

**UNIVERSITA' DEGLI STUDI DI NAPOLI
FEDERICO II**

SCUOLA POLITECNICA E DELLE SCIENZE DI BASE

DIPARTIMENTO DI INGEGNERIA ELETTRICA E TECNOLOGIE DELL'INFORMAZIONE



CORSO DI LAUREA IN INFORMATICA

INSEGNAMENTO INGEGNERIA DEL SOFTWARE ANNO ACCADEMICO 2021/2022

Progettazione e sviluppo di NaTour21

Una moderna piattaforma social per appassionati di escursioni

Gruppo INGSW2122_N_03

Autori:

Pierluigi Supino: N86002833

Pio Francesco Falzarano N86002978

Docenti:

Prof. Sergio di Martino

Prof. Francesco Cutugno

Sommario

1. Document History	4
2. Introduzione alla Documentazione	6

PARTE I: Requisiti Software

3. Analisi dei Requisiti	
3.1 <u>Requisiti Funzionali</u>	8
3.2 <u>Casi d'Uso</u>	10
3.3 <u>Tabelle di Cockburn</u>	12
3.4 <u>Prototipazione Visuale</u>	18
3.5 <u>Prototipazione Funzionale</u>	20
4. Analisi del Contesto	
4.1 <u>Analisi del Target degli Utenti</u>	22
4.2 <u>Valutazione dell'Usabilità a Priori</u>	25
5. Modelli di Dominio	
5.1 <u>Oggetti e Relazioni del Dominio</u>	30
5.2 <u>Class Diagrams di Analisi</u>	30
5.3 <u>Sequence Diagrams di Analisi</u>	34
5.4 <u>Activity Diagrams</u>	36

PARTE II: Design del Sistema

6. Architettura del Sistema	
6.1 <u>Analisi dell'Architettura Esterna</u>	45
6.2 <u>Architettura Backend</u>	46
6.3 <u>Servizi Cloud utilizzati</u>	50
6.4 <u>Architettura Frontend</u>	52

7. Design del Sistema	
7.1 Class Diagrams di Design	54
7.2 Sequence Diagrams di Design	60
7.3 Gerarchie Funzionali	62

PARTE III: Testing e Valutazioni finali

8. Unit Testing	
8.1 Password Policy	64
8.2 Durata Media	70
8.3 Parsing degli Itinerari	74
9. Valutazione sul campo	
9.1 Prototipi finali	81
9.2 Usabilità sul campo	84

*"The function of good software is
to make the complex appear to be simple."*

Grady Booch

1. Document History

Data	Versione	Autore	Descrizione
27/10/2021	0.1	P. Falzarano P. Supino	Creazione del format di base del documento
27/10/2021	0.1	P. Supino	Aggiunta: Requisiti Funzionali
29/10/2021	0.1	P. Falzarano P. Supino	Aggiunta: Use Case e Tabelle di Cockburn
4/11/2021	0.1	P. Supino	Aggiunta: Prototipazione Visuale
12/11/2021	0.1	P. Falzarano P. Supino	Aggiunta: Prototipazione Funzionale
22/11/2021	0.1	P. Falzarano	Aggiunta: Analisi del Contesto
25/11/2021	0.2	P. Supino P. Falzarano	Revisione: Format generale Paragrafi
29/11/2021	0.2	P. Falzarano	Aggiunta: Class e Sequence Diagrams di analisi
07/12/2021	0.2	P. Falzarano P. Supino	Aggiunta: Activity Diagrams
13/12/2021	0.2	P. Falzarano	Modifiche: Class e Sequence Diagrams di Analisi
11/01/2022	0.2	P. Falzarano	Aggiunta: Analisi dell'Architettura
03/02/2022	0.3	P. Falzarano P. Supino	Aggiunta: Class e Sequence Diagrams di Design
07/02/2022	0.3	P. Supino	Aggiunta: Gerarchie Funzionali
15/02/2022	0.3	P. Falzarano P. Supino	Aggiunta: Testing del Sistema
18/02/2022	1.0	P. Supino P. Falzarano	Revisione Finale del Documento

2. Introduzione alla Documentazione

La seguente documentazione segue il processo di sviluppo del Software richiesto dal Committente; ogni fase dello sviluppo è descritta in una sezione:

