Domande orale

⚠ Non è del tutto esaustivo (da approfondire soprattutto la parte di backend avanzato)

▼ ***Differenza tra container pesante e container leggero***

Questa distinzione è un concetto chiave nel contesto dello sviluppo e del deployment di applicazioni web. La differenza si fonda sulla quantità di funzionalità offerte e i loro effetti sulle prestazioni, manutenibilità e scalabilità.

Un container pesante, innanzitutto, fornisce un'ampia varietà di servizi, come la gestione di transazioni, la sicurezza, la gestione delle risorse, pooling di connessioni ecc. Un esempio di container pesante è ***apache tomcat***. Spesso sono più onerosi e la configurazione è più complessa dei leggeri.

Un container leggero ha un ambiente di runtime più snello, offre comunque le funzionalità di base come l'iniezione di dipendenze (*dependency injection*) e il ciclo di vita delle varie componenti, ma lascia da parte alcune funzionalità più pesanti come la gestione di transazioni o la sicurezza a livello enterprise. Un container leggero è più flessibile, usando meno risorse e con tempi di avvio più rapidi, è più semplice da configurare e più estensibile, nel senso che si integra facilmente con altre tecnologie e framework.

La scelta ovviamente dipende dalle necessità. Nel caso di un container pesante, sono offerte più funzionalità, quindi per un'applicazione con requisiti avanzati, ma ha un'impatto maggiore sull'utilizzo delle risorse e prestazioni. Un container leggero, essendo più snello, è più scalabile, più manutenibile, e in generale più agile.

▼ ***Ciclo di vita servlet (normale/multithread e deprecato/singlethread)***

Il ciclo di vita multithread è il modello di default supportato dalle API java servlet in JEE. Si articola in tre fasi principali:

- Inizializzazione: quando il container servlet richiede per la prima volta la Servlet, viene caricata in memoria. Il container quindi invoca

il metodo `init()` della servlet, invocato solo una volta all'inizio del suo ciclo di vita.

- Gestione delle richieste: dopo essere stata inizializzata, la Servlet è pronta a ricevere e servire le richieste. Ogni volta che viene ricevuta una richiesta, viene invocato il metodo `service()` che può essere chiamato nello stesso momento da più thread e determina il tipo di richiesta, delegandola a metodi specifici come `doGet()` e `doPost()`.
- Distruzione: quando il container servlet vuole terminare la Servlet, chiama il metodo `destroy()` , utilizzato per rilasciare le risorse o per altre operazioni di pulizia. Dopo questo metodo, la Servlet non è più in memoria.

Per quanto riguarda il ciclo di vita basato sul modello single-thread, era pensato per servire una richiesta alla volta, per evitare concorrenze. Tuttavia è stato deprecato a seguito di alcune criticità: innanzitutto non è scalabile, in quanto richiede molte istanze della Servlet quando c'è un alto volume di richieste, ognuna delle quali consuma risorse, e il container deve poi gestire più istanze della Servlet.

▼ *Come si fa caching sul web browser in http*

Fare caching sul browser è una tecnica che permette di salvare risorse web (come immagini, pagine html, file javascript ecc) con lo scopo di ridurre i tempi di caricamento e il traffico di rete. Tramite header come `Cache-control` oppure `Expires` possiamo impostare dei valori tali che alcune risorse rimangano in cache, uno degli esempi più frequenti è quello di settare una data di scadenza o un timer, entro i quali i dati salvati saranno validi, oltre i quali saranno scaduti. Un esempio molto usato è `Cache-control: max-age=3600` significa che la risorsa può essere memorizzata in cache (e riusata a sbrega) per un'ora, dopo quest'ora sarà necessaria una nuova richiesta al server. Lato Server, possiamo fare in modo che gli header HTTP vengano modificati tramite `response . setHeader ( "Cache-Control" , "max-age=3600" )` (sempre legato all'ultimo esempio).

▼ *Dependency injection in Spring*

In Spring, la funzionalità più importante è l'iniezione di dipendenze, un meccanismo potente per gestire le dipendenze tra gli oggetti in maniera automatica. In Spring, si può implementare fondamentalmente in due modi: tramite setter o tramite costruttori.

Per esempio, se ho un oggetto `Libro` dotato di un `Autore`, senza la DI dovrei istanziare un nuovo autore nel costruttore del libro, in questo caso c'è un forte accoppiamento tra i due componenti, quindi se voglio cambiare l'implementazione di uno dovrei molto probabilmente cambiare anche il codice dell'altro, anche la testabilità si complica, e non avrei controllo sul ciclo di vita degli autori utilizzati dai libri. Invece, con la DI, posso "iniettare" l'autore nel libro (tramite costruttore o setter) , migliorando la flessibilità del codice, la sua manutenibilità e modularità.

