



UNIVERSITÀ DEGLI STUDI DI NAPOLI

FEDERICO II



UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

Corso di Laurea in Informatica

Progettazione e realizzazione di una piattaforma social per appassionati di escursioni

A cura di:

Cozzolino Francesco

Malangone Luca

Anno accademico 2021/2022

Sommario

1.Introduzione	4
2.0 Modello funzionale	5
2.1 Use case diagram	6
2.1.2 Rappresentazione generale	6
2.1.2 Login e registrazione	7
2.1.3 Itinerari	8
2.1.4 Gestione amministratore	9
2.2 Use case text diagram	9
2.2.1 Inserimento itinerario	10
2.2.2 Segnalazione	12
2.3 Mock-up	13
2.3.1 Inserimento itinerario	13
2.3.2 Segnalazione	14
2.4 Idea progettuale	16
2.5 Target degli utenti	17
2.6 Valutazione dell'usabilità a priori	18
2.7 Statechart dell'interfaccia grafica	21
2.7.1 Aggiungi itinerario	21
2.7.2 Segnalazione	23
2.8 Glossario	24
3.0 Modello di dominio	26
3.1 Class diagram	26
3.1.1 Login e registrazione	29
3.1.2 Visualizzazione post	30
3.1.3 Visualizza itinerario	31
3.1.4 Inserimento itinerario	32
3.1.5 Messaggi	33
3.1.6 Statistiche	34
3.2 Sequence diagram	35
3.2.1 Registrazione	35
3.1.2 Inserimento itinerario	37
3.3 Activity diagram	39
4.0 Design del sistema	45
4.1 Architettura	45

4.1.1. Tecnologie usate	46
4.2 Diagramma delle classi di design	48
4.3 Diagramma di sequenza di design	48
4.3.1 Aggiunta nuovo itinerario	48
4.3.2 Registrazione	50
4.3.3 Login	51
4.3.4 Visualizzazione profilo	52
4.3.5 Chat	52
4.3.6 Aggiunta di segnalazione/correzione	53
4.3.7 Visualizza statistiche	53
4.3.8 Ricerca itinerari	54
4.4 Diagrammi di sequenza di design	54
4.4.1 Invio messaggio	54
4.4.2 Registrazione	54
4.5 Definizione delle gerarchie funzionali	57
5.0 Testing	58
5.1 Test registrazione	58
5.1.1 Metodo da testare	58
5.1.2 Analisi	61
5.1.3 Codice Test	62
5.1.4 Output del test	64
5.2 Test ConvertMap	65
5.2.1 Metodo da testare	66
5.2.2 Analisi	67
5.2.3 Codice test	68
5.2.4 Output del test	69
5.3 Conversione GeoPoint	70
5.3.1 Metodo da testare	70
5.2.2 Analisi	71
5.2.3 Codice test	71
4.2.4 Output test	72
6. Valutazione dell'usabilità sul campo	72
6.1 Test di compito	72
6.2 Monitoraggio attraverso strumenti di logging	74

1.Introduzione

Si vuole realizzare una moderna piattaforma social network, adattabile ai diversi dispositivi mobile, per appassionati di escursioni. Il sistema deve consentire agli utenti di registrarsi e autenticarsi sulla piattaforma. La

registrazione/autenticazione deve essere resa possibile sia in app sia mediante ulteriori piattaforme come Google e Facebook. L'utente autenticato deve essere in grado di visualizzare i post di altri utenti e di poterne creare. La creazione di un post prevede l'inserimento di un nuovo itinerario in piattaforma. Un sentiero è caratterizzato da un nome, una durata, un livello di difficoltà, un punto d'inizio, una descrizione (opzionale) e un tracciato geografico (opzionale) che lo rappresenta sulla mappa. Il tracciato può essere inserito manualmente interagendo con una mappa interattiva oppure tramite file in formato standard GPX. Il sistema deve rendere possibile la richiesta di un filtraggio dei post. Il criterio di selezione deve avvenire per area geografica, livello di difficoltà, durata e per accessibilità. Qualora lo ritenga necessario, un utente può segnalare un itinerario per informazioni inesatte o non aggiornate. Una segnalazione è caratterizzata da un titolo e da una descrizione. In aggiunta, è possibile indicare un punteggio di difficoltà e/o un tempo di percorrenza diverso da quello indicato dal post. In questo caso, il punteggio di difficoltà e il tempo verranno ricalcolati con una media aritmetica. I sentieri per cui è presente una segnalazione mostrano un warning. In aggiunta, un utente può visualizzare la pagina profilo di un altro utente, che riporta le sue attività più recenti (post pubblicati, playlist, foto). Il software rende possibile anche la comunicazione tra utenti attraverso una chat. La comunicazione avviene tra due utenti tramite messaggi privati (PM). Il sistema permette anche una gestione degli amministratori. Essi saranno in grado di effettuare l'accesso tramite l'applicazione e potranno gestire e visionare dati statistici in tempo reale sul sistema (numero di utenti, numero di accessi, numero di ricerche, numero di recensioni, numero di itinerari ed altro).

2.0 Modello funzionale

All'interno del processo di sviluppo di un software l'analisi funzionale è il processo che permette di identificare i processi che costituiscono un sistema informativo. Con tale attività è possibile definire tutte le caratteristiche e le specifiche tecniche delle componenti software da realizzare nell'ambito del sistema sottoposto all'analisi. La fase di analisi funzionale è dunque fondamentale per lo sviluppo del progetto. Tutte le fasi successive sono ben strutturate grazie a tale fase. Le variabili da prendere in considerazione sono molteplici, tra queste spiccano il comportamento del sistema informatico, le limitazioni in fase di sviluppo, tempi, costi e molto altro.

Il processo di analisi funzionale è composto da un processo che prevede:

- confronto con il cliente per raccogliere le informazioni iniziali
- studio e realizzazione di un documento di tesi
- studio e realizzazione dei modelli iniziali
- confronto con il cliente per valutare il lavoro
- modifiche o approvazione del progetto

Al termine dell'analisi funzionale è possibile proseguire con tutte le altre fasi di sviluppo del software.

La rappresentazione dell'analisi funzionale avviene attraverso diagrammi e studio dei target utente, queste raffigurazioni prendono il nome di modello funzionale.

2.1 Use case diagram

Lo use case diagram descrive riassuntivamente gli scenari in cui il sistema interagisce con persone o altri sistemi esterni.

Tale diagramma è stato ideato per:

- Rappresentare gli obiettivi delle interazioni sistema-utente
- Definire e organizzare i requisiti funzionali
- Specificare il contesto e i requisiti del sistema
- Rappresentare la modellazione del flusso di base di eventi in un caso d'uso

2.1.2 Rappresentazione generale

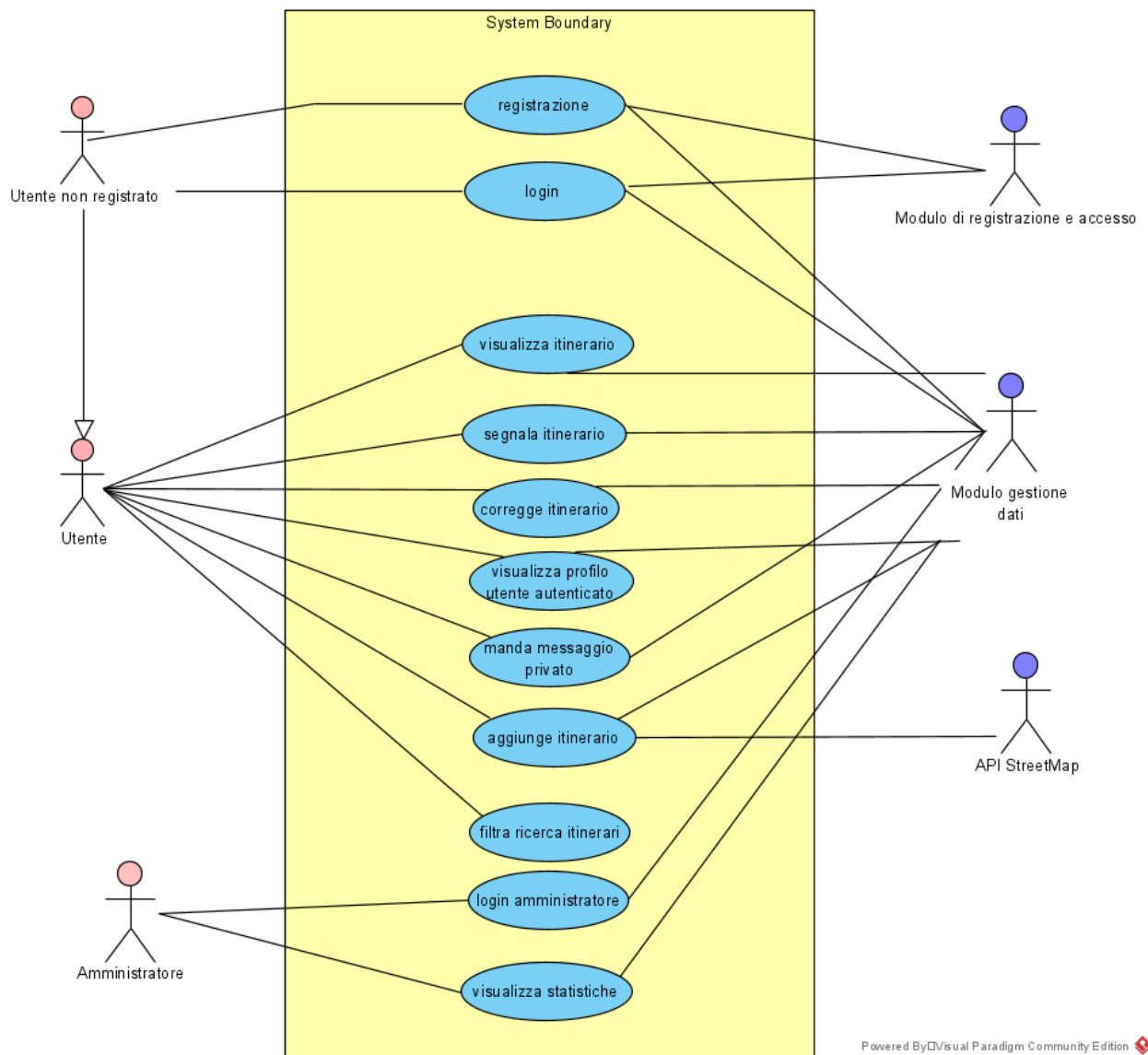
La prima illustrazione mostra uno use case generale.

Si è scelto di identificare tre utenti (utente, utente registrato, amministratore), ognuno con proprie funzionalità.

L'utente non registrato è una persona non registrata/autenticata. Ha la possibilità di registrarsi o di autenticarsi. Sia la registrazione sia l'autenticazione può avvenire tramite il sistema fornito dal sistema o attraverso sistemi esterni come Facebook.

L'utente ha accesso a tutte le funzioni rese disponibili per chi usufruisce dell'applicazione. Ha la possibilità di visualizzare gli itinerari postati da altri utenti autenticati, può inserire itinerari, effettuare segnalazioni, visualizzare un profilo di un utente e scambiarsi messaggi.

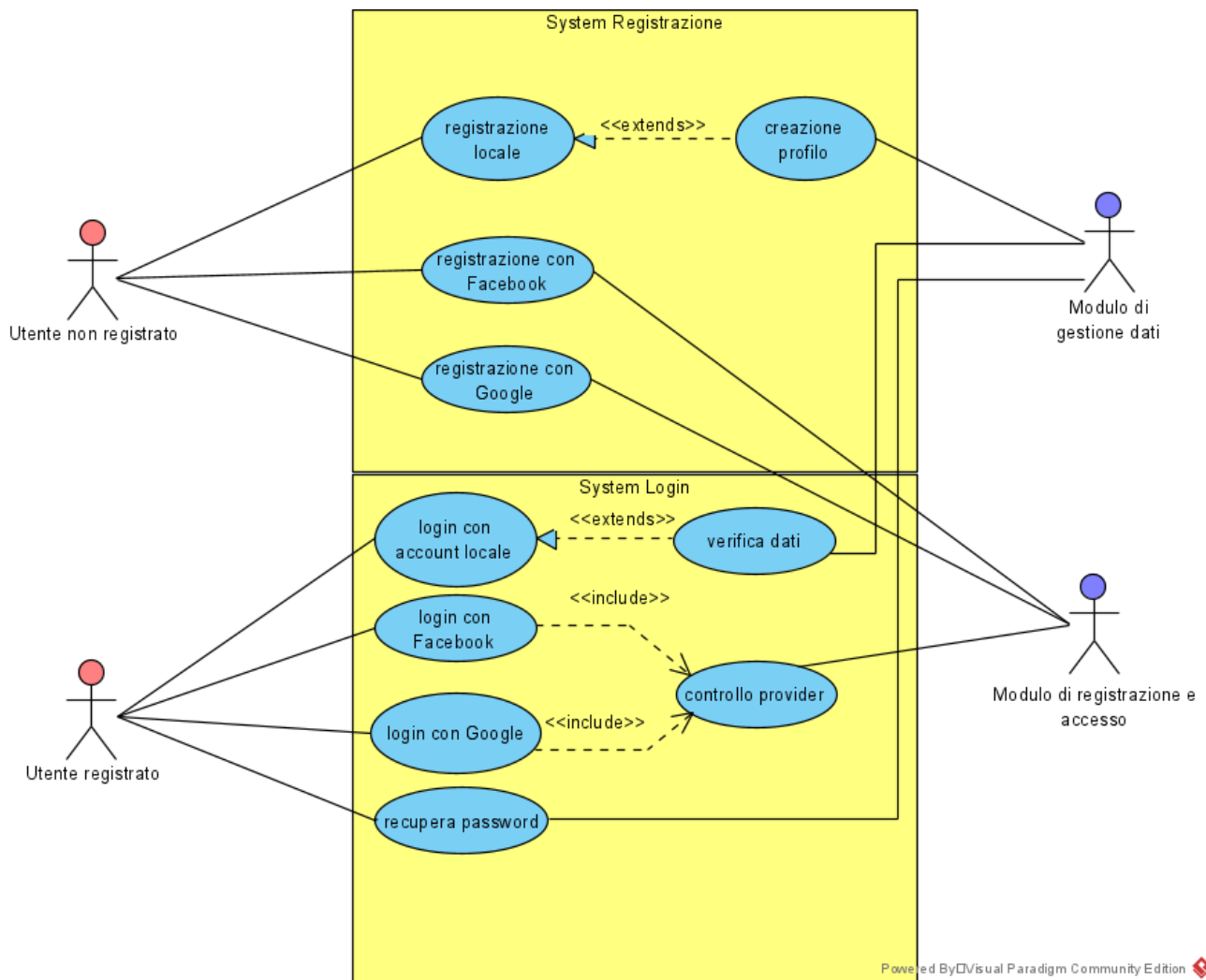
Infine, l'amministratore può visualizzare statistiche relative all'applicativo.



2.1.2 Login e registrazione

Un utente che vuole registrarsi ha la possibilità di registrarsi tramite il sistema o tramite sistemi esterni come Google e Facebook. Nel caso della registrazione tramite il sistema, prima di inviare i dati al database, verrà effettuato un controllo dei dati (dati errati o assenti).

Il login avviene con l'inserimento dell'email e password, in questo caso verrà effettuato un controllo dei dati. In alternativa viene fatto l'accesso tramite Google o Facebook che effettueranno i dovuti controlli di accesso.

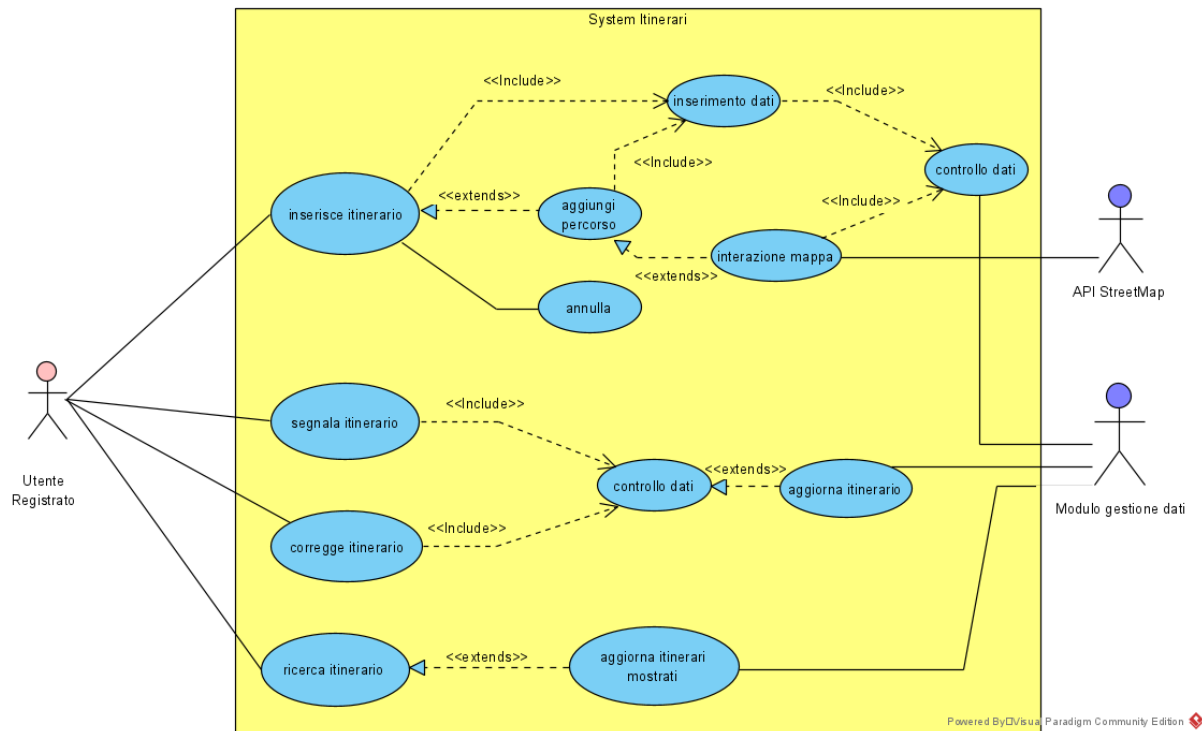


2.1.3 Itinerari

L'utente registrato ha la possibilità di inserire un itinerario. Per l'inserimento sono necessari dati obbligatori, come nome dell'itinerario, durata ed altri, e la possibilità di aggiungere un percorso. Il percorso è possibile aggiungerlo manualmente interagendo con una mappa interattiva oppure caricando un file GPX. Al termine della compilazione, viene effettuato un controllo dei dati inseriti e infine inviati al server.

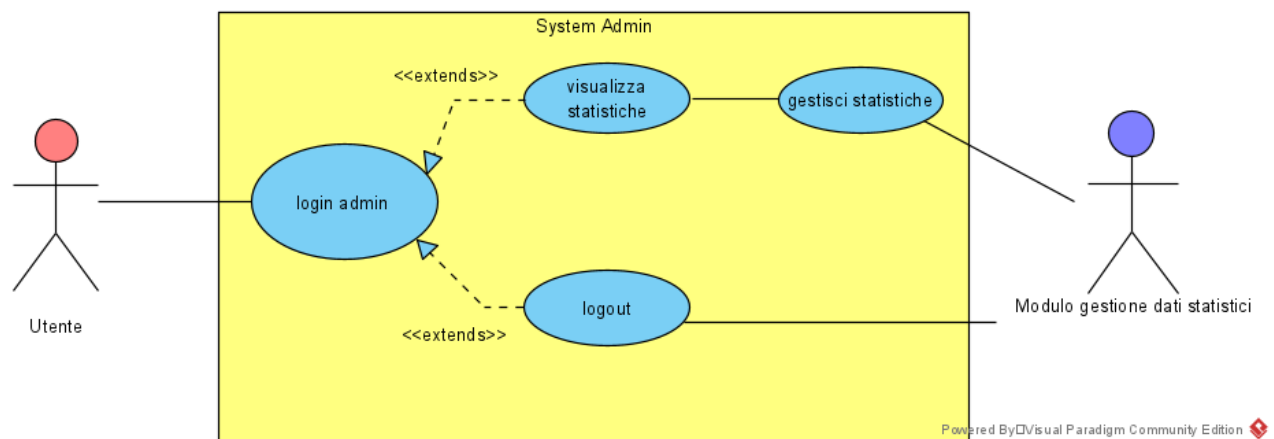
È possibile anche effettuare una segnalazione di un itinerario per dati errati. Il sistema effettuerà un controllo dei dati.

Infine, il sistema rende possibile una ricerca di itinerari tramite filtraggio. Verrà gestito dal server.



2.1.4 Gestione amministratore

L'amministratore avrà la possibilità di visualizzare le statistiche del sistema. Il database invierà le informazioni sugli accessi, utenti registrati, itinerari inseriti ed altro



2.2 Use case text diagram

Il diagramma dei casi d'uso viene accompagnato da una documentazione testuale strutturata o altri diagrammi di maggiore dettaglio. In questa

documentazione viene utilizzata la tabella di Cockburn per due casi significativi. I due casi presi in considerazione sono: inserimento itinerario e segnalazione.

2.2.1 Inserimento itinerario

USE CASE #01	Inserimento itinerario				
Goal in Context	L'utente inserisce un itinerario.				
Preconditions	L'utente ha effettuato l'accesso correttamente.				
Success End Condition	L'utente riesce correttamente ad inserire un itinerario.				
Failed End Condition	L'utente non riesce ad inserire un itinerario.				
Primary Actor	Utente loggato.				
Trigger	Clicca sul bottone "add" nella schermata Home.				
DESCRIPTION	Step n°	Utente Loggato	Sistema	Azure	API gps
	1	Clicca "add" nella schermata "Home".			
	2		Carica il frame "Aggiunta Itinerario".		
	3	Inserisce nome del percorso.			
	4	Inserisce la durata.			
	5	Inserisce il punto d'inizio.			
	6	Inserisce la difficoltà.			
	7	Inserisce una descrizione.			
	8	Inserisce un tracciato			
	9	Clicca "Prossimo".			
	10		Controlla i dati.		
	11		Invia i dati al database.		
	12			Acquisisce i dati	
	13			Ritorna un valore di conferma acquisizione	
	14		Mostra pop-up di inserimento riuscito.		
	15		Mostra frame "Home"		
EXTENSIONS #1 Il sistema rileva campi vuoti	10.a		Il sistema rileva campi vuoti		
	11.a		Evidenzia in rosso il contorno dei campi vuoti		
EXTENSIONS #2 L'utente inserisce il tracciato in formato GPX	8.b	Seleziona l'inserimento tramite file GPX			
	9.b		Permesso di accesso ai file del dispositivo "AgIT_E1"		
	10.b	Accetta i permessi del sistema			

	11.b	Carica file GPX			
EXTENSIONS #3 Il sistema non riesce a caricare il file GPX	11.b.i		Il sistema rileva un formato del file non idoneo		
	12.b.ii		Mostra pop-up di errore "Eglt_E3"		
	13.b.iii	Clicca "OK".			
	14.b.iv		Chiude pop-up.		
EXTENSIONS #4 L'utente inserisce il tracciato manualmente	8.c	Seleziona l'inserimento del tracciato manualmente			
	9.c		Carica nuova schermata "f_mappa"		
	10.c		Richiesta mappa		
	11.c				Restituisce mappa
	12.c		Mostra la mappa e inizializza iterazioni		
	13.c	Indica punto di partenza			
	14.c	Indica punto di arrivo			
	15.c				Elabora e restituisce i dati
	16.c		Controllo dati		
EXTENSIONS #5 Il sistema non riesce a connettersi con l'API gps	11.c.i				Accesso negato
	12.c.ii			Mostra pop-up con messaggio d'errore ""Aglt_E2"	
EXTENSIONS #7 Il sistema non riesce a collegarsi al Database	12.d		Il sistema fallisce il collegamento con il database.		
	13.d		Mostra pop-up "Aglt_E4"		
	14.d	Clicca "OK".			
	15.d		Chiude pop-up.		