- 1) La prima si focalizza sui Requisiti Software, avendo in considerazione le specifiche fornite dal committente e soprattutto dalle esigenze degli utenti finali; vengono descritti tutte le funzionalità offerte dalla piattaforma, dando esempi di prototipazione per mostrare l'interazione col sistema; infine, è dedicata uno spazio all' OOA del dominio, attraverso diagrammi di classi, di sequenza e di attività.
- 2) La seconda espone l'architettura del Sistema, le tecnologie utilizzate per il back-end e l'analisi del design pattern architettonico utilizzato per il front-end, accompagnati da artefatti UML di OOD; una parte è incentrata sulla definizione delle gerarchie funzionali dell'applicativo.
- 3) L'ultima incentrata sul Testing, del codice mediante Unit-Testing effettuato con JUnit, e una Valutazione sull'usabilità con prodotto finito mediante tecniche di Beta Testing e strumenti di logging.

PARTE I: Requisiti Software

3. Analisi dei Requisiti

Il Committente richiede lo sviluppo di un'applicazione su client mobile volta alla commercializzazione del social “NaTour” tra gli amanti delle escursioni. Il fulcro principale della piattaforma social è la condivisione di Itinerari: un utente inserisce un itinerario fornendone una descrizione, gli altri possono visualizzare l’itinerario ed i suoi dettagli. L’esperienza social scaturisce dall’interazione tra utenti, attraverso lo scambio di messaggi, condivisione di foto relative ad un percorso e miglioramento della descrizione di un percorso attraverso riscontri. Ecco le funzionalità in dettaglio:

3.1 Requisiti Funzionali

- Il sistema deve permettere il login all’utente, in particolare:
 - L’utente deve avere la possibilità di accedere utilizzando indirizzo e-mail e password (in caso di avvenuta registrazione con l’indirizzo e-mail corrispondente);
 - L’utente deve poter resettare la propria password;
 - L’utente deve avere anche la possibilità di effettuare il login utilizzando providers esterni (Google);
- Il sistema deve permettere all’utente di registrarsi utilizzando il proprio indirizzo e-mail; inoltre, durante la registrazione, l’utente deve inserire dei campi obbligatori quali nome e password;
- Il sistema deve permettere all’utente di condividere nuovi itinerari caratterizzati da un nome, una durata, un livello di difficoltà, un punto di inizio, una descrizione (opzionale), un insieme di fotografie e un tracciato geografico che lo rappresenta su una mappa, in particolare l’utente può:
 - Decidere di inserire un nuovo itinerario manualmente avvalendosi di una mappa interattiva;
 - Decidere di inserire un nuovo itinerario utilizzando un file in formato GPX;

- Il sistema deve permettere all’utente di visualizzare una lista di itinerari e una pagina di riepilogo di tutte le informazioni riguardanti ogni itinerario, incluse eventuali fotografie caricate da altri utenti;
- L’utente deve aver la possibilità di visualizzare su una mappa il percorso geografico inerente ad un itinerario;
- Il sistema deve permettere all’utente di dare un feedback alla descrizione di un itinerario fornendo un punteggio di difficoltà e/o tempo di percorrenza diverso da quelli indicati nella pagina di riepilogo dell’itinerario; in questo caso il punteggio di difficoltà e il tempo di percorrenza per il sentiero saranno ricalcolati come media delle difficoltà e dei tempi indicati.
- Il sistema deve permettere all’utente di inviare/ rispondere a messaggi privati verso altri utenti.
- Il sistema deve permettere all’utente di visualizzare la lista dei messaggi ricevuti da altri utenti.
- Il sistema deve permettere all’utente di caricare fotografie di punti di interesse scattati sul sentiero di un itinerario; queste saranno visualizzate nella pagina di dettaglio dell’itinerario. Inoltre, se la fotografia ha una posizione geografica di scatto salvata nei metadati, sarà mostrato all’interno della mappa un marker con la fotografia corrispondente.
- Il sistema deve permettere agli Amministratori di rimuovere o modificare itinerari inseriti dagli utenti, ed in caso di modifica verrà mostrato un warning nella schermata di dettaglio dell’itinerario con relativa data di modifica.
- Il sistema deve permettere all’utente di effettuare il logout da un dispositivo (in questo caso dovrà necessariamente effettuare di nuovo il login per utilizzare le funzionalità dell’applicazione).