Non serve più inserire a mano il binding tra i componenti, ma c'è un componente factory, detto *BeanFactory*, che si occupa di ritrovare gli oggetti per nome e gestirne le relazioni. Grazie a ciò, il codice risulta più flessibile, più modulare e più manutenibile.

▼ *Direttiva include jsp*

Al fine di promuovere la manutenibilità, il riutilizzo e la modularità, si possono importare altre pagine all'interno di una JSP grazie alla direttiva `include`. In questo modo, il file che si vuole importare viene incluso nella Java Server Page chiamante nella fase di traduzione, quindi prima che la pagina venga compilata in una servlet.

Altre direttive che abbiamo visto sono `page` che permette di importare package e `taglib` per caricare una libreria di custom tag definite dallo sviluppatore.

▼ *Differenza tra due send su un unico oggetto xmlhttprequest e due send su due oggetti separati*

La differenza sta principalmente nel modo in cui vengono gestite le richieste e le dipendenze (o indipendenze) tra loro.

Con due `send()` sullo stesso oggetto xhr, innanzitutto una volta inviata la prima richiesta, l'oggetto xhr è in uso fino a fine richiesta. Per quanto riguarda lo stato, un secondo tentativo generalmente genera un errore o viene ignorato perché l'oggetto si trova in uno stato tale per cui non è in grado di accettare una nuova richiesta. Se anche questo non generasse problemi, sarebbe comunque complicato gestire le risposte ed ogni volta distinguere a quale richiesta si riferisce l'oggetto.

Con due oggetti distinti, invece, ogni richiesta è indipendente dall'altra e tra loro non iterferiscono, per la gestione dei risultati ogni oggetto avrà il proprio set di callback (come `onreadystatechange`, `onerror`, `onload` ecc) e sarà più semplice gestire le riposte individualmente. Favorisce inoltre il

parallelismo ottimizzando i tempi di risposta, e in caso di errori ovviamente non si influenzano a vicenda.

In conclusione, è consigliato utilizzare oggetti xhr indipendenti tra loro per richieste multiple, essendo inoltre un approccio più conforme agli standard, più facile da gestire e meno soggetto ad anomalie.

▼ *Quando usare jsp e quando usare servlet*

L'utilizzo di servlet è più indicato per le logiche di business complesse e manipolazione dei dati, interagire con i database e offrono un controllo maggiore sulle richieste HTTP.

Le jsp sono più indicate per la presentazione e visualizzazione dei dati, il codice java è ben integrato nelle pagine HTML, e quando la business logic non è particolarmente complessa sono semplici e leggibili.

Utilizzando jsp e servlet in maniera complementare, si può facilmente soddisfare il concetto di "separazione delle preoccupazioni", con le servlet che si occupano di gestire il processamento dei dati mentre le jsp per la loro presentazione.

▼ *Differenza metodi get e post*

Per quanto riguarda l'utilizzo, GET si usa principalmente per richiedere dati da un server, adatto per navigare o per eseguire ricerche, POST invece per inviare dati al server come nella compilazione di form. Nella quantità di dati, GET è più limitato perché vengono allegati all'URL, nel senso che fanno proprio parte della stringa, per lo stesso motivo è meno sicuro, cioè non posso inserire dati sensibili nell'URL, per questo il metodo POST, oltre a poter gestire una quantità maggiore di dati in invio (per il client) inclusi nel body, è più sicuro perché i dati non sono esposti nellURL. Per ciò che concerne la memorizzazione nella cache e nella cronologia, le richieste GET possono essere memorizzate.

In generale, GET è più adatto per richieste semplici, mentre POST è più indicato per trasmettere dati sensibili e in quantità maggiori.

▼ *Come funziona https*

HTTPS è un protocollo di comunicazione che incapsula messaggi HTTP in una connessione TLS (Transport Layer Security). Innanzitutto si stabilisce una connessione sicura, iniziando un processo chiamato "TLS Handshake", dove client e server concordano vari parametri per stabilire una connessione sicura. Durante l'handshake, le due parti si scambiano le

chiavi pubbliche e creano una chiave simmetrica per la sessione. Per garantire autenticità, il server presenta un certificato digitale al client, successivamente inizia lo scambio di messaggi con dati criptati fino a fine sessione. Per ogni nuova sessione, l'handshake e la generazione della chiave simmetrica vengono ripetuti. In conclusione, grazie a HTTPS vengono garantite privacy, integrità dei dati e autenticità del server (quindi del sito visitato).

