Università degli Studi di Verona

Online Library
Documentazione al prototipo

Magdalena M. Solitro

17 settembre 2020

Indice

1 Introduzione

L'obiettivo di questo progetto era quello di sviluppare un software che simulasse un'applicazione online di e-commerce per una libreria.
Questa relazione costituisce la documentazione del prototipo, in cui spiegheremo l'utilizzo del software e descriveremo le scelte progettuali e implementative adottate.

2 Requisiti

I requisiti del progetto sono stati specificati nella consegna, che riportiamo integralmente di seguito:

> *Si vuole progettare un sistema informatico per gestire gli acquisti on-line di una libreria.*
>
> *Per ogni libro si memorizza il titolo, l'autore o gli autori, la casa editrice, l'anno di pubblicazione, il codice ISBN (identificativo), il genere, il prezzo ed una breve descrizione.*
> *Gli utenti possono visualizzare le classifiche di vendita che sono organizzate per genere (novità, narrativa, ragazzi, ...) e vengono aggiornate ogni settimana. Per ogni posizione della classifica si indica da quante settimana il libro è in quella posizione.*
>
> *Il sistema memorizza gli ordini degli utenti. Gli utenti possono essere registrati o meno. Per gli utenti si memorizzano nome, cognome, indirizzo, CAP, città, numero di telefono ed email. Ogni utente registrato accede con email e password ed ha associata una LibroCard per la raccolta punti. Ogni LibroCard ha un numero identificativo, una data di emissione e il totale dei punti raccolti. Gli utenti registrati possono specificare uno o più indirizzi di spedizione diversi da quello di residenza.*
>
> *Ogni libro ha associato il numero di punti che vengono caricati sulle LibroCard in caso di acquisto da parte di utenti registrati.*
>
> *Per ogni ordine si memorizzano il codice (univoco), la data, i libri che lo compongono, l'utente che lo ha effettuato, il costo totale, il tipo di pagamento (carta di credito, paypal o contrassegno) e il saldo punti se l'utente è registrato*
>
> *Il sistema deve permettere agli utenti registrati di accedere al loro profilo, modificare i dati anagrafici, verificare il saldo punti e lo stato dei loro ordini. Ogni utente registrato può vedere tutti gli ordini che ha effettuato nel tempo con il totale*

dei punti accumulati per ogni ordine. Gli utenti non registrati possono accedere agli ordini cha hanno effettuato tramite il codice dell'ordine.

I responsabili della libreria devono poter verificare lo stato degli ordini, e il saldo punti delle LibroCard degli utenti registrati. Inoltre, i responsabili della libreria sono responsabili dell'inserimento dei dati relativi ai libri che si possono ordinare e dell'aggiornamento delle classifiche. Tutti gli utenti sono opportunamente autenticati dal sistema per poter accedere alle funzionalità di loro competenza.

3 UML

3.1 Use Cases

3.2 Activity Diagram

3.3 Class Diagram

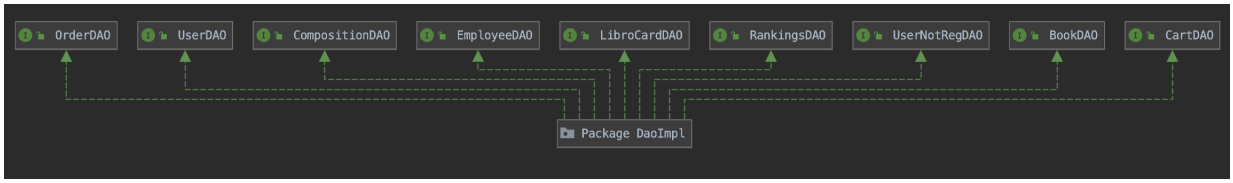


Figura 1: Class Diagram per il package Utils

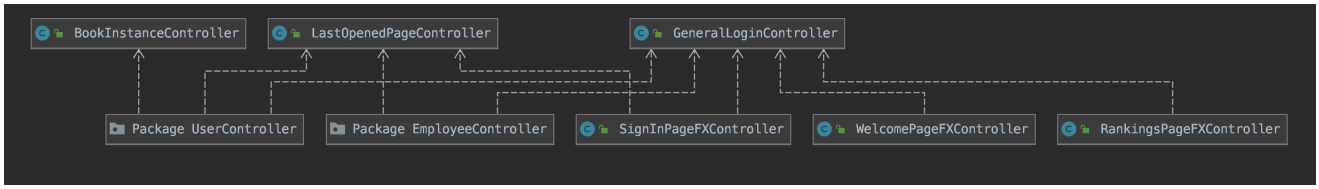


Figura 2: Class Diagram per il package Controller

3.4 Sequence Diagram

4 Scelte progettuali

4.1 Sviluppo

Questo progetto è stato sviluppato principalmente Java, che costituisce un linguaggio estremamente vario e flessibile, e ancora oggi risulta essere molto diffuso nel mercato.
Per l'interfaccia grafica ho deciso di utilizzare due strumenti: JavaFX, una libreria nativa di Java, e FXML, un linguaggio di mark-up basato su XML.
JavaFX rappresenta un'alternativa sicuramente più ricca e versatile rispetto a Swing, che è stato invece il tool proposto a lezione: essa permette infatti di realizzare GUI molto più elaborate, grazie alla presenza di animazioni, oggetti geometrici 2D e 3D, grafici, contenuti multimediali, e altro.
FXML, invece, è stato scelto perché permette di utilizzare un designer di interfacce grafice, Scene Builder, che rende la progettazione sicuramente più semplice e veloce. Inoltre, FXML è particolarmente adatto a interagire con JavaFX, visto che la struttura gerarchica del file rispecchia esattamente la struttura del *scene graph* di JavaFX.
Per stilizzare ulteriormente gli elementi dell'interfaccia grafica si è utilizzato anche CSS, che è completamente supportato sia da JavaFX, sia da FXML.

