Università degli Studi di Salerno
Corso di Ingegneria del Software

BeVoyager
SDD
Versione 2.3

Data: 09/01/2017

Partecipanti:

Nome	Matricola
Donato Tiano	0512102916
Alessandro Longobardi	0512102910
Paolo Zirpoli	0512102862
Salvatore Ruggiero	0512103002

Revision History

Data	Versione	Descrizione	Autore
07/01/2017	1.0	Introduzione, purpose of the system.	
07/01/2017	1.1	Design goals.	
07/01/2017	1.2	Definitions, acronyms and abbreviations.	
07/01/2017	1.3	Current Software Architecture.	
07/01/2017	1.4	Proposal Software Architecture: introduction.	
08/01/2017	1.5	Hardware/Software Mapping.	
08/01/2017	1.6	Subsystem Decomposition (parziale) & Persistent Data Management.	
08/01/2017	1.7	Access Control and Security.	
09/01/2017	1.8	Subsystem Decomposition.	
09/01/2017	1.9	Design Pattern and Global Software Control.	
09/01/2017	2.0	Modifiche Access Control and Security.	
09/01/2017	2.1	Boundary Conditions.	
09/01/2017	2.2	Subsytems Services (part1).	
09/01/2017	2.3	Subsytems Services.	

Sommario

1. Introduction

Il progetto che si vuole proporre è una piattaforma di viaggi, chiamata BeVoyager, la quale nasce dall'idea di un sito all-in-one, cioè una piattaforma dove sia possibile ritrovare tutto quello che all'utente serve per poter organizzare un viaggio in maniera più efficiente. Sul sito saranno unite tutte le comuni ricerche che servono all'organizzazione del viaggio: dalla formazione di un gruppo di persone alla creazione di un luogo, dalla ricerca di itinerari ai feedback su utenti, luoghi e itinerari.

1.1 Purpose of the system

Lo sviluppo di tale progetto è stato pensato al fine di fornire all'utente la possibilità di organizzare viaggi e trovare informazioni su di essi. BeVoyager permetterà all'utente di:

- creare viaggi pubblici, dove ogni iscritto alla piattaforma potrà partecipare e formare un gruppo di persone interessate allo stesso itinerario;
- creare viaggi privati, dove sarà lo stesso utente ad isrivere le persone a tale viaggio in modo da creare un gruppo chiuso di persone conosciute e scegliere insieme l'itinerario di cui il viaggio sarà composto.
- diventare un Tour Operator, ovvero utilizzare la piattaforma per la creazione di viaggi non modificabili da rendere disponibili agli utenti. Dalla vendita di tali viaggi, il Tour Operator ne guadagnerà la percentuale pervenuta.
- lasciare feedback, su persone, itinerari e luoghi, in modo tale da facilitare le ricerche da parte di altre persone sull'utilizzo di un itinerario, sulla scelta di un luogo e sull'aggiunta o meno di un altro utente all'organizzazione di un viaggio.

1.2 Design goals

1.2.1 Criteri di sviluppo e mantenimento

Estensibilità:

- Il sistema verrà progettato in modo tale da facilitare l'introduzione di nuove funzionalità per l'utente. Essendo una web application, verranno usati i linguaggi di scripting come HTML, Javascript, ed il back-end sarà implementato in Java.

Numero minimo di errori:

- Il sistema verrà implementato dando particolare attenzione a minimizzare l'occorrenza di errori durante l'utilizzo.

Interfacciamento con oggetti già esistenti:

- Il sistema deve trattare componenti off-the-shelf come l'utilizzo di librerie Java già esistenti.

Velocità di implementazione:

- Il sistema verrà progettato in maniera tale da massimizzare la velocità di implementazione dello stesso, cercando di garantire uno sviluppo che proceda senza difficoltà

Mantenibilità:

- Il codice sorgente deve essere scritto e documentato con attenzione, permettendo a nuovi programmatori che verranno incaricati di mantenerlo di poter familiarizzare con lo stesso in un periodo di tempo rapido.

Portabilità:

- La piattaforma deve essere aperta all'aggiunta di funzionalità appartenenti ad altre piattaforme. In pratica si deve fare in modo che non venga modificata in maniera lenta nel caso in cui un'altra piattaforma ne richiede l'uso.
- Il sistema deve utilizzare funzionalità esterne in modo tale da non modificare quelle già esistenti. In pratica deve interfacciarsi a sistemi diversi senza subire modifiche a basso livello (codice).

1.2.2 Criteri di affidabilità

Robustezza:

- Il sistema garantisce un funzionamento adeguato anche se vengono inseriti input non validi (fault-tolerance).
- Il sistema deve saper gestire errori che possono verificarsi durante l'uso quotidiano da parte dell'utente.