Il TLS conferisce confidenzialità (quindi protezione, i dati non vengono visti da esterni), integrità (i dati quindi non sono soggetti a modifiche) e autenticità (viene quindi verificata l'identità di chi manda).

▼ *Activation e passivation*

Sono termini comunemente usati in relazione agli Enterprise JavaBeans (EJB), componenti lato server utilizzati per implementare la logica di business delle applicazioni distribuite.

L'activation si verifica quando un client richiede un elemento EJB precedentemente passivato e conservato, generalmente in un database o in un filesystem, e viene caricato in memoria dal contenitore EJB per essere utilizzato. Questo comporta la lettura dello stato dell'EJB dallo storage persistente.

La passivation è il processo inverso. Per ottimizzare le risorse, infatti, lo stato dell'EJB viene serializzato e spostato in uno storage persistente, liberando così memoria dopo che è stato utilizzato, e rimarrà un questo stato passivo finché non servirà nuovamente.

Questi due meccanismi contribuiscono all'efficienza dell'applicazione, soprattutto in termini di memoria e risorse del server.

▼ *JSF, differenza con JSP*

Sono entrambe tecnologie server-side utilizzate per sviluppare interfacce utente nelle app web.

Per quanto riguarda architettura e design, JSF è un framework fortemente basato su componenti, gestione degli eventi e validazione dei dati. JSP è una tecnologia che consente di scrivere HTML misto a codice Java, più semplice e diretta ma meno potente per costruire interfacce.

Per la gestione di componenti, JSF utilizza componenti UI riutilizzabili e gestisce automaticamente lo stato attraverso il ciclo di vita della richiesta, è inoltre fortemente basato sul modello MVC dove JSF stesso funge da

controller, gestendo la navigazione tra le pagine e l'integrazione con il model. JSP non ha un modello a componenti e non fornisce un controllo MVC completo.

A livello di sviluppo, JSF offre un ambiente più standardizzato e più facile da mantenere, e si integra bene con le tecnologie Java Enterprise. JSP può essere più semplice per applicazioni piccole e semplici, ma è meno scalabile e per l'integrazione con altre tecnologie richiede un'onere più manuale.

In sintesi, JSF è più adatto per applicazioni complesse e di grande scala offrendo un framework più robusto basato su componenti, mentre JSP è più semplice ma adatto più a scenari che non richiedono una business logic troppo avanzata.

▼ *Informazioni contenute nei cookie (campi)*

I cookie, strutture dati che vengono salvate sul browser per migliorare le prestazioni e velocizzare l'esperienza utente, contengono i seguenti 7 campi:

- Key : nome identificativo unico del cookie all'interno di un namespace.
- Value : valore del cookie.
- Path : posizione del cookie all'interno del namespace.
- Domain : specifica il dominio al quale appartiene, al di fuori di questo dominio il cookie non può essere utilizzato.
- Max-age : tempo di vita del cookie (opzionale)
- Secure : se settato, il cookie viaggia solo con HTTPS (opzionale)
- Version : Identifica la versione del protocollo di gestione.

▼ *Ciclo di vita jsp*

Partendo dall'inizio del ciclo di vita, il web server (come Tomcat) carica la pagina e ne crea un'istanza, convertendola in una servlet java, se non è stato già fatto. Successivamente viene invocato un metodo di inizializzazione `init()`, che per esempio può creare i collegamenti con database o altri componenti per il corretto funzionamento. Successivamente, ad ogni richiesta viene invocato un metodo `service()` che gestisce il servizio, elabora la richiesta e fornisce una risposta, al termine, ossia quando lo decide il web server, viene invocato un metodo `destroy()`

per terminare, quindi dealloca le risorse, termina i thread ecc. Dopo che la servlet è stata de-istanziata, l'oggetto diventa idoneo per la garbage-collection della JVM.

▼ ***Differenza scope page e scope request***

Questi due concetti si riferiscono a due visibilità differenti dei dati all'interno di un'applicazione web.

Lo scope page si riferisce a tutti gli oggetti che sono accessibili solo all'interno di una pagina web, quindi ogni volta che viene ricaricata la pagina, gli oggetti sono creati e sono disponibili durante tutta la vita della pagina, al ricaricamento della pagina gli oggetti sono distrutti, o anche quando si cambia pagina e ci si ritorna. Un esempio sono le variabili javascript dichiarate in una pagina HTML, accessibili solo tramite script nella pagina stessa.

