Università degli studi di Salerno

***Corso di Laurea in Informatica***

******

***PROGETTO DI INGEGNERIA DEL SOFTWARE – Object Design Document***

***“BUY & SEE”***

**Docente:**

Andrea De Lucia

# Studenti:

##### Nome Matricola

## Fabio Curci 0512103296

## Francesca Di Mauro 05121

## Francesco Ciampa 05121

## Anna Santoro 05121

**Indice**

1. Introduzione

1.1. Object Design Trade-offs

1.2. Linee guida per la documentazione delle interfacce

1.3 Definizioni, acronimi e abbreviazioni

1.4 Riferimenti

2 Packages

2.1.Packages components

3. Class Interfaces

4. Object

**1****. Introduzione**

* 1. **Object Design Trade-off**

Dopo aver stilato il documento di Requirements Analysis e il documento di System Design in cui vi è una descrizione sommaria di ciò che sarà il nostro sistema, definendo i nostri obiettivi ma tralasciando gli aspetti implementativi, andiamo ora a stilare il documento di Object Design che ha come obiettivo quello di produrre un modello che sia in grado di integrare in modo coerente e preciso tutte le funzionalità individuate nelle fasi precedenti. In particolar modo, in tale documento si definiscono le interfacce delle classi, le operazioni, i tipi, gli argomenti e la signature dei sottosistemi definiti nel System Design. Inoltre sono specificati i trade-off e le linee guida.

**Comprensibilità vs Tempo**:

Il codice del sistema deve essere comprensibile il più possibile, in modo da facilitare la fase di testing ed eventuali future modifiche da apportare. Per rispettare queste linee guida il codice sarà accompagnato da commenti volti a semplificarne la comprensione. Ovviamente questo comporterà un aumento del tempo di sviluppo del nostro progetto.

**Prestazioni vs Costi:**

Dato che il nostro progetto è sprovvisto di budget, per poter mantenere prestazioni elevate, in determinate funzionalità verranno utilizzati dei template open source esterni, in particolare Bootstrap.

**Sicurezza vs Efficienza:**La sicurezza, come descritto nei requisiti non funzionali del Requirements Analysis, rappresenta uno degli aspetti importanti del sistema. Tuttavia, dati i tempi di sviluppo molto limitati, ci limiteremo ad implementare sistemi di sicurezza basati su username e password degli utenti.

**1.2 Linee Guida per la Documentazione delle Interfacce**

Gli sviluppatori dovranno seguire determinate linee guida per la stesura del codice:

Naming Convention:

È buona norma utilizzare nomi:

* Descrittivi
* Pronunciabili
* Di uso comune
* Di lunghezza medio-corta
* Non abbreviati
* Utilizzando solo caratteri consentiti (a-z, A-Z, 0-9)

**Variabili**:

* I nomi delle variabili devono iniziare con la lettera minuscola, e le parole successive con la lettera maiuscola. La dichiarazione delle variabili deve essere effettuata ad inizio blocco; in ogni riga vi deve essere una sola dichiarazione di variabile e va effettuato l’allineamento per migliorare la leggibilità.
* In determinati casi, è possibile utilizzare il carattere underscore “\_”, ad esempio quando si fa uso di variabili costanti oppure quando si fa uso di proprietà statiche.

**Metodi**:

* I nomi dei metodi devono iniziare con la lettera minuscola, e le parole successive con la lettera maiuscola. Di solito il nome del metodo è costituito da un verbo che identifica un’azione, seguito dal nome di un oggetto.

I nomi dei metodi per l’accesso e la modifica delle variabili dovranno essere del tipo getNomeVariabile() e setNomeVariabile(). Se viene dichiarata una variabile all’interno di un metodo quest’ultima deve essere dichiarata appena prima del suo utilizzo e deve servire per un solo scopo, per facilitare la leggibilità

* Ai metodi va aggiunta una descrizione, la quale deve essere posizionata prima della dichiarazione del metodo, e deve descriverne lo scopo. La descrizione del metodo deve includere anche informazioni riguardanti gli argomenti, il valore di ritorno, le eccezioni. I metodi devono essere raggruppati in base alla loro funzionalità.

