LORENZO CURRELI 70/89/00431

RELAZIONE PROGETTO FINALE FPW

A) Funzionalità generali

L'applicazione web realizzata si caratterizza per la presenza di una homepage nella quale si possono leggere alcune info sul sito e una barra di navigazione che permette, tra le varie funzionalità, di accedere alla pagina di visualizzazione degli slot. Quest'ultima è l'unica azione permessa all'utente non registrato, mentre le altre due opzioni sono proprio il login e la registrazione. La pagina di login è composta da un form nel quale l'utente indica username e password, oltre a selezionare un campo che indica il tipo di accesso. Si viene quindi reindirizzati alla LoginServlet dove vengono fatti i vari controlli, in particolare oltre a verificare che le credenziali siano corrette rivolgendosi agli opportuni metodi della UtenteFactory, faccio un check sulla tipologia di accesso per fare le relative distinzioni. Dopodiché, passando tramite la servlet AreaPersonale, vengo reindirizzato alla pagina personale dell'utente semplice o amministratore a seconda del tipo di accesso. Salvo come attributi di sessione il tipo di accesso e i dati utente, in modo tale da poter fare dei controlli per garantire che gli utenti semplici accedano solo alle pagine a loro dedicate, così come per gli amministratori. Anche la pagina di registrazione è composta da un semplice form nel quale l'utente compila i campi necessari per registrarsi. Si viene quindi reindirizzati alla servlet NuovoUtente, che dopo aver recuperato i dati inseriti nel form e aver fatto dei controlli, usa il metodo inserisciUtente della UtenteFactory per registrare nel DB il nuovo utente. In particolare ho scelto di far registrare solo utenti maggiorenni o che compiono 18 anni nell'anno corrente, e uso il metodo checkNascita() della classe Utils. Inoltre, qualora l'utente abbia scelto uno username già in uso, poiché deve essere univoco, solleverò un'eccezione. Gli slot disponibili invece vengono mostrati tramite un controllo all'accesso alla pagina dedicata, che fa si che siano recuperati dal DB quelli che hanno almeno un posto disponibile e siano relativi a una data futura. In particolare, si viene reindirizzati alla servlet RecuperoSlot che tramite il metodo recuperaSlot() della SlotFactory, costruisce una lista contenente tanti oggetti di tipo Slot quanti sono quelli disponibili. In particolare, dentro recuperaSlot() uso il metodo checkDate() della classe Utils per vedere se si tratta di una data futura e quindi prenotabile. La lista viene restituita e usata per costruire una tabella con le varie info. Nella pagina in cui sono mostrati gli slot disponibili è mostrato, ma solo se è loggato un utente semplice, un pulsante che reindirizza alla pagina di prenotazione.

B) Funzionalità utente semplice

1) PRENOTAZIONE SLOT: Ho realizzato la pagina di prenotazione con un form che permette di scegliere giorno, fascia e posti. Inoltre ho inserito un pulsante che rimanda alla pagina in cui sono mostrati gli slot disponibili. Dopo aver inserito i dati si viene reindirizzati alla PrenotazioneServlet, dove recupero i parametri della richiesta e faccio dei controlli. Uso il metodo checkDate() per vedere se si sta scegliendo una data futura, dopodiché mi rivolgo al metodo inserisciPrenotazione() della PrenotazioneFactory che verifica se ci sono abbastanza posti disponibili, esegue la prenotazione e aggiorna il numero di posti liberi. A seconda dell'esito della prenotazione imposto i vari parametri che mi permetteranno, tramite dei controlli lato client fatti nella pagina prenotazione.jsp, di mostrare dei diversi messaggi all'utente a seconda che la prenotazione sia avvenuta correttamente, non ci siano abbastanza posti disponibili o abbia scelto una data passata.

