Transport Efficiency Manager

SDD: System Design Document

Transport Efficiency Manager

Riferimento	
Versione	1.1
Data	04/12/2020
Destinatario	Prof.ssa F. Ferrucci
Presentato da	Team NC08
Approvato da	

Revision History

Data	Versione	Descrizione	Autori
28/11/2020	0.1	Creazione parte 1 (Introduzione) e parte 2 (Architettura del sistema corrente)	Francesca Moschella, Federica Attianese, Federica Pica
28/11/2020	0.2	Prima stesura	Francesca Moschella, Federica Attianese, Federica Pica
30/11/2020	0.2.1	Aggiornamento, divisione in sottosistemi e schema E-R	Francesca Moschella, Federica Attianese, Federica Pica
03/12/2020	1.0	Modifiche e revisione stesura	Francesca Moschella, Federica Attianese, Federica Pica
04/12/2020	1.1	Revisione ed ultimazione	Francesca Moschella, Federica Attianese, Federica Pica

Indice dei contenuti

1. Introduzione

1.1 Obiettivi del sistema

La creazione di un programma di corse, per una azienda operante nel settore dei trasporti, è una delle funzionalità principali. Un programma di corse può infatti far la differenza se organizzato in modi diversi, ogni combinazione può infatti diversamente influenzare l'andatura dello svolgimento dell'attività di trasporto. La generazione di un programma che permetta di sfruttare al meglio l'asset aziendale in modo da ottenere le massime prestazioni ed i minimi sprechi è appunto la vision di Transport Efficiency Manager. Questo sistema, nato con questo fine, si realizzerà mediante:

- Una piattaforma web che consentirà alle aziende interessate, di creare un più efficiente programma di corse per la loro organizzazione, sia che queste vi accedano da pc che da smartphone o qualunque altro dispositivo.
- Un database relazionale, contente le informazioni necessarie al sistema per gestire, manipolare e lavorare su dati persistenti, quelli delle aziende.
- La gestione dell'autenticazione per gli utenti e per il controllo degli accessi mediante l'inserimento di credenziali, con particolare attenzione alla sicurezza riguardo la protezione dei dati sensibili.

Infine, il sistema sarà strutturato per poter garantire la semplicità d'uso all'utente finale, consentendo una navigazione, presso la piattaforma, agevole ed intuitiva, senza che sia necessaria la consultazione di documentazione specifica.

1.2 Design Goals

La piattaforma TEM si ripropone di realizzare i seguenti Design Goals, rappresentati nella seguente tabella e di cui sono specificate le relative priorità (1= alta priorità, 2= media priorità, 3=bassa priorità), l'identificativo e la categoria di caratterizzazione. Ogni obiettivo presentato riporta l'identificativo del requisito non funzionale da cui è stato originato ed a cui è quindi associato.

Priorità	ID	Descrizione	Categoria	Origine

2	DG_1	**Tempi di risposta**: il sistema deve essere in grado di elaborare le richieste dell'utente e fornire output in meno di 3 secondi.	Performance	RNF_P_1
1	DG_2	**Throughput:** Il sistema deve consentire e supportare l'utilizzo in contemporanea di almeno 100 utenti diversi.	Performance	RNF_P_2:
2	DG_3	**Usabilità**: Il sistema deve essere intuitivamente utilizzabile dall'utente, qualunque sia il suo grado di familiarità con sistemi tecnologici, inoltre, non deve risultare necessaria la consultazione di eventuale documentazione.	End User	RNF_U_1
2	DG_4	**Usabilità**: l'accesso e la fruibilità del sistema deve essere supportata e assicurata anche su diversi dispositivi, che siano mobile o desktop.	End User	RNF_U_2
3	DG_5	**Utilità:** Il sistema deve consentire all'utente di poter fruire delle funzionalità offerte attraverso l'utilizzo di al più 5 passaggi.	End User	RNF_U_3:
1	DG_6	**Security**: Il sistema deve assicurare ai propri utenti una gestione sicura dei propri dati e delle informazioni inserite; mediante una modalità di autenticazione che li protegga non	Dependability	RNF_A_1:

		permettendo accessi non autorizzati.		
2	DG_7	**Modificabilità**: Il sistema deve poter essere modificabile e se necessario, corretto da parte di altri sviluppatori.	Maintenance	RNF_S_1 RNF_S_3
2	DG_8	**Estensibilità:** *Il* sistema prodotto deve permettere l'estensione o l'aggiunta di funzionalità.	Maintenance	RNF_S_2
1	DG_9	**Robustness**: Il sistema deve resistere a scenari di inconsistenza dei dati e delle informazioni, attraverso il filtraggio dell'input inserito dall'utente.	Dependability	RNF_A_3 RNF_A_2
1	DG_10	**Fault tolerance:** Il sistema deve essere in grado di gestire nella maniera migliore eventuali situazioni di criticità, come problemi dovuti alla rete o tecnici (sovraccarico del database).	Dependability	
2	DG_11	**Costi di sviluppo:** La creazione della piattaforma richiederà costi ridotti sia in termini di risorse umane, sia in termini economici.	Cost	
2	DG_12	**Costi di aggiornamento o manutenzione**: I costi di aggiornamento e/o	Cost	

		manutenzione saranno stabiliti nel momento in cui saranno necessari interventi del genere.		

1.3 Priorità dei Design goal

Per il sistema sviluppato i criteri di dependability sono prioritari, a seguire sono ritenuti importati i criteri di performance, cost, maintenance e i criteri end user.

1.4 Trade-offs

Funzionalità vs. Usabilità

Il sistema mira ad essere intuitivo da utilizzare, user-friendly, qualunque sia il grado di familiarità con tecnologie del genere dell'utente, consentendo una navigazione agevole ed il facile utilizzo di tutte le funzionalità offerte anche senza dover consultare la documentazione.

Tempo di rilascio vs Funzionalità

Il raggiungimento della realizzazione della completezza e totalità delle funzionalità offerte dal sistema prevale su eventuali tempistiche stringenti, si dà priorità quindi alla consegna di un prodotto funzionante e che rispetti quanto previsto nella fase di progettazione.

Prestazioni vs Costi

La garanzia del rispetto del budget prefissato per la realizzazione del sistema prevale sull'integrazione di prestazioni superflue o non particolarmente necessarie, assicurando al cliente di rientrare nei costi senza dover scendere, allo stesso tempo, a compromessi troppo rigidi.

Costi vs Affidabilità

I dati gestiti dal sistema sono sensibili, pertanto la garanzia del loro sicuro trattamento e del rigoroso controllo di input e consistenza, rappresenta la parte del sistema per cui, anche nell'eventualità di tagli dovuti ad una possibile ristrettezza di budget, non si risparmierà e su cui non si cercheranno soluzioni più convenienti a discapito dell'affidabilità.

.

1.5 Definizioni, acronimi e abbreviazioni

TEM: Transport Efficiency Manager

DG: Design goal

RF: Requisito funzionale

RNF: Requisito non funzionale

UC: Use Case

GS: Gestione server

1.6 Riferimenti

- Documentazione reperita tramite ricerche online.
- Materiale relativo al corso di Ingegneria del Software: System Design, System Design-parte2.
- Requisiti funzionali: Sezione 3.2 del RAD.
- Requisiti non funzionali: Sezione 3.3 del RAD.

1.7 Overview

Il secondo punto del documento presenta il sistema corrente.
Il terzo punto presenta l'architettura del sistema proposto, la decomposizione in sottosistemi, il mapping hardware/software, i dati persistenti, il controllo degli accessi e sicurezza, il controllo del flusso globale del sistema e le condizioni limite.
Al quarto punto verranno presentati i servizi dei sottosistemi.

