|  |
| --- |
| http://www.fondazioneantoniodellamonica.com/images/logo_unisa.png  **Corso di Ingegneria del Software** |

****

**Docente:**

Andrea De Lucia

**Studenti:**

Raffaele Vitiello

Alessio Cuccurullo

Francesco Giuliano

**SYSTEM DESIGN DOCUMENT**

**SOMMARIO**

[**1.1 Introduzione** 4](#_Toc532914442)

[**1.2 Obiettivi di design** 4](#_Toc532914443)

[**1.2.1 Criteri di performance** 5](#_Toc532914444)

[**1.2.2 Criteri di affidabilità** 5](#_Toc532914445)

[**1.2.3 Criteri di manutenzione** 6](#_Toc532914446)

[**1.2.4 Criteri per l’utente Finale** 6](#_Toc532914447)

[**1.3 Glossario** 6](#_Toc532914448)

[**1.4 Riferimenti** 6](#_Toc532914449)

[**1.5** **Panoramica** 6](#_Toc532914450)

[**2.** **Architettura del Software Corrente** 7](#_Toc532914451)

[**3.** **Architettura del Sistema Proposto.** 8](#_Toc532914452)

# **Introduzione**

Nell’ambito dei GDR (giochi di ruolo), i diversi giocatori sono soliti utilizzare i propri giochi preferiti in una casa di uno dei giocatori, usare carta e penna per annotarsi tutto quello che succede nella sessione che stanno giocando: modificare e creare personaggi, scrivere e modificare la storia ecc…  
Purtroppo tutte le fasi di gioco, dalla creazione di una storia alla sua fine, avvengono su tool cartacei. L’uso dei fogli presenta però delle pecche, infatti molto spesso questi vengono smariti. Risulta quindi sgradevole, durante le varie fasi di gioco, perdere del tempo per annotare le varie modifiche da effettuare ai PG e quelle fatte alla storia. Dunque, uno degli scopi di questo sistema è quello di far “risparmiare” tempo ai giocatori. Risparmio inteso come:

1. Abbandonare quasi totalmente le soluzioni cartacee.
2. Eliminazione del rischio di perdere i fogli con il rischio di perdere anche le modifiche.

## **1.2 Obiettivi di design**

Il sistema SineCharta deve poter essere il più efficiente ed intuitivo possibile. Tale efficienza sarà costruita attraverso rapidi tempi di risposta ad ogni genere di input ma anche differenti politiche di tolleranza all’errore. In più si punterà ad aver una buona manutenibilità attraverso il facile inserimento di nuove funzionalità. Sarà intuitivo attraverso agevoli interfacce. Tutto ciò ci porta a considerare quattro distinte classi: Performance, Affidabilità, Manutenzione ed Utente  
Finale.

### **1.2.1 Criteri di performance**

|  |  |
| --- | --- |
| **Tempo di risposta** | SineCharta deve assicurare una risposta rapida alle richieste dell’utente. Una semplice richiesta dell’utente deve essere gestita ed elaborata entro 10 secondi, nel caso di una connessione molto lenta (20-25KBps). I tempi di risposta saranno più rapidi quanto più è veloce la tipologia di connessione utilizzata. |
| **Throughput** | Il sistema sarà capace di gestire contemporaneamente diversi utenti. Non è possibile avere una stima certa del carico ma, mediamente, si prevede un carico mensile di 200 studenti. Tutto questo è possibile grazie alla capacità di elaborazione del server che, nel caso la richiesta aumentasse, è dotata di una struttura scalabile. |
| **Memoria** | La quantità di memoria che verrà utilizzata da SineCharta non può essere stimata precisamente. In principio, il sistema dovrà essere sottoposto alla memorizzazione di almeno: 500 Moderatori, 20 giocatori e 1 admin. |

### **1.2.2 Criteri di affidabilità**

|  |  |
| --- | --- |
| **Robustezza** | SineCharta deve gestire eventuali input errati senza interrompere il funzionamento dell’intero sistema, attraverso delle notifiche di errore. |
| **Disponibilità** | SineCharta deve essere disponibile all’uso, 24 ore su 24, da parte degli utenti, grazie ad un server sempre attivo. |
| **Tolleranza all’errore** | SineCharta deve essere capace di operare durante condizioni d’errore. Ciò sarà reso possibile tra tutte quelle sottoparti del sistema che hanno un grado di accoppiamento basso, in modo tale che l’errore in un sottosistema non intacchi le funzionalità di un altro. |
| **Sicurezza** | L’accesso al sistema è controllato da un sistema di autenticazione che categorizza gli utenti non permettendo l’accesso a parti del sistema non autorizzate. |