2) DATI PERSONALI: Avendo salvato l'oggetto utente come attributo di sessione, ho già tutte le informazioni, e le posso mostrare nella pagina datiUtente.jsp costruendo una tabella con le info dell'utente loggato. Per la modifica invece ho associato a ogni voce della tabella un pulsante, il cui click mi scatena un evento che fa comparire sotto la tabella il form per l'inserimento del nuovo valore. Ho gestito tutto tramite jquery, con il file jmodifica.js che contiene tutto il codice relativo alla comparsa del form di modifica dei dati. In particolare quest'ultimo, qualunque sia il campo che vogliamo modificare, reindirizza sempre alla ModificaServlet che recupera dalla richiesta il nome del campo da modificare e il suo valore. Qui viene fatto un test a parte qualora si voglia modificare lo username, perché deve essere univoco, e si usa il metodo cercaUtente() della UtenteFactory che verifica se lo username è già in uso, informando poi l'utente nella error.jsp, e uno per la data di nascita qualora quest'ultimo sia il campo da modificare. A prescindere dal campo scelto, la modifica viene fatta tramite il metodo modificaDati() della UtenteFactory che effettua una update. Ho scelto poi al termine della modifica di invalidare la sessione e reindirizzare alla pagina di login in modo tale che venga salvato nella nuova sessione l'oggetto utente, con tutti i dati aggiornati.

3) LETTURA FATTURE: Ho implementato la lettura delle fatture una alla volta. Al caricamento della pagina leggiFatture.jsp si viene reindirizzati alla servlet LeggiFatture che usa il metodo leggiFatture() della FatturaFactory per recuperare la prima fattura dell'utente loggato. Successivamente l'utente può scorrere cliccando su opportuni pulsanti che scatenano il recupero delle fatture successive sempre con la stessa servlet e gli stessi metodi, ma con offset diverso. Il tutto è gestito nel file jquery.js. Inoltre, ho fatto in modo che la pagina di visualizzazione delle fatture non sia presente nel menù della pagina personale dell'utente che in fase di registrazione abbia scelto di non ricevere fattura.

4) SEZIONE MESSAGGISTICA: La sezione di messaggistica si caratterizza da un semplice form in cui vengono richiesti oggetto e testo del messaggio. Ho implementato dei controlli lato client che fanno si che la lunghezza dell'oggetto sia massimo 30 caratteri e quella del messaggio 500, e inoltre il bottone di invio del messaggio si attiva solo se sono contemporaneamente presenti messaggio e oggetto. La servlet Messaggi recupera quindi i parametri del form e il mittente dallo username della sessione, mentre

il destinatario è sempre l'unico possibile, cioè lo username dell'admin. Viene poi chiamato il metodo inviaMessaggio() della MessaggioFactory che inserisce il nuovo messaggio nel DB, il tutto dopo aver verificato nuovamente lato server che oggetto o messaggio non siano vuoti. Ho realizzato la NotificheServlet per far si che immediatamente dopo il login, venga fatto un controllo di quanti messaggi nuovi ci siano da leggere. Questa informazione mi viene fornita dal metodo daLeggere() della MessaggioFactory. Successivamente l'utente può leggere i messaggi, che vengono recuperati uno alla volta usando le tecniche Ajax tramite la servlet LeggiMessagi, che usa il metodo leggiMessaggi() della MessaggioFactory. Questo metodo recupera ogni singolo messaggio, uno alla volta, e usa il metodo segnaComeLetto() per far si che quel messaggio non risulti più come nuovo. Ogni volta che viene mostrato un messaggio l'utente sa se lo ha già letto o se è la prima volta che lo legge. Il sistema di conteggio delle notifiche e di lettura dei messaggi è lo stesso per utenti semplici e amministratore.