Affidabilità:

- Il sistema deve garantire che le operazioni effettuate dagli utenti vadano a buon fine rispettando le aspettative degli stessi e soprattutto deve essere coerente con le scelte fatte da essi, in quanto le scelte dell'utenza non devono essere altera da un funzionamento

inadeguato.

Disponibilità:

- Essendo il sistema pensato per supportare gli utenti anche durante il viaggio che hanno programmato tramite la piattaforma, si presenta la necessità di rendere il sistema disponibile in maniera continuativa. Bisogna quindi minimizzare i tempi di downtime in caso di manutenzione.

Sicurezza:

- Il sistema deve garantire la sicurezza dei dati inviati dall'utente alla piattaforma. Gli utenti della piattaforma forniranno i loro dati sensibili, che devono essere trattati in maniera congrua, per non permettere possibili furti di dati.

Accesso:

- Il sistema deve gestire l'accesso dell'utenza tramite l'utilizzo di credenziali che serviranno al riconoscimento del singolo utente ed al reindirizzamento al profile personale.
- Il sistema non deve consentire operazioni fatte da utenti a cui non è consentito l'accesso. Deve fornire i permessi solo a coloro che in generale possono effettuare un'operazione che può modificare lo stato di una singola entità.

1.2.3 Criteri utente finale

Tracciabilità:

- Il sistema deve essere in grado di soddisfare tutti i requisiti definiti in fase precedente In pratica le funzionalità del sistema devono essere compatibili con quelle desiderate inizialmente.

Usabilità:

- Il sistema aiuterà l'utente in ogni sua scelta, guidandolo al raggiungimento del proprio obiettivo.
- L'interfaccia utente sarà di facile comprensione, per poter garantire un utilizzo semplificato.

Facilità di riadozione:

- Il design della piattaforma deve permettere all'utente di riprendere da dove aveva interrotto le sue operazioni anche dopo un periodo di assenza dalla piattaforma.

1.2.4 Criteri di performance

Efficienza:

- Il sistema viene progettato in maniera tale da poter fornire risultati rapidamente all'utente. Tempi di risposta accettabili sono nell'ordine di pochi secondi, tenendo conto che la velocità della connessione di ogni utente giocherà un ruolo molto importante nella visualizzazione dei risultati.
- Il sistema dovrà effettuare in tempi accettabili le varie operazioni a runtime nascoste all'occhio dell'utente, ovvero quelle fornite dal server, in modo da ridurre ulteriormente il tempo di risposta prestabilito.

Utilizzo di memoria:

- Il sistema utilizza un database relazionale basato su MySQL. Tutti i dati persistenti verranno salvati in tale database. C'è ovviamente la necessità di garantire uno spazio di archiviazione sufficiente per gestire un numero elevato di accessi e dati.
- L'operazione di lettura dal database deve avere tempi bassi in modo tale da poter gestire i dati in esso velocemente.

1.2.5 Design Pattern

Abstract Factory: Il pattern verrà utilizzato per interfacciare il sistema con componenti già esistenti, quali JDBC e varie classi di servizio.

Bridge: Lo utilizzeremo per la connettività al database e per la mappatura tra boundary e control object

Facade: Utilizzato per creare l'interfaccia utilizzata dai vari sottosistemi

Command pattern: Gestiamo il controllo di un evento tramite oggetti, in modo tale da poter separare la gestione del comando dalla sua origine.

Observer: Viene utilizzato soprattutto nella gestione di notifiche indirizzate ad un utente.

Adapter: Utilizzato per accomunare le operazioni tra Tour Operator e Utente.

MVC: Utilizzato per dividere la logica di business con le interfacce user-friendly.

1.3 Definitions, acronyms and abbreviations

Acronimo	Definizione
SQL	**Structured Query Language, linguaggio usato per interagire col database**
MySQL	DBMS scelto per l'implementazione di BeVoyager
DB	Il database sul quale vengono memorizzati i dati
Java	Linguaggio di programmazione OOP in cui è scritto BeVoyager
Javascript	Linguaggio di scripting per web
HTML	Linguaggio di markup per la strutturazione di pagine web
RAD	Requirements Analisys Document
SDD	Software Design Document
ODD	Object Design Document

2. Current Software Architecture

Dopo aver effettuato varie ricerche sul web, appare evidente che non esistono siti la cui funzione sia simile a quella di BeVoyager. In genere, un sito offre la viasualizzazione di informazioni su luoghi ed eventi di interesse, ma non permette di creare un itinerario personalizzabile, e sopratutto non è stata mai rilevata la funzione di poter creare "viaggi", aggiungendo amici e partecipanti in modo tale da coordinare l'organizzazione e la scelta delle mete.
Quindi, in pratica, a parte qualche funzionalità basilare, non ci sono sistemi già realizzati su cui BeVoyager possa risultare basato.