Lo scope request è una visibilità legata alla richiesta, quindi gli oggetti creati all'inizio della richiesta rimangono disponibili fino a fine richiesta, che di solito si conclude con l'invio di una risposta al client. Dopo il completamento della richiesta, non sono più disponibili. Un esempio sono gli attributi di richiesta in una servlet Java, accessibili durante il processamento della richiesta.

▼ ***Cosa vuol dire MVC e come si usa in spring, ci sono controller in spring? e come si fa un controller?***

MVC, che sta per Model View Controller, è un design pattern usato comunemente nelle applicazioni software. Spring lo supporta pienamente.

- Model rappresenta la struttura dei dati, contiene le entità e la logica di business. Gli oggetti del modello recuperano e memorizzano i loro stati nel database.
- View è la rappresentazione visiva dei dati, cioè l'interfaccia utente.
- Controller è l'intermediario tra il model e la view. Gestisce le richieste dell'utente, lavora con il modello per elaborare e ottenere i dati e invia tutto alla view.

In Spring è permessa la creazione di molti tipi di controller, alcuni controller sono già disponibili, tutti implementano l'interfaccia `Controller`:

- `AbstractController` è una classe base astratta per i controller, fornisce funzionalità di base come la gestione delle sessioni o la definizione

della durata massima della cache per le risposte. I controller che la estendono devono implementare il metodo `handleRequestInternal()`, chiamato per processare una richiesta HTTP.

- `ParameterizableViewController` è l'implementazione di un controller che restituisce una vista specifica. Può essere configurato con una vista, e ad ogni chiamata risponderà con quella vista, spesso usato in pagine statiche.
- `UrlFilenameViewController` mappa automaticamente i nomi degli URL ai nomi delle viste, estraendo il percorso dell'URL e lo usa come nome della vista.
- I `CommandViewController` consentono di mappare dinamicamente parametri di `HttpServletRequest` verso oggetti dati specifici.

▼ ***Punti fondamentali di innovazione di node.js***

Node.js ha introdotto numerose innovazioni nello sviluppo web:

- **Javascript lato server**: prima di Node.js, javascript era principalmente un linguaggio frontend. Node ha esteso l'uso di js anche nel backend, in maniera tale che gli sviluppatori possano utilizzare lo stesso linguaggio sia per il client sia per il server. Questo ha semplificato lo sviluppo web.
- **Modello I/O non bloccante e asincrono**: è un modello che ne costituisce uno dei punti di forza di Node. Questo significa che operazioni di lettura da un database possono essere eseguite in background, permettendo al server di gestire altre richieste nel frattempo.
- **Gestione efficiente delle connessioni**: Node è in grado di gestire un numero elevato di connessioni simultanee con basso sovraccarico, grazie al suo modello ad eventi.
- **Node Package Manager**: NPM è un potente gestore di pacchetti fornito con Node che semplifica la gestione e la condivisione di librerie e contiene un'enorme ecosistema di moduli, rendendo facile per gli sviluppatori trovare e utilizzare codice scritto da altri, e contribuire allo stesso tempo alla community.
- **Capacità multi-threading:** Anche se Node opera con un modello single-threaded, può gestire operazioni multi-threading, questo

permette di sfruttare i multi-core del server senza entrare nella complessità tradizionale del multi-threading.

▼ *Templating jsf*

Una caratteristica generale delle Java Server Faces è quello di facilitare il riutilizzo e l'estensione di codice. Tramite il templating si va a riutilizzare delle pagine "base" per altre pagine.

Tra i tag più frequenti possiamo trovare `ui:insert` di amplissimo utilizzo per inserire contenuto in un template, `ui:component` per definire un componente creato, `ui:include` per incapsulare e riutilizzare un contenuto.

▼ *Eventi lato server websocket*

Le WebSocket forniscono una comunicazione bidirezionale in tempo reale tra client e server. Il server può inviare messagi al client in modo proattivo, utilizzando eventi lato server.

- `onopen`: si verifica quando un client si connette al server e viene aperta la connessione.
- `onclose`: quando un client si disconnette.
- `onmessage`: si verifica quando il server riceve un messaggio dal client.
- `onerror`: quando si verifica un errore durante la comunicazione.

▼ *Polling, long polling, response forever, multiple connections*

Questi termini si riferiscono a diversi modi in cui un client e un server web possono comunicare e condividere informazioni.