C) Funzionalità amministratore

1) INSERIMENTO SLOT: Nella pagina personale dell'amministratore ho inserito varie voci per le funzionalità che esso può eseguire. Per l'inserimento di un nuovo slot ho realizzato la inserisciSlot.jsp che ha un form con tutti i dati dello slot, compreso l'identificativo del bagnino di turno. Cliccando sul bottone "mostra bagnini" viene eseguita una richiesta asincrona al server, che permette di mostrare i bagnini tra i quali è possibile scegliere senza ricaricare la pagina. Le info dei bagnini sono recuperati tramite la servlet LeggiBagnini che restituisce una lista con oggetti di tipo bagnino, recuperati dal DB usando il metodo cercaBagnini() della BagninoFactory. Una volta compilato il form si viene reindirizzati alla servlet AggiungiSlot che fa i seguenti controlli: data scelta futura, slot non presente nel DB, bagnino presente nel DB. Se vengono superati tutti i controlli si usa il metodo inserisciSlot() della SlotFactory per registrare lo slot nel DB, e si settano vari parametri. I controlli lato client permettono di capire all'amministratore quale sia l'esito della richiesta e quale errore abbia commesso(bagnino scelto inesistente, data passata, slot già presente).

2) INSERIMENTO BAGNINO: Anche per l'inserimento del bagnino ho creato un'apposita jsp con un form in cui l'amministratore inserisce i dati e che rimanda alla servlet Bagnino. Ogni bagnino ha un id come chiave primaria, ma ho supposto anche che non possano esistere due bagnini con stesso nome, cognome e numero di telefono. Pertanto uso il metodo cercaBagninoByName() della BagninoFactory per verificare se sia già presente. Se è presente avviso l'admin altrimenti lo registro col metodo inserisciBagnino().

3) PROCESSARE PRENOTAZIONI: Accedendo alla pagina analisiPrenotazioni.jsp si viene reindirizzati alla servlet AnalisiPrenotazioni che usando il metodo recuperaPrenotazioniNonProcessate() della PrenotazioniFactory recupera appunto tutte le prenotazioni non ancora processate. In particolare recupera la prima, poi ho implementato il recupero di tutte le altre prenotazioni senza la necessità di ricaricare la pagina. Sopra ogni prenotazione è mostrato un bottone la cui pressione rimanda alla servlet Processa. In questa servlet si utilizza il metodo vuoleFattura() della FatturaFactory per capire se l'utente che ha effettuato la prenotazione desideri la fattura. Qualora l'abbia richiesta si viene reindirizzati alla compilaFattura.jsp, dove l'admin inserisce gli estremi della fattura. Qui ho impostato un controllo lato client della descrizione, che deve essere di massimo 100 caratteri e ho reso il campo id della prenotazione che si sta fatturando non modificabile. Infine ho realizzato la servlet Fattura che permette di recuperare i dati della prenotazione associata per ottenere l'id del bagnino di turno tramite il metodo cercaBagninoByTurno(), emettere la fattura tramite emettiFattura() e infine processarla.

4) INVIO CODICE SCONTO: Per l'invio del codice sconto ho realizzato una pagina nella quale l'amministratore inserisce lo username, e può leggere la lista dei clienti che ne hanno diritto. Quest'ultima viene mostrata senza ricaricare la pagina, grazie alla servlet LeggiUtenti che, usando il metodo cercaUtentiSconto() recupera gli utenti che ne hanno diritto. Il cupon viene inviato come se fosse un messaggio normale, grazie alla servlet InviaCupon che setta i parametri del messaggio e col metodo dirittoSconto() verifica se l'utente scelto ha realmente diritto allo sconto. Successivamente invia il messaggio. Per generare il codice ho implementato il metodo getMessage() nella classe Utils che genera una sequenza di numeri casuali e la inserisce nel messaggio.

5) INVIO REMINDER: Ho implementato l'invio del reminder in maniera molto simile alla pagina in cui le prenotazioni vengono processate. In questo caso recupero tutte le prenotazioni dal DB, una alla volta, tramite la servlet Prenotazioni che recupera una prenotazione usando il metodo recuperaPrenotazioni() della PrenotazioneFactory. Ho inserito poi un pulsante per l'invio del reminder che si attiva solo qualora sia il giorno prima della data per cui è prevista la prenotazione che si sta mostrando. Ho implementato il metodo checkReminder() della classe Utils che mi permette di capire se sussistano le condizioni per l'invio del reminder. Utilizzo poi la servlet Reminder per inviare il reminder come messaggio normale.

