**­KEY-KNOWLEDGE**



# Nome Documento

Problem Statement Version 1.4.2

SOMMARIO

[**KEY-KNOWLEDGE** 1](#_Toc62639958)

[Nome Documento 1](#_Toc62639960)

[TABELLA REVISIONI 2](#_Toc62639961)

[Design Goals 3](#_Toc62639962)

[Trade-off: 4](#_Toc62639963)

[Architettura del Sistema proposto 5](#_Toc62639964)

[Decomposizione in sottosistemi 5](#_Toc62639965)

[Mapping hardware/software 7](#_Toc62639966) [Rappresentazione UML del DataBase noSQL proposto 8](#_Toc62639967)

# ****TABELLA REVISIONI****

|  |  |  |  |
| --- | --- | --- | --- |
| **DATA** | **VERSIONE** | **DESCRIZIONE** | **AUTORE** |
| **17/10/2020** | **1.0** | **Prima versione del problemStatemant** | **Crescenzo Manzone**  **Franco Nicola Fernando**  **Giovanni Battista Mercurio** |
| **23/10/2020** | **1.1** | **Modifica scenari** | **Crescenzo Manzone** |
| **24/10/2020** | **1.2** | **Modifica Requisiti Funzionali** | **Franco Nicola Fernando** |
| **25/10/2020** | **1.3** | **Modifica Requisiti non Funzionali** | **Giovanni Battista Mercurio** |
| **27/10/2020** | **1.4** | **Unione delle modifiche effettuate** | **Crescenzo Manzone**  **Franco Nicola Fernando**  **Giovanni Battista Mercurio** |
| **6/11/2020** | **1.4.2** | **Revisione requisiti** | **Crescenzo Mazzone** |

# Design Goals

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **CATEGORIA** | **Nome** | **Descrizione** | **Priorità** |
| DG\_1 | Criteri di End User: | Usabilità | Il Sistema dovrà avere un’interfaccia funzionale e di facile comprensione. L’interfaccia dovrà adattarsi ad ogni device di ogni utente , includendo widget di facile utilizzo. | Alta |
| DG\_2 | Criteri di Affidabilità: | Attendibilità | Il Sistema garantirà l’attendibilità, intesa come correttezza e affidabilità dei dati. | Alta |
| DG\_3 | Criteri di Affidabilità: | Disponibilità | Il Sistema dovrà essere permanentemente fruibile dagli utenti. | Alta |
| DG\_4 | Criteri di Affidabilità: | Sicurezza | Ogni Utente è dotato di nickname e password con cui poter effettuare l’accesso al Sistema e utilizzare solo le funzionalità associate alla propria categoria. | Bassa |
| DG\_5 | Criteri di Affidabilità: | Tolleranza ai guasti | Ogni guasto parziale non influenzerà l’intera applicazione, se una funzionalità non è disponibile , l’utente ne verrà a conoscenza solo se la richiederà. | Bassa |
| DG\_6 | Criteri di Performance: | Tempi di risposta | Il prodotto Software deve consentire una partita con tempi di attesa brevi ai vari utenti, le domande saranno caricate sul client all’inizio di ogni partita, per ottimizzare le prestazioni. | Bassa |
| DG\_7 | Criteri di Performance: | Memoria | Il Sistema memorizzerà uno storico delle risposte nel client di ogni utente. Le domande di ogni partita vengono scaricate dal DataBase e cancellate nel momento in cui termina la partita. | Bassa |
| DG\_8 | Criteri di Manutenibilità: | Modificabilità | Il Sistema dovrà essere facilmente modificabile. | Alta |
| DG\_9 | Criteri di Manutenibilità: | leggibilità | Il sistema dovrà essere facilmente comprensibile leggendo il codice.Ci dovranno essere opportuni commenti in tutti I file di codice. | Alta |
| DG\_10 | Criteri di Manutenibilità: | Tracciabilità | Il codice sarà suddiviso in porzioni in base ai requisiti. | Alta |

# Trade-off:

* **Attendibilità vs Tempi di risposta**

Il Sistema sarà implementato in modo tale da preferire il tempo di risposta all’attendibilità, in modo tale

da garantire attese brevi agli utenti.

* **Disponibilità vs Tolleranza ai guasti**

Il Sistema deve sempre essere disponibile all’Utente in caso di errore in una funzionalità, anche al

costo di rendere non disponibile quest’ultima per un lasso di tempo.

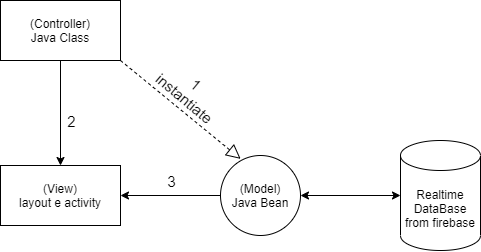
* **Criteri di Manutenibilità vs Criteri di Performance**

Il Sistema sarà implementato preferendo la manutenibilità ai tempi di risposta in modo da facilitare gli sviluppatori nel processo di aggiornamento del software a discapito delle performance del sistema.

# 

# Architettura del Sistema proposto

**Panoramica**

Il pattern architetturale scelto è MVC (Model, View, Control).

**Model:** contiene i le classi DAO con i rispettivi getters e setters.

**View:** si occupa di visualizzare i dati all’Utente e gestisce l’interazione fra quest’ultimo e l’infrastruttura

sottostante.