4.2 Metodologia di sviluppo

Il progetto è stato sviluppato con il supporto di Git come sistema di code versioning e di GitHub per l'hosting del repository. È stata seguita una metodologia di sviluppo Agile ...

4.3 Database

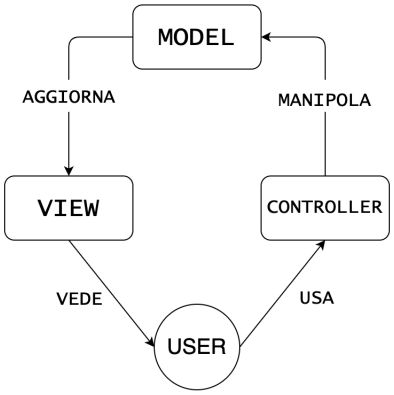
Fin dall'inizio, risultava evidente la necessità di mantenere i dati nel tempo: per questo motivo ho deciso di far uso di un database, sfruttando così anche le conoscenze apprese durante il corso di Database frequentato durante lo sviluppo del progetto.
Dato che la gestione e l'utilizzo del database non rientravano negli obiettivi del progetto, ho scelto di utilizzare SQLite, che costituisce un RDBMS semplice, leggero e open-source.
SQLite è un database engine *"serverless"*, che viene quindi memorizzato interamente in locale, ed è inoltre scritto interamente in C, il che lo rende particolarmente veloce.
Questo strumento presenta tuttavia anche dei limiti ...

4.3.1 Schema logico

4.3.2 Schema relazionale

4.4 MVC Pattern

Il funzionamento del progetto è basato sul MVC pattern, che prevede la suddivisione logica delle classi in tre componenti:



...

4.5 Singleton Pattern

Il *Singleton Pattern* è stato utilizzato per limitare l'instanziazione di un oggetto a uno solo. In termini pratici, l'applicazione può essere usata da un solo utente alla volta, e quindi non è possibile essere loggati durante la stessa sessione come due utenti diversi. Nonostante l'applicazione simuli idealmente un sito di e-commerce, quindi un'applicazione web a cui possono accedere più utenti contemporaneamente, nella pratica è stata sviluppata come un'applicazione desktop: di conseguenza il software non è utilizzabile da più persone contemporaneamente, e questo giustifica l'utilizzo di questo design pattern. L'implementazione di questo pattern è stata sviluppata nella classe `GeneralLoginController`: la classe contiene l'attributo `loginInstance`, inizialmente settato a `null`, la cui funzione è memorizzare una stringa che identifica univocamente l'utente che ha eseguito il login.
Quando l'utente esegue il logout, il valore della stringa torna ad essere `null`.

4.6 Observer Pattern

L'*Observer Patter* prevede che un oggetto possa impostare una lista di "ascoltatori" (tecnicamente chiamati *Observers* o *Listeners*) che vengono notificati automaticamente ogni qualvolta

lo stato dell'oggetto cambia. Questo pattern è stato utilizzato nell'interfaccia grafica, per ottenere dei cambiamenti dinamici in risposta alle interazioni dell'utente con determinati elementi. Più precisamente, è stato impiegato nei seguenti contesti:

- nella `MainPage` dei clienti (registrati e non), per visualizzare i libri appartenenti a un determinato genere;
- nelle `RankingsPage` dei clienti e dei responsabili, per visualizzare le classifiche relative a un determinato genere;
- in `AllLibroCardsPage`, ovvero la pagina che consente ai responsabili della libreria di controllare le LibroCard degli utenti. In questo caso, l'interfaccia cambiava a seconda che si volesse effettaure la ricerca della LibroCard con l'e-mail dell'utente o con la cardID;
- in `AllOrdersPage`, ovvero la pagina che permette ai responsabili di controllare lo stato degli ordini. In questo caso, l'insieme degli ordini visualizzato cambiava in base allo stato che veniva selezionato.

4.7 Data Access Object Pattern

Per la gestione del database ho deciso di adottare il *Data Access Object Pattern*, che permette di separare l'insieme delle azioni che l'applicazione ha bisogno di eseguire sul database dal modo in cui esse sono concretamente implementate.
Per ogni classe del `Model` è stata quindi definita un'interfaccia `DAO`, contenente la dichiarazione dei metodi tramite cui l'oggetto poteva interagire con il database. Ad ogni interfaccia corrisponde una classe `DaoImpl` che la implementa, sfruttando l'API di Java per l'interazione con i DBMS, JDBC, e semplici query SQL.

5 Validazione

6 Conclusioni