6) RICERCA UTENTI: Per la ricerca utenti ho usato un'unica pagina nella quale ho inserito due form. Il primo form permette di scegliere da un elenco l'ordine nel quale visualizzare gli utenti, e rimanda alla servlet RecuperaUtenti che tramite il metodo getUtenti() restituisce la lista di utenti nell'ordine che è stato scelto per la visualizzazione e poi viene usata per costruire una tabella. Per la ricerca del singolo utente ho usato invece la servlet CercaUtente che recupera la stringa inserita dall'admin e ne verifica la validità. Uso poi i metodi cercaByNomeECognome() e cercaByNomeECognomeEData() a seconda di quanti parametri sono stati inseriti. Una volta recuperati i dati dell'utente, se questo desidera la fattura, recupero le sue prenotazioni col metodo getPrenotazioni(). Nella digitazione del l'utente da cercare l'admin è assistito dall'auto completamento che ho implementato con tecniche Ajax, sfruttando la servlet Suggerimenti.

D) Struttura database

Ho realizzato il database creando 6 tabelle, che contengono i dati delle entità utente, messaggio, prenotazione, slot, fattura e bagnino.

1) UTENTE: La tabella utente ha la colonna username come chiave primaria, poi ho inserito come attributi tutti gli altri campi che l'utente indica in fase di registrazione, quali nome, cognome, sesso, password, email, data di nascita, codice fiscale, numero di telefono, fattura e foto. In più ho inserito un'ulteriore colonna chiamata 'tipo' che contiene il carattere 'u' per gli utenti semplici e 'a' per l'amministratore, in modo tale da poter fare distinzione tra i due.

2) MESSAGGIO: La tabella messaggio ha come chiave primaria un ID seriale che viene assegnato a ogni messaggio, i campi destinatario e mittente, che sono delle chiavi esterne che fanno riferimento allo username dell'utente che riceve e invia il messaggio, e due campi testuali contenenti l'oggetto e il contenuto vero e proprio del messaggio. Ho inserito infine un campo booleano chiamato 'letto' che viene impostato di default a false e che viene cambiato a true solo quando il messaggio viene letto dal destinatario.

3) SLOT: La tabella slot contiene le info dello slot, cioè il giorno e la fascia, che formano la chiave primaria, e il numero di posti prenotati. Ho inserito poi un ID come chiave esterna, che fa riferimento alla chiave primaria dell'entità bagnino per poter associare a ogni slot il bagnino di turno.

4) BAGNINO: Per l'entità bagnino ho usato un ID seriale come chiave primaria, accompagnato da nome, cognome e numero di telefono che formano una chiave candidata. Completano la tabella un campo testuale contente gli attestati posseduti dal bagnino e uno contenente l'email del bagnino.

5) PRENOTAZIONE: La tabella prenotazione ha ancora una volta un'ID seriale come chiave primaria. Ho inserito come altri attributi della prenotazione il giorno, la fascia e il numero di posti richiesti. La colonna username è invece una chiave esterna che fa riferimento all'utente che ha effettuato la prenotazione. Infine ho inserito un campo booleano 'processata' che al momento della prenotazione viene impostato false di default, e che viene impostato true solo nel momento in cui l'amministratore decide di processare quella prenotazione.

6) FATTURA: L'ultima tabella è quella della fattura, nella quale la chiave primaria è ancora un ID seriale. Inserisco inoltre una data, relativa al giorno in cui la fattura viene emessa, e due campi numerici relativi al numero di posti prenotati e al prezzo del servizio. Un campo testuale è stato inserito nella tabella per contenere la descrizione del servizio. Infine troviamo come chiavi esterne l'ID della prenotazione, che riporta alla prenotazione alla quale è associata la fattura, e l'ID del bagnino di turno.

*NOTA: Ho inserito ulteriori spiegazioni delle scelte implementative come commenti all'interno del codice

USERNAME	PASSWORD
curreli (account amministratore)	00431
mariorossi	mariorossi
flaviana	flaviana58
luigiarcuri	luigiarcuri