3.2 Casi d'Uso

Di seguito sono riportati i requisiti funzionali introdotti precedentemente attraverso rappresentazione grafica con l'utilizzo di Use Case Diagrams; ogni diagramma è relativo a macro-funzionalità del Sistema:

- Gestione del Profilo Utente;
- Condivisione di Itinerari;
- Messagistica tra Utenti;
- Funzionalità dell'Amministratore;

Gli attori secondari che riguardano web services sono stati omessi per brevità, visto che nella parte II del documento è dedicato un paragrafo ai servizi cloud utilizzati.

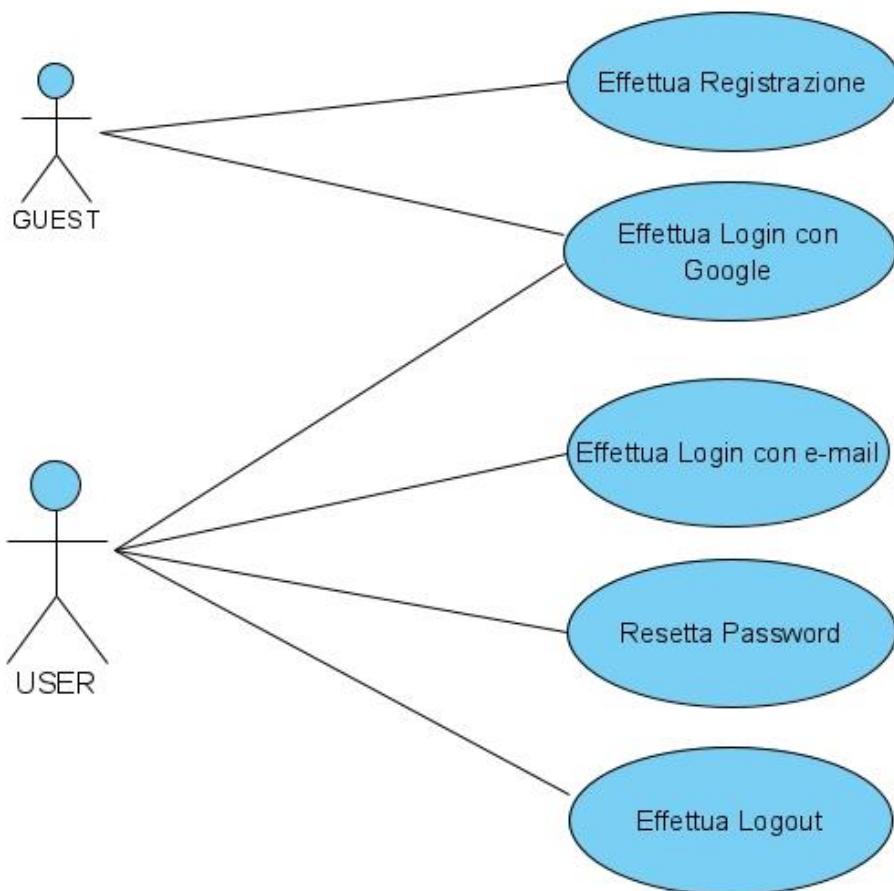


Figura 1: Use Case Diagram relativo alla gestione del Profilo Utente

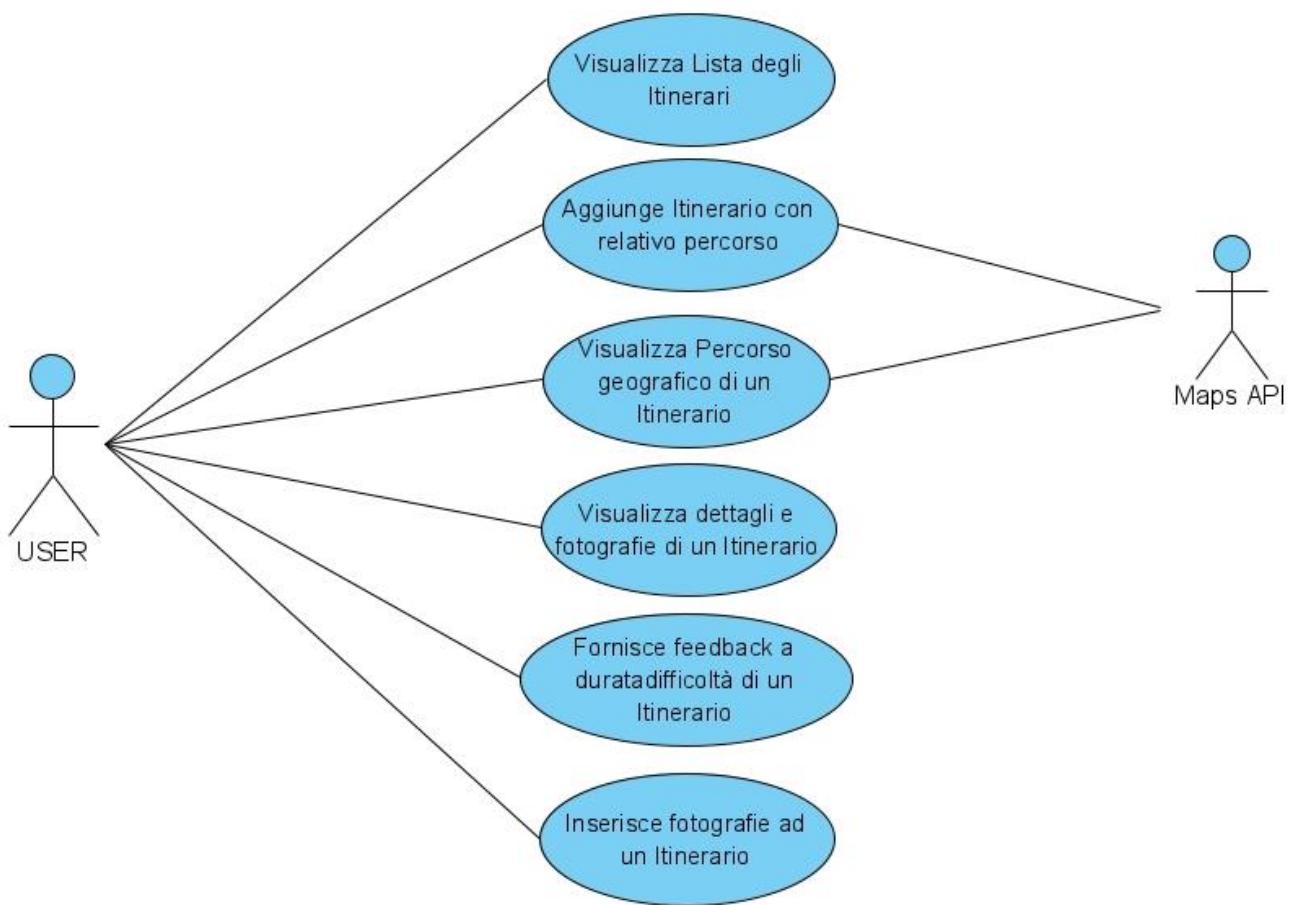


Figura 2: Use Case Diagram relativo alla condivisione di Itinerari

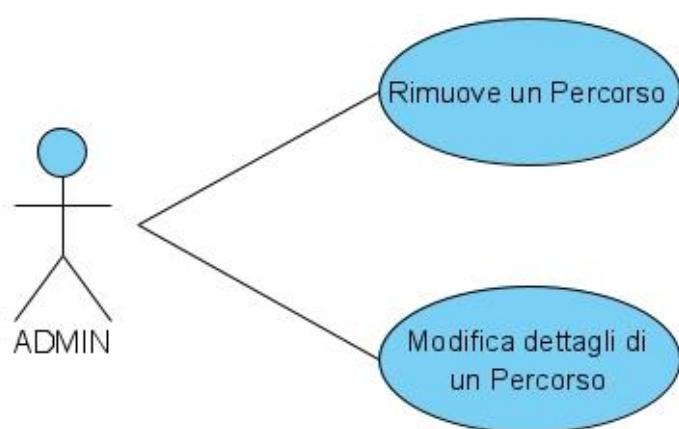


Figura 3: Use Case Diagram relativo all'Amministratore
(include anche i casi d'uso del normale utente)