2. Architettura del sistema corrente

La piattaforma TEM nasce per offrire una soluzione alternativa e più efficiente, al metodo di organizzazione delle corse utilizzato al momento dalle aziende nel campo dei servizi di trasporto. Attualmente per la creazione di un programma di corse le variabili di scelta prese in considerazione non si rifanno a precisi criteri; l'organizzazione dei diversi mezzi o conducenti non è strutturata quindi tenendo conto di possibili fermate più affollate di altre o di giornate con più traffico durante determinati viaggi o tratte, ma tutto è combinato "casualmente". Se infatti un mezzo non permette a tutti gli utenti di effettuare la tratta, a causa di mancanza di posti disponibili, questi sono costretti a dover aspettare la corsa successiva, o a trovare altre alternative, per arrivare addirittura a non avere la possibilità di usufruire del tutto del servizio di trasporto. Scenari come questi sono familiari a pendolari, ma soprattutto agli studenti, prevalentemente utenti abbonati, che non possono fruire di un servizio per cui hanno già pagato e che quindi gli dovrebbe essere assicurato.
Il normale svolgimento di una corsa, in scenari che presentano situazioni di disagio come la sopra nominata indisponibilità di posti a sedere, per la mole di utenti alle fermate, prevede quindi che il controllore permetta ai passeggeri di salire fino a completare la capienza del bus e lasci i restanti utenti alla fermata, che impieghi del tempo a quella stessa fermata per gestire il sovraffollamento e che quindi tardi l'ora di arrivo a destinazione per quella corsa. Solo il passare del tempo, lo svolgimento di corse ed i conseguenti feedback degli autisti, che ormai hanno acquisito abitudinarietà svolgendo le stesse tratte settimanalmente, portano ad una modifica dell'organizzazione di una determinata corsa, processo che può richiedere mesi.

3.Architettura del sistema proposto

3.1 Panoramica

Il sistema che proponiamo è una piattaforma web che permetta:

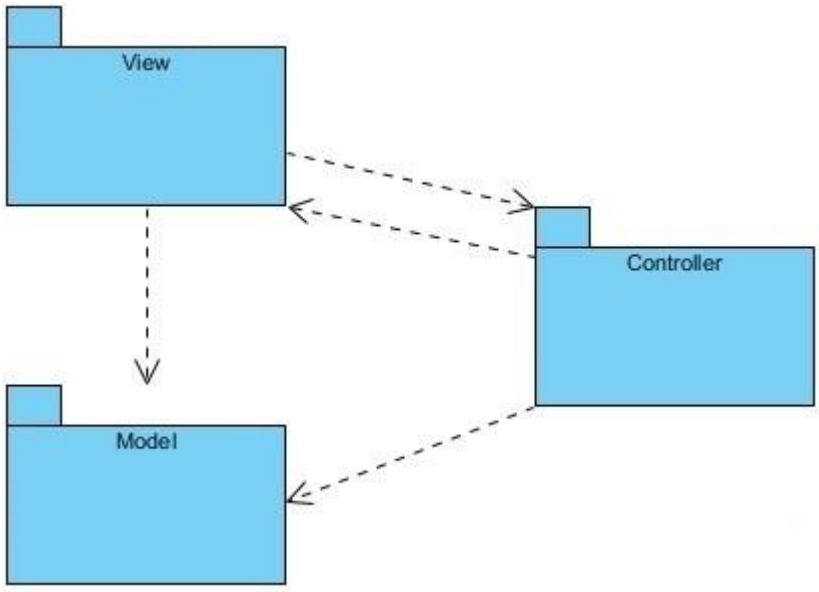
1. L'autenticazione dell'azienda e di un admin preposto alla gestione della piattaforma.
2. La possibilità di due diversi tipi di generazione del programma di corse; uno manuale ed uno automatico sulla base di previsioni in base a dati empirici.
3. L'opportunità, da parte dell'azienda, di poter inserire e consultare le risorse disponibili.
4. La possibilità di poter consultare e modificare comodamente il programma di corse in ogni momento sulla piattaforma.

Nei punti successivi saranno trattate nel dettaglio le restanti varie fasi del System design.