2.2.2 Segnalazione

USE CASE #01	Segnalazione di un itinerario			
Goal in Context	L'utente invia la segnalazione di un itinerario.			
Preconditions	L'utente ha effettuato l'accesso correttamente e sta visualizzando un itinerario.			
Success End Condition	L'utente riesce correttamente ad inviare una segnalazione di un itinerario.			
Failed End Condition	L'utente non riesce ad effettuare la segnalazione di un itinerario.			
Primary Actor	Utente loggato.			
Trigger	Clicca sul bottone "add" nella schermata Home.			
DESCRIPTION	Step n°	Utente Loggato	Sistema	Azure
	1	Clicca "Segnala" nella schermata "Seg_U1".		
	2		Carica il frame "Seg_U2".	
	3	Inserisce titolo della segnalazione.		
	4	Inserisce una descrizione.		
	5	Clicca "Invia".		
	6		Controlla i campi.	
	7		Invia i dati al database	
	8			Controllo dati
	9			Acquisisce i dati
	10		Mostra pop-up di successo "Seg_U3".	
	11	Clicca "ok"		
	12		Chiusura pop-up	
	13		Mostra icona warning in "Seg_U1"	
	14		Mostra numero di warning aggiornato	
EXTENSIONS #1 Il sistema rileva campi vuoti	7.a		Il sistema rileva campi vuoti	
	8.a		Evidenzia in rosso il contorno dei campi vuoti	
EXTENSIONS #2 L'utente ha già effettuato una segnalazione	9.b			Rileva errore
	10.b		Mostra pop-up "Seg_E1"	
	11.b	Clicca "OK"		
	12.b		Chiusura "Seg_E1"	
EXTENSIONS #3 Il sistema non riesce a connettersi al server	8.c		Mostra pop-up "Seg_E2"	
	9.c	Clicca "OK"		

2.3 Mock-up

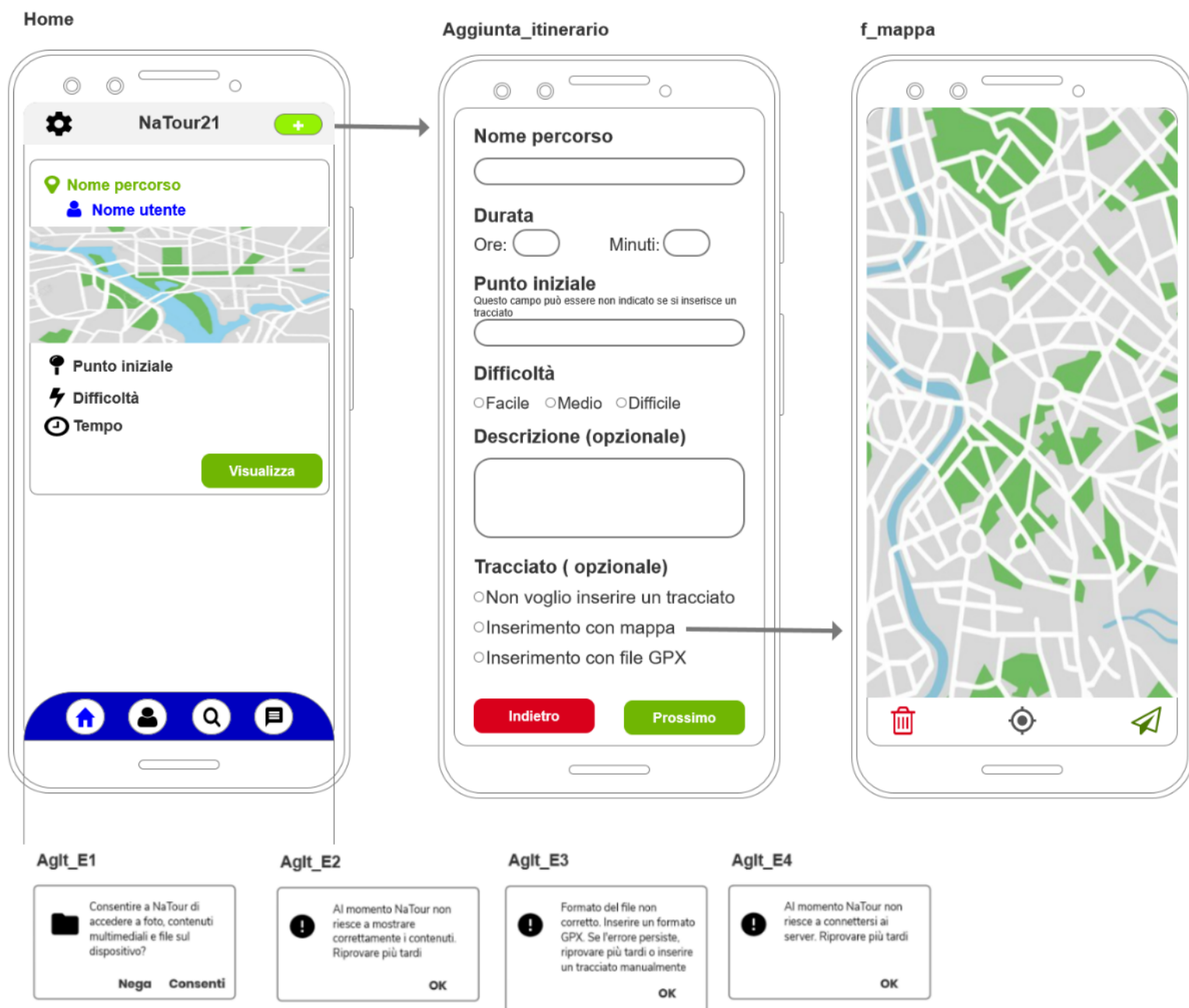
Il mock-up permette la rappresentazione totale o in parte del sistema. In questa documentazione si mostrano i mockup dei due casi trattati nella sezione precedente (Use case text diagram): inserimento itinerario e segnalazione.

2.3.1 Inserimento itinerario

Il mockup rappresenta come è possibile aggiungere un nuovo itinerario.

Dopo aver cliccato il pulsante "add" in Home (in alto a destra), il sistema carica una nuova schermata dove è possibile aggiungere i dettagli dell'itinerario. Alcuni attributi sono opzionali quindi non presi in considerazione in fase di controllo. L'inserimento del tracciato ha diverse esecuzioni in base a ciò che viene selezionato. Se si sceglie di inserire un tracciato manualmente (Inserimento con mappa), viene aperta una schermata da cui è possibile interagire con una mappa. Se si sceglie di inserire un tracciato tramite file (Inserimenti con file GPX), il sistema chiederà l'accesso ai dati del dispositivo, se non richiesti in precedenza, successivamente sarà possibile inserire il file.

Nell'immagine successiva sono mostrate le schermate di dialogo e gli eventuali pop-up di errore.



2.3.2 Segnalazione

Il mockup illustra come effettuare la segnalazione per un itinerario.

Dopo aver cliccato "SEGNALA" in Seg_U1, si apre un pop-up (Seg_U2) per l'inserimento dei dati della segnalazione. Alcuni dati sono opzionale come aggiorna difficoltà e aggiorna tempo. Dopo aver cliccato su "INVIA" viene aggiornato il numero di segnalazioni nella schermata dell'itinerario.

Nell'immagine successiva sono mostrate le schermate di dialogo e gli eventuali pop-up di errore.

Seg_U1

Nome percorso
Post di: Nome utente

⚡ Difficoltà: Difficoltà:
📍 Punto iniziale: Punto iniziale
🕒 Durata: Durata

Correggi Segnala

"Descrizione"

Seg_U4

Crea nuova correzione

Durata:
Ore: Minuti:

Difficoltà:
▼ Nessuna scelta
Facile
Medio
Difficile

Elimina INVIA

Seg_U5

! Segnalazione effettuata con successo.
Grazie del feedback

OK

Seg_U2

Titolo Segnalazione

Descrizione

Elimina INVIA

Seg_U3

! Segnalazione effettuata con successo.
Grazie del feedback

OK

Seg_E1

! Segnalazione già effettuata per questo itinerario

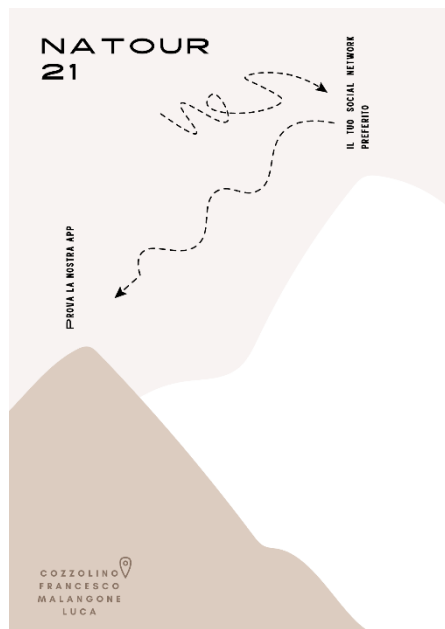
OK

Aglt_E2

! Al momento NaTour non riesce a connettersi ai server. Riprovare più tardi

OK

2.4 Idea progettuale



NaTour21 è stato sviluppato con l'obiettivo di creare un social network per escursionisti. Con la sua grafica semplice e intuitiva, permette di svolgere le principali attività di qualsiasi app social.

Che cos'è un social network?

Con riferimento a "Enciclopedia TRECCANI"

<< Con l'espressione social network si identifica un servizio informatico on line che permette la realizzazione di reti sociali virtuali. Si tratta di siti internet o tecnologie che consentono agli utenti di condividere contenuti testuali, immagini, video e audio e

di interagire tra loro. Generalmente i social network prevedono una registrazione mediante la creazione di un profilo personale protetto da password e la possibilità di effettuare ricerche nel database della struttura informatica per localizzare altri utenti e organizzarli in gruppi e liste di contatti. Le informazioni condivise variano da servizio a servizio e possono includere dati personali, sensibili (credo religioso, opinioni politiche, inclinazioni sessuali ecc.) e professionali. Sui social network gli utenti non sono solo fruitori, ma anche creatori di contenuti.>>

Cosa offre Natour21?

- Autenticazione tramite Google: Gli utenti avranno la possibilità di registrarsi in piattaforma o tramite il proprio account di Google
- Inserimento di nuovi itinerari: Gli elementi principali del nostro social network sono i tracciati. Ogni utente avrà la possibilità di condividere con altri utenti un tracciato ben dettagliato
- Ricerche di itinerari: E' possibile visualizzare ogni itinerario inserito in piattaforma attraverso una intuitiva ricerca
- Segnalazioni: Noi sviluppatori teniamo che i tracciati inseriti siano il più precisi possibile. Per questo ogni utente può segnalare un qualsiasi tracciato per dati non corretti
- Visualizzare altri utenti: Ogni utente ha un proprio profilo che mostra i post e altre informazioni (implementazioni future)
- Chat: I social network oltre ad essere un ritrovo di informazioni, sono utili anche per fare nuove amicizie o chiedere ulteriori informazioni sui tracciati. Per tale motivo sono stati implementati i messaggi tra utenti

- Sviluppo: Gli amministratori sono sempre al corrente dei dati statistici dell'applicazione. Questi dati sono utili per i nostri sviluppi futuri

Caratteristiche

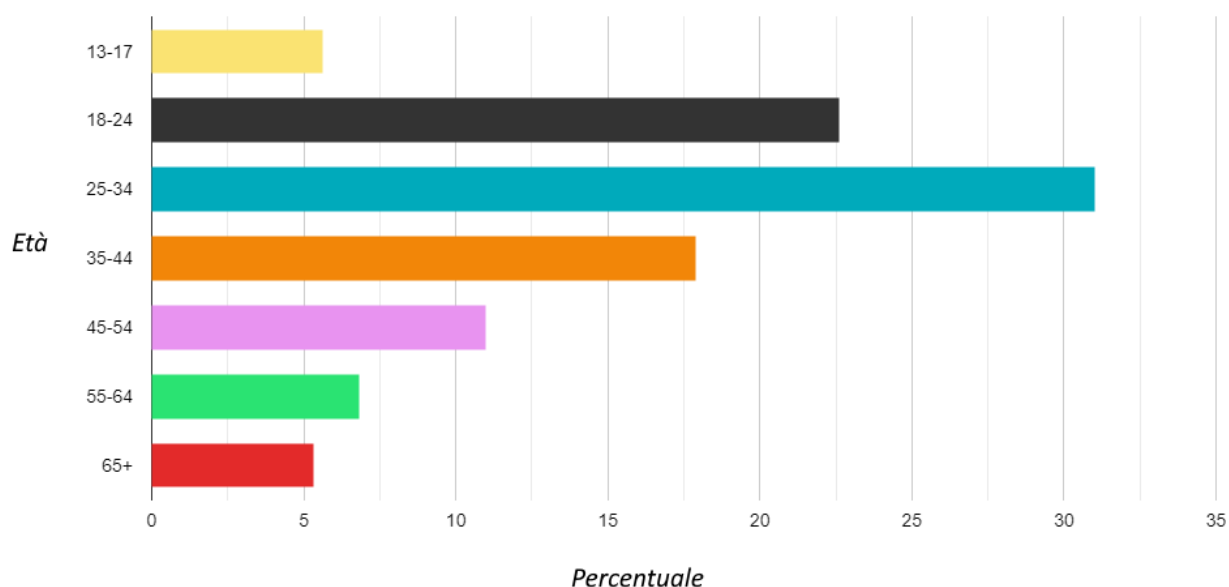
- VISUALIZZAZIONE
 - Interfaccia moderna
 - Interfaccia intuitiva
 - Elementi scalabili
- MAPPE
 - Mappa interattiva
 - Chiara da leggere
 - Lettura del file GPX

2.5 Target degli utenti

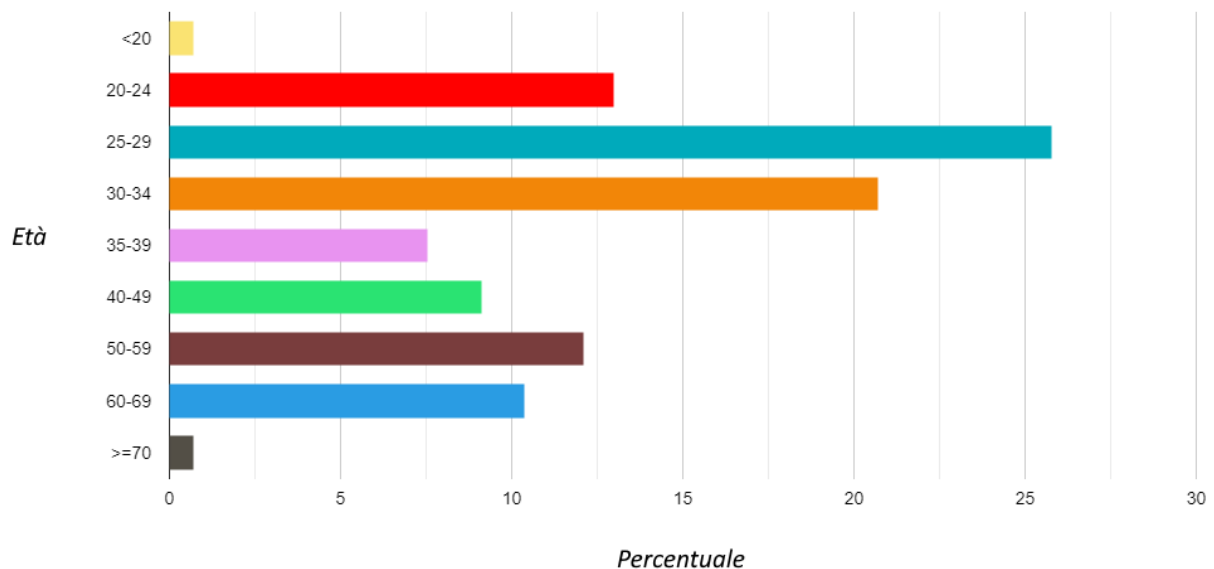
Per prima cosa, abbiamo deciso di occuparci dell'individuazione del target degli utenti del sistema, in maniera da poter influenzare le nostre decisioni verso una user experience quanto più gradevole possibile per i nostri futuri utenti.

Abbiamo approcciato tale compito concentrandoci sulla fascia d'età a cui dare più attenzione, cercando online statistiche in merito riguardanti (1) l'utilizzo dei social media per fascia d'età e (2) le fasce d'età in cui si trovano più escursionisti.

Eseguendo quest'indagine, abbiamo prodotto i seguenti grafi:



Grafo 1 - Utenti dei social media divisi per fasce d'età



Grafo 2 - Escursionisti divisi per fasce d'età

Da questa analisi si evince un dato molto importante: sia gli utilizzatori dei social media sia gli escursionisti sono per la maggior parte concentrati nelle stesse fasce d'età, ed in particolare tra i 20 e i 40 anni. La nostra utenza, quindi, potrebbe essere composta da utenti già abituati in precedenza all'utilizzo di applicazioni social media, e che non solo sia rivolta ad escursionisti esperti, ma anche ad escursionisti novizi o persone che si avvicinano per la prima volta a questo mondo, che hanno già familiarità con altri social. Queste informazioni hanno quindi assunto una grande importanza in fase di stesura dei mockup, i quali presentano notevoli analogie con social media già esistenti.

2.6 Valutazione dell'usabilità a priori

Un test di usabilità ha lo scopo di ricavare informazioni per migliorare il sistema. Chi conduce l'esperimento dovrà esaminare in dettaglio le operazioni svolte dagli utenti per capire problemi, difficoltà o errori del sistema. Le tecniche per svolgere un test sono varie, nel caso di questo progetto si è optato di svolgere una valutazione veloce e il più affidabile possibile. La scelta è ricaduta sulla

valutazione euristica di Nielsen, costituita da dieci regole che permettono al valutatore di inquadrare i problemi del sistema.

Tale valutazione corrisponde alle regole del modello ISO 9241. Rappresentazione in Figura 1 (Riferimento al libro "Facile da usare").

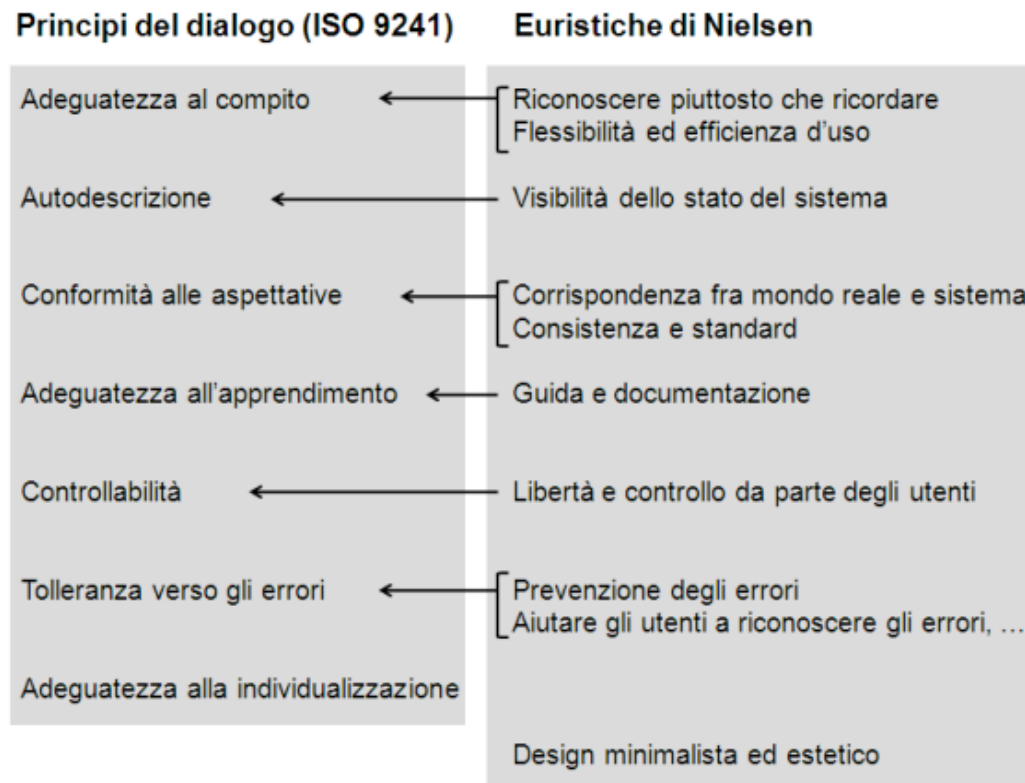




Figura 1 Confronto fra i principi del dialogo dell'ISO 9241 e le euristiche di Nielsen

I test d'usabilità sono stati svolti con mockup interattivi di Axure, sfruttando la rapidità e la semplicità con le quali il programma permette la loro realizzazione. Axure permette di simulare il processo, in parte, del sistema (simula la pressione dei bottoni, effetti visivi, apertura delle pagine ed altro).

L'immagine 2 mostra l'esempio di un mockup interattivo mostrato all'utente. I bottoni o i testi cliccabili sono segnati da un'icona , la quale è assegnata un'azione, mentre altri elementi sono segnati con  che indica una nota allegata.

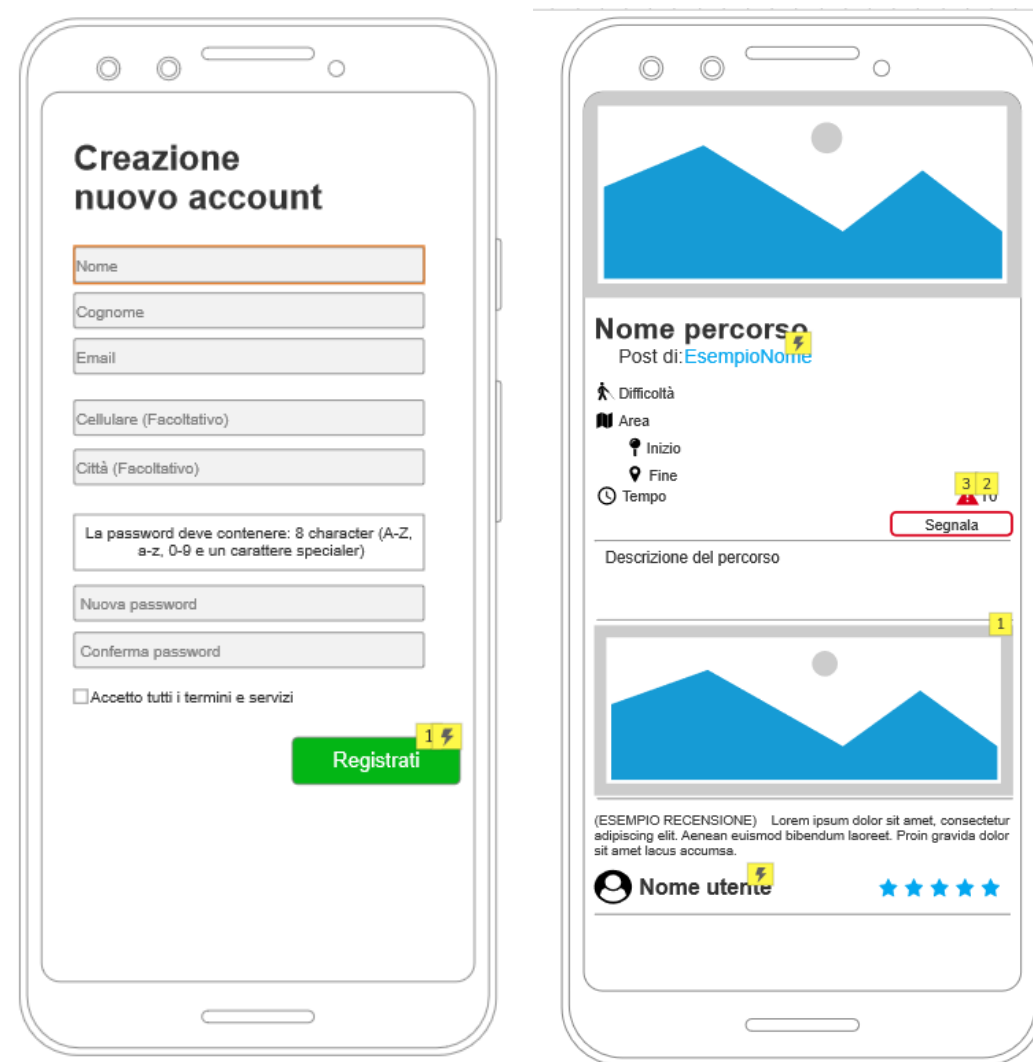


Figura 2. Raffigura due esempi di mockup interattivo

I mockup sono stati presentati a cinque tester. Ad essi sono stati assegnati compiti da eseguire e sono stati monitorati e registrati nella loro esecuzione. I filmati sono successivamente stati analizzati per verificare possibili problemi/errori.