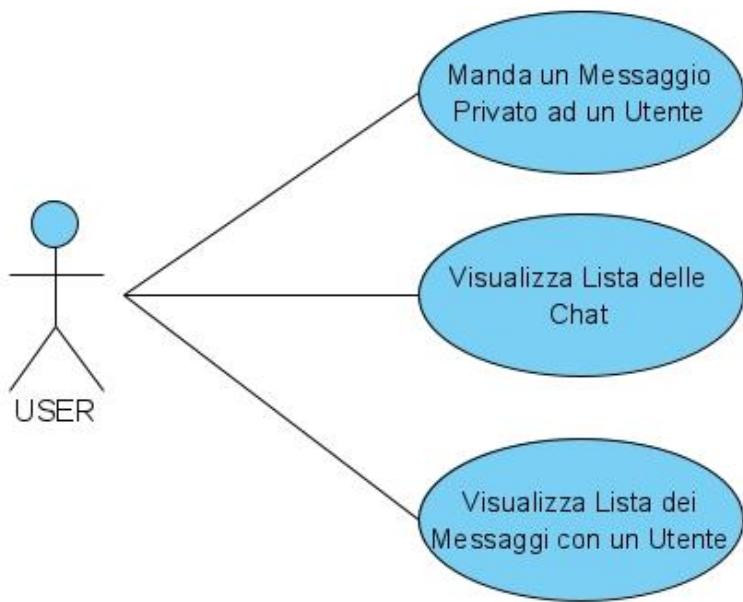


Figura 4: Use Case Diagram relativo alla Messaggistica

3.3 Tabelle di Cockburn

In questa sezione vengono analizzati due casi d'uso: il primo “Aggiunge Percorso” (Use Case #1) cardine della piattaforma, e il secondo “Fornisce feedback alla descrizione di un Percorso” (Use Case #2) per migliorare l'accuratezza dei dettagli dei percorsi secondo le esperienze personali degli utenti.

Al fine di garantire una coerente rappresentazione e una facile interpretazione si è fatto uso del formalismo delle Tabelle di Cockburn, riportate in seguito, con allegati mock-up grafici dell'interfaccia utente e statecharts nelle sezioni successive.

USE CASE #1	Aggiunge Itinerario		
Goal in Context	Permettere all'utente di inserire un percorso e condividerlo sulla piattaforma;		
Preconditions	L'utente deve aver effettuato il login;		
Success End Condition	L'utente è riuscito ad inserire il percorso nella piattaforma;		
Failed End Condition	Il Percorso non è stato inserito nella piattaforma;		
Primary Actor	Utente		
DESCRIPTION	Step	Utente	Sistema
	1	L'utente clicca sul pulsante “+” nella schermata Main-Page	
	2		<i>Il sistema mostra la schermata Add-Itinerary</i>
	3	L'utente inserisce un nome per l'itinerario	
	4 (*)	L'utente inserisce una descrizione per l'itinerario	
	5	L'utente inserisce una difficoltà per l'itinerario	
	6	L'utente inserisce una durata per l'itinerario	
	7	L'utente clicca sul pulsante “Select Photos”	
	8		<i>Il sistema mostra il File Explorer del dispositivo</i>
	9	L'utente seleziona le foto da allegare all'itinerario e conferma	
	10		<i>Il sistema chiude il File Explorer e associa le foto al percorso</i>
	11	L'utente clicca sul pulsante “Select GPX file”	

	12		<i>Il sistema mostra il File Explorer</i>
	13	L'utente seleziona il file GPX da allegare all'itinerario e conferma	
	14		<i>Il sistema chiude il File Explorer ed effettua il parsing del file a JSON</i>
	15	L'utente clicca sul pulsante "Add"	
	16		<i>Il sistema aggiunge l'itinerario nella piattaforma e ne mostra la schermata Itinerary-Detail</i>
EXTENSION #1 Errore Generico	Step	Utente	Sistema
	1		<i>Il sistema mostra un messaggio di errore non riuscendo a caricare l'itinerario</i>
	2	L'utente riprova a salvare l'itinerario (ritorno al punto 15 del normal flow)	
EXTENSION #2 File non supportati	Step	Sistema	
	1		<i>Il sistema mostra un simbolo di warning e un messaggio di errore accanto ai file caricati</i>
EXTENSION #3 Le fotografie hanno una posizione di scatto distante dal tracciato geografico	Step	Sistema	
	1		<i>Il sistema mostra un messaggio di errore quando l'utente cerca di caricare l'itinerario, invitandolo ad eliminare le fotografie non attinenti al percorso</i>

