Tecnologie Web T
28 Giugno 2016 – Compito

Tempo a disposizione: 3 ore

La soluzione comprende la **consegna elettronica** dei seguenti file mediante l'apposito applicativo Web **esamix** (**http://esamix.labx**):

CercaLibri.zip	file zip contenente il sorgente java/class e pagine Web per punto 1
Hotel.zip	file zip contenente il sorgente java/class e file xml/xsd per punto 2
Chef.zip	file zip contenente il sorgente java/class per punto 3

Ogni file .zip consegnato DEVE CONTENERE TUTTI e SOLI i file creati/modificati e/o ritenuti importanti in generale ai fini della valutazione (ad esempio, descrittori, risorse statiche o dinamiche, codice Java e relativi .class, ecc.) e NON dell'intero progetto

N.B. Per superare la prova scritta di laboratorio ed essere ammessi all'orale, è necessario totalizzare almeno 18 punti (su un totale disponibile di 33), equamente distribuiti sui tre esercizi, ovvero almeno 6 punti sul primo esercizio, 6 punti sul secondo esercizio e 6 punti sul terzo esercizio

---

Studenti in debito di Tecnologie Web L-A
Viene richiesto lo svolgimento dei soli esercizi 1 (17 punti) e 2 (16 punti). Tempo a disposizione: 2 ore.
I 18 punti necessari per l'ammissione all'orale sono così distribuiti: almeno 10 punti sul primo esercizio e almeno 8 punti sul secondo

---

ESERCIZIO 1 (11 punti)
Si realizzi un'applicazione Web per la ricerca di libri di interesse in un catalogo bibliografico mantenuto in memoria lato server.
L'applicazione deve consentire all'amministratore (autenticato tramite username=admin; password=admin) di inserire libri nel catalogo tramite un semplice form che preveda come campi di input <autore, titolo, editore, isbn>; tali dati devono essere passati alla servlet **salvaNuovoLibro** in formato JSON; il catalogo deve essere mantenuto in memoria centrale, senza requisiti di fault-tolerance.
Una volta popolato il catalogo, ogni utente deve avere la possibilità di fare ricerche per Autore nel catalogo (pagina **cerca.jsp**). Ogni utente può eseguire al massimo 3 ricerche concorrenti AJAX, inserendo le relative stringhe *nomeAutore_i* in un apposito form. Una servlet **dispatcher** deve occuparsi di ricevere la richiesta, verificare la correttezza del formato di *nomeAutore_i* (NO caratteri numerici); dipendentemente dal numero di elementi totali nel catalogo, dovrà poi **delegare il compito di ricerca a [1, n] slave**, ciascuno dei quali lavorerà su 10 elementi del catalogo. Al termine della ricerca, l'applicazione Web (pagina **cerca.jsp**) deve essere in grado di mostrare all'utente la lista dei libri trovati per gli autori specificati e i risultati analoghi ottenuti in precedenza per al massimo 3 richieste precedenti di quello stesso utente.
Inoltre, si giustifichi nel sorgente la scelta di implementare gli slave come oggetti Java, Javabeans o altre servlet. Problemi in caso di accesso concorrente alla stessa struttura dati catalogo? Vantaggi/svantaggi nell'uso del modello di concorrenza deprecated per servlet?

ESERCIZIO 2 (11 punti)

Si progetti una grammatica **XML Schema**, e un suo **documento XML** di esempio, per la modellazione delle informazioni di un **sito Web per la navigazione di alberghi** nel rispetto delle specifiche dettagliate di seguito.

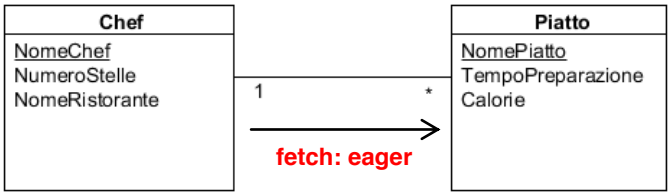
Ciascun documento XML modella una navigazione di un utente registrato al sito Web. Nel dettaglio, si tiene traccia di (i) informazioni utente quali email (di tipo "email" e obbligatorio) e password (di tipo "password" e obbligatorio) (ii) attributi per specificare una ricerca da parte dell'utente; in particolare, "nazione", "città", "numero stelle" e "fascia prezzo" (tutti obbligatori, tranne fascia prezzo che è opzionale). Nel dettaglio, "nazione" e "città" sono stringhe, "numero stelle" è un intero compreso tra 1 e 5, mentre "fascia prezzo" rappresenta il costo/notte dell'albergo in Euro e può appartenere a 4 possibili fasce: bassa (inferiore a €50), media (tra €50 e €120), alta (da €121 a € 180) e lusso superiore a €180). Il tipo "password" è una stringa di 8 caratteri che deve contenere almeno un numero e un carattere speciale tra {?!*$}; il tipo "email" è una stringa formata da due sottostringhe separate dal carattere speciale "@" e in cui la seconda sottostringa contiene un "."

Il risultato della ricerca è modellato a sua volta come un insieme di "oggetti" complessi (cardinalità dell'insieme da 0 a N), ognuno formato da "nome hotel" (tipo stringa), "fotografia hotel" (rappresentata dal nome del file jpeg corrispondente), "giudizio utenti", tra {eccellente, buono, medio, sufficiente, mediocre}, "prezzo" (espresso in Euro) e "link a sito Web hotel dedicato" (tipo indirizzo Web). Tutti i campi sono obbligatori. Durante la navigazione del risultato, l'utente può visitare il link del sito Web dedicato dell'albergo di interesse.

Si realizzi l'**applicazione Java "IndagineMercatoHotel"** che, facendo uso del **parser DOM**, esponga il metodo `getSelezione()`, unitamente a suo un `main` di prova, in grado di restituire la lista dei nomi degli alberghi del risultato della ricerca per i quali l'utente ha fatto "click" sul link al sito Web dedicato. Si stampi la lista sul file **Hotel.txt**.

ESERCIZIO 3 (11 punti)

Partendo dalla realtà illustrata nel **diagramma UML** di seguito riportato, si fornisca una soluzione alla gestione della persistenza basata su **Pattern DAO** in grado di "mappare" efficientemente e con uso di ID surrogati il modello di dominio rappresentato dai **JavaBean Chef e Piatto del diagramma UML** con le corrispondenti **tabelle relazionali derivate dalla progettazione logica del diagramma** stesso.



Nel dettaglio, dopo aver creato da applicazione Java gli **schemi delle tabelle** all'interno del proprio schema nel database **TW_STUD** di **DB2** (esplicitando tutti i **vincoli** opportuni), **implementato** i **JavaBean** e **realizzato le classi** relative al **Pattern DAO** per **l'accesso CRUD** alle tabelle**,** si richiede l'implementazione di un **opportuno metodo per il supporto della seguente operazione**: "per ogni chef pluristellato (ovvero con 2 stelle o più), il nome del piatto proposto dallo stesso che richiede il minore tempo di preparazione".

Si crei poi un **main di prova** che: (i) inserisca due o piu' tuple nelle tabelle di interesse; (ii) faccia uso corretto del metodo realizzato al punto precedente al fine di produrre una stampa del risultato sul file **Chef.txt**.

N.B. L'implementazione del **Pattern DAO deve limitarsi** al solo **DBMS DB2**. La soluzione **deve sfruttare il mapping 1-N specificato nell'UML e propendere per il caricamento indicato nello stesso**. Ogni ulteriore scelta da parte dello studente deve essere opportunamente giustificata con commenti nel codice.