Nella scelta degli utenti per il test, si è optato di consultare utenti con caratteristiche diverse ma in modo selettivo. Come mostrato in Figura 3. , la scelta è ricaduta su cinque categorie:

- Utenti giovani e con esperienze nelle escursioni
- Utenti giovani e interessati, ma non esperti, alle escursioni
- Utenti dai 30 ai 40 anni e con interessi/esperienze nelle escursioni
- Utenti che già utilizzano app per escursioni

	Età		
	<30	30-40	40+
Non escursionista			
Interessati all'escursionismo	✓	✓	
Escursionista	✓	✓	✓

Figura 3. Tabella tester

I compiti assegnati ai tester sono i seguenti:

1. Registrazione al sistema
2. Visualizzazione di un itinerario
3. Inserimento di un itinerario attraverso la mappa interattiva
4. Invio di un messaggio ad un utente

I dati osservati mostrano che gli utenti hanno trovato intuitivo il design dell'applicazione e sono riusciti con facilità a svolgere i compiti a loro assegnati. Concludendo, si è ritenuto il sistema usabile.

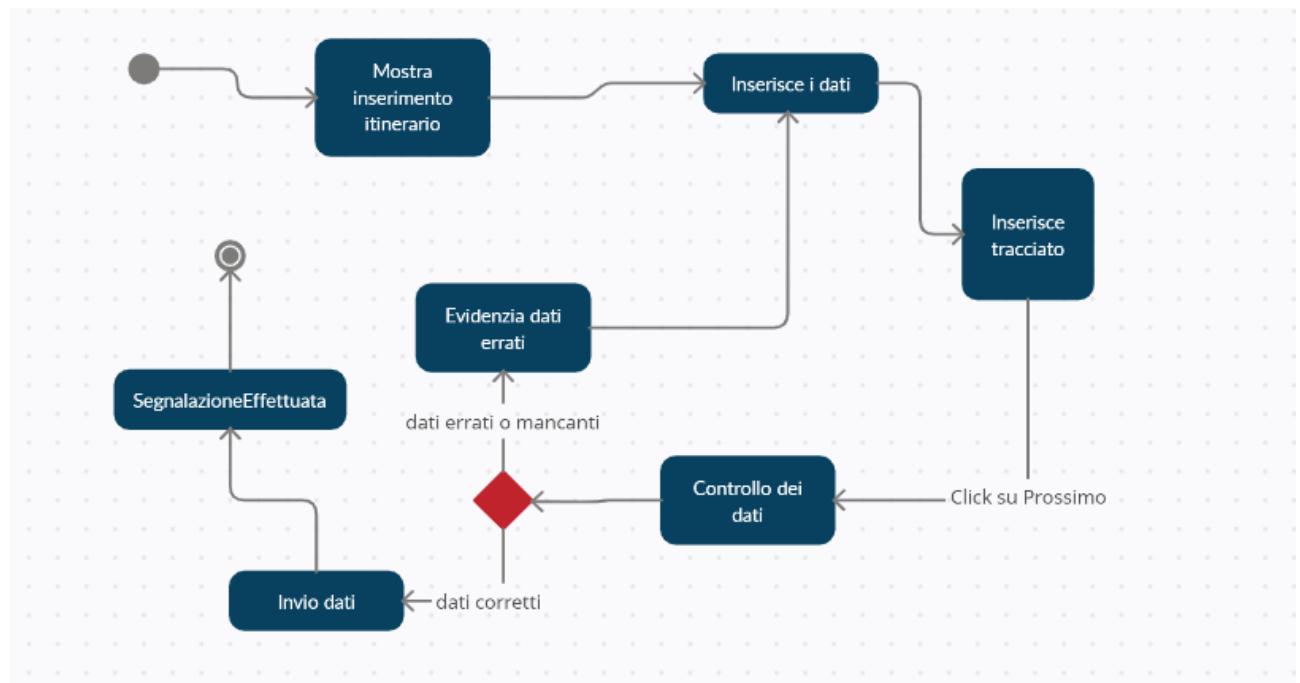
2.7 Statechart dell'interfaccia grafica

Lo statechart diagram mostra gli stati che sono assunti dall'entità o dalla classe in risposta ad eventi.

Le immagini seguenti mostrano il diagramma dei soli casi presi in considerazione in precedenza per le tabelle di Cockburn e Mock-up dell'interfaccia.

2.7.1 Aggiungi itinerario

Gli eventi mostrano l'inserimento di un tracciato all'interno del sistema. Dopo l'inserimento dei dati, viene effettuato un controllo e, in base alla risposta, vengono prodotti due risultati diversi. L'uno evidenzia un errore nell'input e richiede un nuovo inserimento dei dati, l'altro accetta i dati e crea un nuovo itinerario



L'inserimento del tracciato viene esteso per avere una miglior comprensione degli eventi (Figura 4). Il percorso può essere inserito in tre modi diversi. Il primo non prevede l'inserimento di nessun tracciato e quindi continua il suo percorso di eventi al di fuori dell'estensione. Nella seconda scelta, viene inserito tramite mappa interattiva. In questo caso viene aperta una pianta geografica interattiva e, parallelamente, aggiorna la mappa e i dati. Infine l'inserimento tramite importazione di un file GPX, viene controllato il file prima di aggiornare i dati.

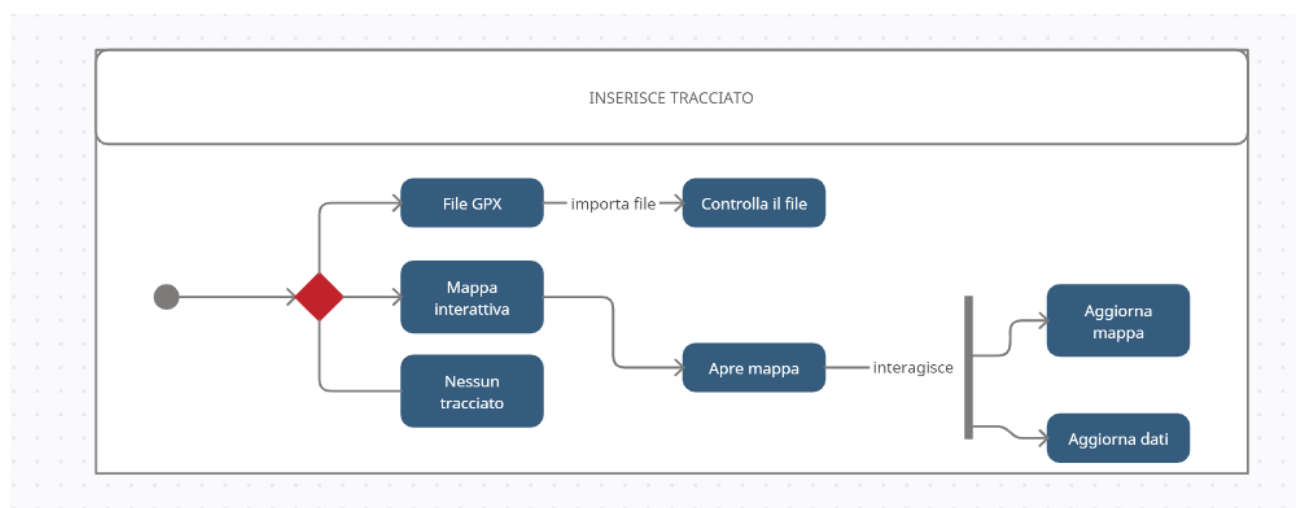
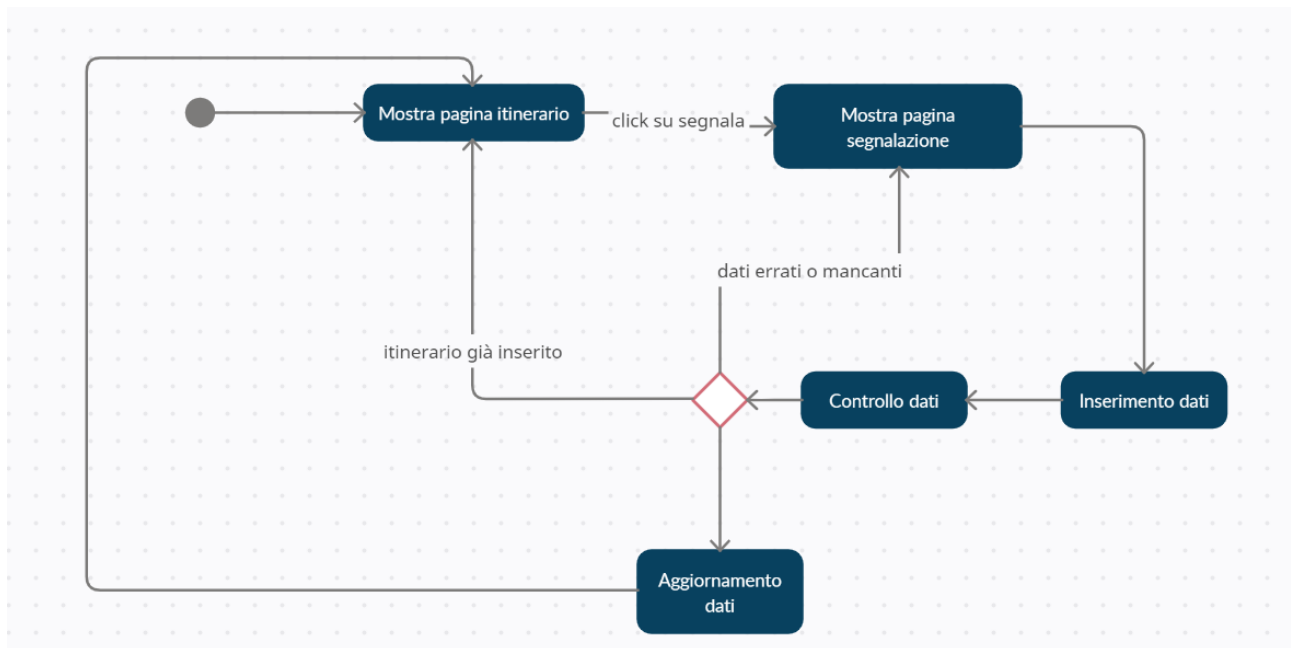


Figura 4. Estensione di Inserimento tracciato

2.7.2 Segnalazione

La Figura 5. mostra gli eventi della segnalazione. Nella pagina di un itinerario, con click sul bottone Segnala viene aperta una pagina di segnalazione. Dopo aver inserito i dati essenziali viene effettuato un controllo che produce due effetti. Nel caso che i dati sono errati o mancanti, viene richiesto di inserire nuovamente le informazioni. Nella seconda ipotesi, un utente ha già inviato una segnalazione per quell'itinerario, dunque si viene riportati alla finestra del percorso. Nell'ultimo caso, i dati vengono aggiornati e si viene riportato nella pagina dell'itinerario con numero di segnalazioni aggiornato.



2.8 Glossario

Adattabilità	La capacità del prodotto software di adattarsi a diversi ambienti specificati, senza utilizzare azioni o mezzi diversi da quelli forniti a tale scopo dal software stesso. [ISO/IEC 25000].
Activity diagram	Un diagramma di attività presenta visivamente una serie di azioni o flusso di controllo in un sistema simile a un diagramma di flusso o a un diagramma di flusso di dati. I diagrammi di attività vengono spesso utilizzati nella modellazione dei processi aziendali.
Analisi di sistema	Un insieme di attività, metodi, tecniche e strumenti incentrati sulla traduzione dei requisiti aziendali in requisiti di sistema.
Analisi dei requisiti	Una serie di compiti, attività e strumenti per determinare se i requisiti elicitati non sono chiari, incompleti, ambigui o contraddittori e successivamente il documentare i requisiti in forma di modello consistente.
Class diagram	Una tipologia di diagramma di struttura statico che descrive la struttura di un sistema, mostrandone le classi, i loro attributi, le relative operazioni (o metodi) e le relazioni tra le diverse classi.
Efficienza	La capacità del prodotto software di fornire performance appropriate, relativamente alla quantità di risorse utilizzate nell'ambito di condizioni prefissate [ISO/IEC 25000].
Entità	Un elemento o insieme di elementi aventi un'esistenza distinta e separata, benché non debba necessariamente essere un'esistenza materiale.
Errore	Un'azione umana che produce un risultato scorretto [IEEE 610].
Funzione	Una descrizione di "cosa" fa un sistema. Una funzione ha un corrispondente scopo implicito, ed è una parte fondamentale

	del sistema descritto: un sistema è costituito da attributi di funzione, attributi di prestazione, attributi di risorse (costo) e attributi di progettazione. Tutti gli attributi esistono rispettando le condizioni specificate. Una funzione può spesso essere scomposta in un insieme gerarchico di sotto-funzioni
Obiettivo	Uno stato o un risultato desiderato di un'attività intrapresa. Gli obiettivi dovrebbero essere misurabili e definiti nel tempo, in modo che il loro avanzamento possa essere monitorato.
Input	Un dato ricevuto da un componente o da un sistema da una sorgente esterna.
Output	Dati trasmessi da un componente o da un sistema ad una destinazione.
Progetto	Un insieme di attività coordinate e controllate con date di inizio e di fine, intraprese per realizzare un obiettivo che sia conforme ai requisiti specificati, compresi i vincoli temporali, di costo e di risorse.
Qualità	Il grado con il quale un componente, un sistema od un processo soddisfa i requisiti specificati e/o le esigenze e le aspettative dell'utente/cliente [IEEE 610].
Sequence diagram	Una rappresentazione strutturata UML di un comportamento, espresso come una serie di passaggi sequenziali nel tempo. Il sequence diagram è una sorta di interaction diagram che mostra come i processi operano l'uno con l'altro e in quale ordine.
Software	Programmi, procedure, possibilmente con associata documentazione e dati pertinenti all'operatività del sistema informatico [IEEE 610]
Sistema	Un insieme di elementi interagenti, organizzato per ottenere una specifica funzione od un gruppo di funzioni [IEEE 610].

Testabilità	La capacità di un prodotto software di essere testato efficacemente ed efficientemente [ISO/IEC 25000].
Use Case diagram	Un diagramma UML che mostra use case, attori e loro interrelazioni.
Use Case Text diagram	Documentazione testuale di uno Use Case diagram
Utente	Una persona che utilizza un prodotto software.

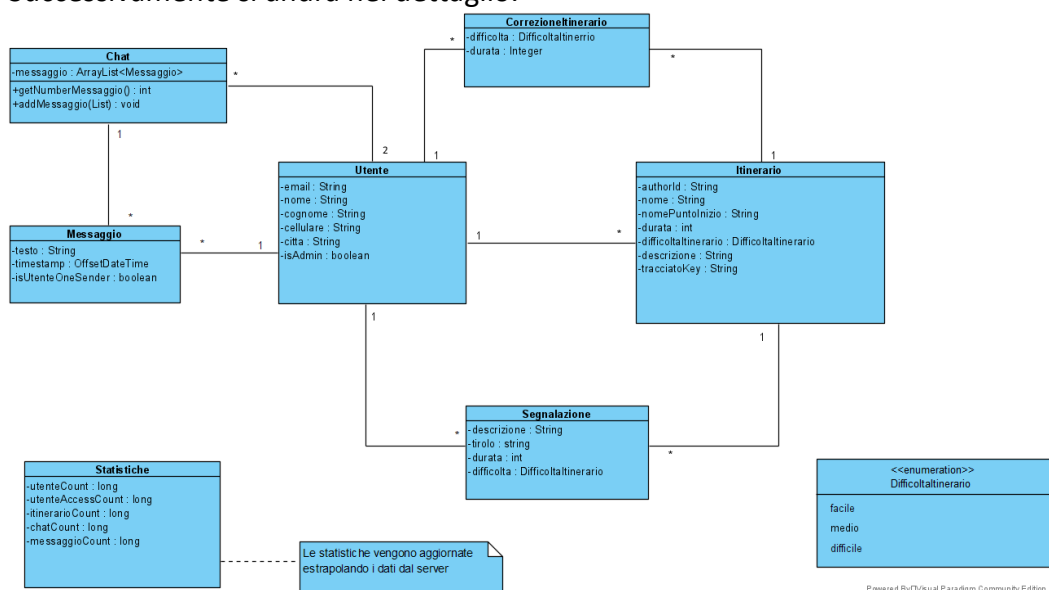
3.0 Modello di dominio

Il modello di dominio descrive le varie entità che fanno parte o hanno rilevanza nel sistema e le loro interazioni. Il loro uso è importante per fornire, a tutti coloro che dovranno sviluppare un sistema, concetti sulla logica e sulle funzioni del software.

I modelli sviluppati per questo progetto sono: Class diagram, Sequence diagram e Activity diagram.

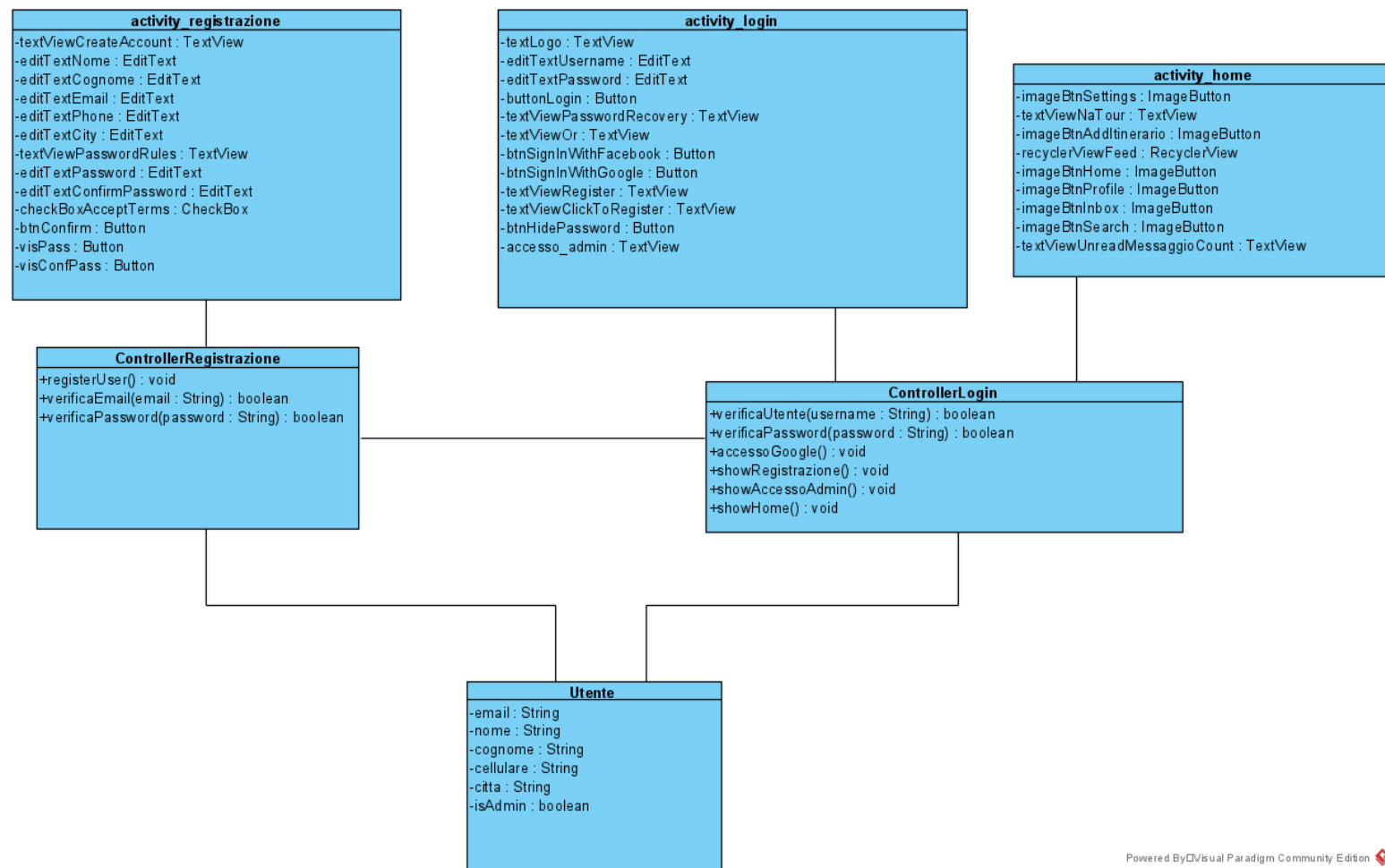
3.1 Class diagram

Il primo class diagram mostrato è il diagramma di entità relazione generale. Successivamente si andrà nel dettaglio.

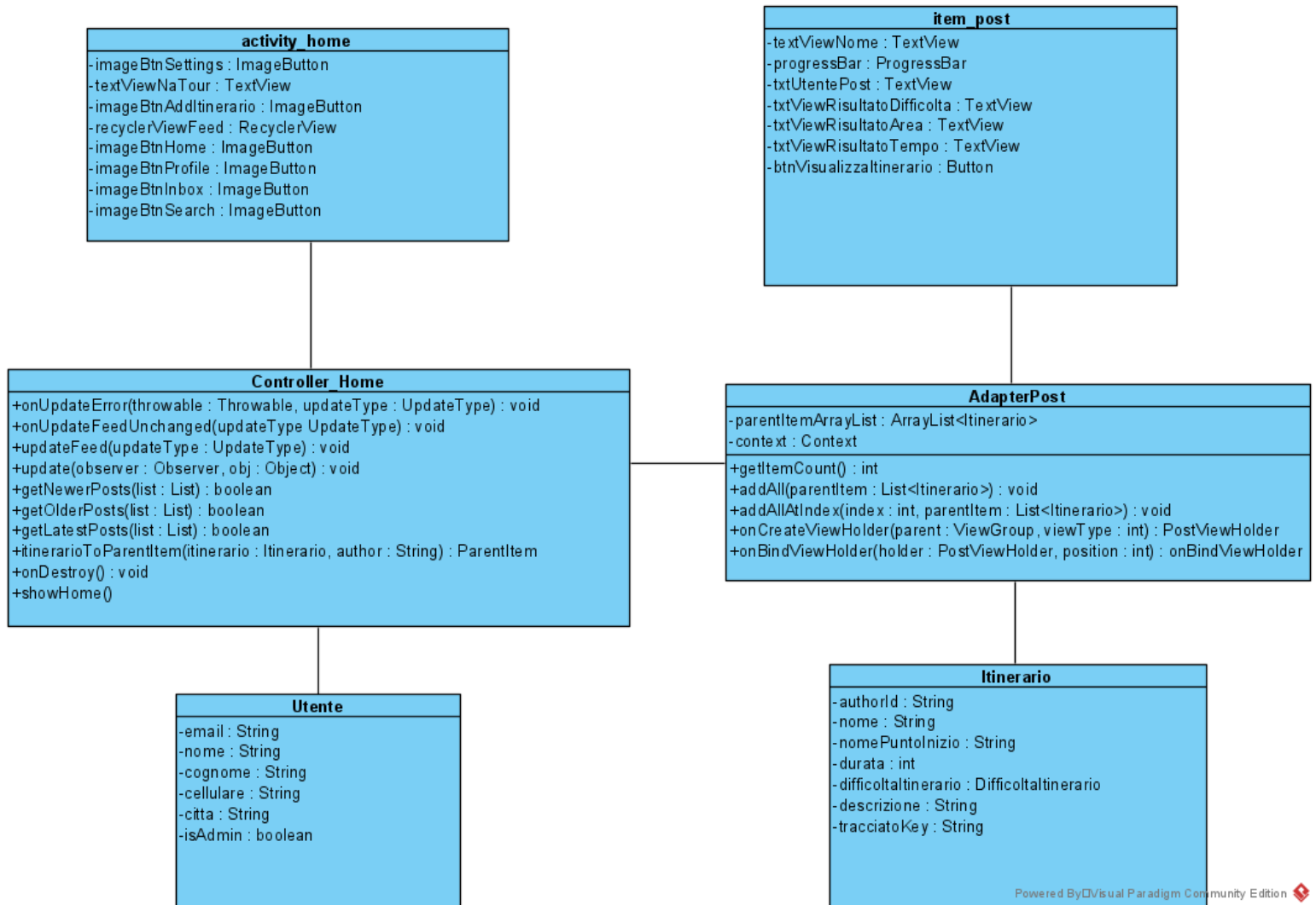


- Utente – Chat : Un utente ha da zero a molte chat. Chat ha 2 utenti associati.
- Chat – Messaggio: Chat ha da zero a molte chat. Un messaggio è associato ad una sola chat.
- Messaggio – Utente: Messaggio è associato ad un solo utente. Utente può avere da zero a molti messaggi.
- Utente – Itinerario: un utente può pubblicare da zero a molti tracciati. Un Itinerario è pubblicato da un solo utente.
- Itinerario – Segnalazione : un itinerario può avere da zero a molte segnalazioni. Una segnalazione è associata a solo un itinerario.
- Segnalazione – Utente: Una segnalazione è effettuata da un solo utente. Un utente può fare da zero a molte segnalazioni.
- Correzioneltinerario – Utente: Una correzione è effetuata da un solo utente. Un utente può fare da zero a molte correzioni.
- Itinerario – Correzioneltinerario: Un itinerario può avere da zero a molte segnalazioni. Una correzione può essere associata ad un solo itinerario.

