System Design Document

Versione	Cambiamenti	Autori
0.1	Struttura del Documento, bozza di Design Goals e Suddivisione in Sottosistemi	Pietro Negri
0.3	Scelta del Database relazionale, costruzione dello schema EER, tabelle delle entità	Antonio Trovato
0.4	Hardware/Software Mapping	Pietro Negri
0.5	Controllo di Accesso, tabella di accesso e sicurezza	Antonio Trovato
0.7	Boundary Conditions	Mafalda Ingenito
0.8	Correzione Boundary Conditions	Mafalda Ingenito
0.9	Flusso di Controllo Globale	Giovanni Renzulli
1.0	Correzioni generali, ampliamento delle sezioni e correzione suddivisione in sottosistemi	Pietro Negri, Antonio Trovato, Mafalda Ingenito, Giovanni Renzulli
1.1	Miglioramento Decomposizione in Sottosistemi, Correzione Tabelle, pulita sezione Dati Persistenti.	Pietro Negri, Antonio Trovato

Indice generale

1. Introduzione

1.1 Obiettivi del Sistema

Il sistema che si intende realizzare ha come scopo la facilitazione della gestione interna di una Scuderia di Formula 1.

L'obiettivo è realizzare un sistema che permetta a un membro dello Staff di comunicare in maniera intuitiva con gli altri membri durante il week-end di gara. Bisogna inoltre consentire agli utenti di svolgere particolari mansioni, o visualizzare determinate informazioni, relative alla propria posizione all'interno della squadra.

Il cliente richiede che sia possibile, tramite un'interfaccia che sia facilmente comprensibile da ogni tipologia di utente, riuscire a realizzare setup o strategie complicate che di norma richiederebbero giorni di analisi dei dati e di lavoro se dovessero essere decise a voce o a penna, rischiando errori che comprometterebbero i risultati della squadra dal punto di vista sportivo ed economico, dando quindi priorità all'usabilità.

Il sistema è realizzato tramite un'interfaccia web ed è sostanzialmente organizzato come un sito suddiviso in diverse sezioni, tramite le quali le diverse tipologie di utenti hanno accesso alle funzioni a loro assegnate.

Un pilota ha tutto l'interesse a sfruttare la piattaforma di messaggistica per comunicare con i Tecnici e gli Strateghi durante le prove e le qualifiche per richiedere particolari assetti o modifiche all'auto senza dover scendere o spiegarsi a voce.

Un tecnico invece deve avere a disposizione i dati del circuito su cui si sta correndo, come possono essere per esempio l'altitudine e il numero di curve, per compilare un assetto che sia in grado di rendere l'auto veloce sul circuito in esame. Questo è importante anche nell'ottica del modulo di IA, che dovrebbe ulteriormente facilitare il tecnico nell'interpretazione dei dati fornendo una base di partenza su cui costruire l'assetto in vista delle varie prove a cui la macchina sarà sottoposta durante il week-end. Gli assetti, dal canto loro, per una questione di organizzazione devono essere assegnati a un circuito in particolare al fine di fornire i dati corretti per la loro finalizzazione.

Uno Stratega, il cui lavoro è molto simile a quello del Tecnico, ha anch'esso bisogno dei dati relativi alle caratteristiche del tracciato. Ma il suo impiego è quello di, una volta osservati i risultati delle prove e i dati ricavati dal tracciato, trovare un giusto equilibrio tra un numero adeguato di soste, la scelta della miscela di gomme e del carburante con cui partire. Ciò significa dover gestire dati di natura diversa da quelli del Tecnico, seppur strettamente correlati, in cui interviene allo stesso modo l'Intelligenza Artificiale per fornire una base di partenza.

Devono anche essere offerte delle funzionalità tramite le quali ogni Tecnico/Stratega può visualizzare dati relativi a un circuito in particolare tra quelli presenti nel calendario dell'annata in corso. Relativamente al sistema, è richiesto che gli utenti possano autenticarsi e gestire in modo basilare il proprio profilo.Inoltre è richiesto che il sistema fornisca all'utente gli strumenti necessari per caricare file contenenti particolari setup oppure di esportare dei setup salvati collegati al proprio account

1.2 Design Goals

I design goals identificati per il sistema AM-GP sono i seguenti:

1 Criteri di Performance

- Response Time:
  - Si assicura un tempo medio di risposta inferiore al secondo per tutte le richieste, in particolare:
    - Per le gestioni di Setup, Strategie e funzionalità di visualizzazione elenchi, il tempo di risposta è di 1 secondo.
    - Per l'aiuto dell'IA il tempo di risposta è inferiore ai 5 secondi.
    - Per l'invio di un messaggio, la funzionalità assicura un tempo di risposta inferiore al secondo.
- Throughput:
  - Si assicurano le funzionalità del sistema per almeno 50 utenti concorrenti, collegati in locale.