- **Polling**: il client invia continuamente richieste al server per verificare se ci sono dati disponibili. Risulta semplice da implementare ma inefficiente in termini di risorse.
- **Long polling**: Simile al polling, solo che dopo una richiesta iniziale viene mantenuta aperta la connessione finché ci sono dati disponibili da mandare. Questo è più efficiente del polling normale ma mantiene connessioni persistenti che non risolvono del tutto l'overhead.
- **Streaming/Forever response**: Dopo una richiesta iniziale, il server mantiene la connessione aperta, appena i dati sono pronti il server risponde con risposte parziali (half duplex solo StoC, complicato con

proxy). Non è adatto a connessioni numerose perché è molto dispendioso.

- **Multiple connections**: Il client apre molteplici connessioni con un server, questo può aumentare la reattività però non è un vero modello real-time e incrementa notevolmente il carico.

▼ ***Differenza stato props e stato state react***

La differenza essenziale tra props e state è la loro origine e il modo in cui possono venir modificati.

Le props (*properties*) sono valori iniettati da un componente genitore, sono immutabili all'interno del componente figlio che le riceve e vengono utilizzate per trasmettere dati tra componenti.

Lo stato è un oggetto interno e mutabile che rappresenta lo stato corrente del componente. Può essere modificato solo dallo stesso componente tramite il metodo `setState()` e viene utilizzato per memorizzare dati dinamici che cambiano in base all'interazione utente.

▼ ***Differenza http 1.0, http 1.1 e http 1.1 con pipelining***

HTTP 1.0 è la prima versione originale del protocollo HTTP, rilasciata nel 1996, questa versione richiede una nuova connessione TCP per ogni richiesta e non supporta il pipelining, ovvero l'invio di più richieste all'interno della stessa connessione prima di ricevere la risposta alle richieste precedenti, è meno efficiente delle versioni successive a causa dell' overhead di handshake per ogni richiesta.

HTTP 1.1 viene rilasciato nel 1997, introduce funzionalità cruciali come le connessioni persistenti, che permettono di riutilizzare la stessa connessione per più richieste/risposte, il pipelining è implementabile manualmente (non c'è supporto nativo) e per questo può generare problemi di congestione anche se è comunque più efficiente di HTTP 1.0.

HTTP 1.1 con pipelining permette l'invio di più richieste nella stessa connessione TCP prima di ricevere la risposta alla prima, questo può migliorare le prestazioni riducendo l'overhead di handshake, ma non è supportata universalmente da tutti i client e server e deve essere attivata esplicitamente da entrambi per il corretto funzionamento.

▼ ***All'interno di una servlet posso vedere un javabean con scope session definito in una jsp? come posso accedervi?***

Sì, è possibile.

Tramite l'oggetto `HttpSession` ottenuto dalla servlet con il metodo `request.getSession()` e il JavaBean si recupera da questo oggetto con il metodo `getAttribute()` specificando il nome dell'attributo.

▼ *Operazioni sui file in node*

Node offre un modulo `fs` per gestire diverse operazioni sui file, come la lettura, scrittura, creazione directory, eliminazione, copiare un file, rinominarlo, modificarne i permessi.

▼ *Cookie in javascript*

Per l'impostazione di un cookie si utilizza il metodo `document.cookie` per accedere ai dati cookie, come parametri andremo ad inserire nome, valore e due opzionali che sono scadenza e percorso.

Per l'accesso, si usa lo stesso metodo che restituisce una stringa contenente tutti i cookie separati da punti e virgola. Tramite i metodi `split` e `indexOf` o regular expression possiamo estrarre il valore specifico di un cookie.

▼ *Direttiva page*

Le direttive page nelle JSP sono istruzioni che controllano come il contenitore JSP elabora e compila la pagina, influenzando il comportamento generale e la sua traduzione in servlet.

Alcuni esempi sono `page contentType` per specificare il tipo MIME del contenuto (*Multipurpose Internet Mail Extensions*), `pageEncoding` per definire la codifica di caratteri, `import` per importare classi o package Java, `session` per indicare se la pagina richiede una sessione attiva, `errorPage` per specificare una JSP da visualizzare in caso di errori.

▼ *Definizione di pagine web dinamiche e pagine web attive e differenze*

Le pagine web dinamiche sono generate al momento della richiesta, e possono variare in base ai dati utente o dati esterni. Vengono utilizzate diverse tecnologie lato server e possono essere più complesse da sviluppare e gestire rispetto alle pagine statiche.

Le pagine attive sono pagine dinamiche che utilizzano tecnologie lato client per rispondere alle azioni dell'utente in modo più immediato e interattivo, e possono quindi essere più adatte alle situazioni in cui l'utente necessita una riposta in tempo reale e un'esperienza più interattiva.