**Controller:** riceve i comandi dell’Utente attraverso il View e reagisce eseguendo delle operazioni che possono interessare il Model e che portano generalmente ad un cambiamento di stato del View.

# Decomposizione in sottosistemi

Il Sistema è stato diviso in quattro sottosistemi principali:

* **Gestione Account**

Il Sottosistema ***Gestione Account*** gestisce l’autenticazione dell’Utente, in particolare le funzioni di Login, Logout, Recupero Password e Registrazione. L’utente può accedere alla sua area personale e visualizzare i suoi dati, in particolare: nickname, nome, cognome, email, foto. L’utente inoltre può accedere alla sua area KNOWLEDGE che gli permette di visualizzare i dati relativi alle domande risposte nel corso delle sue partite. Inoltre l’UserManager può sospendere o eliminare utenti accedendo alla sua area gestionale.

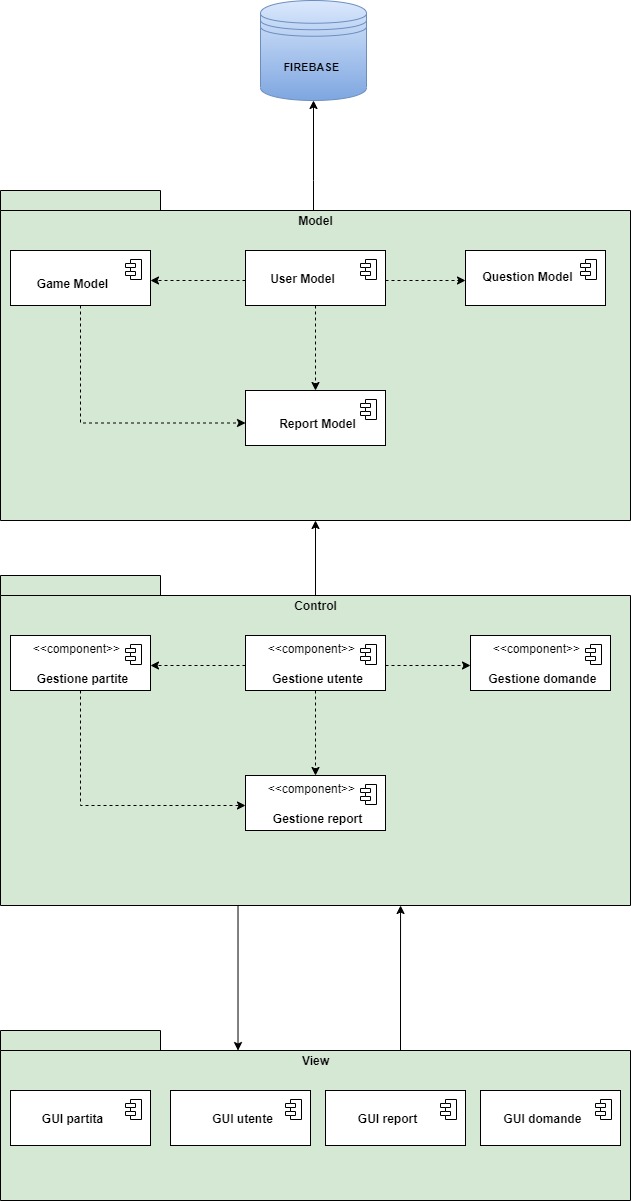
* **Gestione Domande**

Il Sottosistema ***Gestione Domande*** gestisce l’elenco di domande, ogni domanda può essere cancellata o modificata accedendo alla relativa categoria dal Question Manager (inserendo testo domanda, difficoltà, risposte (4), risposta esatta). Possono essere aggiunte nuove categorie con un nome univoco.

* **Gestione Partite**

Il Sottosistema ***Gestione Partite*** gestisce la creazione di partite da parte degli utenti. Ogni partita si svolge tra due utenti. Ogni partita deve contenere gli identificativi dei due utenti coinvolti, chi ha vinto la partita e la sua modalità (MISC, RESTART, CLASSIC). Inoltre si gestisce la possibilità da parte degli utenti di terminare anticipatamente la partita.

* **Gestione Report**

Siccome ogni utente ha la possibilità di creare domande nella modalità MISC è stato introdotto il sottosistema Gestione Report. Ogni utente che risponde a una domanda creata può farne il report semplicemente cliccando un pulsante per segnalare l’utente avversario. Il Report viene gestito dall’UserManager che lo visualizza e può sospendere o eliminare l’utente che ha generato la domanda.

# Mapping hardware/software

Il sistema proposto sarà sviluppato come un’applicazione android che si connette a un server remoto per la memorizzazione e il recupero dei dati nel Database. Si utilizza un’architettura client/server dove un server fornisce servizi a più client. Su un dispositivo android è eseguita un’applicazione client che consente all’utente di interagire attraverso il livello View inviando richieste attraverso il livello Control e Model al server. La comunicazione tra applicazione client android e server avviene tramite json. Le specifiche hardware e software necessarie per il client sono avere un dispositivo con sistema operativo android che può connettersi alla rete. Per la parte server, invece, il client accede a un server firebase con un database. Quest’ultimo non è altro che un servizio offerto da Google, quindi la parte server è gestita dal servizio utilizzato. Le specifiche software per il server comprendono un Database noSql per la gestione dei dati persistenti.

# Rappresentazione UML del DataBase noSQL proposto