3. Proposed Software Architecture

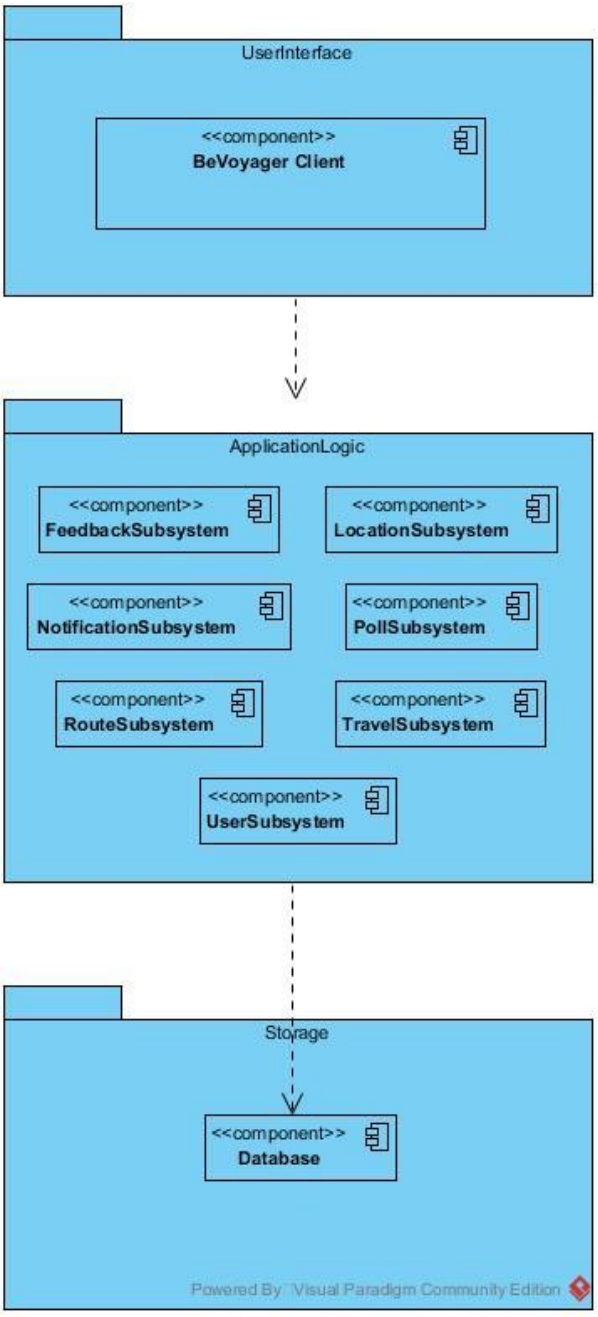
3.1 Introduction

Nella fase di analisi del sistema, è stata individuata una prima divisione logica del sistema proposto, ed alcune delle relazioni tra i sottosistemi trovati.
Per la struttura del sistema, essendo una web application, si è ritenuto opportuno utilizzare lo stile architetturale Client-Server, per poter semplificare le operazioni logiche del sistema.
Il client, eseguito in locale sulla macchina personale dell'utente, comunica esclusivamente con il server, del quale utilizza i servizi offerti per portare a termine le operazioni scelte.
Il server si occupa di gestire la logica dell'applicazione e di memorizzare i dati sul database.
Per la memorizzazione dei dati è ovviamente necessario l'utilizzo di un sistema per la gestione dei dati persistenti a lungo termine. Si è deciso di optare, quindi, per un database relazionale, che permette di gestire le varie operazioni sui dati in maniera corretta ed efficace.
I servizi offerti dal database sono accessibili solamente al server.

3.2 Subsystem decomposition

Il Sistema è stato suddiviso in sottosistemi in modo tale da sezionare entità e operazioni. Tale suddivisione comporta l'esplicazione delle interazioni tra le varie parti.

Di seguito il diagramma della decomposizione in sottosistemi:



Ogni sottosistema offre dei servizi specifici riguardanti il concetto relativo al dominio dell'applicazione che astrae.

Per quanto riguarda le relazioni tra i sottosistemi, ogni sottosistema può interagire con il Database attraverso il Manager contenuto in esso. L'UserSubsystem è in grado di relazionarsi con ogni altro sottosistema, ad esempio UserManager chiama TravelSubsystem per la creazione di un viaggio, RouteSybsystem per la creazione di un itinerari, LocationSubsystem per la creazione di un luogo, NotificationSubsystem per 'invio di una notifica, FeedbackNotification per l'invio di un feedback e PollSubsystem per la Creazione di un sondaggio. TravelSubsystem chiama RouteSubsystem che, a sua volta, chiama LocationSubsystem. PollSubsystem ha la necessità di sapere a quale luogo di un viaggio è legato. FeedbackSubsystem si relazione con un utente, un luogo o un itinerario.