3.2 Decomposizione in sottosistemi

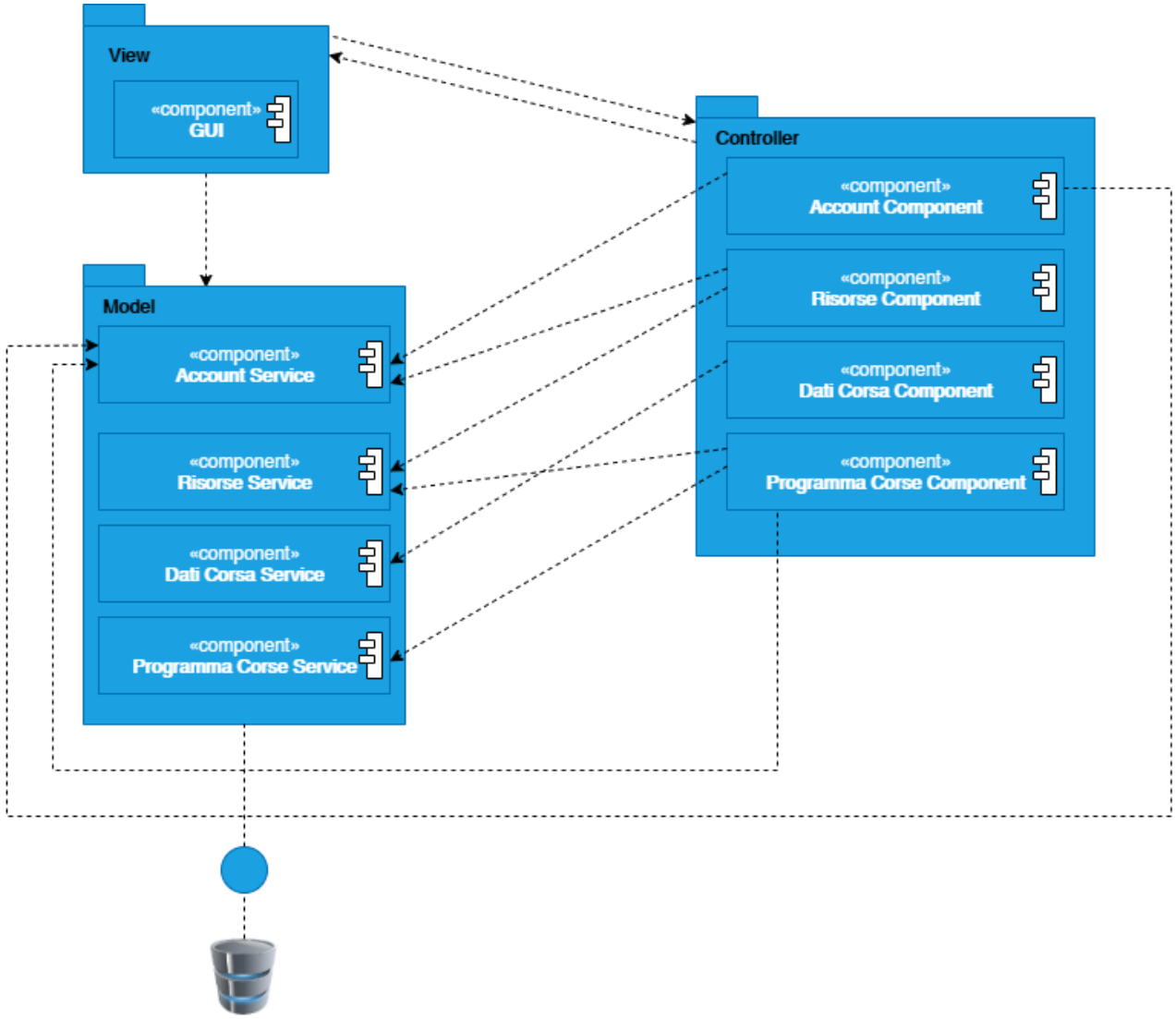
La decomposizione prevista per questo sistema è formata da tre layer che si occupano di gestire aspetti e funzionalità differenti. Nello specifico, l'architettura utilizzata segue lo stile Model/View/Controller, che distingue tre tipi di sottosistemi:

- Sottosistema View: gestisce l'interfaccia grafica e gli eventi generati dall'utente, mostrandogli, se necessario, gli oggetti del dominio
- Sottosistema Model: gestisce i dati e i metodi per accedervi
- Sottosistema Controller: gestisce la sequenza di interazioni con l'utente e della logica del sistema.