2 Criteri di Affidabilità

- Robustezza:
  - Si prevede una doppia convalida, lato client e lato server, per l'input inserito da parte dell'utente, in ogni form i cui vincoli siano stati specificati nel documento di Analisi dei Requisiti.
  - Si prevede una convalida lato server per l'estensione e il formato dei file caricati sulla piattaforma.
- Reliability:
  - I dati che possono essere osservati all'interno del sistema sono attendibili e consistenti con il comportamento tenuto dagli utenti in precedenza. Essi rispecchiano infatti la reale situazione in funzione dell'aggiornamento dei dati attuato dai diversi tipi di utenti che possono modificare, inserire o cancellare dati persistenti.
- Fault Tolerance:
  - Il sistema può subire guasti dovuti al sovraccarico del database. Per ovviare al problema è previsto un salvataggio periodico delle informazioni sotto forma di codice SQL, per rigenerare il database in caso di malfunzionamenti.
- Security:
  - L'accesso è garantito da una ID, rilasciata dal fornitore di servizi, e una password.
  - L'applicazione è protetta da SQLInjection e attacchi XSS.

3 Criteri di Costo

- Costo di Sviluppo:
  - Si stima un costo di sviluppo di circa 160 ore tra progettazione e sviluppo, 40 per ogni Team Member.

4 Criteri di Manutenzione

- Portabilità:
  - La portabilità è assicurata dal fatto che l'interazione avviene mediante un browser, per cui non avviene un'interazione diretta con il sistema sottostante. Ciò rende l'applicazione indipendente dal Sistema Operativo utilizzato.
- Tracciabilità dei Requisiti:
  - La Tracciabilità dei Requisiti è garantita dalla presenza di matrici RF-UC, UC-SUBSYS, RNF-DG.

5 Criteri per l'Utente Finale:

- Utility:
  - L'applicazione proposta cerca di facilitare il lavoro degli utenti in determinate aree di competenza, cercando di fornire uno strumento che schematizzi, semplifichi e renda facile organizzare grandi quantità di dati, rendendo più facile e più trasparente il lavoro rispetto a una rappresentazione a voce o a penna delle stesse informazioni.
- Usability:
  - Per garantire la massima usabilità da parte dell'utente, questo sistema si propone come un sistema semplice da apprendere, grazie all'essenzialità delle funzioni proposte, ai suggerimenti della UI presenti per ogni funzionalità del sistema, e allo stile grafico minimale, con bottoni molto grandi, utilizzabili facilmente da touch screen, e scritte ben leggibili ad alto contrasto.

1.3 Definizioni, Acronimi e Abbreviazioni

Formula 1: La massima categoria (in termini prestazionali) di vetture monoposto a ruote scoperte da corsa su circuito.

Tecnico: Membro dello staff le cui mansioni risiedono nella gestione dell'auto.

Stratega: Membro dello staff le cui mansioni risiedono nell'elaborazione di una strategia per vincere la corsa.

Pilota: Membro dello staff le cui mansioni risiedono nel guidare la monoposto durante la corsa.

Setup: Particolare insieme di modifiche effettuate ai componenti dell'auto.

Strategia: Particolare insieme di soste, quantità di carburante e gomme da montare alla vettura.
SQL: Strucured Query Language, linguaggio standardizzato per database basati sul modello relazionale.

Greenfield Engineering: Tipologia di sviluppo, innescata da una particolare necessità in un determinato ambito, che comincia da zero, in cui nessun sistema esiste precedentemente e i requisiti vengono estratti dall'utente e dai clienti.

DB: Database.

Matrice RF-UC: Matrice che assegna a ogni Requisito Funzionale un Caso D'Uso che lo realizza.

Matrice UC-SUBSYS: Matrice che assegna a ogni caso d'uso un Sottosistema.

Matrice RNF-DG: Matrice che assegna a ogni Requisito non Funzionale, un Design Goal.