3.3 Hardware/Software Mapping

BeVoyager è un sistema che si basa su una piattaforma web. Essendo tale, è naturalmente divisibile in più nodi che gestiscono le operazioni. Nel nostro caso, abbiamo deciso di suddividere il sistema in due nodi di tipo hardware:

- Dispositivo dell'utente (computer, laptop, smartphone): esso presenta l'interfaccia utente che permette di utilizzare i servizi offerti dalla piattaforma.
- Il Server: esegue la logica dell'applicazione e memorizza i dati in maniera persistente.

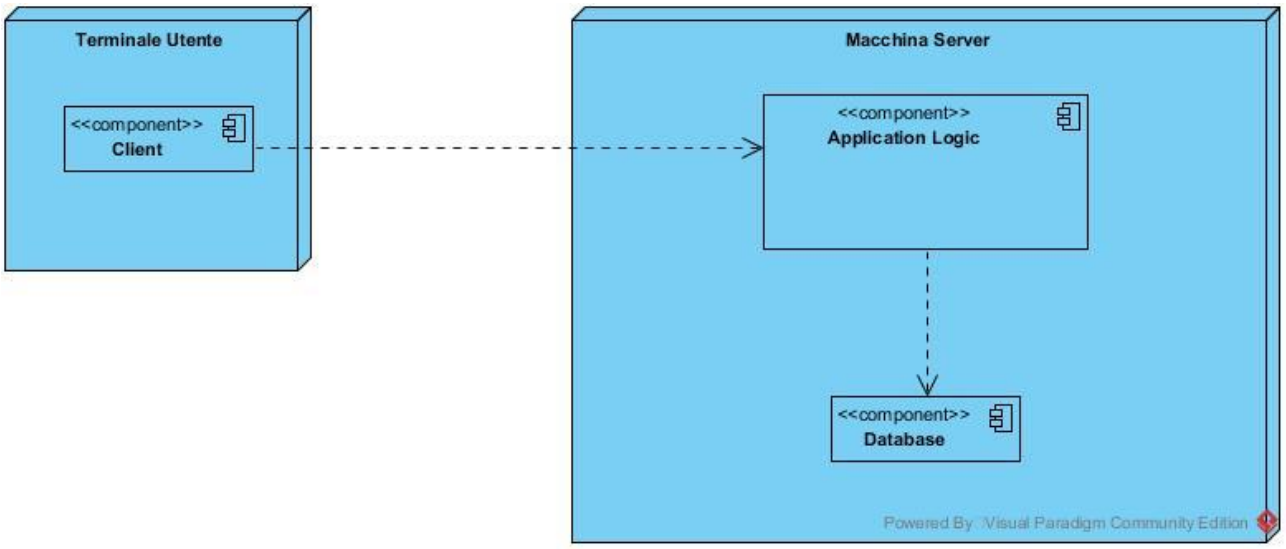
Il dispositivo utente può avere svariate configurazioni hardware (differente quantità di memoria di massa o RAM, differente CPU, etc…). Di questo viene tenuto conto in quanto l'applicazione web viene sviluppata in modo da non richiedere risorse eccessive all'utente. Per ottenere buone prestazioni, l'implementazione del client è effettuata con le tecnologie web più comuni, quali HTML, CSS, Javascript.

Il server ha invece configurazione hardware fissa, quindi è possibile configurarlo in partenza in base alle risorse richieste dall'applicazione e dal database. Detto ciò, il server ha comunque bisogno di essere di veloce esecuzione per poter rispettare i criteri di efficienza descritti precedentemente.
La logica del server dell'applicazione verrà implementata in Java, usando le tecnologie Servlet e JSP. Come webserver verrà utilizzato Tomcat. Il database viene invece implementato utilizzando un

database relazionale. Come DBMS abbiamo scelto MySQL, e le query utilizzate per interagire con il database verranno scritte in SQL.

In presenza di un singolo processore, il tasso di calcolo richiesto dall'operazione non pesa sulle prestazioni di tale processore.
Le operazioni di interazione tra utente e database non inficiano sull'efficienza del processore.
La gran parte del sistema proposto non sarà programmato in maniera distribuita. Logicamente, l'utilizzo della programmazione distribuita aumenterebbe le prestazioni della piattaforma. Si utilizzeranno le "session" per migliorare le sue prestazioni.
La memoria attribuita al sistema dal punto di vista software sarà tale da gestire molteplici richieste al database. Saranno necessari controlli sull'accesso ai server in modo da non incontrare problemi di inconsistenza di dati.
Le operazioni di input-output sono gestite da i Manager che fungono da intermediari tra l'User Interface (parte vista dall'utente tramite Gui) e il server del sistema. La dinamicità di tali operazioni non pesa sulla larghezza di banda disponibile per la comunicazione tra hardware connesso e operazioni server. L'interazione tra i vari sottosistemi sarà gestita da software tramite chiamate tra i Manager e Classi interne.
Il tempo di risposta di operazioni input-output dovrà essere quanto più piccolo possibile, in quanto il sistema interagisce dinamicamente con l'utente che potrà indirizzare al software nuove funzioni in input o visualizzare l'output finale della sua operazione.