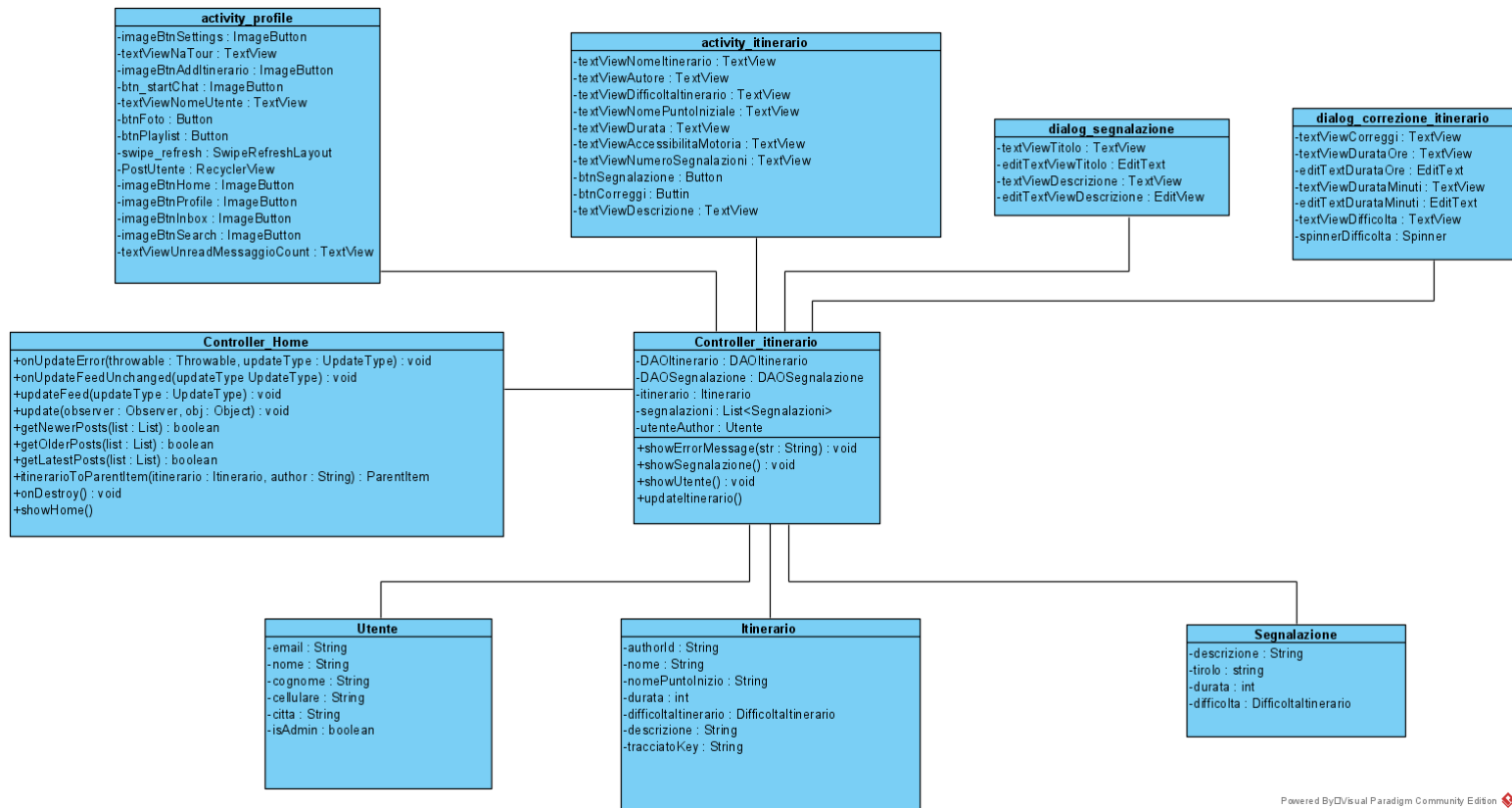
3.1.1 Login e registrazione



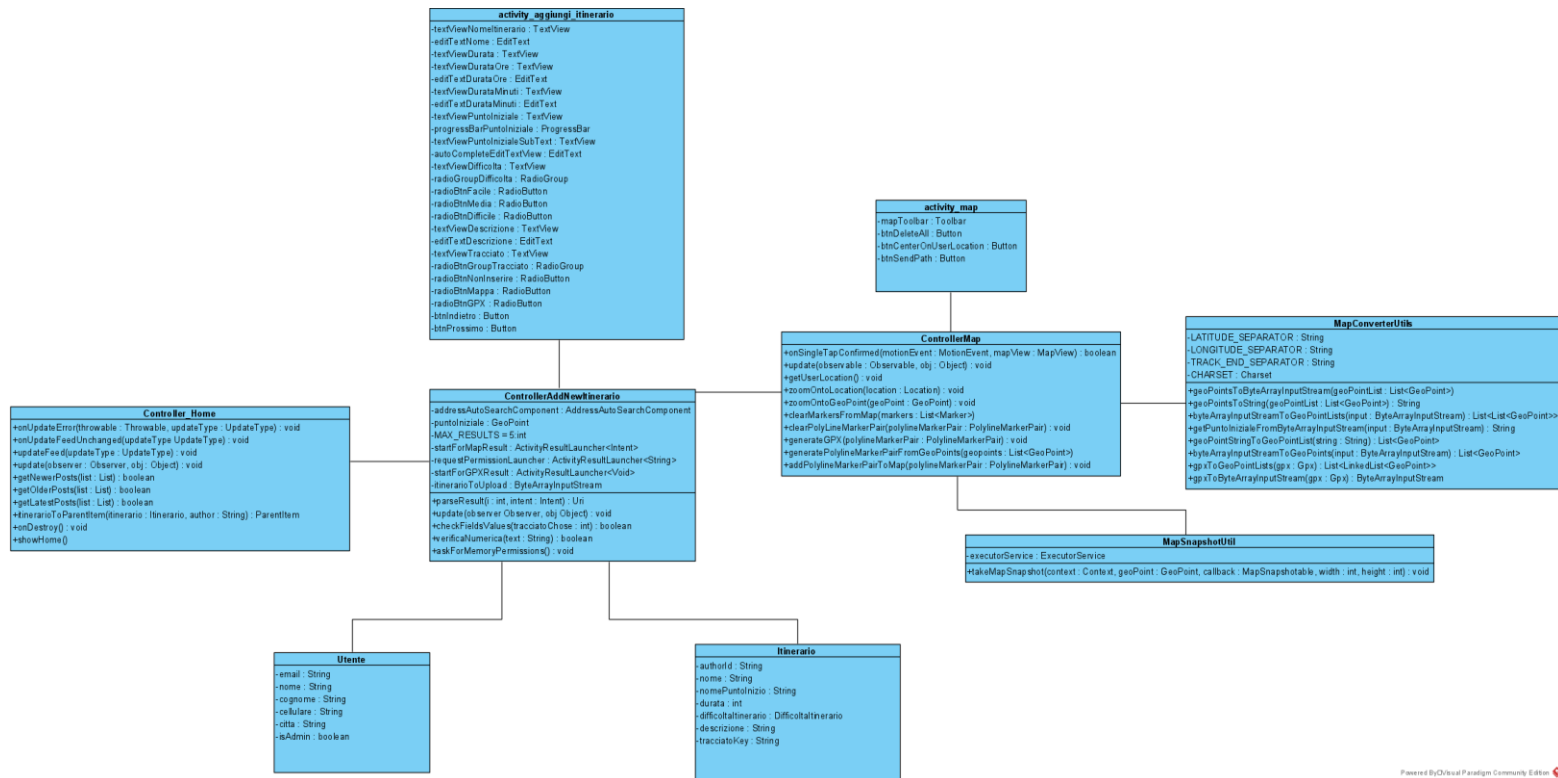
3.1.2 Visualizzazione post



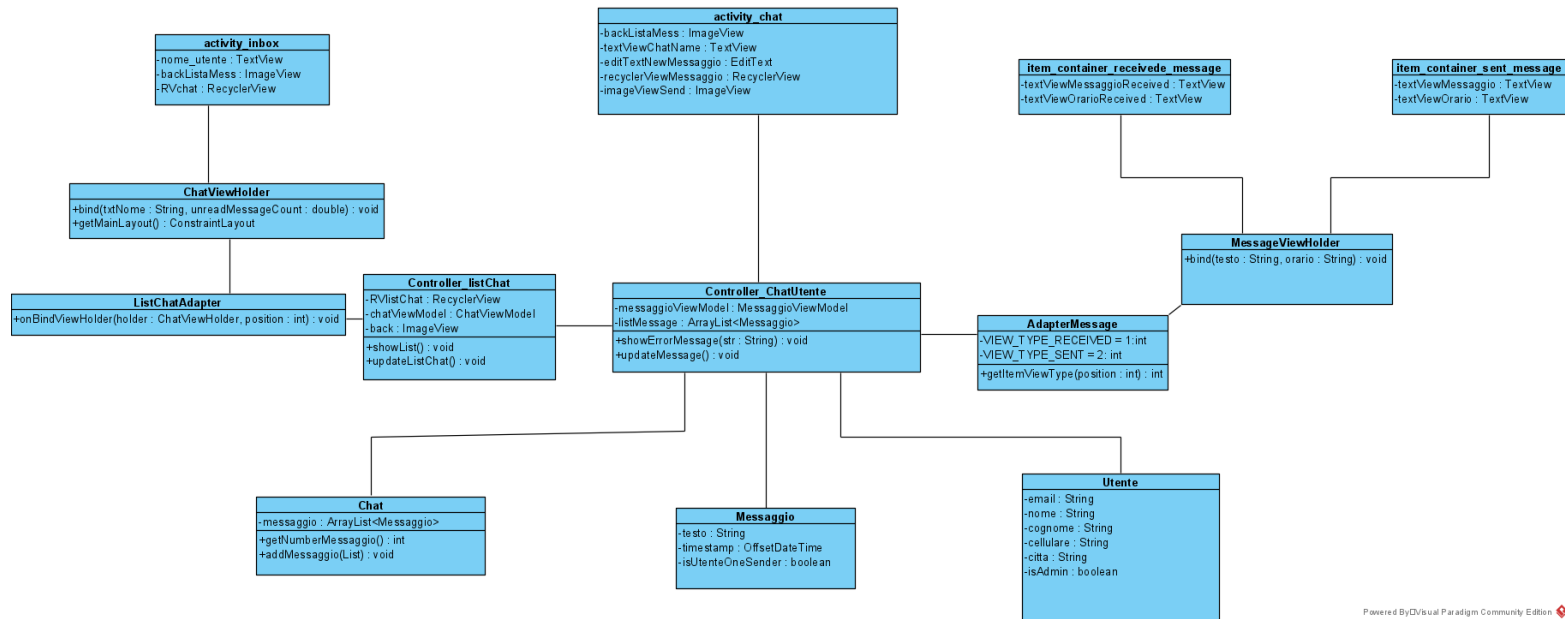
3.1.3 Visualizza itinerario



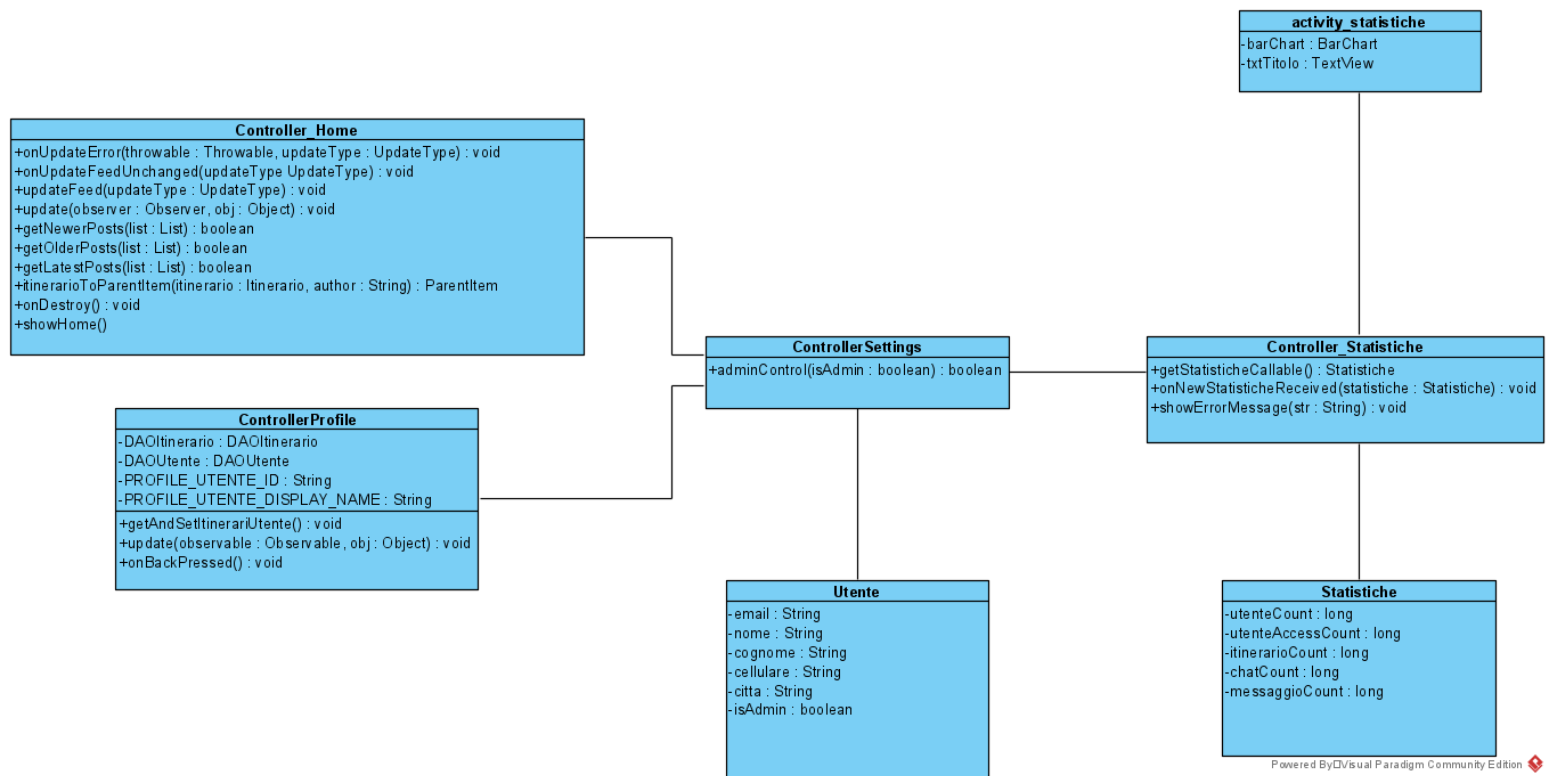
3.1.4 Inserimento itinerario



3.1.5 Messaggi



3.1.6 Statistiche



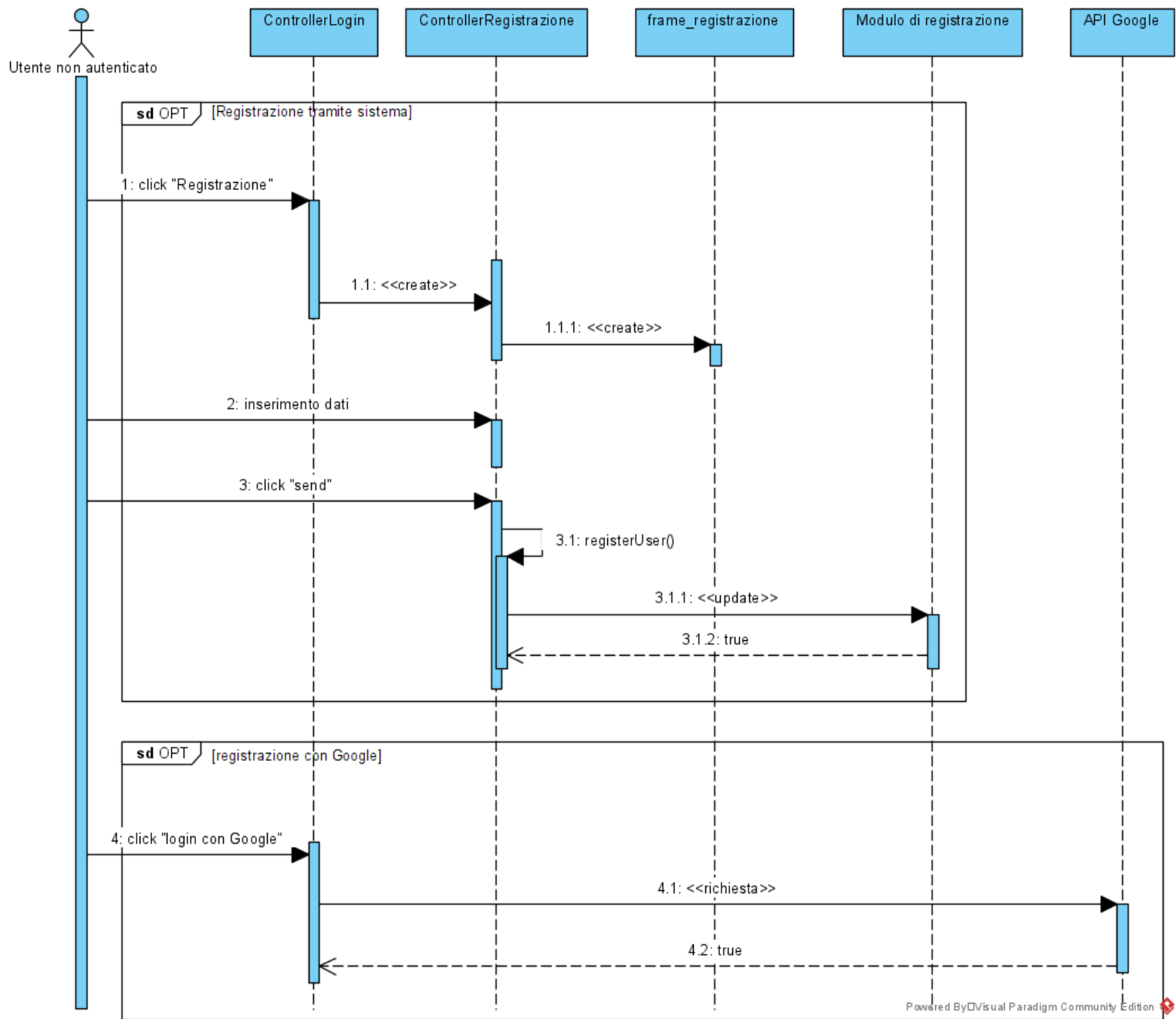
3.2 Sequence diagram

I Sequence diagram illustrati di seguito descrivono scenari di casi d'uso significati. I due scenari mostrati sono la registrazione e l'inserimento di un itinerario, considerati eventi fondamentali in questo progetto.

3.2.1 Registrazione

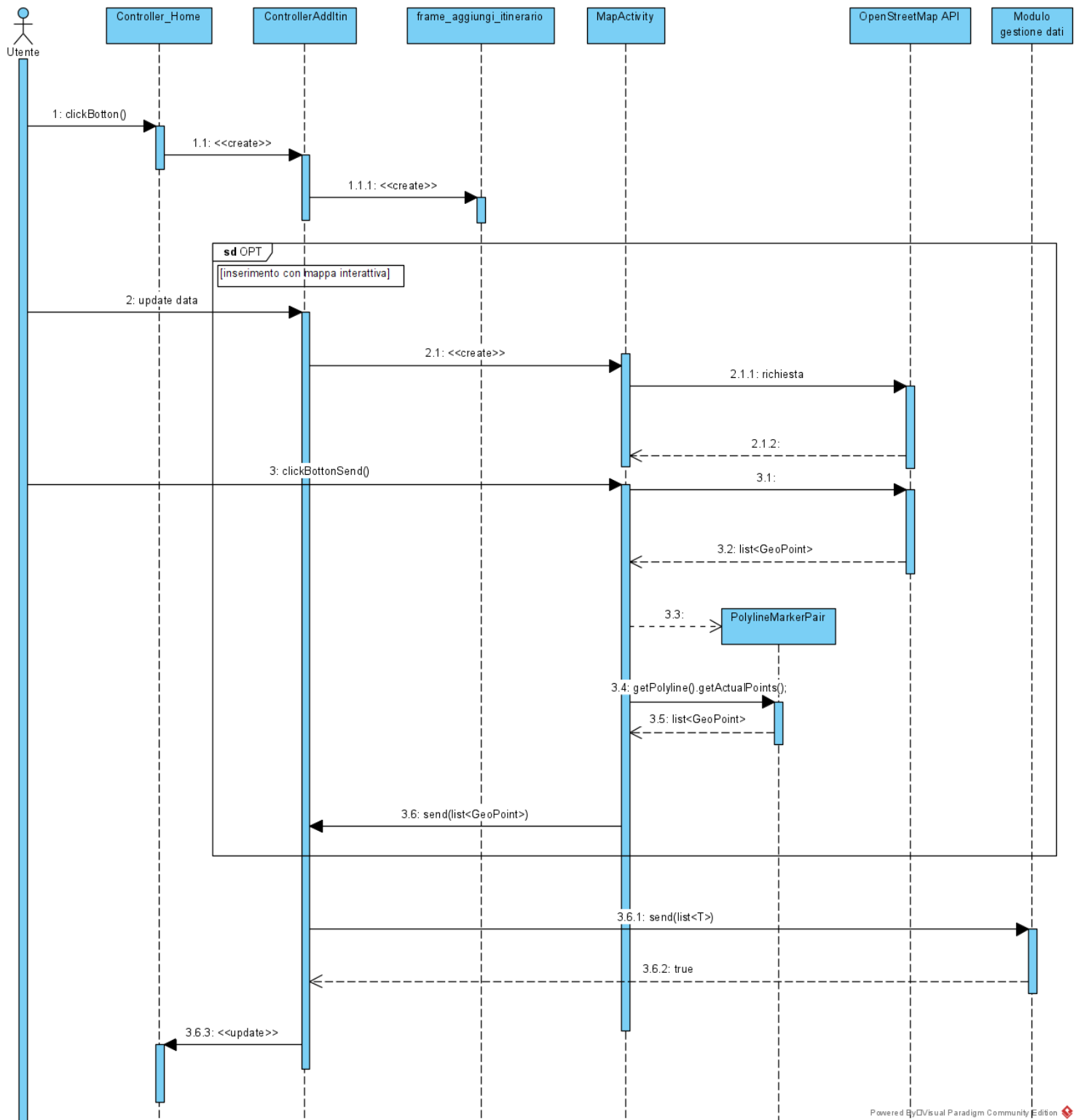
L'attore è l'utente non registrato. In base al tipo di registrazione scelto, si hanno due scenari differenti. Nel primo caso l'utente decide di registrarsi tramite il sistema. Viene aperta la schermata di registratore dove l'utente potrà interagire inserendo i dati obbligatori. Dopo i dovuti controlli, i dati vengono salvati su database.

Nel secondo caso, l'attore decide di registrarsi tramite l'account di Google. In tal caso, viene inviata una richiesta all'API di Google.