Questa architettura è stata scelta perché, nonostante non sia minimo il coupling tra il sottosistema Model che mantiene la conoscenza del dominio e gli altri due, ogni sottosistema ha delle responsabilità ben definite; questo rende più agevole la manutenibilità in quanto, trovandoci di fronte ad un sistema interattivo che utilizza view multiple per ogni modello, si avrà necessità di modificare le interfacce utente più spesso rispetto ai dati del dominio.

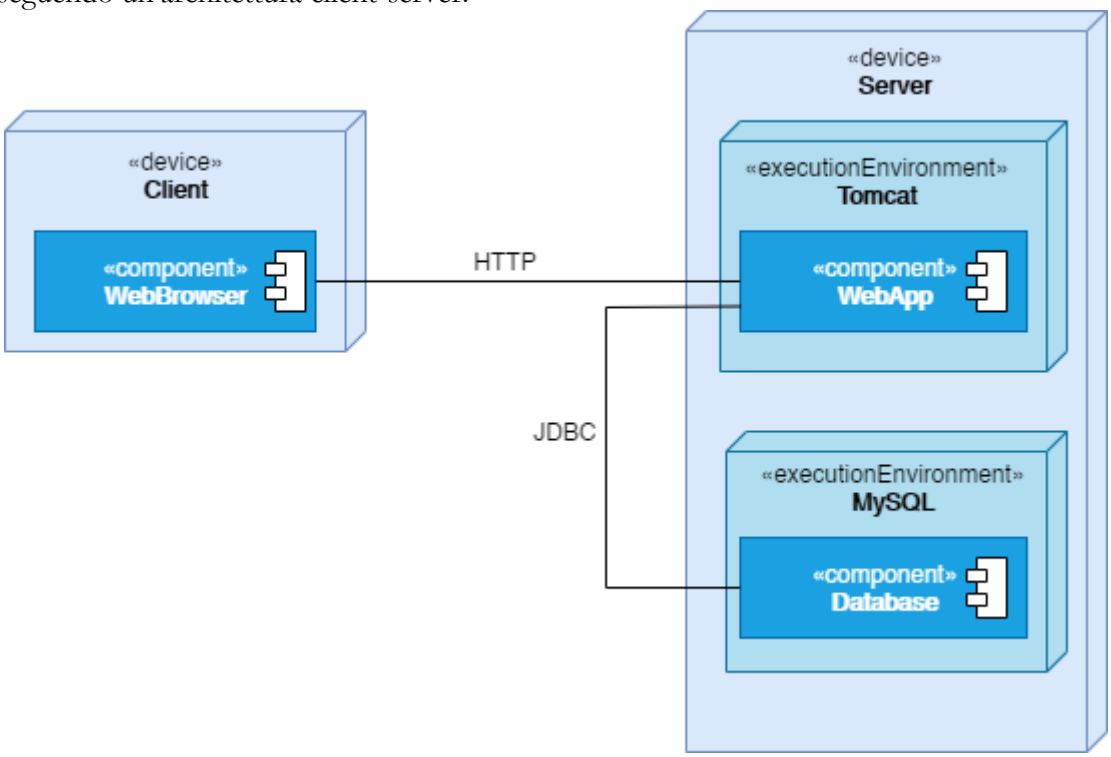
3.2.1 Decomposizione in sottosistemi



3.3 Mapping hardware/software

TEM consiste in un'applicazione web-based installabile su qualsiasi server capace di eseguire Java e MySQL.

L'app interagisce con un database relazionale, MySQL. Dato il basso numero di dati gestiti e di utenti che utilizzeranno il sito, si è scelto di installare tutte le componenti sulla stessa macchina, seguendo un'architettura client-server.