Di seguito il *Deployment Diagram*:



3.4 Persistent Data Management

Il sistema utilizza un database relazionale per semplificare la gestione dei dati persistenti e le operazioni da eseguire su di essi.

Di seguito sono elencate le tabelle che rappresentano i dati da memorizzare sul database.

Tabella	Colonna
RegisteredUser	id (Long) P.K. username (String) UNIQUE name (String) lastName (String) email (String) UNIQUE password (String) birthDate (String) age (Short)
SystemAdmin	id (Long) P.K. email (String) UNIQUE password (String)
TourOperator	id (Long) P.K. package (String) earnings (Double) purchaseDate (String) expired (Boolean)
Travel	id (Long) P.K. startDate (String) endDate (String) routeID (Long) F.K. TO Route destination (String) creatorID (Long) F.K. TO RegisteredUser Type (Boolean)
UserTravelMatch	(partecipantID (Long) F.K. TO RegisteredUser,

	travelID (Long) F.K. TO Travel) P.K. entryDate (String)
Route	id (Long) P.K. description (String)
Location	id (Long) P.K. description (String)
RouteLocationMatch	(locationID (Long) F.K. TO Location, routeID (Long) F.K. TO Route) P.K. entryDate (String)
Poll	id (Long) P.K. description (String) positive (Integer) negative (Integer) date (String) idRoute (Long) F.K. TO Route idLocation (Long) F.K. TO Location
FeedbackUser	id (Long) P.K. senderID (Long) F.K. TO RegisteredUser recipientID (Long) F.K.TO RegisteredUser message (String) sendDate (String) sendTime (String)
FeedbackRoute	id (Long) P.K. senderID (Long) F.K. TO RegisteredUser routeID (Long) F.K.TO Route message (String) sendDate (String) sendTime (String)

FeedbackLocation	id (Long) P.K. senderID (Long) F.K. TO RegisteredUser locationID (Long) F.K.TO Location message (String) sendDate (String) sendTime (String)
Notification	(id (Long), senderID (Long) F.K. TO RegisteredUser recipient (Long) F.K. TO RegisteredUser) P.K. body (String) sendDate (String)

I dati memorizzati devono essere permanenti ed accessibili in qualsiasi momento si verifichi una richiesta da parte della logica dell'applicazione. Le varie entità sono individuabili grazie a degli ID univoci che vengono dati in input alle operazioni.

Si è deciso di creare tre differenti tipi di feedback per i seguenti motivi:

1. Efficienza nelle operazioni di lettura, facilitata dalla distinzione tra le varie categorie di feedback. Avere tabelle separate è una situazione ottimale per diminuire il numero di letture da fare rispetto a se ci trovassimo di fronte ad un'unica tabella.
2. La distinzione in tre tabelle rispetto ad averne anche una comune è utile per la riduzione di spazi.
3. Tre tabelle diverse fa sì che non ci siano query complesse sullo stesso elemento feedback, ma solo interrogazioni semplici.

3.5 Access Control and Security

BeVoyager supporta tre categorie di utenti: Utente non registrato, Utente Registrato, TourOperator. In più è prevista l'inclusone di un System Admin per la gestione amministrativa della piattaforma.

Un utente non registrato ha solo funzionalità di visualizzazione sulla piattaforma; può effettuare ricerche e visualizzare dati di base.

Un utente registrato, che cioè possiede un account su BeVoyager presente nel database, può invece creare e modificare viaggi e itinerari, rilasciare feedback e ricevere notifiche.

Un Tour Operator è una categoria che può creare viaggi con l'obiettivo di guadagnare una percentuale dalla loro vendita.

Il System Admin, come già detto, si occupa della gestione amministrativa; può effettuare operazioni su feedback, viaggi, itinerari, etc...