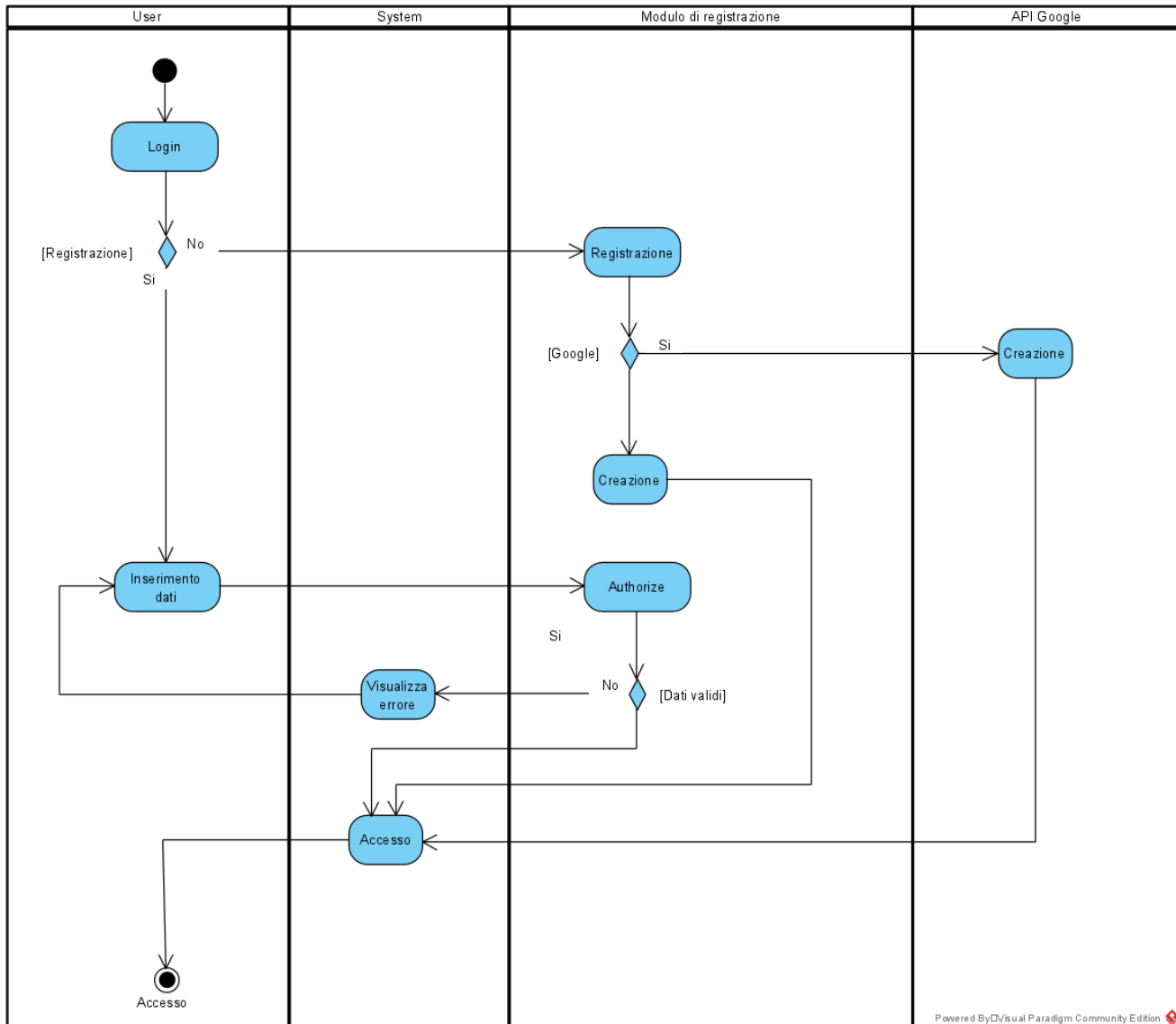
3.1.2 Inserimento itinerario

L'inserimento dell'itinerario risulta essere più complesso del Sequence Diagram precedente. L'attore è l'utente (autenticato). Dopo aver inserito i dati del percorso, l'utente può scegliere di inserire il tracciato attraverso una mappa interattiva. Nell'intraprendere questa scena, vengono effettuate varie richieste all'API OpenStreetMap che, in base alla richiesta, restituisce liste di valori. Alla fine del ciclo, i dati vengono salvati su PostgresSql.



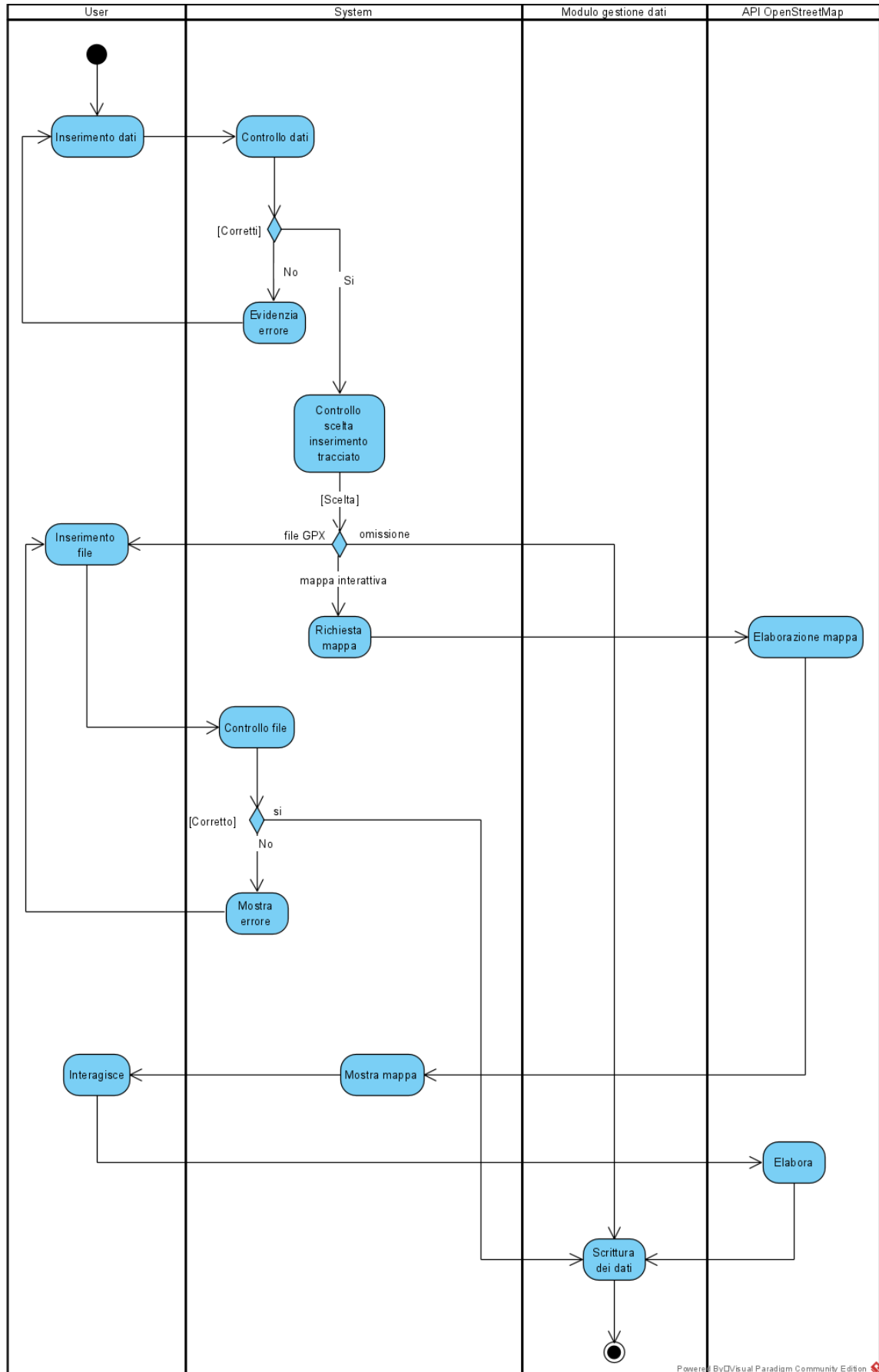
3.3 Activity diagram

L'activity diagram è stato progettato per le funzioni principali del sistema.
L'immagine seguente mostra le azioni per l'autenticazione o registrazione di un utente.



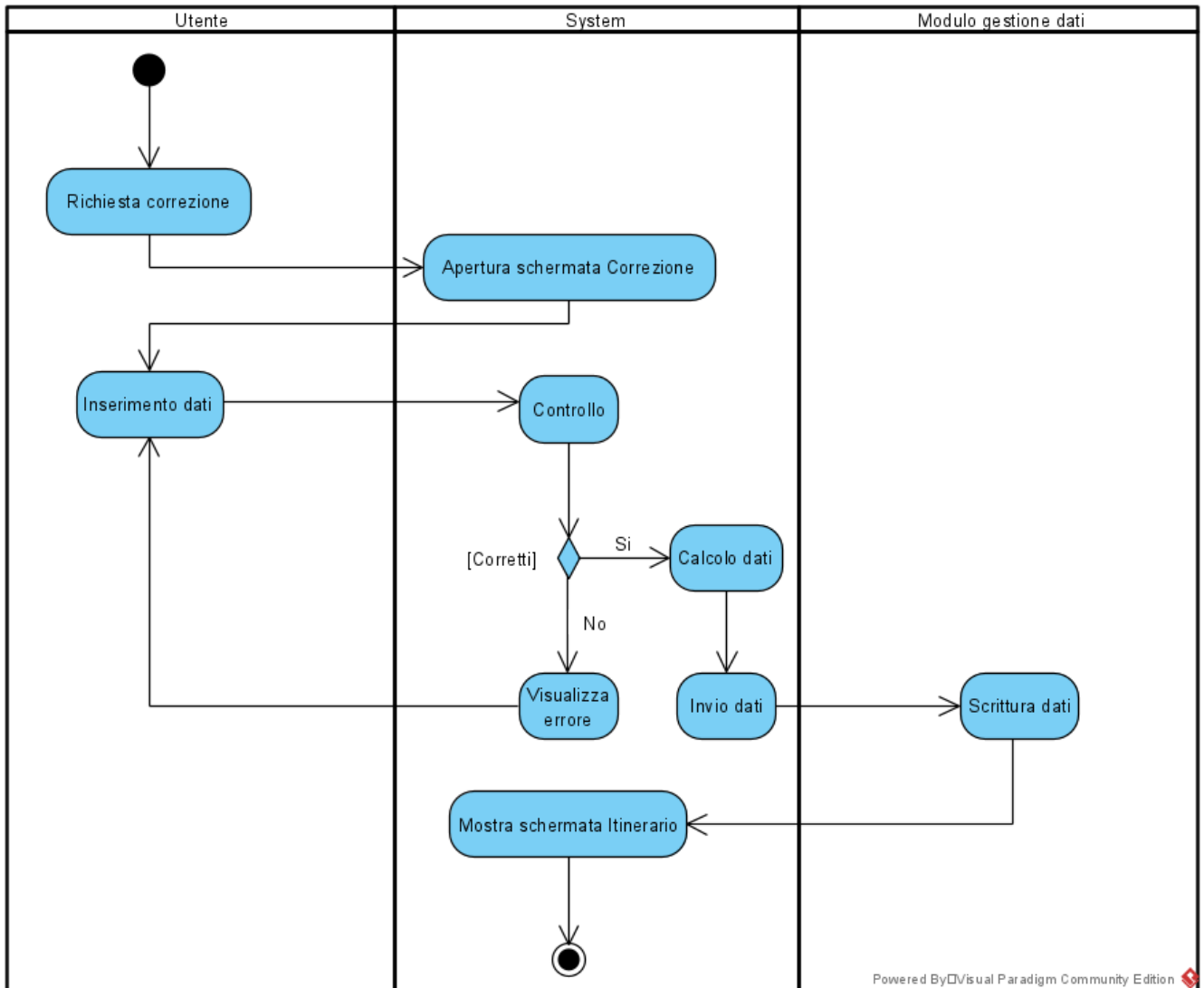
Sono presenti quattro attori: utente, sistema, modulo di registrazione e API.
Durante la registrazione l'utente dovrà inserire i dati obbligatori, verrà effettuato un controllo, e verrà salvato all'interno del Database. Un'alternativa è l'accesso tramite Fb/Google, in tal caso viene gestito dal dovuto API.

La prossima immagine illustra l'inserimento di un itinerario nel sistema.



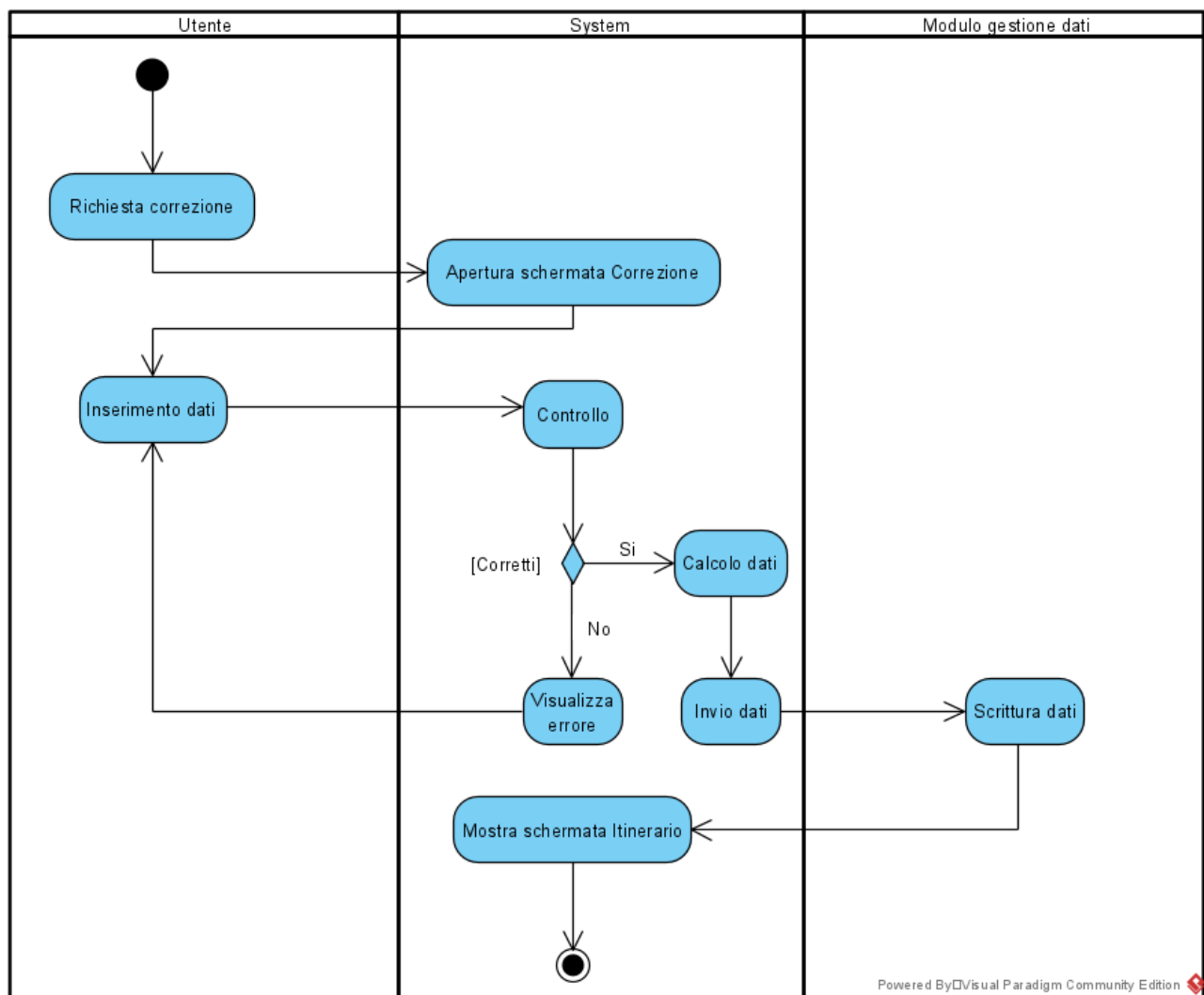
Gli attori sono l'utente, il sistema, postgres e API (OpenStreetMap). Il sistema effettuerà vari controlli nell'inserimento dei dati e dell'estensione del file nel caso venga richiesto. L'API verrà interpellato nel caso in cui l'utente decida di inserire un tracciato tramite mappa interattiva. Infine i dati verranno salvati su PostgreSQL.

La terza immagine è inerente alla segnalazione di un itinerario.

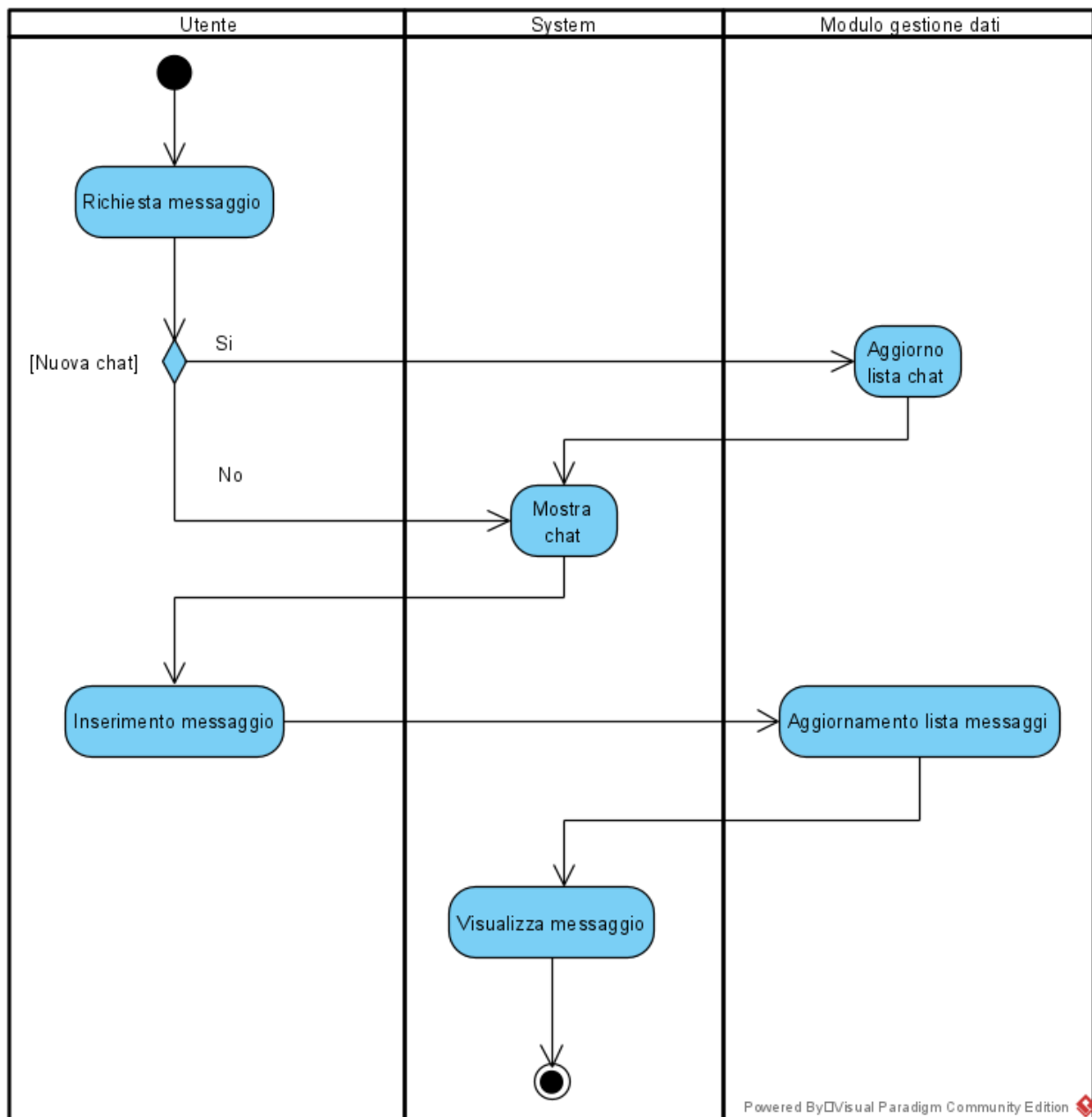


In questo caso sono solo tre gli attori: utente, sistema e PostgreSQL. Alla richiesta di una segnalazione, il sistema effettuerà un controllo dei dati e successivamente salvato sul Db.

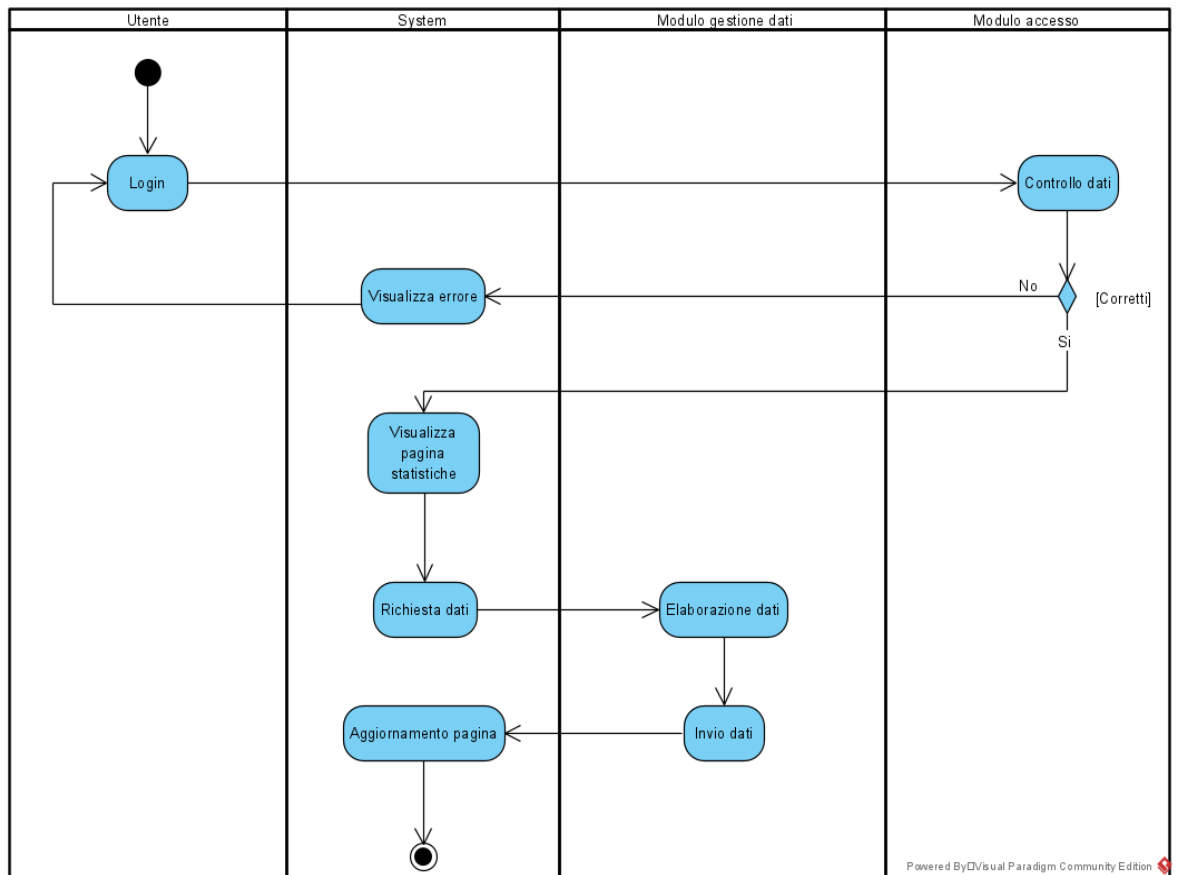
La correzione dell'itinerario è analoga.



Un'altra funzione che può svolgere l'utente è l'invio di messaggi.



Alla richiesta di un messaggio, il sistema controlla se l'utente è già presente nella lista chat. Lo scambio di messaggio avviene tramite un salvataggio testuale all'interno del database e mostrato in tempo reale all'utente.



Infine, l'ultima immagine mostra le azioni di un amministratore.

L'amministratore dopo aver effettuato l'accesso, con dovuti controlli da parte del modulo d'accesso, visualizza tutte le statistiche del sistema. Le statistiche vengono aggiornate in tempo reale.

4.0 Design del sistema

Il System Design definisce la struttura hardware e software del sistema. Utilizzato per identificare gli obiettivi e suddividere il progetto (decomporre il sistema in sottosistemi) cercando di essere il più coerenti possibile e definendo un accoppiamento debole.