Event-Driven:
XSS: Cross-site Scripting.

UI: User-Interface, Interfaccia Utente.

IA: Intelligenza Artificiale.

CRUD: Create, Retrieve, Update, Delete
JDBC: Connettore per database che consente l'accesso e la gestione della pesistenza dei dati sulle basi dati da qualsiasi programma scritto in Java.

1.4 Riferimenti

- Bernd Bruegge & Allen H.Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, (3rd Edition), Prentice Hall 2010.
- AM-GP RAD V1.0
- https://www.raeng.org.uk/publications/other/14-car-racing

1.5 Panoramica

Al secondo punto del documento presentiamo in modo generale il sistema corrente.

Al terzo punto presenteremo l’architettura del sistema, proponendo innanzitutto una panoramica della suddivisione in sottosistemi. Poi andremo ad esaminare più nel dettaglio i sottosistemi, esamineremo il mapping tra hardware e software, i dati persistenti, il controllo degli accessi e la sicurezza, il controllo del flusso globale e le boundary condition. Infine, tratteremo le interfacce proposte per ogni sottosistema.

2. Architettura del Sistema Corrente

Attualmente non esistono sistemi software che si occupano di gestire questa tipologia di problematiche.

I software utilizzati sono perlopiù per l’interpretazione di dati telemetrici, meteorologici e monitoraggio dell’auto.

Trattandosi di sistemi non compatibili con il nostro background di studenti di Informatica del terzo anno, e non avendo a disposizione le risorse per poterci interfacciare con questo tipo di sistemi, abbiamo cercato di orientarci verso qualcosa che abbracciasse alcune necessità lasciate irrisolte perché ritenute secondarie.
Di conseguenza questo progetto rientra nell’ambito della Greenfield Engineering, nasce cioè sulla base di bisogni dell’utente ed è il risultato dell’interazione stretta tra Cliente, Utente e Team di Sviluppo.

3. Architettura del Sistema proposto

3.1 Panoramica

Il sistema proposto è fondamentalmente un'applicazione web, in locale per motivi relativi al suo impiego e utilizzo nel contesto di un paddock.

L'obiettivo è fornire un gestionale che aiuti il team nella comunicazione e nell'organizzazione.

Il sistema è suddiviso in client e server, dove il client gestisce la parte di presentazione e la parte di logica relativa all'interfaccia grafica mentre il server si occupa della logica di business, che nel nostro caso è principalmente un'interazione, mediata da JDBC, verso un database, che ci consente di gestire i dati persistenti, ovvero di effettuare le operazioni CRUD.

La suddivisione in sottosistemi è stata effettuata sulla base delle funzionalità offerte, ovvero si è cercato di creare dei sottosistemi adibiti alla gestione delle entità di cui tenere tracce.

Non mancano anche sottosistemi di supporto che offrono funzionalità non strettamente legate alla gestione dei dati persistenti, che diventano sottosistemi separati per mantenere elevata la coesione del sistema.

Si è cercato di mantenere un compromesso tra un'elevata coesione e un accoppiamento ridotto, per quanto sia chiaro che aumentare la coesione significa anche aumentare l'accoppiamento.

La suddivisione è stata effettuata anche sulla base delle tecniche di layering e di partitioning.