Il nostro sistema tratta dati sensibili, come i dati anagrafici, quindi si presenta la necessità di una gestione sicura dei dati memorizzati. Ciò viene implementato tramite un sistema di autenticazione che un utente deve utilizzare per accedere alla piattaforma, tramite la quale si rende possibile l'accesso alle funzionalità del sito. L'autenticazione in questo caso è realizzata tramite un sistema di login, in cui l'utente specifica l'email con cui si è registrato alla piattaforma, e una password per proteggere l'account. Il sistema di login utilizza un HashMap a liste di trabocco, in modo tale da ottenere da una parte uno spazio di memorizzazione ampio per inserire ciascun utente, dall'altra invece possiamo ottenere efficienza attraverso la gestione di una chiave che viene assegnata all'utente al momento dell'accesso.
Per quanto riguarda il login, esso non necessita di un server per la relativa gestione, in quanto l'oggetto che se ne occupa si basa solo sul salvare il riferimento dell'utente in una struttura dati. Il login stesso, infatti, viene visto solo come una form per l'utente tramite la quale può realizzare la richiesta di accesso.
Il servizio conosce il sistema a tempo di compilazione, ovvero tutti i componenti del sistema, dovranno prima essere compilati e poi deployati, quindi il servizio dovrà avere tutto sotto controllo già a tempo di compilazione.

Oggetto **Attore**	**User Manager**	**TourOperator Manager**	**Travel Manager**
User	CreazioneUtente, Login utente, Cerca utente da nome, Visualizza informazioni utente	Controllo T.O., Visualizza informazioni T.O.	Visualizza informazioni viaggio
RegisteredUser	Login utente, Cerca utente da nome, Viaggi già conclusi, Cancella account, Partecipa ad un viaggio, Compra pacchetto T.O., Crea un viaggio, Visualizza informazioni utente	Controllo T.O., Visualizza informazioni T.O.	Visualizza informazioni viaggio, Creazione Viaggio, Cancella Viaggio, Chiudi Viaggio, Aggiorna viaggio, Cerca viaggio attraverso luogo

TourOperator	Login utente, Cerca utente da nome, Viaggi già conclusi, Cancella account, Partecipa ad un viaggio, Crea un viaggio, Visualizza informazioni utente	Controllo scadenza pacchetto T.O., Gestione ricavi T.O., Controllo T.O., Rinnovo pacchetto	Visualizza informazioni viaggio, Creazione Viaggio, Cancella Viaggio, Chiudi Viaggio, Aggiorna viaggio, Cerca viaggio attraverso luogo
SystemAdmin	Login utente, Cerca utente da nome, Viaggi già conclusi, Cancella account, Partecipa ad un viaggio, Crea un viaggio, Cerca utente da id, Visualizza informazioni utente	Controllo T.O., Visualizza informazioni T.O.	Visualizza informazioni viaggio, Creazione Viaggio, Cancella Viaggio, Chiudi Viaggio, Aggiorna viaggio, Cerca viaggio attraverso luogo

Oggetto **Attore**	**Location Manager**	**Route Manager**	**Poll Manager**
User	Cerca luogo attraverso il nome, Visualizza informazioni luogo	Informazioni itinerario, Ricerca itinerari per luogo, Ricerca itinerari per data	
RegisteredUser	Cerca luogo attraverso il nome, Visualizza informazioni luogo, Crea luogo	Crea itinerario, Aggiorna itinerario, Informazioni itinerario, Ricerca itinerari per luogo, Ricerca itinerari per data, Aggiungi luogo ad itinerario, Rimuovi luogo da itinerario, Proporre luogo in itinerario, Filtra itinerario per luogo	Crea un Poll, Elimina un Poll, Aggiorna Poll, Vota Poll

TourOperator	Cerca luogo attraverso il nome, Visualizza informazioni luogo, Crea luogo	Crea itinerario, Aggiorna itinerario, Informazioni itinerario, Ricerca itinerari per luogo, Ricerca itinerari per data, Aggiungi luogo ad itinerario, Rimuovi luogo da itinerario, Proporre luogo in itinerario, Filtra itinerario per luogo	Crea un Poll, Elimina un Poll, Aggiorna Poll, Vota Poll
SystemAdmin	Cerca luogo attraverso il nome, Visualizza informazioni luogo, Crea luogo, Cancella luogo	Crea itinerario, Aggiorna itinerario, Elimina itinerario, Informazioni itinerario, Ricerca itinerari per luogo, Ricerca itinerari per data, Aggiungi luogo ad itinerario, Rimuovi luogo da itinerario, Proporre luogo in itinerario, Filtra itinerario per luogo	Crea un Poll, Elimina un Poll, Aggiorna Poll, Vota Poll

Oggetto **Attore**	**Feedback Manager**	**Notification Manager**
User	Trova feedback di un generico elemento	
RegisteredUser	Crea un feedback, Trova feedback di un generico elemento	Crea notifica, Imposta notifica come già letta, Invia notifica, Ricevi notifica
TourOperator	Crea un feedback, Trova feedback di un generico	Crea notifica, Imposta notifica come già letta, Invia notifica, Ricevi

	elemento	notifica
SystemAdmin	Crea un feedback, Elimina feedback, Trova feedback di un generico elemento	Crea notifica, Invia notifica, Imposta notifica come già letta, Ricevi notifica

3.6 Global Software Control

Il nostro sistema si baserà essenzialmente sul pattern MVC, quindi sarà orientato ad eventi, ovvero, ogni qualvolta un utente esegue operazioni sul nostro sistema, tali operazioni verranno reindirizzate e gestite attraverso dei dispatcher.
Quindi, per natura, il nostro sistema sarà decentralizzato, ossia ogni oggetto provvederà a risolvere una singola classe di problemi, quindi per poter eseguire delle operazioni, verranno chiamate in causa vari oggetti, su cui ognuno di essi eseguirà azioni a lui competenti.

