID (Redundant array of independent disks) per i dati.

Replicazione: applicazione

- L'unico stato necessario a livello applicativo è la sessione.
- Può accadere però che l'application server utilizzi oggetti o componenti con stato per motivi di performance (cache) o altre necessità specifiche.
- Alcuni framework disponibili sul mercato permettono la replicazione attraverso tecniche di clustering molto simili a quelle dei sistemi database
- Altri framework non sono in grado di replicare orizzontalmente.
- Se si mantiene lo stato concentrato all'interno della sessione, e la sessione viene gestita attraverso i cookie, è possibile realizzare un framework applicativo completamente stateless,
- Si ottiene così una configurazione completamente replicabile orizzontalmente.

Replicazione: web server

- Il web server è stateless, non crea problemi nella replicazione.
- L'unicità degli URL può essere gestita attraverso diverse soluzioni sia hardware che software.
- Si possono applicare politiche di **load balancing** con diverse euristiche usando dispositivi appositi.