3.2 Decomposizione in Sottosistemi

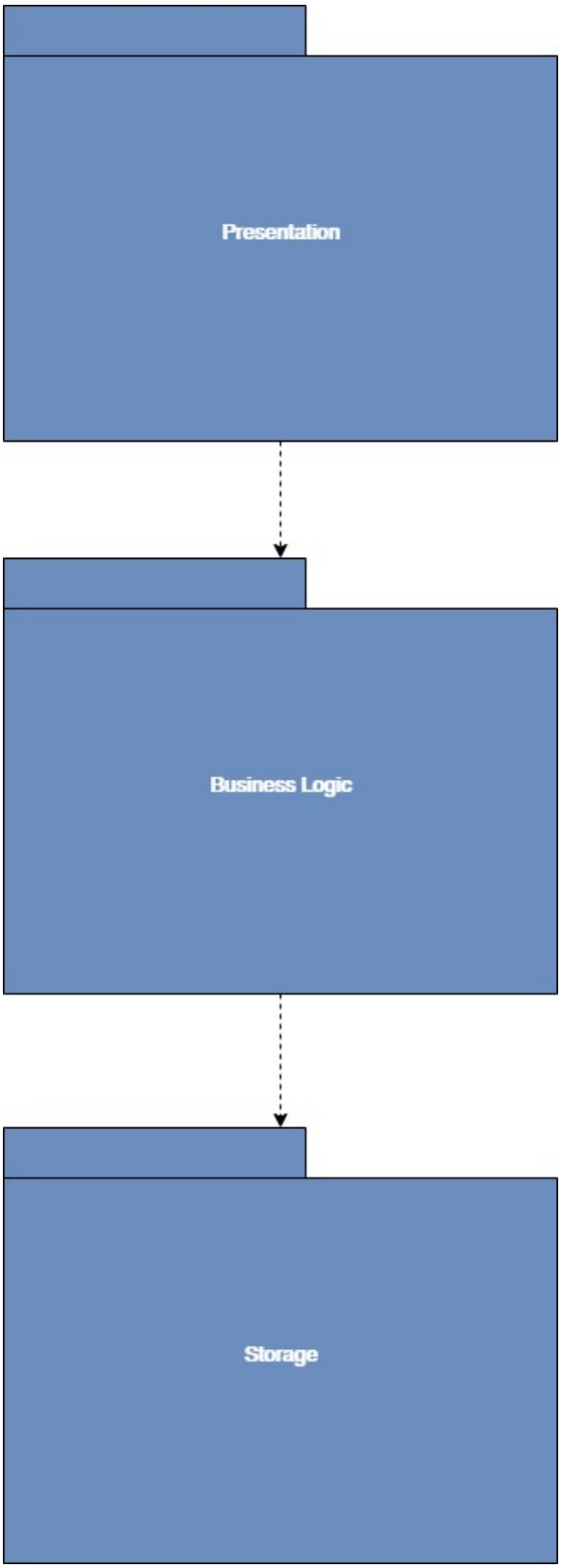
3.2.1 Layering

Il sistema adotta un'architettura client-server three-tier, in cui le funzionalità sono suddivise in tre layer che si occupano di gestire funzionalità differenti.

I vantaggi di questo tipo di architettura risiedono nella modularità e nella sostanziale indipendenza dei tre layer.

- Presentation: Si occupa di gestire l'interfaccia grafica e la logica di business ad essa associata, come la raccolta degli eventi generati dall'utente.
- Business Logic: Si occupa della gestione della logica del sistema, dell'interazione tra i sottosistemi e del controllo delle richieste ricevute dallo strato sovrastante.
- Storage: Si occupa di gestire i dati persistenti, necessari al funzionamento dell'applicazione.