CGI

Common Gateway Interface è uno standard per la comunicazione tra un web server e programmi esterni. Permette ai server web di eseguire script e programmi per generare contenuti dinamici per le pagine web, anziché fornire solo pagine statiche pre-generate. Questo è molto utile nell'interazione dinamica con l'utente e nell'interattività.

EJB session bean stateful e stateless

I session bean, sia stateless che stateful, sono componenti server-side utilizzati per incapsulare la logica di business in applicazioni Java Enterprise.

Differenza tra java model 1 e java model 2

Sono due modelli di progettazione software per la creazione di applicazioni web Java.

Nel java Model 1 troviamo un'architettura più diretta e lineare adatta ad app web più piccole, e le JSP per gestire sia la presentazione che la logica di business, per applicazioni complesse risulta più difficile da gestire.

Nel Model 2 viene offerta un'architettura basata sul pattern MVC per una migliore separazione delle responsabilità tra presentazione (JSP) e business logic (Servlet). Questo fornisce maggiore leggibilità e scalabilità del codice.

Cookie: che cos'è e come si può usare in java lato server

Un cookie è un piccolo file di testo che un server web invia al browser. Il cookie viene memorizzato sulla macchina dell'utente e può essere utilizzato dal server per visite successive del client.

Lato server in Java troviamo la classe Cookie, dove il costruttore accetta due stringhe: nome e valore, successivamente tramite setter è possibile impostare gli altri campi come il max-age o il path.

Possono essere molto utili per memorizzare le preferenze utente, tenere traccia di un'autenticazione o tracciare le attività di un utente.

HTTP, altre richieste oltre get e post (put, delete, option, head, trace)

- **PUT**: ha lo scopo di aggiornare o creare una risorsa sul server, che gli viene inviata, usata tipicamente per ricevere file o aggiornare record in database.

- **DELETE**: offre la funzionalità di eliminare una risorsa dal server, questa risorsa viene specificata dall'URI della richiesta.
- **OPTION:** ha lo scopo di ottenere informazioni sulle opzioni di comunicazione supportate dal server per una specifica risorsa, restituisce informazioni sui permessi e altre opzioni supportate dal server.
- **HEAD:** grazie ad esso possiamo recuperare l'header di una risorsa senza restituire il corpo della risposta. Si può usare per esempio per verificare l'esistenza di una risorsa.
- **TRACE**: ha lo scopo di diagnosticare errori o tracciare il percorso di una richiesta attraverso server intermediari. Può essere usata nel debug di problemi di comunicazione, analisi di sicurezza e sviluppo.

▼ ***Aggiornamento proxy cache (Freshness, Validation, Invalidation)***

Sono tre metodi per gestire l'aggiornamento della cache.

- **Freschezza**: definisce quanto tempo un oggetto in cache può essere considerato valido prima di richiedere nuovamente al server una conferma della validità.
- **Convalida**: consiste nel contattare un server prima di fornire un oggetto dalla cache, per verificare se è ancora valido, questo evita di fornire contenuti obsoleti.
- **Invalidazione**: rimuove un oggetto dalla cache prima che scada la sua validità, può essere attivata dal server o da eventi esterni.

▼ ***Differenza include/forward***

La include inserisce il contenuto di una pagina in un'altra, senza modificare l'URL della pagina includente. In HTML, viene utilizzato il tag `<include>` per includere un file esterno.

La forward invia la richiesta a un'altra pagina web e reindirizza il browser a questa pagina, la pagina di destinazione è quindi vista come una pagina nuova, con un altro URL. In HTML viene usato l'attributo `action` del tag `<form>` per reindirizzare, oppure lato servlet con l'oggetto `RequestDispatcher` c'è il metodo `forward`.

▼ ***Perché le websocket non sono un protocollo ma un upgrade***

Le websocket sono un upgrade del protocollo HTTP. Utilizzano infatti la stessa connessione TCP di un'istanza esistente HTTP, in cui l'handshake iniziale avviene con messaggi HTTP.

Viene offerta però una comunicazione bidirezionale, molto adatto a scenari come chat, giochi online o aggiornamenti automatici, e la possibilità del server di fornire messaggi in maniera proattiva risulta una delle funzionalità chiave. Sono più efficienti, l'overhead è ridotto rispetto a HTTP, perché non serve creare una nuova connessione a ogni richiesta.

Le websocket sono supportate dalla maggior parte dei browser moderni.