3.7 Boundary Conditions

Il sistema avrà avvio con il caricamento del database e di tutti i dati contenuti al suo interno. Il sistema risulta pronto all'uso e disponibile all'invio dei dati in esso.
L'UserInterface carica la gli script necessari alla composizione ed alla visualizzazione della homepage. Da qui un utente può effettuare login per poi interagire con ciò che è all'interno del database.
Ogni singolo sottosistema non viene terminato ma, resta latente se non direttamente invocato dal sistema. Se un sottosistema risulta momentaneamente non disponibile, allora il resto del sistema ne verrà informato attraverso un cambio comportamentale del sistema dovuto alla gestione delle eccezioni. Di conseguenza, il comportamento conseguente ad un malfunzionamento, non causerà un riavvio di sistema ma si adatterà al tipo di failure in cui si imbatterà.

StartServer

Attore: SystemAdmin

Entry condition: Il SystemAdmin ha effettuato il login sul server

Flusso di eventi:

- Il SystemAdmin esegue il comando di startup del sistema

  Il sistema esegue le operazioni di iniziallizzazione, andando a leggere dal database i dati necessari alle normali funzionalità

Exit condition: Il sistema è inizializzato correttamente ed è disponibile

ShutdownServer

Attore: SystemAdmin

Entry condition: Il SystemAdmin ha effettuato il login sul server

Flusso di eventi:

- Il SystemAdmin esegue il comando di shutdown del sistema

  Il sistema esegue le operazioni di terminazione, rilasciando tutte le risorse del utilizzate e salvando i dati rimanenti sul database per recuperarli in seguito.

Exit condition: Il sistema è spento correttamente

4. Subsystem Services

UserSubsystem

Servizio	Descrizione
Creazione utente	Crea un utente registrato, prendendo i dati passati e rendendoli persistenti.
Login utente	Dato un utente, ci dice se esso è loggato o no alla piattaforma. (Possibile concorrenza)
Viaggi a cui ha partecipato (in server)	Dato un utente, restituisce i viaggi a cui ha partecipato.
Cancella account	Dato un utente, elimina questo, e di conseguenza anche i suoi viaggi, dal database.
Partecipa ad un viaggio (in server)	Dato un Travel e un utente, permette di associare fare partecipare l'ultimo al primo, e di salvare questa associazione nel database.
Cerca utente dal suo nome	Data una stringa, permette di cercare gli utenti che hanno nome simile ad essa. Restituisce questi utenti (Possibile concorrenza)
Cerca Utente tramite mail	Data una mail, restituisce l'utente corrispondente.
Cerca utente dal suo id	Dato un id, permette di trovare l'utente associato. Restituisce l'utente che questo id.
Compra pacchetto T.O.	Dato in input il nome di un pacchetto TourOperator, crea, rende persistente e restituisce un utente di tipo TourOperator.
Crea un viaggio	Dato in input un RegisteredUser, crea un nuovo Travel, associa il primo a questo, lo salva in database.
Visualizza informazioni utente	Dato in input l'id di un utente, questo viene letto dal database e restituito
Cambio password	Modifica la password nel database.
Login utente	Dato un utente, disconnette tale utente.

TravelSubsystem

Servizio	Descrizione
Creazione Viaggio	Dati in input i dati di un Travel e l'id di un utente, questi vengono salvati in database e restituiti all'utente stesso.
Cancella Viaggio	Dato un Travel, questo viene cancellato dal database.
Chiudi Viaggio (in server)	Dato un Travel, questo viene chiuso, ovvero viene reso immodificabile.
Visualizza informazioni viaggio (in server)	Dato l'id di un Travel, esso viene letto dal database, e contemporaneamente, vengono lette tutte le sue informazioni in altri servizi. Va a raccogliere in particolare gli utenti partecipanti.
Aggiorna viaggio	Dato in input un Travel, esso viene aggiornato, a seconda degli attributi modificati.
Cerca viaggio attraverso luogo (in server)	Dato in input un luogo, questo restituisce una collezione di viaggi che lo contengono
Salva viaggio in database	Dato un viaggio, lo registra nel database.
Cerca se utente partecipa al viaggio(in server)	Dati userId e travel Id, dice se l'utente partecipa a tale viaggio.
Aggiungi utente al viaggio(in server)	Dati userId e travel Id, aggiunge l'utente a tale viaggio.
Rimuove utente da viaggio(in serve)	Dati travelId e UserId, rimuove l'utente da tale viaggio.
Partecipanti al viaggio	Dato travelId, restituisce i suoi partecipanti.
Cerca viaggio attraverso id (in server)	Dato in input un id, restituisce il viaggio corrispondente.
Sondaggio su viaggio	Dato travelId, restituisce il sondaggio su tale viaggio.
Filtra viaggi	Dato un luogo, restituisce i viaggi che lo contengono; presa in input una data, restituisce i viaggi con quella data.