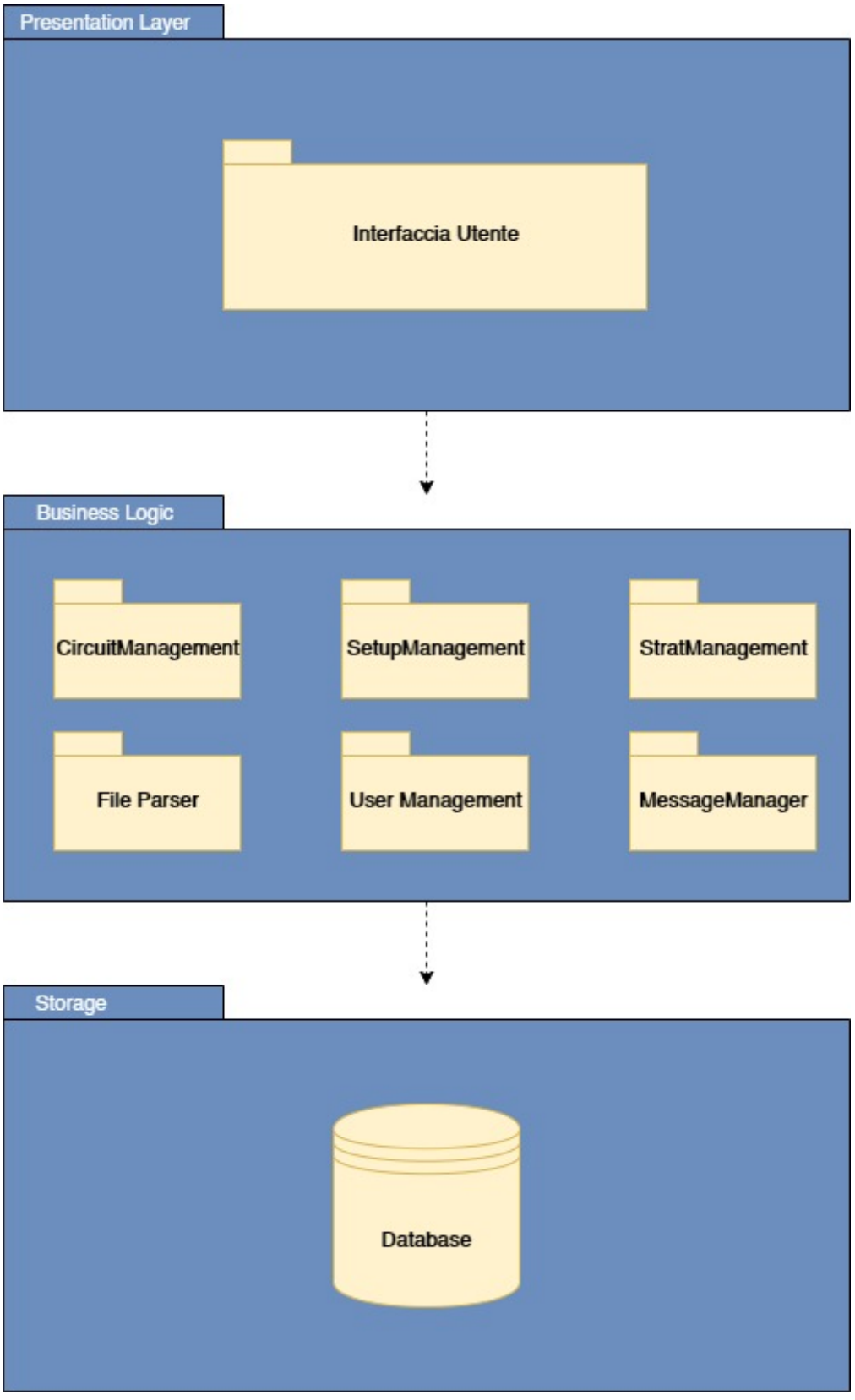
1



1 Gli strati sono rappresentati come package UML.

3.2.2 Struttura dei Sottosistemi

[2]



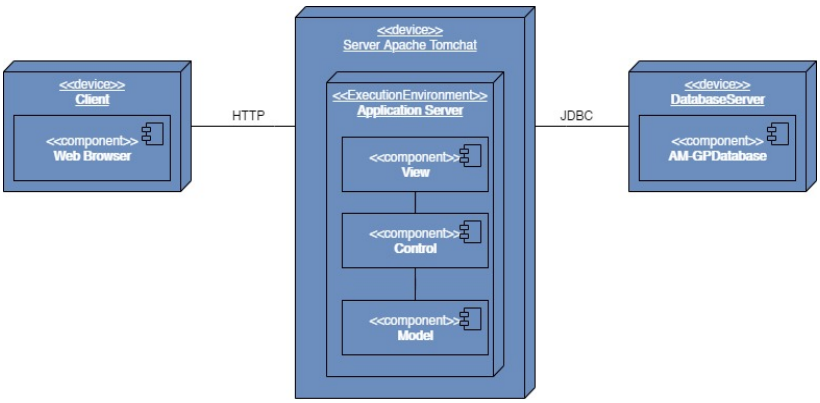
2 I layer sono rappresentati come package UML all'interno del quale troviamo i sottosistemi

Abbiamo deciso di dividere il sottosistema in questo modo sulla base delle funzionalità offerte, cercando di privilegiare innanzitutto la coesione (affidando a ogni sottosistema dei compiti ben specifici). Questo ci ha portato ad avere purtroppo un po' di accoppiamento in più rispetto a una situazione in cui alcuni compiti potevano essere condivisi da un singolo sottosistema. Ma siamo comunque riusciti a mantenere un accoppiamento accettabile.

Una scelta particolare è stata quella di prevedere un sottosistema dedicato allo storage che si interfaccia poi con il database vero e proprio, per disaccoppiare i singoli sistemi con il database vero e proprio: In caso di modifica della base di dati sottostante, l'importante è che i servizi offerti siano sempre gli stessi per non dover modificare l'intero sistema.

Abbiamo aggiunto un sottosistema di File Parser, sfruttato dal Manager di Messaggi e dal Manager di Setup, che si occupa di verificare la correttezza dei file caricati sulla piattaforma.