### **1.2.3 Criteri di manutenzione**

|  |  |
| --- | --- |
| **Estendibilità** | SineCharta dovrà essere propenso all’inserimento di nuove funzionalità. Quindi, è necessario che il codice scritto sia ben strutturato, dunque, di facile comprensione per evitare ambiguità nella scrittura successiva. |
| **Modificabilità** | Deve essere possibile effettuare eventuali modifiche al sistema (ad es. in caso di bug). Anche in questo caso, è necessario che il codice sia ben strutturato e di facile comprensione. |
| **Leggibilità** | Il codice sarà comodo da leggere grazie ad una accurata indentazione di quest’ultimo. Inoltre, sarà facile da comprendere le diverse parti di cui il codice è composto ci saranno opportuni commenti che ne spiegano il funzionamento. |
| **Tracciabilità dei requisiti** | Grazie alla tracciabilità dei requisiti, sarà possibile effettuare le modifiche necessarie al corretto funzionamento del sistema, valutando correttamente i costi e i rischi che le modifiche porteranno. |

### **1.2.4 Criteri per l’utente Finale**

|  |  |
| --- | --- |
| **Usabilità** | SineCharta rende ogni funzione di semplice uso, garantendo un’ottima esperienza lavorativa all’utente grazie ad interfacce grafiche intuitive. |

## **1.3 Glossario**

## **1.4 Riferimenti**

* 1. **Panoramica**

Prima di parlare dell’architettura, è importante fare un accenno alle attività di system design  
che costituiscono le fondamenta per l’architettura software del sistema.

* Decomposizione del sistema, in cui il sistema viene suddiviso in diversi sottosistemi  
  ognuno dei quali, a sua volta, è caratterizzato da servizi che offre ad altri sottosistemi.

L’insieme dei servizi sarà denominato Interfaccia.

* Mapping Hardware/Software, riguardante la scelta della configurazione hardware del  
  sistema, la comunicazione tra nodi, il come vengano incapsulati i servizi di un  
  sottosistema.
* Gestione dei dati persistenti, nel quale di individuano gli oggetti che devono essere  
  resi persistenti e quale genere di infrastruttura si deve usare per memorizzare tali  
  oggetti.
* Politiche di accesso e Sicurezza, che ci aiuta a rappresentare tramite delle tabelle le  
  operazioni ed informazioni utilizzabili da ogni singolo attore.
* Controllo del software globale, che ci guida su quali operazioni eseguire ed in che  
  ordine, per garantire il corretto flusso di controllo del sistema.
* Condizioni Boundary, che includono oltre l’avvio e lo shutdown anche la gestione dei  
  fallimenti dovuti all’invecchiamento del sistema, interruzione di corrente o anche a  
  errori di progettazione.

1. **Architettura del Software Corrente**

In commercio non è presente un sistema cosi concepito, dunque non è possibile descrivere l’architettura Software esistente.

1. **Architettura del Sistema Proposto**
   1. **Panoramica**

Il tipo di sistema proposto è un’applicazione web, composta da un architettura client/server. Tale sistema deve rispondere alle richieste effettuate da parte degli utenti, in base alla tipologia dell’utente che lo utilizza. Nel caso di un utente moderatore il sistema dovrà interagire con esso mostrando le storie e le sessioni di gioco attive, mentre per un utente giocatore il sistema interagirà con esso mostrando informazioni relative alle proprie sessioni di gioco e i personaggi completi di tutte le statistiche aggiornate.

Le motivazioni che hanno portato ad una scelta di un architettura client/server sono le seguenti:

* **Portabilità:** il sistema essendo un’applicazione web potrà essere utilizzata su diversi dispositivi e macchine differenti.
* **Performance:** il sistema garantisce tempi di risposta rapidi, ma comunque essendo un’applicazione web i tempi saranno in dipendenza della connessione di rete.
* **Scalabilità:** il sistema è in grado di supportare e gestire diverse richieste da parte di molti utenti contemporaneamente collegati all’applicazione.
* **Flessibilità:** il sistema, basato su due tipologie di utenza, visualizza due interfacce simili e diverse funzionalità specifiche per tipologia d’utente.
* **Affidabilità:** le componenti sia client che server garantiscono l’affidabilità anche in presenza di guasti e situazioni impreviste, quindi deve essere possibile effettuare dei backup periodici al database.
  1. **Decomposizione del sistema**

Per la realizzazione del sistema è stata utilizzata un’architettura MVC, poiché l’applicazione si occupa di gestire direttamente i dati, la logica e le funzionalità. I tre ruoli principali dell’applicazione sono:

* **Model:** si occupa di gestire i dati utili all’applicazione, comunicando direttamente con il database.
* **View:**