Il sistema che verrà realizzato si basa su un'architettura Web-based:

- **Protocollo richiesto**: HTTP
- **Memorizzazione dei dati**: MySQL
- **WebServer**: Apache Tomcat
- **Tecnologie utilizzate**: Javascript, CSS3, HTML5
- **Framework necessari**: Spring Boot MVC

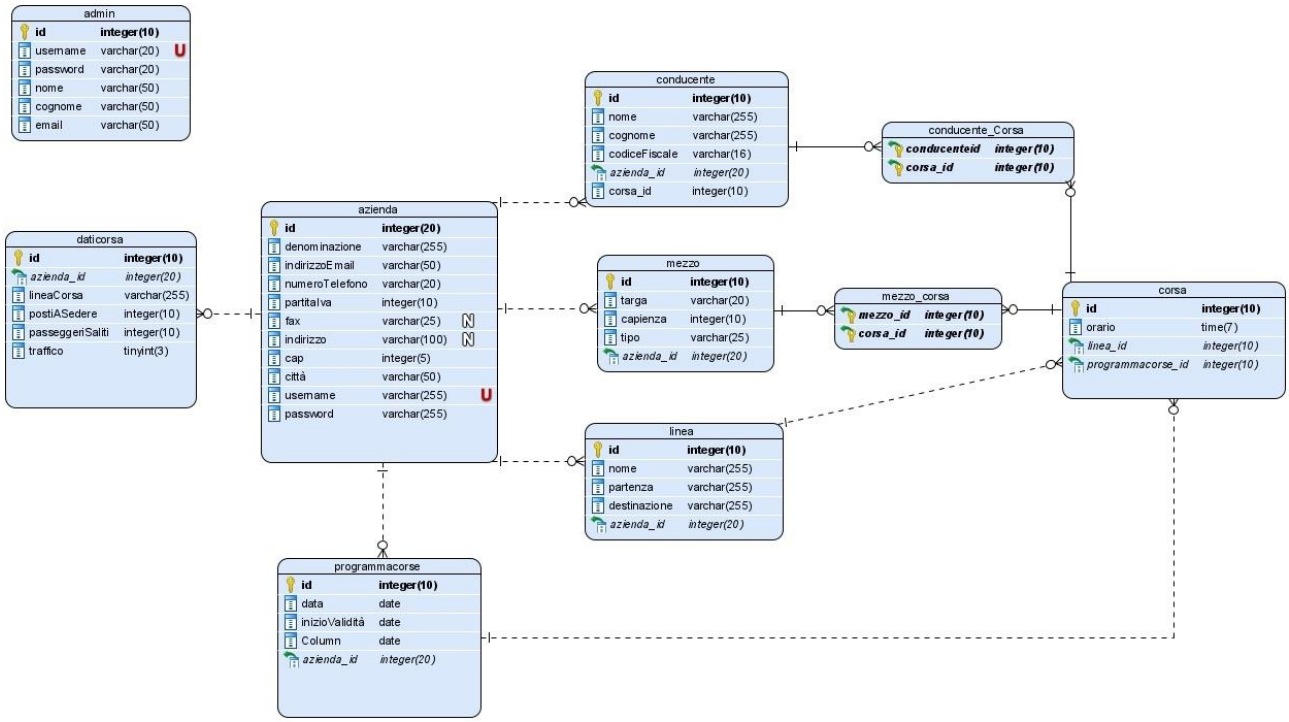
Il sistema sarà accessibile attraverso browser web (lato Client) installati sui dispositivi degli utenti.
L'app (lato Server) si suddivide in due componenti principali:

1. **Webapp**, che rappresenta il core dell'applicazione, a cui saranno allocati i sottosistemi della decomposizione precedente, cioè Model, View, Controller;
2. **Database**, che gestisce la persistenza dei dati.

Il sistema dev'essere installato su una macchina in grado di supportare Apache Tomcat, in modo da garantire il funzionamento e l'operabilità della Webapp, e MySQL per garantire l'operabilità del database con cui l'app si interfaccia.
Le dipendenze sono gestite tramite il gestore pacchetti **Maven**.