▼ *Oggetti built-in jsp*

Gli oggetti built-in di JSP sono oggetti predefiniti disponibili in ogni pagina JSP, e sono accessibili senza bisogno di crearne istanze o dichiarazioni esplicite. Forniscono informazioni sul contesto specifico della pagina. Alcuni esempi:

- `page` rappresenta la servlet associata alla pagina JSP, fornisce l'accesso all'oggetto `ServletContext`
- `pageContext` fornisce un contesto per contenere informazioni sulla pagina, sulla sessione o sulla richiesta.

Abbiamo poi oggetti relativi alla richiesta:

- `request` rappresenta l'oggetto `HttpServletRequest` della servlet che contiene informazioni sulla richiesta.
- `requestScope` per l'accesso ad attributi di richiesta.

Oggetti relativi alla sessione:

- `session` rappresenta l'oggetto `HttpSession` della servlet.
- `sessionScope` per accedere agli attributi di sessione.

Analogamente, abbiamo oggetti relativi all'applicazione. Questi oggetti built-in migliorano la leggibilità e la manutenibilità del codice JSP e riducono la necessità di scrivere codice boilerplate per le informazioni comuni.

▼ *Server proxy e gateway*

Entrambi sono intermediari tra client e server.

Un proxy serve da intermediario per richieste e risposte e offre funzionalità come caching, autenticazione per controllare l'accesso a risorse, filtraggio

per bloccare l'accesso a siti dannosi o indesiderati, e traduzione per convertire protocolli di comunicazione.

Un gateway serve a connettere reti diverse e gestirne il traffico, fornisce funzionalità come il routing che determina il percorso migliore per l'instradamento del traffico, un firewall per filtrare il traffico in entrata e in uscita per maggior sicurezza, la traduzione degli indirizzi di rete (NAT) per la condivisione di un singolo indirizzo IP pubblico, o anche VPN per creare una connessione sicura tra reti geograficamente disperse.

Troviamo inoltre il concetto di tunnel: è una connessione virtuale criptata che collega due punti.

▼ ***Tipi di web cache (user agent e proxy)***

Lo user agent, che tipicamente è il browser (quindi si parla di una tecnologia client-side), mantiene una cache delle pagine web visitate dall'utente, è tuttora molto usato per i dispositivi mobili, molto utile nei casi in cui la connettività sia intermittente o per ridurre la latenza nel caricamento di elementi statici.

Come proxy cache abbiamo le *Forward Proxy Cache*, che mirano a ridurre la banda utilizzata, il traffico viene intercettato e le pagine sono messe in cache in modo tale che il server non debba continuamente scaricare le stesse risorse. Abbiamo inoltre le *Reverse Proxy Cache*, il quale scopo è di ridurre il carico delle macchine, rappresenta la base del funzionamento delle Content Delivery Network (come MS Azure o Amazon CloudFront), le CDN sono sistemi di server che si occupano di velocizzare la distribuzione di contenuti web, ridurre la latenza e fornire resistenza a guasti o picchi di traffico.

▼ ***Javascript engine. oggetti o classi? è a eventi? DOM e accesso cookie/schermo.***

L'engine di JavaScript serve a convertire correttamente il codice JS in codice macchina ed eseguirlo. Innanzitutto si parte dal file .js che viene passato al Parser, il quale analizza il codice e lo converte in un albero di sintassi astratta (AST), l'Abstract Syntax Tree non è altro che una struttura gerarchica del codice che facilita l'analisi. Successivamente l'AST viene letto dall'interprete, e può accedere o modificare la memoria, chiamare funzioni o valutare espressioni, opzionalmente poi troviamo il Profiler che monitora le prestazioni. Abbiamo poi il Compiler che converte l'AST in codice macchina ottimizzato da eseguire. La Heap memory memorizza i

dati del codice inclusi variabili e oggetti, mentre lo Stack tiene traccia delle chiamate di funzioni in esecuzione con anche i valori dei parametri e le variabili locali.

Javascript è un linguaggio a oggetti basato su prototipi per l'ereditarietà. Non ci sono classi come in Java o C++, ogni oggetto ha una semplice sintassi di creazione, e ha un prototipo dal quale eredita proprietà e metodi. JS è inoltre orientato agli eventi, cioè azioni che possono essere intercettate e gestite dal codice JavaScript (eventi come click su un pulsante o caricamento di una pagina), che lo rende molto versatile, potente e utile per l'interattività.

Tramite l'oggetto document e metodi come `getElementById()` possiamo accedere agli elementi del DOM per l'identificativo oppure con `getElementsByClassName` che restituisce tutti gli elementi di una determinata classe (attributo `class`). Troviamo anche il metodo `querySelector` per selezionare elementi dal CSS. Per navigare nel DOM abbiamo attributi e metodi come `parentNode`, `childNodes`, `nextSibling` o `previousSibling`.

