# INTRODUZIONE

* 1. Object design Trade-offs

Dopo la realizzazione dei documenti RAD e SDD abbiamo descritto in linea di massima quello che sarà il nostro sistema e gli obbiettivi da seguire, tralasciando gli aspetti implementativi. Il seguente documento ha lo scopo di produrre un modello capace di integrare in modo coerente e preciso tutte le funzionalità̀ individuate nelle fasi precedenti. In particolare, definisce le interfacce delle classi, le operazioni, i tipi, gli argomenti e le signature dei sottosistemi definiti nel System Design. Inoltre, sono specificati i trade-off e le linee guida.

**Prestazioni vs Costi**

Non avendo finanziamenti esterni, si utilizzeranno delle tecnologie open-source in grado di gestire il sistema in maniera gratuita. Nello specifico, verrà utilizzato un database relazionale come repository centrale per i dati gestiti dal sistema e un web server monolitico per la gestione dell’interazione con gli utenti.

**Sicurezza vs Efficienza**

Il sistema si baserà prevalentemente sulla gestione della sicurezza per evitare accessi non autorizzati cosi da proteggere informazioni personali quali e-mail, password e carte di credito.

**Comprensibilità vs Tempo**

Il codice deve essere il più chiaro possibile, ogni componente deve essere accompagnato da un commento in grado di descrivere quali operazioni si stanno implementando. Questa forma di comprensibilità del codice porterà dei rallentamenti in fase implementativa e di testing andando a creare, però, vantaggi sulla comprensione globale del sistema e delle sue componenti.

**Interfaccia vs Usabilità**

L’interfaccia verrà gestita in modo tale da poter essere il più semplice ed intuitiva possibile, attraverso l’uso di form e bottoni di facile comprensione per l’utente finale.

1.2 Linee guida per la documentazione delle interfacce

1.3 Definizioni, acronimi e abbreviazioni

**Acronimi:**

• SDD: System Design Document • ODD: Object Design Document • RAD: Requirements Analysis Document

**Abbreviazioni:**

• DB: Database • DBMS: Database Management System

**Definizioni:**

• Servlet: Classi ed oggetti Java per la gestione di operazioni su un Web Server

1.4 Riferimenti

Il contesto è ripreso dal RAD e dall’ SDD del progetto YouLearn.

È stato anche usato come riferimento il libro: Object-Oriented Software Engineering: Using UML, Patterns, and Java, 3rd Edition Prentice Hall, Upper Saddle River, NJ, September 25, 2009.

inoltre, sono stati usati dei materiali di supporto visionabili al link: <https://wwwbruegge.in.tum.de/lehrstuhl_1/component/content/article/217-OOSE>.

Infine è stato usato anche il libro : Design Patterns: Elements of Reusable Object-Oriented Software, Gang of four, 1994.

# Design Pattern

* 1. Design Pattern Globali

Utilizzeremo il Singleton pattern per le classi di tipo Manager.

Immagine che contiene screenshot

Descrizione generata automaticamente

Il singleton è un design pattern che ha lo scopo di garantire che di una determinata classe venga creata una e una sola istanza, e di fornire un punto di accesso globale a tale istanza.

L'implementazione più semplice di questo pattern prevede che la classe singleton abbia un unico costruttore privato, in modo da impedire l'istanziazione diretta della classe. La classe fornisce in un metodo getter statico che restituisce l'istanza della classe (sempre la stessa), creandola preventivamente o alla prima chiamata del metodo, e memorizzandone il riferimento in un attributo privato anch'esso statico.