LocationSubsystem

Servizio	Descrizione
Crea luogo	Dati in input dati relativi ad un luogo, questo viene salvato sul database e viene restituito all'utente
Cerca luogo attraverso il nome	Dato in input il nome di un luogo, viene restituita una collezione di luoghi
Cerca luogo attraverso id	Dato id luogo, restituisce il luogo.
Cancella luogo	Dato in input un luogo, lo cancella dal database.
Visualizza informazioni luogo	Dato in input l'id di un luogo, vengono restituiti tutti le informazione relative a questo.
Salva luogo in db	Dato un luogo, lo registra nel database.
Aggiorna luogo	Dato un luogo, ne aggiorna le informazioni.

RouteSubsystem

Servizio	Descrizione
Crea itinerario	Dati in input dati relativi ad un itinerario, questo viene salvato in database e viene restituito ad un utente
Aggiorna itinerario	Dato in input un Route, questo viene aggiornato in database.
Elimina itinerario	Dato in input un Route, questo viene eliminato dal database
Informazioni itinerario (in server)	Dato in input un Route, vengono raccolte tutte le informazioni, quindi i feedback e i luoghi che contiene.
Ricerca itinerari per luogo (in server)	Dato in input un Location, vengono raccolti tutti gli iitinerari che lo contengono
Aggiungi luogo ad itinerario (in server)	Dato in input un Location e un Route, il primo viene aggiunto a quest'ultimo.
Rimuovi luogo da itinerario (in server)	Dato in input un Location e un Route, il primo viene cancellato da quest'ultimo.

Salva itinerario in db	Dato un itinerario, lo salva nel database.
Ricerca itinerario per id (in server)	Dato un routeId, restituisce l'itinerario corrispondente.
Ricerca itinerario per nome (in server)	Dato un nome, restituisce una lista di itinerari.
Restituisce luoghi in itinerario	Dato routeId, restituisce i luoghi in esso contenuti.
Ricerca utente in itinerario.	Dati userId e routeId, dice se l'utente partecipa a tale itinerario.

PollSystem

Servizio	Descrizione
Crea un Poll	Dati in input dati relativi ad una proposta, questa viene salvata in database e restituita
Elimina un Poll	Dato un Poll, questo viene eliminato dal database
Aggiorna Poll	Dato un Poll, questo viene aggiornato con nuovi dati
Vota Poll (in server)	Dato un Poll, questo viene incrementato negativamente o positivamente.
Salva Poll in db	Dato un Poll, lo salva nel database.
Inserisci sondaggio utente (in server)	Dati pollId e userId, inserisce il poll dell'utente.
Cerca sondaggio utente (in server)	Dati pollId e userId, dice se il poll è legato a tale utente.

FeedbackSubsystem

Servizio	Descrizione
Crea un feedback	Dato un Feedback, l'id del sender, e l'id del ricevitore generico, questo viene salvato in database e restituito.
Elimina feedback	Dato un generico Feedback, questo viene eliminato dal database.
Trova feedback (in server)	Dato un Feedback, ci dice se esso è presente.
Modifica feedback (in server)	Dato un feedback, questo viene modificato.

Cerca feedback tramite id (in server)	Dato id, restituisce il feedback o su utente, o su itinerario o su luogo.
Salva feedback in db	Dato un feedback, lo salva nel database.

NotificationSubsystem

Servizio	Descrizione
Crea notifica	Dati in input i dati relativi ad una notifica, questa viene salvata sul database
Aggiorna notifica	Data una Notifica, questa viene aggiornata in database, utile per impostarla come già letta
Imposta notifica come già letta	Data una notifica, questa viene impostata come già letta, e aggiornata in database
Invia notifica (in server)	Dato un utente e una notifica, manda questa alla sezione notifiche.
Consegna notifica (in server)	Data una notifica ed utente destinatario, consegna la notifica all'utente
Salva notifica in db	Data una notifica, la salva nel database.
Elimina notifica	Dati in input id della notifica, essa viene eliminata.
Retituisce notifica	Dato input notifica, restituisce tale notifica.
Notifiche utente	Dato un userId, ritorna le notifiche ad esso associate.