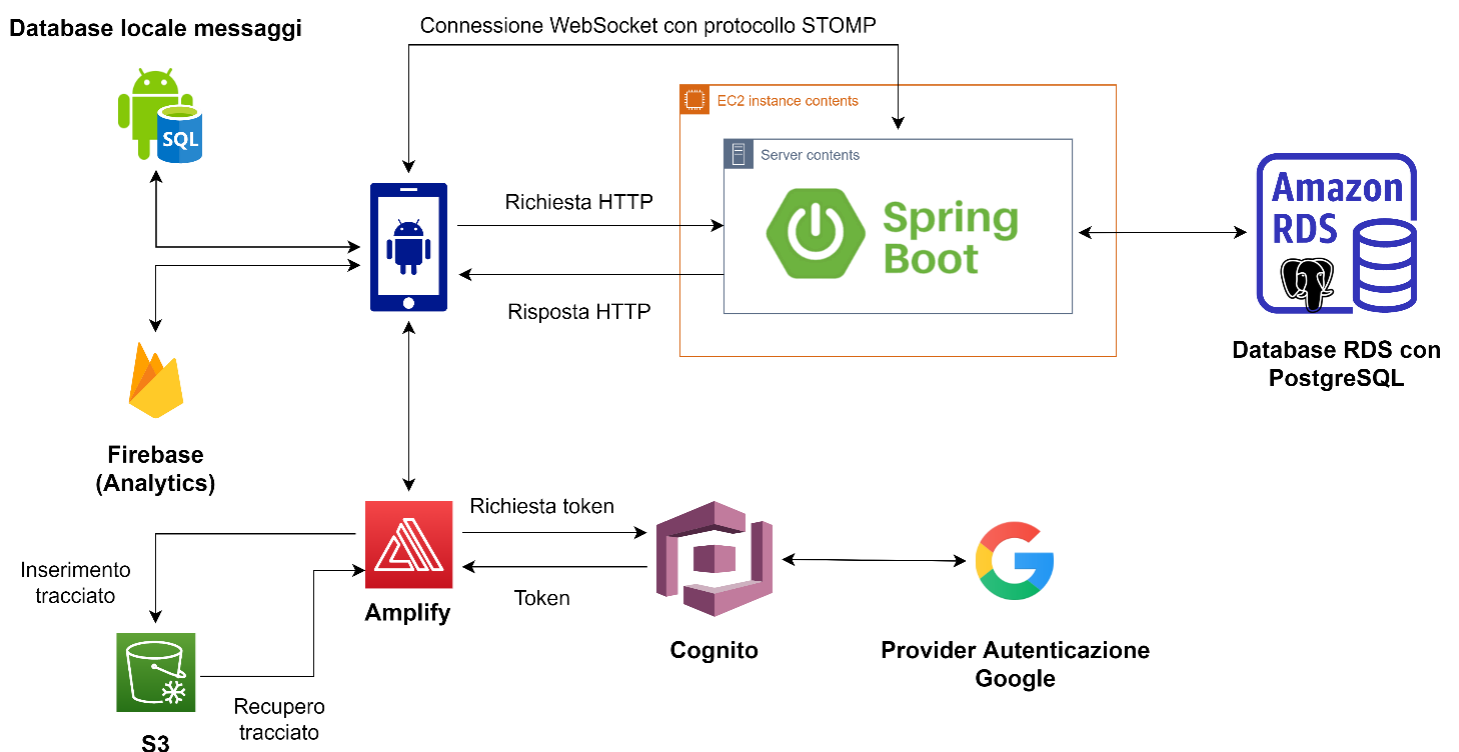
4.1 Architettura

Tra le varie architetture possibili (client-server, repository, layered, three tier, n thier) si è scelto di adottare l'architettura client-server, dove viene utilizzato il protocollo **HTTP** per comunicare tra le parti. Specificamente per la parte relativa alle chat, invece, è stato impiegato il protocollo di comunicazione **STOMP**, usato con WebSocket.

Questo sistema ha due moduli. Il server esegue le operazioni necessarie per svolgere le varie richieste dell'utente. Permette la gestione di banche dati, effettua richieste di inserimento e aggiornamento dati e visualizza informazioni in tempo reale.

Il client può effettuare alcune operazioni, per fare ciò richiede un terminale con capacità elaborative (è sufficiente uno smartphone). Il client gestisce l'interfaccia utente dell'applicazione NaTour, verifica i dati inseriti e provvede ad inviare al server le richieste formulate dall'utente.

La logica dell'architettura client-server è riportata nell'immagine seguente.



Il frontend è stato realizzato seguendo il pattern **MVC** (Model-View-Controller), mentre il backend è stato realizzato seguendo le linee guida dello sviluppo nel framework **Spring**, sfruttando alcuni dei suoi pilastri fondamentali (dependency injection, inversion of control, beans) per rendere l'applicazione il più modulare possibile, favorendo il mantenimento di essa.

Andremo ora ad analizzare specificamente, pezzo per pezzo, le tecnologie utilizzate.

4.1.1. Tecnologie usate

Servizio di public cloud computing

Si è fatto uso di servizi di public cloud computing per garantire la massima scalabilità della nostra applicazione. In particolare, sono stati usati i servizi di **Amazon Web Services (AWS)**, proprietà di Amazon, poiché essi sono della qualità da noi ricercata ed offrono le loro risorse a prezzi competitivi.

In particolare, si è fatto uso di:

- **EC2**: fornisce capacità di calcolo scalabile, con ambienti virtuali liberamente configurabili e scalabili a nostro piacimento, attraverso criteri ben definiti. Di tipo **Infrastructure as a Service (IaaS)**, è usato per portare in vita il nostro backend.
- **RDS**: è una raccolta di servizi gestiti che rende semplice impostare, operare e scalare i database nel cloud. Permette l'uso di **PostgreSQL**, l'engine scelto per il nostro database, ed in particolare della sua estensione **PostGIS**, necessaria per la funzionalità di ricerca geografica degli itinerari.
- **Cognito**: permette di aggiungere strumenti di registrazione degli utenti e di accesso e controllo degli accessi alle app Web e per dispositivi mobili. Cognito permette di ridimensionare le risorse per milioni di utenti e supporta l'accesso con provider di identità social quali **Apple**, **Facebook**, **Google** e **Amazon** e provider di identità aziendali tramite **SAML 2.0** e **OpenID Connect**.
- **S3 (Simple Storage Service)**: è un servizio di archiviazione di oggetti che offre scalabilità, disponibilità dei dati, sicurezza e prestazioni all'avanguardia nel settore. Usato per conservare i tracciati, in quanto quest'ultimi sono **BLOB**.

Google Analytics

Parte della piattaforma di sviluppo **Firebase**, **Google Analytics** è una soluzione totalmente gratuita per monitorare l'utilizzo dell'applicazione da parte dei nostri utenti. Fornisce statistiche su quali parte dell'applicazione generano più engagement, sulla fidelizzazione, sugli utenti che usano l'app. Inoltre, abbiamo configurato il modulo in maniera che fornisca anche statistiche sugli itinerari più visualizzati dagli utenti, aiutandoci a comprendere quali sono i contenuti migliori da mostrare ad essi. Assieme ad Analytics usiamo anche il modulo **Crashlytics**, utile per monitorare eventuali errori e crash durante l'utilizzo dell'applicazione.

Spring Boot

Una versione del noto framework di programmazione Java **Spring** che richiede minima configurazione per essere utilizzato. Ci ha permesso di sviluppare il nostro backend rapidamente ed in totale sicurezza, rivelandosi affidabile, stabile, ed efficiente.

Spring ci ha permesso, inoltre, di implementare la parte del server di messaging con pochissime linee di codice, grazie al suo supporto ai WebSocket ed al protocollo STOMP.

Tecnologie e moduli client

Sistemi operativi e linguaggio di programmazione

Per il client si è deciso di sviluppare un'applicazione nativa, realizzata per dare la miglior esperienza possibile ad i nostri utenti. Il tutto è stato pensato per **Android**, il sistema operativo mobile più utilizzato al mondo. Il linguaggio di programmazione scelto, invece, è stato **Java**, usato nel settore da decenni, semplice da usare e conosciuto dalla stragrande maggioranza dei programmatori.

RxJava

RxJava è un'implementazione per la JVM delle **API ReactiveX**. Il suo scopo è di rendere semplice la creazione di codice asincrono, astruendo dal low-level e le sue varie problematiche. Usa il pattern **Observer**.

OSMDroid

OSMDroid è una libreria per Android che mira a fornire funzionalità utili per le applicazioni che vogliono fare uso di mappe. Usa **OpenStreetMap**, da cui il nome (OpenStreetMap Android). In particolare, è stata scelta questa libreria rispetto a quella di Google Maps alla luce della libertà d'uso che se ne può fare.

Retrofit

Retrofit è un client HTTP per la JVM. Trasforma le API del nostro backend in interfacce Java, ed esegue richieste con pochissimo codice.

StompProtocolAndroid

StompProtocolAndroid è un client STOMP (su WebSocket) per Android. Insieme a Spring (impiegato nel backend), ha reso semplicissimo lo scambio di messaggi tra client e server.

Librerie Android Jetpack

Jetpack è un insieme di librerie che funziona consistentemente tra diverse versioni di Android, mirato a rendere semplice lo sviluppo di alcune funzionalità incoraggiando gli sviluppatori a seguire “best practices” e ridurre il codice “boilerplate”. In particolare, si noti l’uso delle seguenti, impiegate per fornire funzionalità aggiuntive:

- **Room:** Fornisce un layer di astrazione su SQLite per Android. Usata per memorizzare le chat dell’utente.
- **Work:** Permette di svolgere operazioni in background in maniera affidabile, rispettando il sistema operativo Android. Usata per il caricamento degli itinerari.
- **Preference:** Permette di gestire facilmente le impostazioni di un’applicazione attraverso la creazione di una UI semplice da usare per l’utente, senza aver bisogno di preoccuparsi dell’archiviazione.

4.2 Diagramma delle classi di design

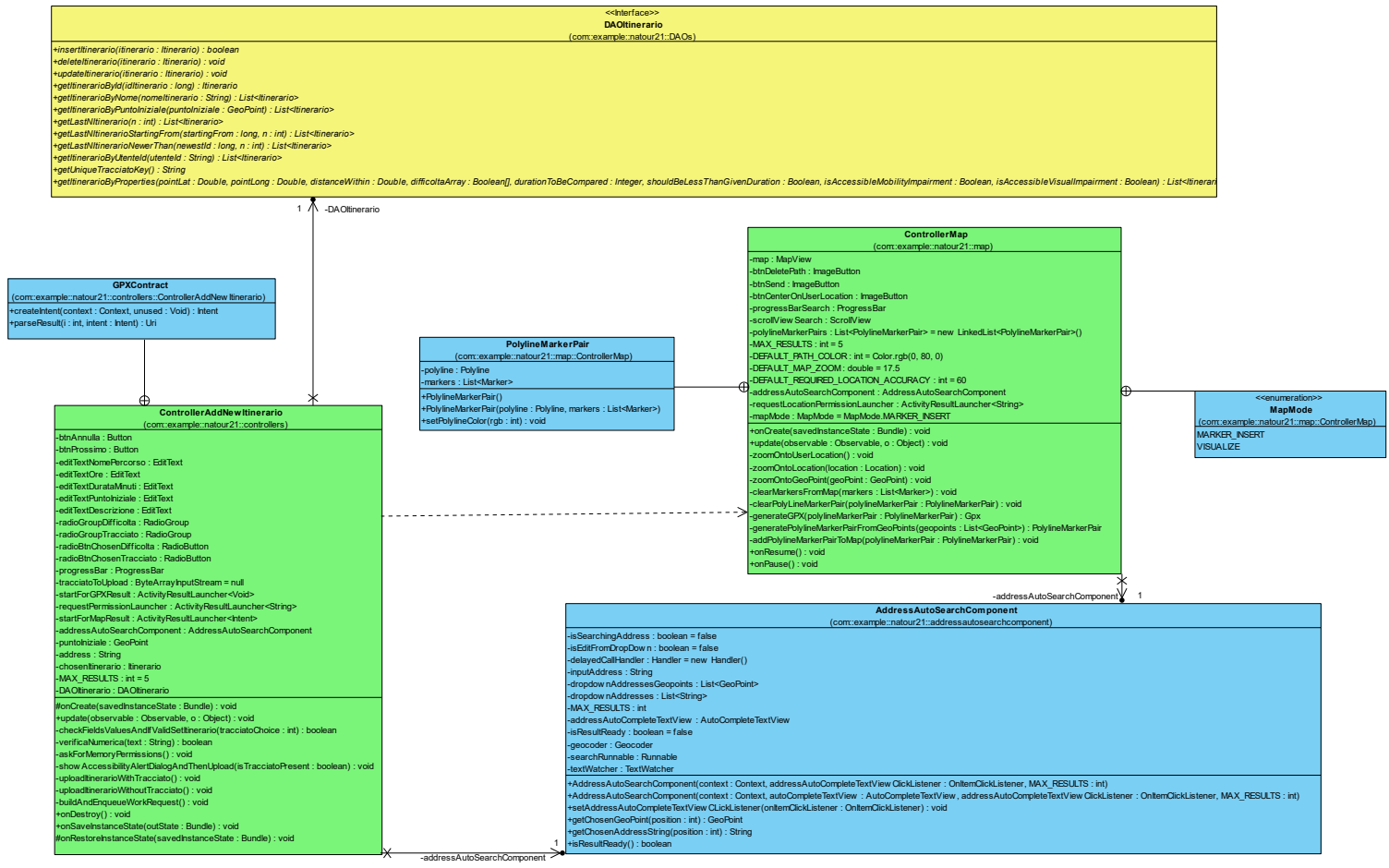
I class diagram di seguito illustrati mostrano lo sviluppo del sistema.

Essendo una struttura molto ampia, i class diagram sono stati suddivisi in sotto-diagrammi tale da permettere una visualizzazione migliore.

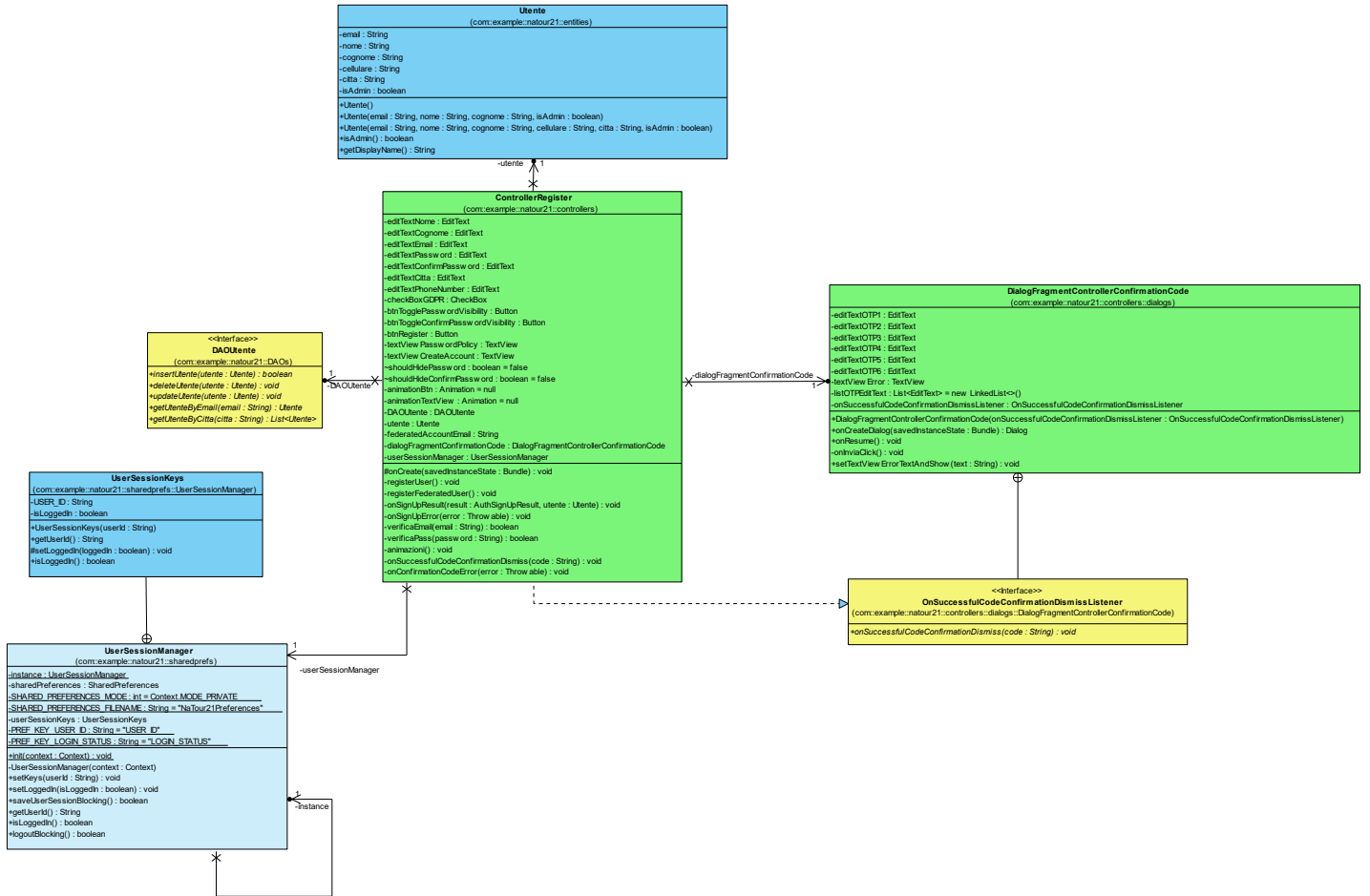
4.3 Diagramma di sequenza di design

Seguono quindi i diagrammi delle classi di design relativi ai requisiti funzionali:

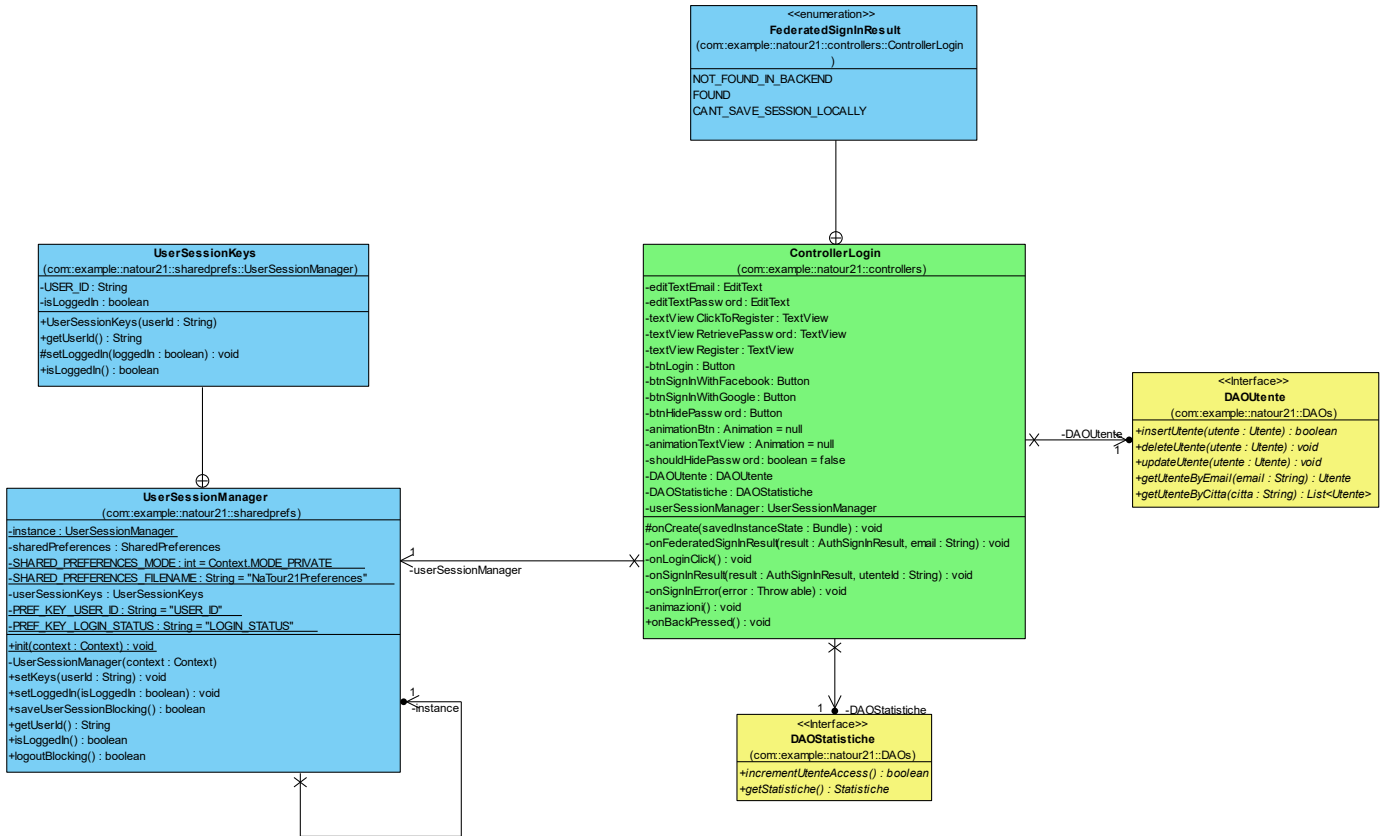
4.3.1 Aggiunta nuovo itinerario



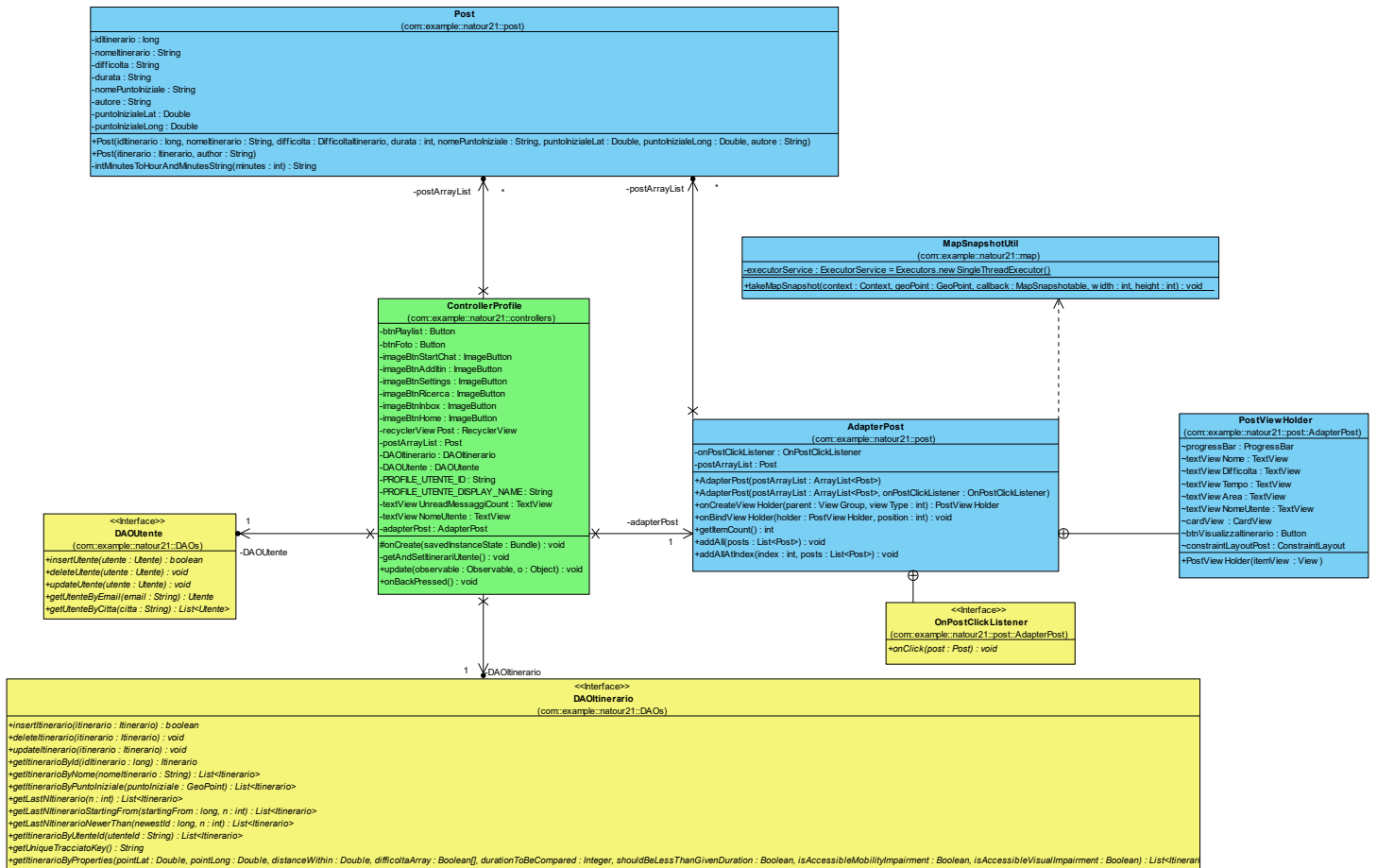
4.3.2 Registrazione



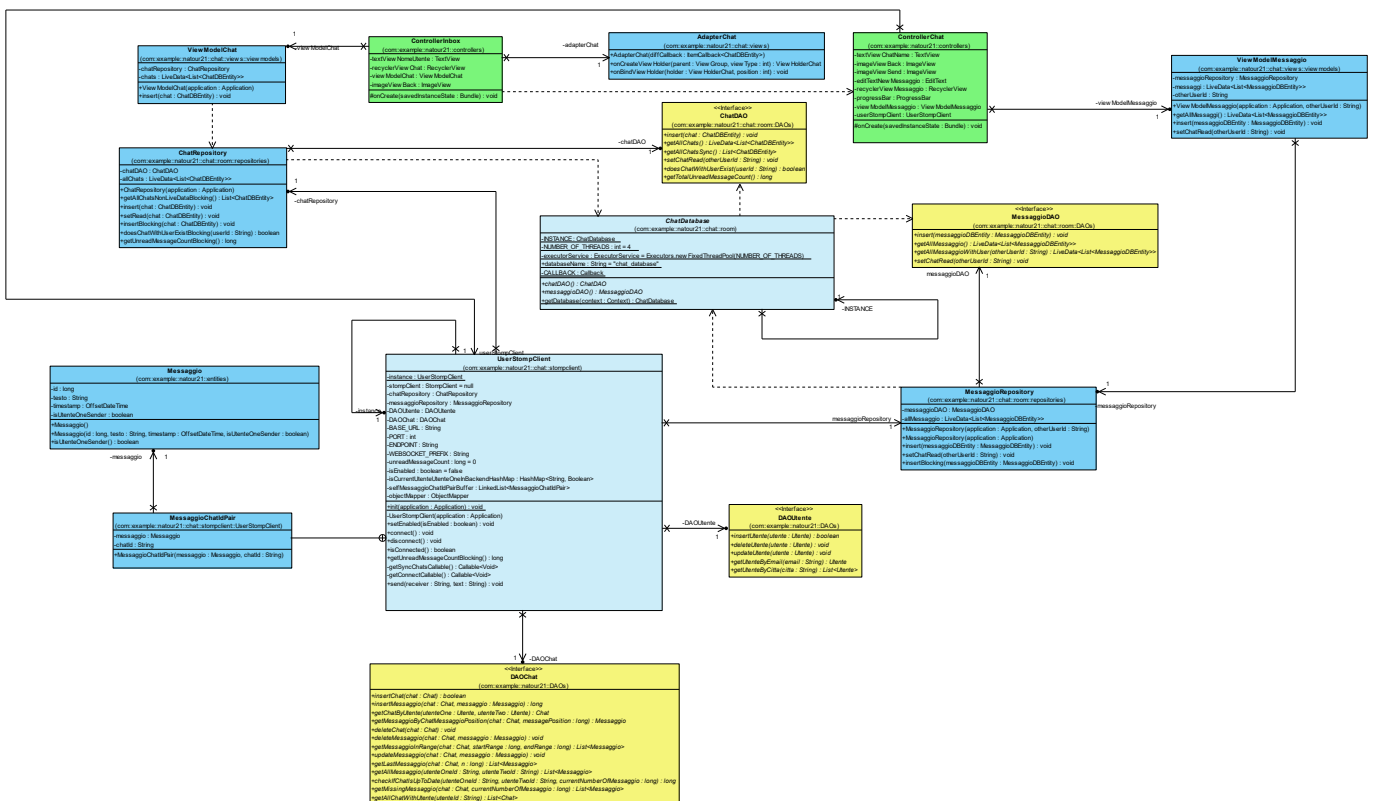
4.3.3 Login



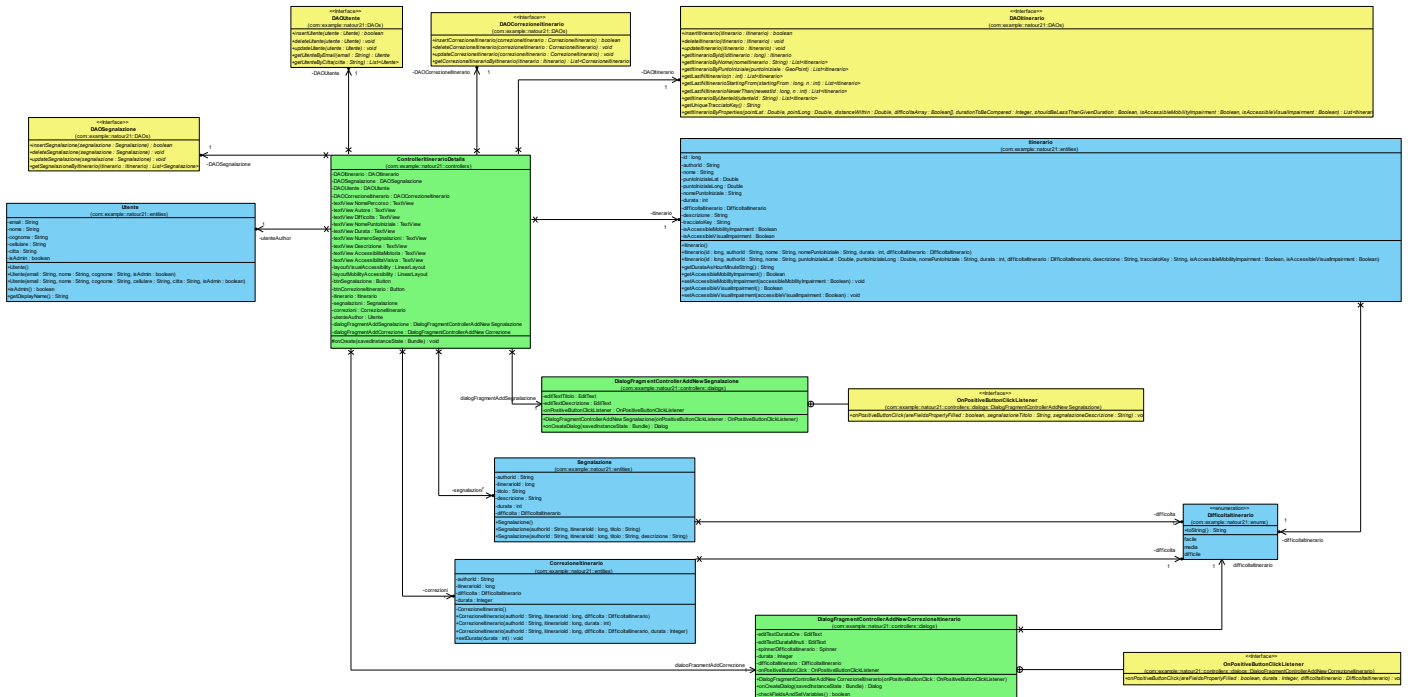
4.3.4 Visualizzazione profilo



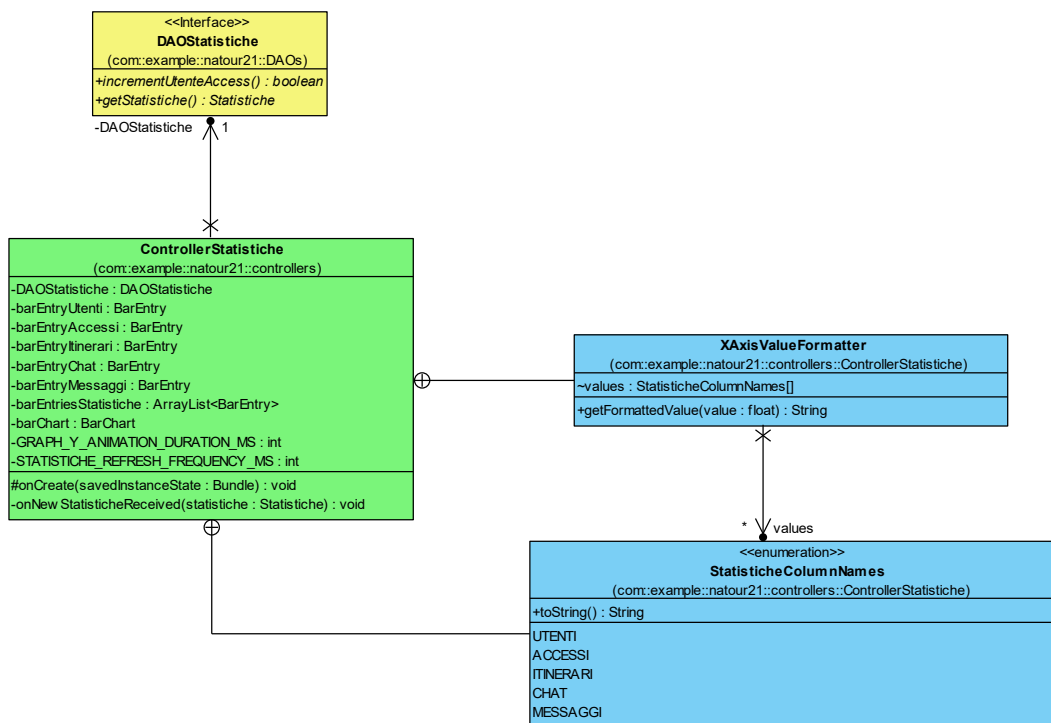
4.3.5 Chat



4.3.6 Aggiunta di segnalazione/correzione



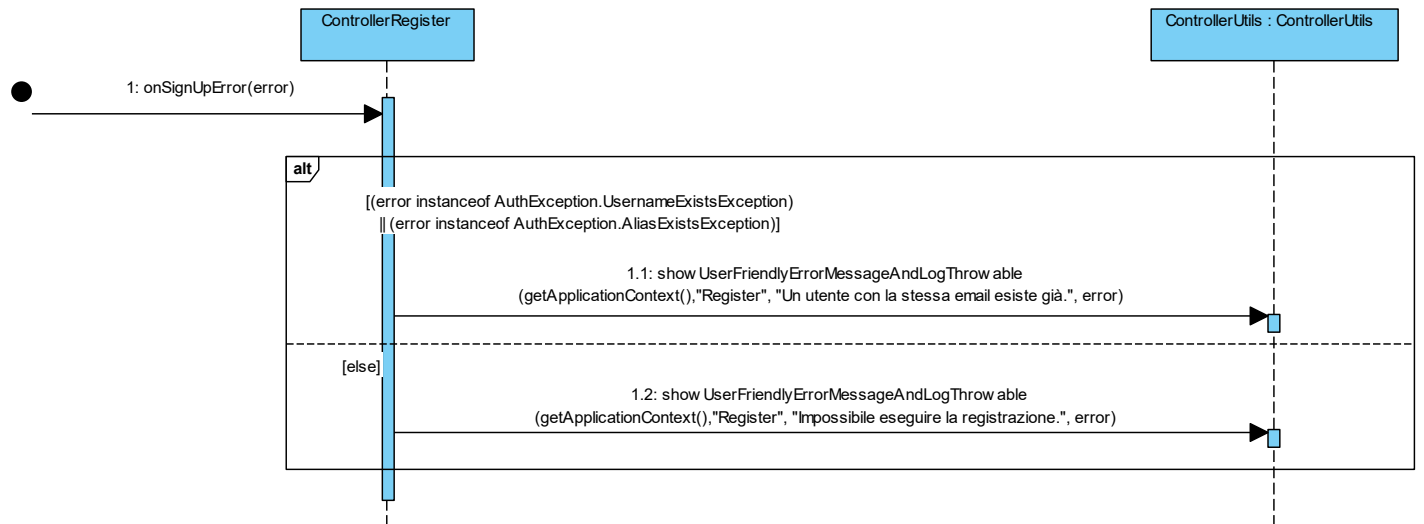
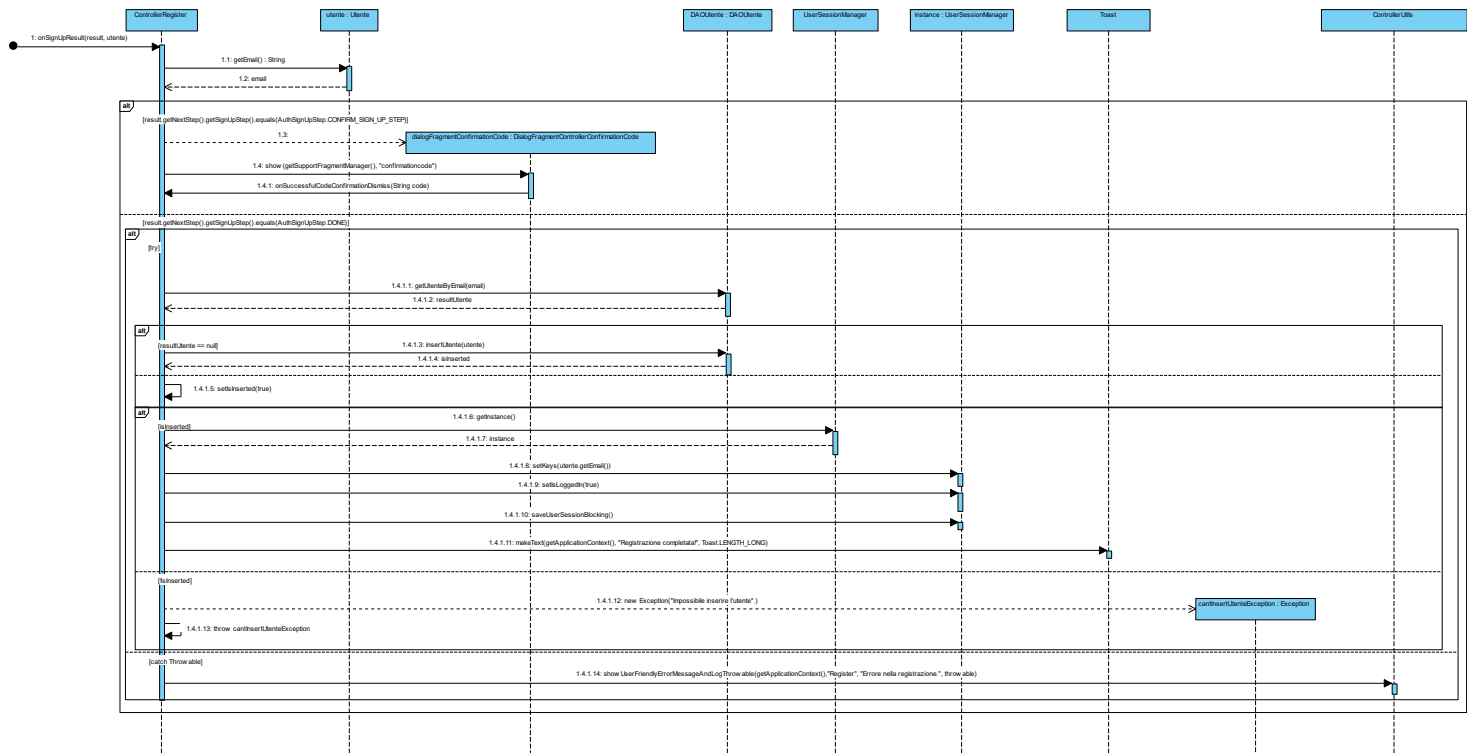
4.3.7 Visualizza statistiche

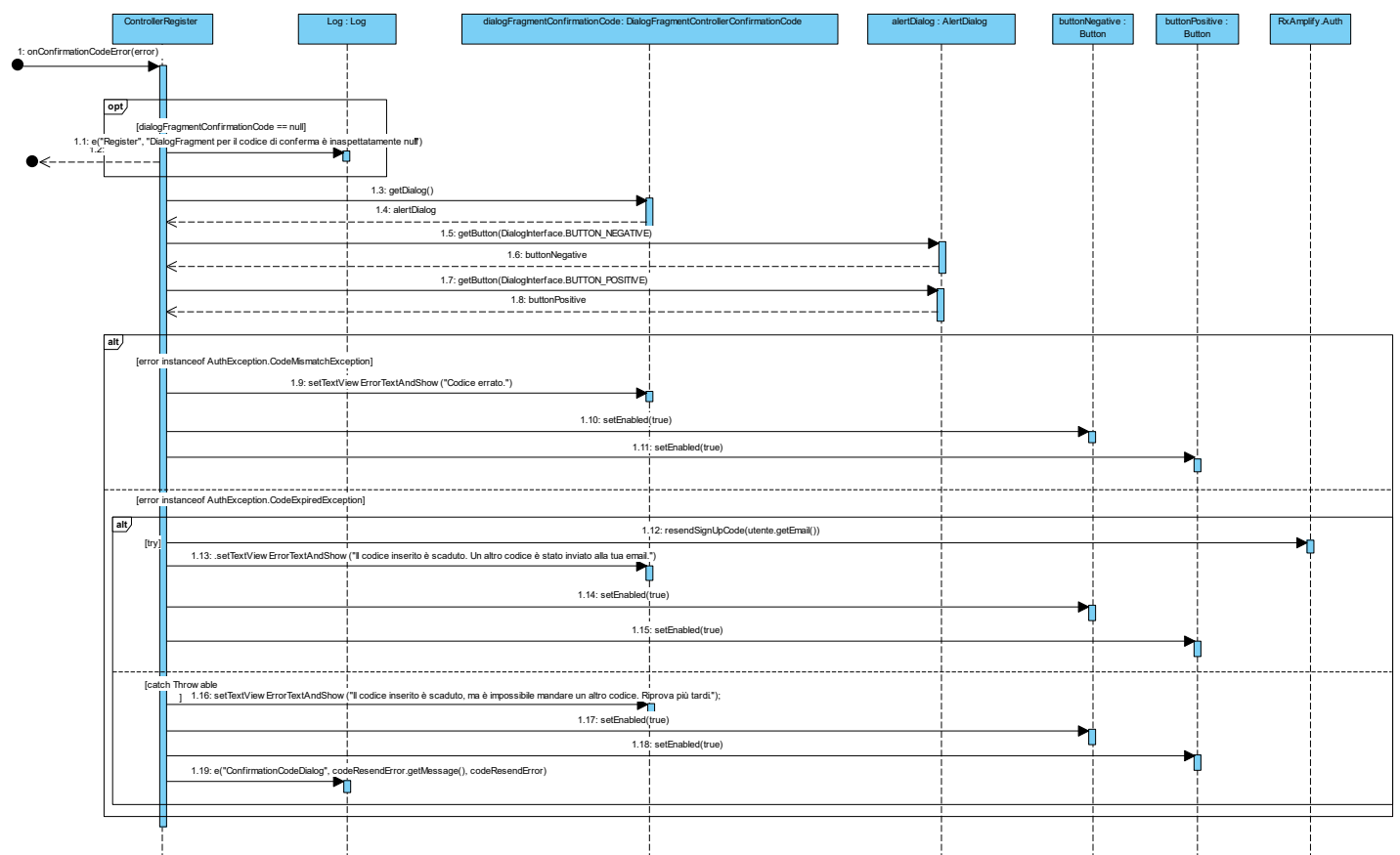
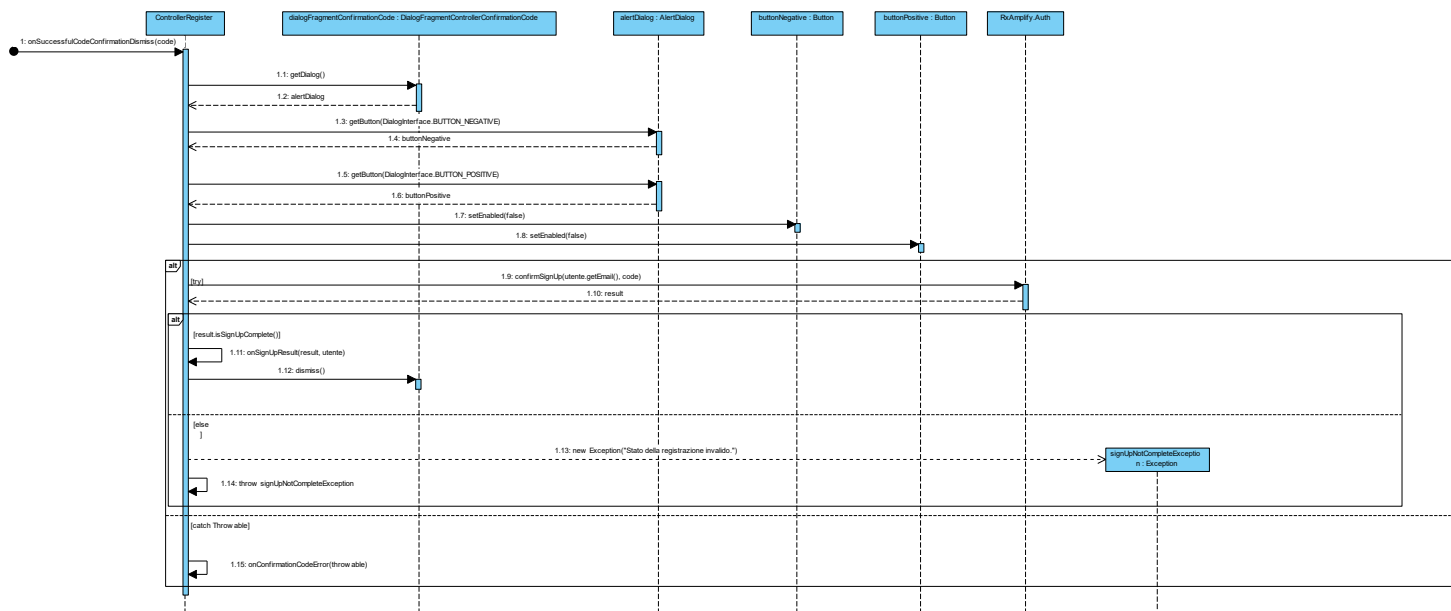


4.4 Diagrammi di sequenza di design



Per semplicità, sono stati omessi dettagli implementativi legati all'esecuzione di codice asincrono attraverso RxJava, ed il codice viene mostrato come se fosse eseguito in maniera sincrona. L'unica eccezione è il primo diagramma, dove viene mostrata una chiamata al metodo "subscribe" di `Single<AuthSignUpResult>` per esplicitare le callback utilizzate. Inoltre, per i metodi utilizzati dal metodo principale, è stato omesso l'attore.