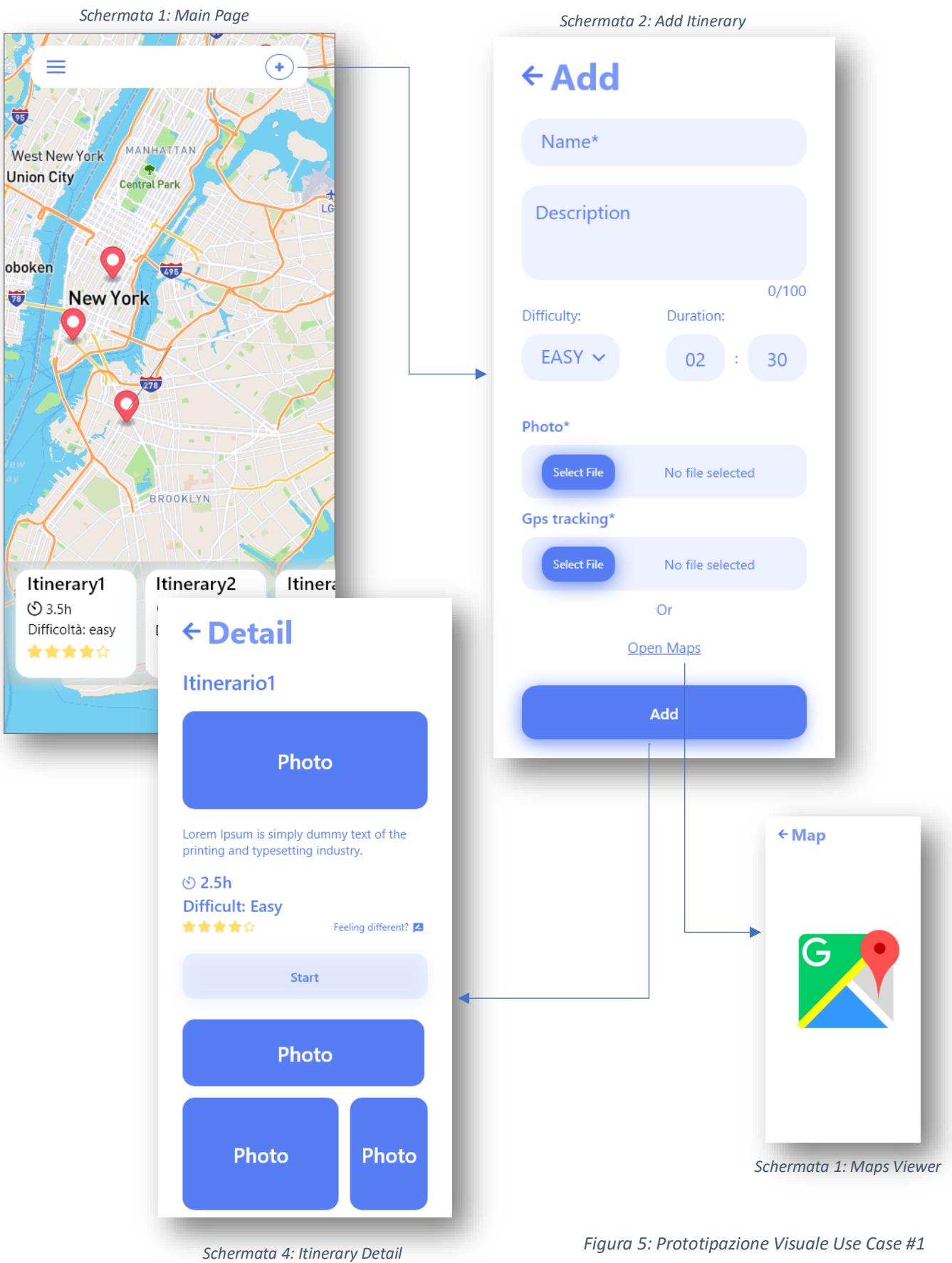
	<i>Step</i>	Sistema		
EXTENSION #4 Durata nulla				
	1	<i>Il sistema mostra un messaggio di warning e un messaggio di errore accanto al widget per la selezione della durata</i>		
EXTENSION #5 Nome Obbligatorio	<i>Step</i>	Sistema		
	1	<i>Il sistema mostra un messaggio di warning e un messaggio di errore accanto alla TextField per l'input del nome dell'Itinerario</i>		
SUBVARIATIONS	<i>Step</i>	Utente	Maps Api	Sistema
	11.a	L'utente clicca su "Open Maps"		
	12.a			<i>Il sistema apre Maps-Viewer</i>
	13.a	L'utente crea un percorso sulla mappa e lo conferma		
	14.a		<i>Salva il percorso in formato JSON</i>	
	15.a			<i>Il sistema associa il JSON al percorso e (ritorno al punto 15 del normal flow)</i>