3.3 Hardware/Software Mapping



Il sistema sarà implementato tramite un Web Server Apache Tomcat, all'interno del quale troviamo l'ambiente di esecuzione in cui vengono accolte e gestite le richieste. Il web server si interfaccerà, tramite JDBC, a un DBMS MySQL per la gestione dei dati persistenti importanti per l'applicazione.

Questa mappatura è strettamente correlata alla scelta di progettare il sistema con un'architettura di tipo Client-Server.

Avremo un Client, che corrisponde a una macchina separata su cui è in esecuzione un web browser connesso alla rete locale. Su un altro dispositivo troveremo il web server su cui è in esecuzione il sistema, che è collegato a un database che può essere, a seconda delle necessità, sia sulla stessa macchina che su una macchina diversa.

3.4 Gestione dei Dati Persistenti

Il sistema ha bisogno di effettuare query complesse sui dati, inoltre si ha bisogno di gestire la memorizzazione persistente e fornire meccanismi di backup: abbiamo pensato di sfruttare le potenzialità dei database relazionali per la memorizzazione dei dati e lo sfruttamento di procedure MySQL per la rigenerazione del database.

Per la nostra tipologia di sistema, il database relazionale è più semplice da gestire rispetto a una gestione esclusivamente tramite file (è comunque possibile esportare dei dati dal database e scriverli su di un file) e fornisce prestazioni migliori rispetto a un database object-oriented.

Gli oggetti persistenti provengono dagli entity individuati in fase di Analisi, e lo schema del DB è pesantemente influenzato dal class diagram del RAD.

Le relazioni sono esattamente le stesse e il mapping è sostanzialmente 1 a 1.

Per rappresentare la tassonomia dell'utente viene utilizzato il vertical mapping. (Presenza di attributo "ruolo" nella classe Utente). Lo schema nel dettaglio e il suo mapping sono descritti in un documento a parte, chiamato MappingDB.

3.5 Controllo di Accesso e Sicurezza

Il controllo degli accessi è garantito tramite l'utilizzo di id e password per ogni tipo di utente. L'accesso ai dati riguardanti il setup è reso sicuro poiché solo l'utente tecnico, tramite autenticazione, può accedere a tali dati. L'accesso ai dati riguardanti la strategia è reso sicuro poiché solo l'utente stratega, tramite autenticazione, può accedere a tali dati. L'accesso ai dati riguardanti la classifica è reso sicuro poiché solo l'utente pilota, tramite autenticazione, può accedere a tali dati.
Le operazioni che gli utenti possono fare sono:

	Messaggio	Circuito	Setup	Strategia	Tabellone
Pilota	sendMessage() getMessage() getMessaggiInviati() getMessaggiRicevuti()				viewTabellone()
Tecnico	sendMessage() getMessage() getMessaggiInviati() getMessaggiRicevuti()	0	getSetup() downloadSetup() saveSetup() getSetupList()		
Stratega	sendMessage() getMessage() getMessaggiInviati() getMessaggiRicevuti()	0		downloadStrat() saveStrategy() getStrategyList() getStrategy()	

3.6 Flusso di Controllo Software Globale

Il sistema è un'unione di due modelli.

Nel nostro caso, trattandosi di un'architettura client-server, in cui un server aspetta delle richieste inviate da un client, le processa e le invia eventualmente a Servlet oppure JSP ,si tratta di un sistema fondamentalmente event-driven. Cioè abbiamo sostanzialmente degli eventi (l'arrivo di una richiesta) e dei gestori (le Servlet) che si occupano poi di processare le richieste.

Allo stesso tempo però, il Web Server scelto si comporta in modo da ottimizzare questo comportamento allocando, per ogni richiesta, un nuovo thread che si occupa di tutta la gestione della singola richiesta, in maniera concorrente rispetto alle altre. Questo assicura che quando una singola richiesta richieda più tempo di un'altra, essa non tolga troppo tempo alle altre.

Inoltre, questo assicura un importante design goal, pur aumentando la complessità, cioè la necessità di avere un numero di utenti concorrente discretamente voluminoso, trattandosi di un sistema che verrà usato durante un week-end da non meno di 50 persone e deve mantenere una performance elevata per le necessità di lavoro degli utenti coinvolti.