Per accedere ai cookie c'è l'oggetto `document` con attributo `cookie`, analogamente per le proprietà dello schermo abbiamo `document.window`, grazie al quale poi possiamo accedere a informazioni come altezza e larghezza.

▼ *JQuery*

JQuery è una libreria open-source di javascript che offre un'ampia selezione di funzioni per accesso a elementi del DOM e la loro manipolazione, le animazioni, AJAX e molto altro ancora, in generale semplifica il JavaScript puro anche se le differenze sono limitate.

▼ *HTML: head tag, iframe, importazione css e js, àncore, ipertesti*

Il linguaggio HTML (HyperText Markup Language) è un linguaggio di markup utilizzato per creare la struttura di una web page, si fonda su una serie di tag per la definizione del contenuto e la formattazione della pagina.

Nella parte di `<head>` andiamo a definire tutte le informazioni non direttamente visualizzate come la codifica caratteri o i fogli di stile CSS, tra i più importanti troviamo `<title>` (titolo della pagina), `<meta>` che fornisce informazioni aggiuntive sulla pagina come l'autore o la codifica.

Un iframe (*Inline Frame*) è un elemento HTML che consente di incorporare una pagina all'interno di un'altra, utile per visualizzare contenuti esterni

come mappe o video.

Per importare CSS, nell'header andiamo a utilizzare il tag `<link>` con l'attributo `rel =" stylesheet "`. Per importare file JS esterni c'è il tag `<script>` con l'attributo `src` dove andiamo a scrivere il percorso del file.

Per le ancore abbiamo il tag `<a>` , è un elemento HTML che fornisce un collegamento ad un'altra parte della pagina stessa, oppure ad un'altra pagina, l'attributo `href` specifica la destinazione.

Un ipertesto è un sistema di collegamenti che permette la navigazione tra diverse pagine web. Grazie alle ancore realizziamo i collegamenti ipertestuali.

▼ *Come funziona un sistema di server web*

Il Web segue un modello Client-Server, dove il Client è un elemento attivo che con il protocollo HTTP si connette al server, richiedendo pagine web (risorse identificate con URL) ai server e visualizzandone il contenuto. Il Server rimane in attesa, in ascolto di eventuali connessioni, riceve le richieste HTTP e fornisce le risorse ai Client.

Un problema iniziale da risolvere consiste nel superare i limiti del web statico. La prima soluzione per rendere più dinamico e interattivo il web prende il nome di *Common Gateway Interface*, già presente da HTTP-1.0, che permette al web server di interfacciarsi con programmi esterni, eseguiti dinamicamente come risposta alla chiamata, che producono un output dinamico come response HTTP.

Nel suo percorso di evoluzione il web dinamico trova successivamente l'introduzione del concetto di *application server*: un contenitore nel quale stanno le funzioni server-side, che si interfaccia con il web server e il database. Risulta una soluzione più modulare. Esso supporta automaticamente molte funzionalità frequenti e comuni come il ciclo di vita, il sistema di nomi o la sicurezza.

Riassumendo: il web server riceve la richiesta, per alcune operazioni la delega all'application server che si occupa di molte funzionalità più specifiche dell'applicazione web, il quale è collegato al database per tutti i dati necessari.

▼ *JavaScript: dispositivi diversi, selettori e gerarchia CSS*

Javascript è un linguaggio di scripting che trova le sue origini nel 1995, sviluppato inizialmente per dare interattività client-side nella pagine HTML,

evoluto poi anche in utilizzo server-side con Node.js.

JavaScript si utilizza spesso per adattare e riutilizzare lo stesso DOM in dispositivi diversi come tablet o televisioni, posso accedere alle dimensioni dello schermo con `document.window.width/height`.

Per selezionare gli elementi del DOM (al fine di manipolarli e modificarne contenuto o comportamento) ci sono diversi tipi di selettori: per identificativo, per nome di classe, selettori CSS (es. “selezionami tutti i paragrafi”, quindi elementi con tag `<p>`), selettori di discendenza o di fratellanza.

La gerarchia CSS determina un ordine assiologico nell’applicazione di stili CSS ad elementi HTML in caso di conflitti. Solitamente, la priorità sale grazie a quanto è specifica una regola, quanto è recente e se è definita nell’head del documento (rispetto al body). Per forzare una regola c’è anche la keyword `!important`.

💡 A me all’orale ha chiesto JSON, Message Driven Bean (PB); poi differenze tra Java e JS, differenze tra JSON e XML, caching in generale (GDM).