protocollo di livello applicativo → basato su TCP

HTTP → REQUEST-RESPONSE → trasferisce risorse web da server a client → richieste (URL) risposte (pagine)

- STATELESS → ONE-SHOT
  - però ci sono i COOKIE → per dare stato ad http
    - collezioni di stringhe che si muovono come un token tra client e server
    - server riconosce PC dell'utente
    - AUTENTICAZIONE
      1. filtro su indirizzi IP
      2. form x username o password
      3. HTTP Basic (usn e pwd in base64)
      4. HTTP digest → deprecato → funzione hash
    - SICUREZZA
      - HTTPS → http + TLS → Transport Layer Security → connessione criptata → crittografia asimmetrica
        - o SSL → Secure Sockets Layer
        - confidenzialità
        - integrità
        - autenticità
      - PROXY → applicativo che agisce sia come client che server per fare richieste per conto di altri client → serve per → filtrare le richieste / fare caching → scarica in locale le risorse web x successive richieste → riduce uso banda e carico sul server
      - GATEWAY → intermediario x altri server, instrada pacchetti a diff. del proxy, riceve come se fosse server originale e client non distingue
      - TUNNEL → applicativo che agisce come "blind relay" tra 2 connessioni
- messaggio http → body / header
  - REQUEST
    - request line
      - tipo di comando
      - risorsa richiesta
      - versione di HTTP
    - GET → richiede risorsa al server → parametri → lunghezza limitata
    - POST → la risorsa richiesta non sta nell'URL ma nel body (non come GET) → no limiti lunghezza / no creazione risorsa
    - PUT → memorizza su server una risorsa all'URL specificato → si crea risorsa
    - DELETE → elimina risorsa all'URL specificato
    - HEAD → come GET ma riceve solo header
    - OPTIONS → richiede informazioni sulle opzioni disponibili
    - TRACE → loop-back della richiesta → diagnostica e test
  - RESPONSE
    - status line
    - header
    - body
    - status code (4xx-5xx)
- HTTP 1.0 → non persistente → connessione apposita per ogni risorsa
- HTTP 1.1 → persistente → server lascia aperta la connessione x ricevere altre richieste → chiude quando lo dice l'header o per time-out
- HTTP 1.1 pipelining → client invia molteplici richieste prima di ricevere risposte → risposte arrivano nell'ordine delle richieste

CACHE

- HTTP CACHE → http non ha cache interna → 3 meccanismi
  - freshness
    - server → "Expires" in response header
    - client → direttiva Cache Control: max-age
  - validation
  - invalidation
- Proxy Cache → proxy server
  - forward p.c. → riduce banda, pagine in cache
  - reverse p.c. → Gateway cache → opera x conto del server
- User Agent Cache → browser → pagine visitate dall'utente

HTML → HyperText Markup Language → linguaggio di codifica del testo a marcatori (markup) usato x descrivere le pagine (nodi dell'ipertesto)

- composto da → tag → insieme di istruzioni → caratteristiche del documento
  - semantica
  - grammatica

identificate con session ID ← condivisa fra tutte le richieste dello stesso client

↑

dove c'è personalizzazione delle richieste c'è stato (es. autenticazione) ↑ entità gestita dal web container

↑

WEB SERVER → STATO → lato client → cookie → basso livello
STATO → lato server → sessioni → conversazione di pagine visitate dall'utente → alto livello

↓ ↑

APPLICATION SERVER → CONTAINER → entità supervisore a cui vengono delegate funzioni → componenti trasportabili riutilizzabili e mobili → SERVLET

↓ ↑

DATABASE SERVER

CONTAINER → trasformate in servlet dai → HTML + Java

HTML + Java → JSP → delega la risorsa → forward

JSP → JAVA BEAN → statico o dinamico → include

include, forward → tramite RequestDispatcher

JAVA BEAN: modello "base" per componenti Java, il più semplice → classe public, costruttore senza argomenti, accessors → può avere scope page (default) → fino a quando la pagina viene completata o fino al forward

SERVLET: classe Java che fornisce risposte a richieste HTTP → o + in generale → fornisce un servizio comunicando con il client mediante protocolli request/response → eseguono direttamente nel web container

servlet container controlla e supporta il ciclo di vita → chiama doGet o doPost → chiamato ad ogni http request → init() service() destroy() → metodi

lifespan e accessibilità della servlet definiscono → SCOPE (AMBITO) → request, session, application → di oggetti quali ServletContext, HttpSession, Request ..

MULTI-THREAD → CONCORRENZA → più thread condividono la stessa istanza di una servlet

SINGLE-THREADED (DEPRECATO) → Container crea istanza servlet per ogni richiesta concorrente

le applicazioni web tradizionali usano un modello di interazione rigido → click-wait-refresh → sincrono → proposta di un modello asincrono

AJAX → Asynchronous Javascript And Xml → basato sull'oggetto javascript XmlHttpRequest → modalità alternativa per gestire gli eventi remoti → si guadagna in espressività ma si perde linearità dell'interazione

AJAX → elimina i caricamenti di pagina (SPA) → Single Page Application → a cui si ispira anche

REACT.JS → libreria javascript per la creazione di interfacce utente web → sviluppo front-end → esegue all'interno del browser → manipola un virtual DOM → copia esatta → NON interagisce col back-end