**Classi e pagine:**

* I nomi delle classi e delle pagine devono iniziare con la lettera maiuscola, e anche le parole successive all’interno del nome devono iniziare con la lettera maiuscola.

I nomi delle classi e delle pagine devono essere evocativi, in modo da fornire informazioni sullo scopo di quest’ultime.

Ogni file sorgente .java contiene una singola classe e dev’essere strutturato in un determinato modo:

* Una breve introduzione alla classe

L’introduzione indica: l’autore, la versione e la data.

/\*\*

\* sommario dello scopo della classe.

\*

\* @author [nome dell’autore]

\* @version [numero di versione della classe]

\* @since [versione di partenza]

\*/

* L’istruzione import che permette di importare all’interno della classe gli altri oggetti che la classe utilizza.

* La dichiarazione di una classe è caratterizzata da:

1. Dichiarazione della classe pubblica
2. Dichiarazioni di costanti
3. Dichiarazioni di variabili di classe
4. Dichiarazioni di variabili d’istanza
5. Costruttore
6. Commento e dichiarazione metodi e variabili

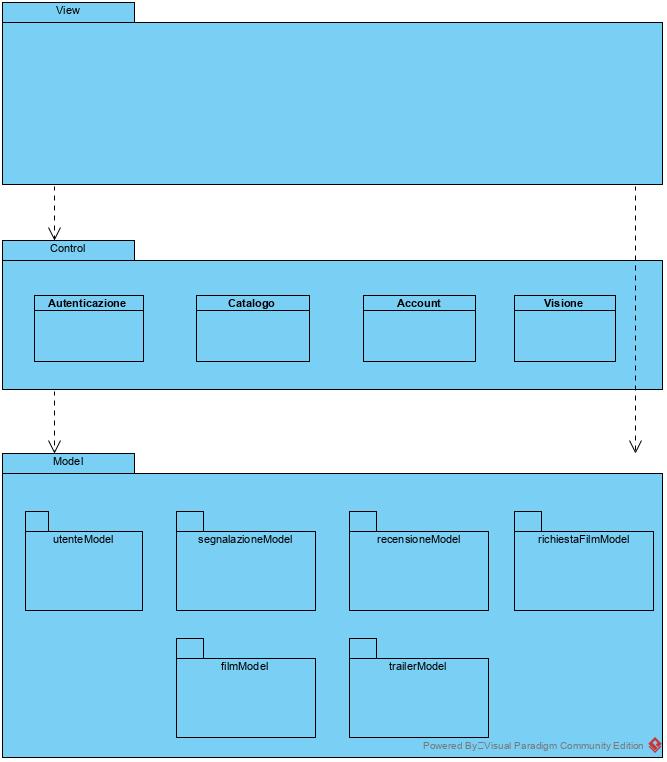
**1.3 Definizioni, acronimi e abbreviazioni**

**1.4 Riferimenti**

**2.** **Packages**

La struttura del nostro sistema, Buy&See, e è strutturata secondo una divisione in package e sottopackage che raggruppano le classi che hanno il compito di gestirne la logica in base alle richieste dell’utente che ne fa uso.