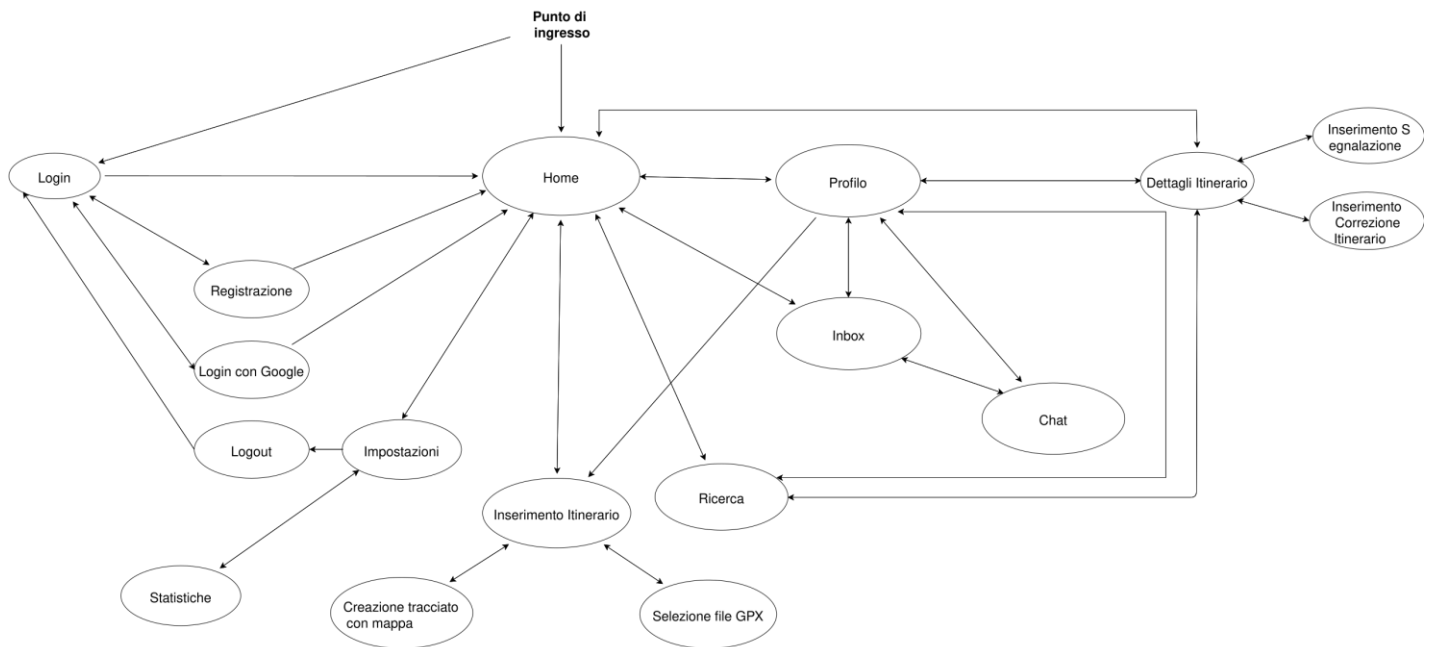




4.5 Definizione delle gerarchie funzionali

Di seguito è riportata la definizione delle gerarchie funzionali. Da notare che:

1. l'applicativo sviluppato è capace di ricordare la sessione dell'utente (se autenticato), e quindi può presentargli direttamente la home se la sessione è ancora valida;
2. le statistiche sono accessibili solo agli admin.



5.0 Testing

I seguenti test sono stati realizzati con Junit. I test mostrano tre metodi non banali.

5.1 Test registrazione

5.1.1 Metodo da testare

Codice del metodo da testare:

```

import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class registrazione {

    public boolean registerUser(String nome,String cognome,String
email,String pass,String confPass,String citta,String cell) {

        String nameText = nome;
        String surnameText = cognome;
        String emailText = email;
        String passwordText = pass;
        String confPasswordText = confPass;
        String cityText = citta;
        String telephoneText = cell;

```

```

        if (nameText.isEmpty()) {
            System.out.println("Campo obbligatorio.");
            return false;
        }

        if (surnameText.isEmpty()) {
            System.out.println("Campo obbligatorio.");
            return false;
        }

        if (emailText.isEmpty()) {
            System.out.println("Campo obbligatorio.");
            return false;
        }

        if (passwordText.isEmpty()) {
            System.out.println("Campo obbligatorio.");
            return false;
        }

        if (confPasswordText.isEmpty()) {
            System.out.println("Campo obbligatorio.");
            return false;
        }

        if (!verificaPass(passwordText)) {
            System.out.println("La password non rispetta le
policy.");
            return false;
        }

        if (!(passwordText.equals(confPasswordText))) {
            System.out.println("Le password non corrispondono.");

            System.out.println("Le password non corrispondono.");

            return false;
        }

        if (!(verificaEmail(emailText))) {
            System.out.println("Email non valida.");
            return false;
        }

        if(verificaNumerica(nameText, surnameText,
cityText,telephoneText) == false) {
            return false;
        }

```

```

        if(!telephoneText.isEmpty() &&
        controlloCell(telephoneText) == false) {
            System.out.println("Il numero deve essere formato da 10
        numeri.");
            return false;
        }
        System.out.println("Inserimento riuscito");
        return true;
    }

    private boolean verificaEmail(String email) {
        String expressionPlus="^[\\w\\-]([\\.\\w])+[\\w]+@([\\w\\-
    ]+\\.)+[A-Z]{2,4}$";
        Pattern pPlus = Pattern.compile(expressionPlus,
        Pattern.CASE_INSENSITIVE);
        Matcher mPlus = pPlus.matcher(email);
        boolean matchFoundPlus = mPlus.matches();

        return matchFoundPlus;
    }

    private boolean verificaPass(String password) {
        return password.matches("(?=.*\\d)(?=.*[a-z])(?=.*[A-
    Z])(?=.*[!&#x21;^*ç@#$$]).{8,15}");
    }

    private boolean verificaNumerica(String nome, String cognome,
    String citta, String numero) {
        if(nome.matches(".*\\d+.*")) {
            System.out.println("Il nome deve essere formato da sole
        lettere.");
            return false;
        }

        if(cognome.matches(".*\\d+.*")){
            System.out.println("Il cognome deve essere formato da
        sole lettere.");
            return false;
        }
        if(citta.isEmpty()==false && citta.matches(".*\\d+.*")){
            System.out.println("La città deve essere formato da
        sole lettere.");
            return false;
        }
        if( !numero.isEmpty() ){
            if(numero.matches("[a-zA-Z].*")) {

```

```

        System.out.println("Il cellulare deve essere
formato da soli numeri.");
        return false;
    }

    return true;
}
private boolean controlloCell(String num){
    if(num.length() != 10)
        return false;
    return true;
}
}

```

5.1.2 Analisi

PROBLEMA: Effettuare una registrazione dell'utente

INPUT: Nome, cognome, email, password, città, cellulare

OUTPUT: Registrazione riuscita o messaggio di errore

Esaminando gli input:

- Nome = {caratteri}, {""}, {0-9}
- Cognome = {caratteri}, {""}, {0-9}
- Email = {caratteri + @ + .}, {""}, Email \notin { caratteri + @ + }
- Password = Password \in { lettere maiuscole e minuscole + numeri + caratteri speciali}, {""}, Password \notin { lettere maiuscole e minuscole + numeri + caratteri speciali}, Password \neq ConfermaPassword
- Cellulare = {0-9}, {""}, {caratteri}, { x < 10 }, { 10 < x }

Test	CE1	CE2	CE3	CE4
Nome	✓	Campo vuoto	Valori numerici	
Cognome	✓	Campo vuoto	Valori numerici	
Email	✓	Campo vuoto	Email non corretta	
Password	✓	Non rispetta i criteri di sicurezza	Diversa da conferma password	
Città	✓	Valori numerici		
Cellulare	✓	Valori non numerici	Meno di 10 caratteri	Più di 10 caratteri

5.1.3 Codice Test

```

import static org.junit.jupiter.api.Assertions.assertEquals;

import org.junit.jupiter.api.Test;

public class TestJUnit {

    @Test
    public void registrazioneTest1() {
        registrazione test = new registrazione();
        System.out.print("Test1 (Campi corretti): ");
        assertTrue(test.registerUser("Francesco", "Esposito",
"fra@gmail.com", "PASSWORD12!",
"PASSWORD12!", "Roma", "3338989740"));
    }

    @Test
    public void registrazioneTest2() {
        registrazione test = new registrazione();
    }
}

```

```

        System.out.print("Test2 (Nome mancante): ");
        assertFalse(test.registerUser("", "Esposito",
"fra@gmail.com", "PASSWORD12!",
"PASSWORD12!", "Roma", "3338989740"));
    }
    @Test
    public void registrazioneTest3() {
        registrazione test = new registrazione();
        System.out.print("Test3 (Cognome mancante): ");
        assertFalse(test.registerUser("Francesco", "",
"fra@gmail.com", "PASSWORD12!",
"PASSWORD12!", "Roma", "3338989740"));
    }
    @Test
    public void registrazioneTest4() {
        registrazione test = new registrazione();
        System.out.print("Test4 (Email mancante): ");
        assertFalse(test.registerUser("Francesco", "Esposito",
"", "PASSWORD12!", "PASSWORD12!", "Roma", "3338989740"));
    }
    @Test
    public void registrazioneTest5() {
        registrazione test = new registrazione();
        System.out.print("Test5 (Email non corretta): ");
        assertFalse(test.registerUser("Francesco", "Esposito",
"fragmail.com", "PASSWORD12!",
"PASSWORD12!", "Roma", "3338989740"));
    }
    @Test
    public void registrazioneTest6() {
        registrazione test = new registrazione();
        System.out.print("Test6 (Nome con numeri): ");
        assertFalse(test.registerUser("Franc234esco",
"Esposito", "fra@gmail.com", "PASSWORD12!",
"PASSWORD12!", "Roma", "3338989740"));
    }
    @Test
    public void registrazioneTest7() {
        registrazione test = new registrazione();
        System.out.print("Test7 (Cognome con numeri): ");
        assertFalse( test.registerUser("Francesco",
"Espo464sito", "fra@gmail.com", "PASSWORD12!",
"PASSWORD12!", "Roma", "3338989740"));
    }
    @Test
    public void registrazioneTest8() {
        registrazione test = new registrazione();
        System.out.print("Test8 (Password non rispetta la
sicurezza): ");

```

```

        assertFalse( test.registerUser("Francesco", "Esposito",
"fra@gmail.com","password", "password","Roma","3338989740"));
    }
    @Test
    public void registrazioneTest9() {
        registrazione test = new registrazione();
        System.out.print("Test9 (Password e conferma password
diversi): ");
        assertFalse( test.registerUser("Francesco", "Esposito",
"fra@gmail.com","PASSword12!", "PASS","Roma","3338989740"));
    }
    @Test
    public void registrazioneTest10() {
        registrazione test = new registrazione();
        System.out.print("Test10 (Cellulare con lettere): ");
        assertFalse( test.registerUser("Francesco", "Esposito",
"fra@gmail.com","PASSword12!",
"PASSword12!","Roma","333prova94"));
    }
    @Test
    public void registrazioneTest11() {
        registrazione test = new registrazione();
        System.out.print("Test11 (Cellulare con 6 numeri): ");
        assertFalse( test.registerUser("Francesco", "Esposito",
"fra@gmail.com","PASSword12!", "PASSword12!","Roma","333974"));
    }
    @Test
    public void registrazioneTest12() {
        registrazione test = new registrazione();
        System.out.print("Test12 (Cellulare con 12 numeri):
");
        assertFalse( test.registerUser("Francesco", "Esposito",
"fra@gmail.com","PASSword12!",
"PASSword12!","Roma","333974666666"));
    }
}

```

5.1.4 Output del test



L'output riporta a sinistra il numero del test seguito dalla descrizione dell'input inserito, a destra riporta la risposta del software


Nome test (input) : output


Test1 (Campi corretti): Inserimento riuscito!
Test2 (Nome mancante): Campo obbligatorio.
Test3 (Cognome mancante): Campo obbligatorio.
Test4 (Email mancante): Campo obbligatorio.
Test5 (Email non corretta): Email non valida.
Test6 (Nome con numeri): Il nome deve essere formato da sole lettere.
Test7 (Cognome con numeri): Il cognome deve essere formato da sole lettere.
Test8 (Password non rispetta la sicurezza): La password non rispetta le policy.
Test9 (Password e conferma password diversi): Le password non corrispondono.
Le password non corrispondono.
Test10 (Cellulare con lettere): Il cellulare deve essere formato da soli numeri.
Test11 (Cellulare con 6 numeri): Il numero deve essere formato da 10 numeri.
Test12 (Cellulare con 12 numeri): Il numero deve essere formato da 10 numeri.













Risultato Junit:

Finished after 0,151 seconds

Runs: 12/12  Errors: 0  Failures: 0



▼  TestJUnit [Runner: JUnit 5] (0,001 s)

-  registrazioneTest1() (0,000 s)
-  registrazioneTest2() (0,000 s)
-  registrazioneTest3() (0,000 s)
-  registrazioneTest4() (0,000 s)
-  registrazioneTest5() (0,000 s)
-  registrazioneTest6() (0,000 s)
-  registrazioneTest7() (0,000 s)
-  registrazioneTest8() (0,000 s)
-  registrazioneTest9() (0,000 s)
-  registrazioneTest10() (0,001 s)
-  registrazioneTest11() (0,000 s)
-  registrazioneTest12() (0,000 s)

5.2 Test ConvertMap

Il metodo converte i valori di altitudine e longitudine (estratti dalla mappa interattiva) in String

5.2.1 Metodo da testare

```
import java.util.List;
import java.util.List;

public class MapConvert {

    final private static String LATITUDE_SEPARATOR = "$";
    final private static String LONGITUDE_SEPARATOR = "#";
    final private static String TRACK_END_SEPARATOR = "%";

    public static String geoPointsToString(List<GeoPoint>
geoPointList){
        if (geoPointList == null)
            return null;
        String outputString = "";
        for (GeoPoint gp: geoPointList){
            outputString += gp.getLatitude()+LATITUDE_SEPARATOR
                +gp.getLongitude()+LONGITUDE_SEPARATOR;
        }
        outputString += TRACK_END_SEPARATOR;
        System.out.println("SUSOUTPUTSTRING: " + outputString);
        return outputString;
    }
}
```

Il prossimo codice è una estrazione della classe GeoPoint per comprendere meglio il codice precedente.

```
public class GeoPoint {

    private double mLongitude;
    private double mLatitude;
    private double mAltitude;

    public GeoPoint(double mLatitude, double mLongitude) {
        super();
        this.mLongitude = mLongitude;
        this.mLatitude = mLatitude;
    }

    public double getLongitude() {
        return this.mLongitude;
    }
}
```

```
public double getLatitude() {  
    return this.mLatitude;  
}  
}
```

5.2.2 Analisi

PROBLEMA: Estrazione dei dati dalla mappa interattiva e convertirli in stringhe

INPUT: latitudine e longitudine in double

OUTPUT: String

Esaminiamo l'input:

- Latitudine = { $-90 - 90$ }, { $-90 < x < 90$ }, { "testo" }, { null }; (Nota. La mappa ha solo valori compresi tra -90 e 90, quindi ulteriori controlli sono inutili.
- Longitudine = { $-180 - 180$ }, { $-180 < x < 180$ }, { "testo" }, { null }; (Nota. La mappa ha solo valori compresi tra -180 e 180, quindi ulteriori controlli sono inutili.

TEST	CE1	CE2
Latitudine	✓	Null
Longitudine	✓	Null

5.2.3 Codice test

```
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertNull;
import java.awt.List;
import java.util.ArrayList;
import java.util.Random;
import java.util.concurrent.ThreadLocalRandom;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

public class MapConvertTest {

    MapConvert map = new MapConvert();
    GeoPoint geoPoint1, geoPoint2, geoPoint3;
    int maxY = 90, minY = -90;
    int maxX = 180, minX = -180;
    double x, y, x2, y2, x3, y3;

    @BeforeEach
```

```

    public void inizializza() {
        ThreadLocalRandom tlr = ThreadLocalRandom.current();
        x = tlr.nextInt(minX, maxX + 1);
        y = tlr.nextInt(minY, maxY + 1);
        x2 = tlr.nextInt(minX, maxX + 1);
        y2 = tlr.nextInt(minY, maxY + 1);
        x3 = tlr.nextInt(minX, maxX + 1);
        y3 = tlr.nextInt(minY, maxY + 1);
        geoPoint1 = new GeoPoint(y, x);
        geoPoint2 = new GeoPoint(y2, x2);
        geoPoint3 = new GeoPoint(y3, x3);
    }

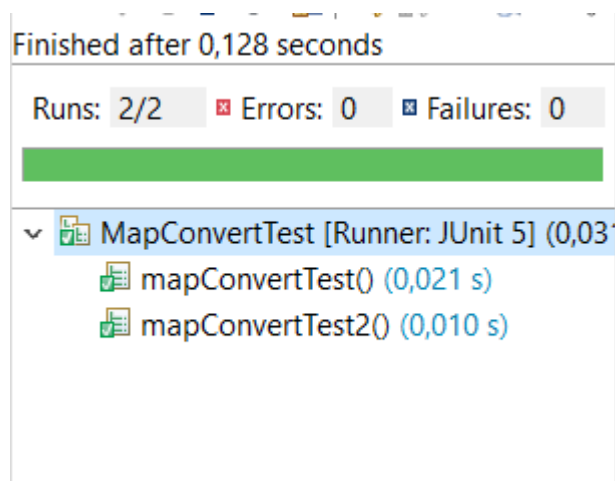
    @Test
    public void mapConvertTest() {

        ArrayList<GeoPoint> list = new ArrayList<GeoPoint>();
        list.add(geoPoint1);
        list.add(geoPoint2);
        list.add(geoPoint3);
        assertEquals(y + "$" + x + "#" + y2 + "$" + x2 + "#" + y3 +
"$" + x3 + "#%", map.geoPointsToString(list));
    }

    @Test
    public void mapConvertTest2() {
        ArrayList<GeoPoint> list2 = null;
        assertNull(map.geoPointsToString(list2));
    }
}

```

5.2.4 Output del test



5.3 Conversione GeoPoint

Il metodo accetta una stringa e restituisce un oggetto GeoPoint (coordinate di un punto con riferimento a latitudine e longitudine).

5.3.1 Metodo da testare

```
public class ListGeoPoint {
    enum ReadMode{
        LATITUDE,
        LONGITUDE
    }
    final private static String LATITUDE_SEPARATOR =
"$";
    final private static String LONGITUDE_SEPARATOR =
"#";
    final private static String TRACK_END_SEPARATOR =
"%";

    public List<GeoPoint>
geoPointStringToGeoPointList(String string){
    double[] values = new double[2];
    int index;
    ReadMode readMode = ReadMode.LATITUDE;
    List<GeoPoint> output = new
LinkedList<GeoPoint>();
    char discriminator;
    int fieldsCounter = 0;
    while(string.length() != 0){
        if (readMode.equals(ReadMode.LATITUDE))
            discriminator =
LATITUDE_SEPARATOR.toCharArray()[0];
        else
            discriminator =
LONGITUDE_SEPARATOR.toCharArray()[0];
        index = string.indexOf(discriminator);
        if (index == -1) {
            System.out.println("Stringa GeoPoint
invalida!");
            return null;
        }

        values[fieldsCounter] =
Double.valueOf(string.substring(0, index));
        fieldsCounter++;
        string = string.substring(index+1);
        readMode =
(readMode.equals(ReadMode.LATITUDE)) ?
ReadMode.LONGITUDE :
ReadMode.LATITUDE;
        if (fieldsCounter == 2){
```

```

        fieldsCounter = 0;
        GeoPoint newGeoPoint = new
GeoPoint(values[0], values[1]);
        output.add(newGeoPoint);
    }
}
return output;
}

```

5.2.2 Analisi

PROBLEMA: Si vuole inserire una stringa e convertirla in coordinate GeoPoint

INPUT: String

OUTPUT: oggetto GeoPoint

Esaminiamo l'input:

- String : { numeri + \$ + numeri + #}, \$ ∈ { String} e # ∈ { String}
 Nota : La stringa in input è sempre formata da numeri, non richiede controlli sul contenuto di caratteri non numerici

5.2.3 Codice test

```

import static
org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertNull;
import org.junit.jupiter.api.Test;

public class ListGeoPointTest {
    ListGeoPoint t = new ListGeoPoint();

    @Test
    public void mapTest() {
        String s = "75$20#";
        for(GeoPoint i:
t.geoPointStringToGeoPointList(s)) {
            System.out.println("Test1 :");
            System.out.println("    Input -->
75$20#");
            System.out.println("    Conversione --> "
+ i.getLatitude() + " " + i.getLongitude());
            GeoPoint g = new GeoPoint(75, 20);
            assertEquals(g.getLatitude(),
i.getLatitude());
            assertEquals(g.getLongitude(),
i.getLongitude());

```

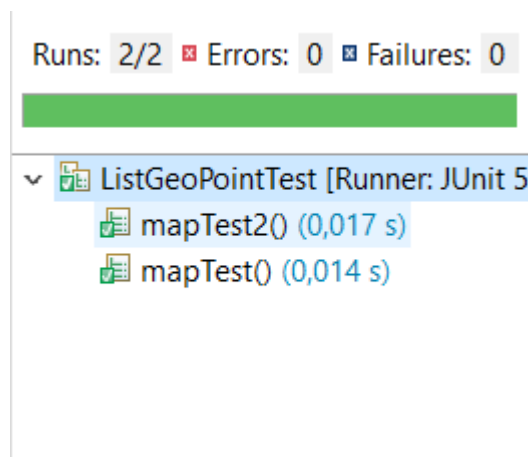
```

    }
}

@Test
public void mapTest2() {
    String s = "7520#";
    System.out.println("Test2 :");
    System.out.println("    Input --> 7520#");
    System.out.print("    Output -->  ");
    assertNull(
t.geoPointStringToGeoPointList(s));
}
}

```

4.2.4 Output test



```

Test2 :
    Input --> 7520#
    Output --> Stringa GeoPoint invalida!
Test1 :
    Input --> 75$20#
    Conversione --> 75.0 20.0

```

6. Valutazione dell'usabilità sul campo

6.1 Test di compito

Per la valutazione dell'usabilità sul campo si è scelto di impiegare tecniche simili a quelle usate per la valutazione dell'usabilità a priori (vedere 2.6). In particolare,

in prima fase, si è scelto di far provare l'applicativo in beta testing a tester scelti dalle stesse cinque categorie precedentemente individuate, al fine di constatare l'effettiva usabilità del prodotto realizzato. Per comodità, si riportano di nuovo le categorie individuate:

- utenti giovani e con esperienze nelle escursioni (**Lorenzo R.**);
- utenti giovani e interessati, ma non esperti, alle escursioni (**Andrea D.**);
- utenti dai 30 ai 40 anni e con interessi (**Francesca B.**) / esperienze nelle escursioni (**Roberta C.**);
- utenti escursionisti oltre i 40 anni (**Giuseppe R.**).

NaTour

Tabella tester

	Età		
	<30	30-40	40+
Non escursionista			
Interessati all'escursionismo	✓	✓	
Escursionista	✓	✓	✓

Figura 3. Tabella tester

Per evitare una valutazione incorretta scaturita da una familiarità con i mockup interattivi, abbiamo scelto tester diversi da quelli precedentemente impiegati, e abbiamo proposto loro altri quattro compiti:

1. registrati alla piattaforma;
2. inserisci un itinerario, usando la mappa interattiva per specificare un tracciato ad esso associato, il quale parte dal luogo dove ti trovi ora;
3. segnala un itinerario;
4. cerca un itinerario nei pressi di Napoli che sia facile, e che sia accessibile a persone con disabilità motorie.

	Compito 1	Compito 2	Compito 3	Compito 4
Lorenzo R.	S	S	S	S
Andrea D.	S	S	S	S
Francesca B.	S	S	S	S
Roberta C.	S	P	S	S
Giuseppe R.	S	P	S	F

Abbiamo notato, con soddisfazione, un tasso di successo del **90%**.

Andremo ora ad approfondire i successi parziali ed i fallimenti:

- Roberta C. e Giuseppe R. hanno usato la barra di ricerca per trovare il luogo dove si trovavano, mentre invece avrebbe dovuto usufruire del tasto dedicato per il riposizionamento della mappa alla loro posizione. Poiché hanno terminato il compito con successo, ma usando un metodo meno efficiente, abbiamo segnato il successo come parziale. Attribuiamo questo risultato ad una scarsa familiarità con altre applicazioni basate sull'utilizzo di mappe.
- Giuseppe R. ha digitato "Napoli" nel campo di ricerca legato alla località, senza però aspettare che appaia un risultato geolocalizzato come suggerimento. Ha poi effettuato la ricerca, non ottenendo i risultati sperati. Per risolvere questa problematica, abbiamo deciso di notificare l'utente di una selezione "incompleta" aggiungendo un piccolo messaggio di errore sotto il campo.

6.2 Monitoraggio attraverso strumenti di logging

Al test precedentemente eseguito abbiamo aggiunto l'utilizzo di strumenti di logging per registrare le interazioni e l'utilizzo da parte dell'utente dell'applicativo. A questo fine, sono stati utilizzati strumenti di logging come la libreria di log di Android e **Google Analytics** (precedentemente approfondito in 4.1.1, "Tecnologie utilizzate"). Il logging effettuato con la libreria di Android riguarda vari aspetti dell'applicazione, come l'inizializzazione di componenti, l'uso di alcune feature, e lo stato della connessione STOMP. Per visualizzare un esempio del logging effettuato durante una normale sessione di utilizzo dell'app, ci si può riferire al file di log allegato al progetto.