REACT.JS → JSX → Javascript XML → tag HTML dentro codice js → Babel pre-compila

REACT.JS → element → semplice immutabile

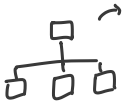
REACT.JS → component → complessi e dinamici → mattoncini → indipendenti riusabili

component → class, function → stateless

component → state → proprietà che variano nel tempo

component → props → proprietà immutabili → configurazione

state → gerarchia di componenti → mantengono stato → passano props ai figli

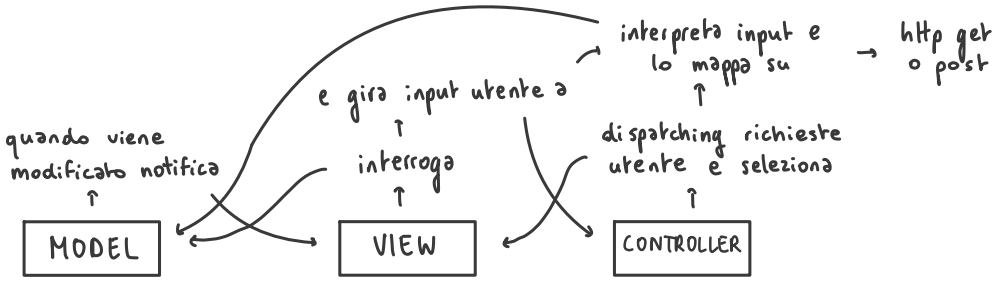


MVC

JAVA MODEL 2

JAVA MODEL 1

solo servlet e JSP



EJB

Framework Java per lo sviluppo di app aziendali

Enterprise Java Bean ← eseguiti in un container EJB ← principali componenti

session bean

utilizzato per eseguire attività di business a livello di sessione

NO PERSISTENTI

VITA BREVE (failure EJB)

- 1 istanza x cliente
- short-lived (vita bean = vita client)
- NO CONCORRENTI

2 tipi di session bean

STATELESS

ogni invocazione di metodo è indipendente dalle precedenti → è stateless solo x l'annotazione!

resource pooling → pooling dei componenti server-side da parte del container EJB → x evitare di mantenere un istanza separata EJB x ogni cliente

ciclo di vita

1. no state → non istanziato
2. Pooled state → ancora non associato a richiesta cliente
3. Ready state → pronto a rispondere a una invocazione di metodo

si applica anche a message-driven bean

ogni EJB container contiene molti pool, ciascuno composto da istanze con la stessa destination JMS

STATEFUL → gestione della coppia oggetto EJB + istanza bean

trasparente x il client

→ ACTIVATION → dissociazione, con salvataggio dell'istanza su memoria (serializzazione)

→ PASSIVATION → recupero da memoria (deserializzazione) e riassociazione con oggetto EJB

NO CONCORRENZA

ogni istanza associata a un singolo cliente

entity bean

rappresenta un oggetto di dominio

tipo cliente o ordine

PERSISTENTI

LONG-LIVED

ACCESSO CONDIVISO

- istanza condivisa da + clienti
- long-lived
- persistente
- transazionale

message-driven bean

gestisce messaggi asincroni inviati a una coda JMS (Java Message Service)

stateless

SPRING → piattaforma di sviluppo applicazioni JAVA → CONTAINER LEGGERO → core a cui si agganciano solo i moduli che servono

DEPENDENCY INJECTION → flessibilità, testing, manutentibilità

i componenti non devono cercare a runtime le risorse necessarie → componenti sono Spring Beans

dichiarano le risorse di cui hanno bisogno e il container gliele passa → INVERSION OF CONTROL

sistema DISACCOPPIATO e MANUTENIBILE

Factory per componenti → XML BeanFactory

globale, ritrova oggetti per nome e gestisce relazioni tra di essi

spring inietta tramite costruttore o con metodi set/get

CORE PACKAGE

carica le risorse dichiarate dai componenti e gliele passa

MVC PACKAGE

spring offre supporto a componenti "controller"

DISPATCHER SERVLET ← si avvale di servlet che fa da dispatcher delle richieste

vista come "Front controller"

1. intercetta richieste HTTP che arrivano al web container
2. cerca un controller x gestire la richiesta e lo invoca
3. riceve model
4. cerca ViewResolver per scegliere una view a cui fare forwarding del model
5. view crea Http response

JSF → FACELETS → linguaggio x costruire pagine JSF → albero dei componenti → popolato da managed beans

↳ TEMPLATING → creare template → modello di layout → può includere elementi fissi e variabili

WEB SOCKET → protocollo di messaggistica → ASINCRONA, FULL-DUPLEX → su una singola connessione TCP

Javascript lato client e JEE lato server

CONTENUTI INTERATTIVI IN TEMPO REALE

UPGRADE request tramite handshake su HTTP

→ POLLING → client fa polling a intervalli prefissati e server risponde subito → ritardi, anche se non ci sono messaggi server viene bombardato di richieste

→ LONG POLLING → client manda richiesta iniziale e server attende finché non ha dati da inviare → poi nuova richiesta → CONTRO: ogni request/response si appoggia su una nuova connessione

→ STREAMING/FOREVER RESPONSE → client manda solo la richiesta iniziale → poi connessione sempre aperta per aggiornamenti push → client manda una prima richiesta GET → server poi manda dati quando gli va

HALF-DUPLEX → solo server to client

→ MULTIPLE CONNECTIONS → long polling su due connessioni HTTP separate

una per il long polling tradizionale
una per dati client verso server

→ CONTRO: overhead di 2 connessioni per ogni client

NODE.JS → Javascript sia lato client che server → NO DOM lato server

↳ ASINCRONO → ogni operazione delegata all'OS → poi callback → wrapping di tutte le chiamate bloccanti di so

↓ ↘

MODULI EVENT LOOP → singolo thread → gestione eventi su coda degli eventi → ogni operazione esegue come chiamata dall'event loop

↳ fs per gestire file

↓

supporta stream, networking ecc.