3.4 Gestione dati persistenti

Per la gestione dei dati persistenti, TEM si affida ad un DBMS, gestito tramite MySQL. La struttura interna del database segue il seguente schema:



3.5 Controllo degli accessi e sicurezza

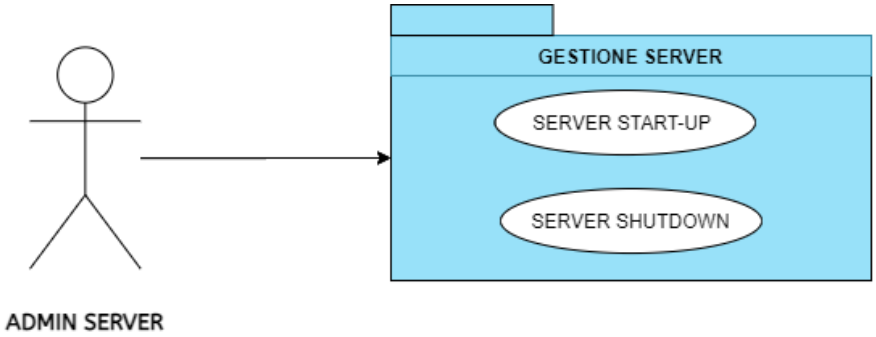
Il controllo degli accessi è garantito tramite l'utilizzo di username (in questo caso l'username è l'e-mail dell'azienda) e password, che verranno richieste per ogni singolo accesso.
In una fase di sviluppo iniziale, non sarà prevista una crittografia dei dati all'interno del database, pertanto le password saranno salvate in chiaro.
La matrice di accesso che segue modella i diritti di accesso su una classe. In particolare, è stata usata una matrice di tipo Capability, in cui, ad ogni tupla (classe, operazione) si associano uno o più attori che possono effettuare quella specifica operazione su quella specifica classe.

	Risorse	Account	DatiCorsa	ProgrammaCorse
Inserimento	Azienda	Admin	Azienda	
Creazione				Azienda
Modifica	Azienda	Admin	Azienda	Azienda
Consultazione	Azienda, Admin	Admin	Azienda, Admin	Azienda, Admin
Cancellazione	Azienda	Admin	Azienda	Azienda
Registrazione		Ospite		

3.6 Controllo flusso globale del sistema

Il controllo di flusso globale adottato è di tipo thread-driven, in quanto Apache Tomcat è in grado di gestire in maniera concorrente l'interazione tra la webapp e più clients. Più specificamente, ogni richiesta da parte di un utente genera un thread dedicato, attraverso il quale essa sarà gestita.

3.7 Condizione limite



3.7.1 Start-up sistema

Identificativo UC_GS1	**START-UP**	**Data**	**30/11/2020**
		Vers.	*0.00.001*
		Autore	*Team **F** ³ (NC08)*
Descrizione	Lo UC fornisce definisce la funzionalità di avvio del sistema per il gestore del server.		
Attore Principale	**Gestore/admin del server** È interessato ad avviare il sistema.		
Attori secondari	NA		
Entry Condition	Il gestore ha accesso alla macchina su cui è installato il sistema.		
Exit condition On success	L'avvio del sistema è stato effettuato con successo.		
Exit condition On failure	Il sistema non è stato avviato.		
Rilevanza/User Priority	Elevata		
Frequenza stimata	1 uso/anno		
Extension point	NA		
Generalization of	NA		
Flusso di Eventi Principale/Main Scenario			
1	Gestore del server:	Accende il server	
2	Gestore del server:	Lancia il servizio del DBMS ed il web container tramite gli appositi comandi.	
3	Sistema:	Comunica al gestore che lo start-up è avvenuto con successo	
Scenario/Flusso di eventi di ERRORE: non è possibile avviare il sistema			
3.1	Sistema:	Mostra al gestore del server un messaggio che spiega il motivo dell'insuccesso.	
3.2	Sistema	Termina con un insuccesso.	

