

Commenti sulla soluzione e correzione del compito

Il codice deve girare! Se il codice non esegue, non è codice, e il compito è considerato non valutabile. Si valuta quel che avete fatto, non ciò che avreste avuto intenzione di fare. Non a caso le consegne sono sempre organizzate per punti progressivi, in modo che possiate procedere passo passo. Solo quando un punto è raggiunto e il codice relativo esegue si fa il passo successivo. La cosa da fare è quindi: implementare una funzionalità, che giri correttamente, salvare copia del progetto. Ripetere poi per le altre funzionalità. Lavorando in questo modo avreste SEMPRE qualcosa di sensato (anche se incompleto) da consegnare.

Nel dettaglio delle valutazioni, alcune righe sono tutti 0: se non c'è quasi nulla, o se l'approccio è completamente sbagliato (dato on session, vedi sotto) non procedo ad indicare il dettaglio.

Veniamo alla questione fondamentale: come deve essere strutturata l'applicazione? Per deciderlo dobbiamo porci alcune domande:

- Quali dati sono generali e devono essere presenti in tutta l'applicazione?
- Quali invece sono legati allo stato del singolo utente?

1) I dati trasversali a tutta l'applicazione

Sono i dati per l'autenticazione, le stanze presenti, i messaggi in ciascuna stanza. Questi devono essere disponibili in tutta l'applicazione, quindi devono stare nell'area comune raggiungibile tramite il ServletContext, alias "application" nelle JSP. In alternativa possono essere immagazzinati in una risorsa sempre raggiungibile (il DB): quest'ultimo è preferibile se abbiamo bisogno di persistenza (ovvero se i dati devono sopravvivere al restart del server).

Volendo implementare una soluzione "in memoria" (ovvero nel servletContext) che poi possa evolvere in una soluzione persistente la cosa migliore è usare un approccio DAO: nella forma più semplice possiamo avere una funzione dove incapsuliamo il controllo di username e password: in una prima versione questa lavorerà con il servletContext, poi potremo sostituirla con una che operi con il DB. Entrambe avranno bisogno di una inizializzazione (nel primo caso per creare e dati, nel secondo per stabilire una connessione con il DB) che potrà essere implementata nella init di una servlet.

(Avevo raccomandato di procedere alla implementazione in memoria e solo dopo passare a quella basata su DB).

Lo scopo dell'app è di far condividere i dati tra più utenti: dunque la lista delle stanze, e per ogni stanza la lista dei messaggi devono essere nell'area condivisa. Metterli in session è sbagliatissimo, e comporta l'immediata insufficienza del compito.

2) I dati specifici dello stato dell'utente

Nel nostro caso, e solo il fatto che l'utente sia già autenticato o meno. Potremmo usare un cookie, ma è assai più semplice e comodo lavorare con una session, e tenere in sessione questa informazione (per esempio, se l'utente è autenticato, vi sarà in sessione lo username, se non lo è tale dato non sarà presente). Il logout potrà allora essere ottenuto con un semplice invalidate sulla sessione.

3) Verifica autenticazione e controllo credenziali

Il controllo delle credenziali non si fa confrontandole una ad una con tutti i record disponibili! Per l'approccio in memoria, meglio avere una mappa con tutti gli utenti, e controllare se la mappa contiene il record che descrive l'utente (ci sono metodi nelle API per questo: la ricerca sequenziale è di ordine N, quella implementata nelle API è log N). Nel caso del database, si fa una query specifica per vedere se il record cercato è presente. NON si scarica l'intera tabella del DB per poi iterare su di essa (cosa che purtroppo ho visto fare in più casi...).

Per controllare che un utente abbia i permessi di accedere alle informazioni, ci sono apposta i Filter. Non si mette il controllo in ogni servlet! Lo si centralizza nel Filter. E' poi fondamentale che il filtro protegga tutti i punti di accesso, e quindi va configurato correttamente! (o tramite annotation, o con la WEB-INF).

4) Lo header

La parte comune della pagina (tipicamente lo header, ma potrebbe essere anche il footer) va scritta una volta sola e inclusa tutte le volte che è necessario: così anche eventuali variazioni future sarà necessario farle una sola volta.

5) MVC

Era richiesta una soluzione MVC: quindi soluzioni basate solo su JSP o solo su servlet non vanno bene. Ovviamente era fondamentale definire dei business object: almeno stanza e messaggio (ma poteva esser e comodo anche avere Lista di Stanze e Liste di Messaggi).

Mettere nelle JSP del codice statico HTML tramite delle `out.println()` è considerato sbagliato. Le `out.println()` vanno usate solo per stampare valori di variabili (dati dinamici), e comunque ad essa va preferita la sintassi `<%= %>` (salvo i casi in cui la scrittura basata sulla `out.println` risulti più leggera, ad esempio perché siamo dentro un loop con poco output).

6) Ricaricamento della pagina

Il ricaricamento ogni 10 sec. poteva essere fatto facilmente con Javascript.

Attenzione però: se si fa un reload della pagina corrente vengono passati anche i dati, quindi se non si fanno le cose per bene il testo di un messaggio può essere ricaricato a ogni giro! Se invece si carica una URL, bisogna garantirsi che se vi sono dati attesi nella request o nel context, questi siano presenti, altrimenti si va incontro a null pointer exceptions!

7) Come testare il tutto?

Usando due browsers diversi (in modo che ciascuno abbia la propria sessione) accediamo alla chat e proviamo. Se abbiamo fatto l'errore di mettere le stanze in sessione, ci accorgeremo subito che la cosa non funziona!

Potremo anche accedere alla webapp con delle url intermedie (ad esempio quella che mostra le stanze, o quella che mostra i messaggi. Se lo facciamo da un browser in cui non ci siamo ancora autenticato, o sul quale abbiamo fatto logout, e abbiamo fatto le cose per bene, dobbiamo essere rediretti alla pagina di login

8) Note varie

Un modo per far sì che la web app si avvii con la pagina di catalogo è usare la `<welcome-file-list>` del `web.xml`.

Come sempre, piccole penalizzazioni di punteggio sono state date a chi non ha usato nomi di package canonici e a chi non ha seguito le regole per il nome del progetto date nelle istruzioni generali. Attenzione poi: se nel test c'è scritto che il DB si deve chiamare "MyDerbyDB", perché in vari lo chiamano "ExamDerbyDB"? Copia dagli appunti, evidentemente. Lecito, ma gli appunti non vanno copiati pari pari, devono essere ricontestualizzati! (Per inciso, non avete idea dell'overhead di tempo che si ha nella correzione se gli standard dati non sono rispettati!).