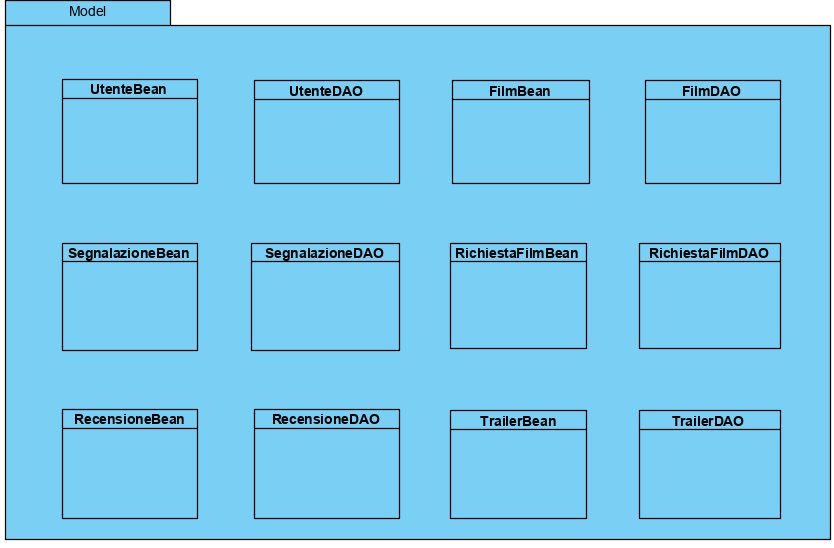
**2.1 Buy&See**



**2.2 Descrizione delle classi**

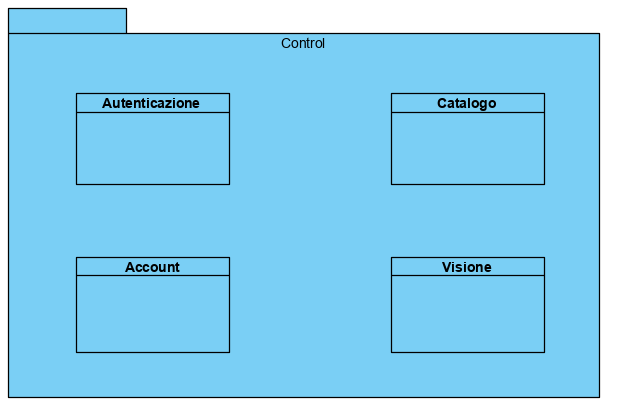
**2.2.1 Model**

Il sottopackage “Model” è presentato nel seguente schema e contiene le classi java rappresentanti le entità presenti all’interno del sistema.



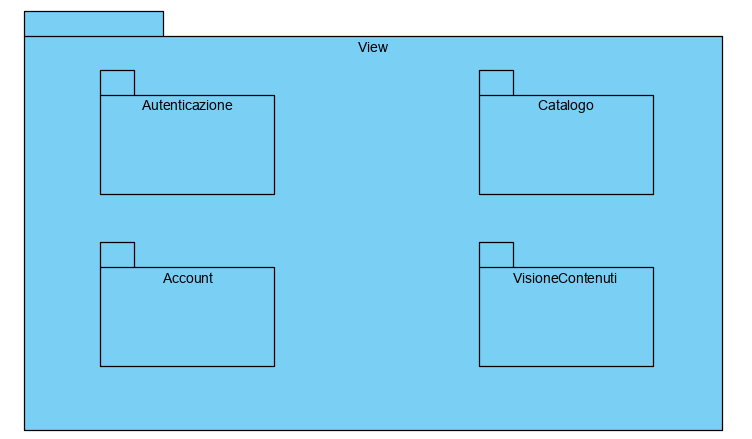
**2.2.1 Control**

Il sottopackage “Control” è presentato nel seguente schema e contiene le classi Java che si occupano della logica di controllo del sistema.

****

**2.2.1 View**

Il sottopackage “View” è presentato nel seguente schema e contiene le classi Java che si occupano della logica di presentazione del sistema.

****

**3. Class Interface**

|  |  |
| --- | --- |
| UtenteDAO | |
| Servizio | Descrizione |
|  |  |
|  |  |
|  |  |
|  |  |

Panoramica di ogni metodo DAO e Manager

|  |  |
| --- | --- |
| RecensioneDAO | |
| Servizio | Descrizione |
| public void doSave(RecensioneBean recensione) | Il sottosistema permette di effettuare una recensione ad un film compilando un form. L’oggetto recensione, passato come parametro, verrà salvato nel database |
| public void doDelete(RecensioneBean recensione) | Il sottosistema permette di |
| public List<RecensioneBean> doRetrieveAll() |  |
|  |  |
| public void doUpdate(RecensioneBean recensione) | Il sottosistema permette di modificare una recensione già presente nel sistema |

**3.1 Object Constraint Language**