3.7.2 Shutdown del sistema

Identificativo UC_GS2	**SHUTDOWN**	**Data**	**30/11/2020**
		Vers.	*0.00.001*
		Autore	*Team **F** ³ (NC08)*
Descrizione	Lo UC fornisce definisce la funzionalità di terminazione del sistema per il gestore del server.		
Attore Principale	**Gestore/admin del server** E' interessato a terminare il sistema.		
Attori secondari	NA		
Entry Condition	Il gestore ha accesso alla macchina su cui è installato il sistema.		
Exit condition On success	Il sistema è terminato correttamente.		
Exit condition On failure	Il sistema non è stato terminato.		
Rilevanza/User Priority	Elevata		
Frequenza stimata	1 uso/anno		
Extension point	NA		
Generalization of	NA		

Flusso di Eventi Principale/Main Scenario		
1	Gestore del server:	Termina il servizio del web container chiudendo la shell in cui questo è stato avviato.
2	Sistema:	Comunica al gestore che il servizio è stato terminato con successo.
3	Gestore del server:	Termina il servizio del DBMS tramite l'apposito comando.
4	Sistema:	Comunica al gestore che il servizio è stato terminato correttamente
Scenario/Flusso di eventi di ERRORE: non è possibile terminare il servizio		
3.1	Sistema:	Mostra al gestore del server un messaggio che spiega il motivo dell'insuccesso.
3.2	Sistema	Termina con un insuccesso.

4. Subsystem services

4.1 Model

Sottosistema	Risorse Service
Descrizione Sottosistema	Sottosistema che gestisce le operazioni relative alle risorse dell'azienda.

Sottosistema	DatiCorsa Service
Descrizione Sottosistema	Sottosistema che gestisce le operazioni relative ai dati relativi alle corse effettuare.

Sottosistema	ProgrammaCorse Service
Descrizione Sottosistema	Sottosistema che gestisce le operazioni relative alla generazione del programma di corse.

Sottosistema	Account Service
Descrizione Sottosistema	Sottosistema che gestisce le operazioni relative al servizio di autenticazione e gestione dell'account personale.

4.2 View

Sottosistema	GUI
Descrizione Sottosistema	Sottosistema che gestisce l'interfaccia grafica di tutti i servizi.

4.3 Controller

Sottosistema	Risorse Component
Descrizione Sottosistema	Sottosistema che gestisce la logica delle operazioni relative ai servizi di inserimento dei dati relativi alle risorse dell'azienda.

Sottosistema	DatiCorsa Component
Descrizione Sottosistema	Sottosistema che gestisce la logica delle operazioni relative ai servizi di inserimento dei dati delle corse effettuate.

Sottosistema	ProgrammaCorse Component
Descrizione Sottosistema	Sottosistema che gestisce la logica delle operazioni relative ai servizi di creazione del programma di corse.

Sottosistema	Account Component

Descrizione Sottosistema	Sottosistema che gestisce la logica delle operazioni relative al servizio di autenticazione e gestione dell'account personale.

5.Glossario

Java: linguaggio di programmazione ad alto livello orientato agli oggetti

Applicazione web: programma accessibile tramite browser web ed in grado di elaborare richieste e risposte

Throughput: misura della capacità del sistema di condurre task contemporanei

Server: macchina connessa alla rete dotata di un ambiente di esecuzione

Application Server: sistema software per la gestione delle richieste/risposte provenienti dai client

Tomcat: server web open source sviluppato dalla Apache Software Foundation

DBMS: sistema software per la gestione dei dati persistenti su database.

MySQL: relational database management system composto da un client a riga di comando e un server.

Mvc: Model-view-controller, pattern architetturale per lo sviluppo di sistemi software.

CSS3: terza versione di CSS; è un linguaggio di programmazione web utilizzato per descrivere l'aspetto e la formattazione di un sito web al browser lato client.

HTML5: linguaggio di markup per la strutturazione delle pagine web.

HTTP: protocollo a livello applicativo usato come principale sistema per la trasmissione d'informazioni sul web.

Spring: framework open source per lo sviluppo di applicazioni su piattaforma Java.