La scelta dell'ambiente di sviluppo di Apache ci consente di non dover gestire in prima persona le problematiche relative alla sincronizzazione dell'accesso alle risorse, tipiche della gestione dei Thread, riducendo i costi.

3.7 Boundary Conditions

3.7.1 Deploy

Il SystemOperator deploya l'applicazione sul server e, se questo era stato arrestato correttamente, l'inizializzazione viene completata permettendo all'utente di poter avviare i vari casi d'uso.

NOME	StartUpServer()
ATTORI PARTECIPANTI	SystemOperator
ENTRY CONDITION	• Il server è nello stato di arresto
FLOW OF EVENT	1 System Operator deploya l'applicazione sul server.
EXIT CONDITION	• L'applicazione è deployata e attiva sul server.

3.7.2 Shutdown

Il SystemOperator arresta il server che, prima di terminare l'esecuzione, verifica se ci sono dati persistenti da salvare. Se la verifica e l'eventuale salvataggio va a buon fine, il server è ufficialmente arrestato e potrà eventualmente essere attivato se l'utente fa partire il caso d'uso per lo sturt-up del server.

NOME	ShutdownServer()
ATTORI PARTECIPANTI	SystemOperator
ENTRY CONDITION	• Il server è in esecuzione
FLOW OF EVENT	1 SystemOperator arresta il server; 2 Il sistema verifica se ci sono dati persistenti da salvare ; 3 Il sistema termina l'esecuzione;
EXIT CONDITION	• Il sistema AM-GP non è più in esecuzione;

3.7.3 LostConnectionException()

La perdita della connessione, causata da vari fattori quali, ad esempio, la lontananza dal un trasmettitore, è una delle eccezioni che il sistema si ritrova a dover affrontare. In questi casi, infatti, è necessario che il sistema salvi lo stato corrente in modo tale da poterlo ripristinare al ritorno della connessione;

NOME	LostConnectionException()
ATTORI PARTECIPANTI	Utente Registrato, Ospite
CASI D'USO ESTESI	Il caso d'uso "LostConnectionException() " estende tutti i casi d'uso in cui durante l'esecuzione del sistema, si ha una perdita di connessione
ENTRY CONDITION	• L'utente sta utilizzando il sistema • Si ha una perdita di connessione
FLOW OF EVENT	1 Il sistema salva il suo stato corrente 2 Il sistema notifica la perdita di connessione all'utente
EXIT CONDITION	L'utente si trova nella pagina che mostra l'errore appena avvenuto.
ERROR MESSAGE	• "LOST CONNECTION EXCEPTION"

3.7.4 ServerFailure()

In seguito ad un fallimento del server e ad una conseguente chiusura inaspettata di quest' ultimo, l'utente viene informato tramite un messaggio di notifica d'errore.

NOME	ServerFailure()
ATTORI PARTECIPANTI	Utente Registrato, Ospite
CASI D'USO ESTESI	Il caso d'uso "ServerFailure() " estende tutti i casi d'uso in cui durante l'esecuzione del sistema, si ha un improvviso fallimento del server
ENTRY CONDITION	• L'utente sta utilizzando il sistema • Si ha un fallimento del server
FLOW OF EVENT	1 Il sistema notifica all'utente il server failure;
EXIT CONDITION	• L'utente si trova nella pagina che mostra l'errore appena avvenuto.
ERROR MESSAGE	"server failure improvviso!"

3.7.5 PersistentDataException()

Durante il caricamento dei dati, il sistema si rende conto di dati persistenti corrotti e, non potendo continuare il caricamento, notifica l'utente tramite messaggio d'errore

NOME	PersistentDataException()
ATTORI PARTECIPANTI	Utente Registrato, Ospite
CASI D'USO ESTESI	Il caso d'uso "PersistentDataException()" estende tutti i casi d'uso in cui durante il caricamento di dati persistenti, di verifica la presenza di dati corrotti
ENTRY CONDITION	• L'utente sta utilizzando il sistema • Il sistema sta caricando dati persistenti
FLOW OF EVENT	1 Il sistema riscontra dei dati persistenti corrotti; 2 Il sistema notifica all'utente l'errore indirivuato;
	• L'utente si trova nella pagina che mostra l'errore appena avvenuto.
ERROR MESSAGE	"DATI PERSISTENTI CORROTTI !"

4. Interfacce dei Sottosistemi

Saranno descritte in dettaglio dopo/durante la fase di Object Design.