(*) Opzionale

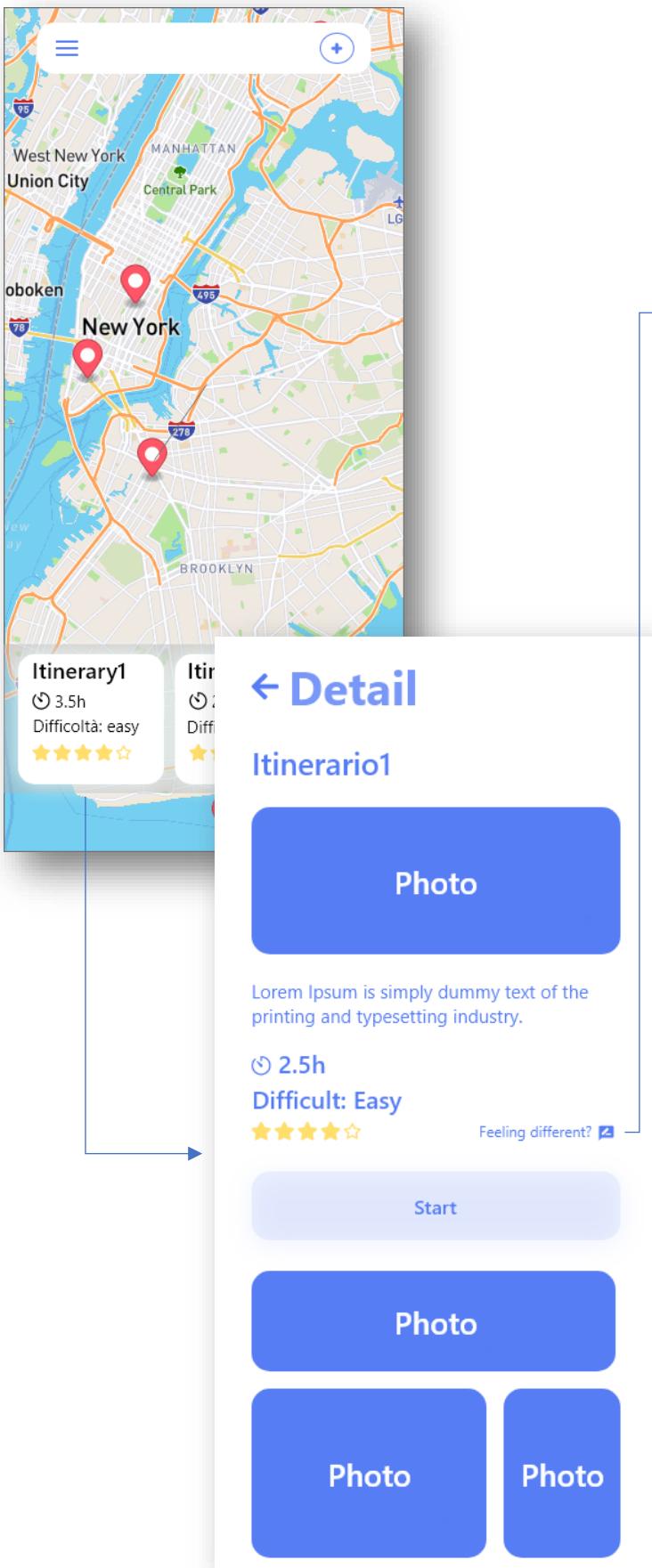
USE CASE #2	Fornisce feedback ad un Itinerario		
Goal in Context	Permettere all'utente di indicare un diverso punteggio di difficoltà e/o tempo di percorrenza di un itinerario;		
Pre conditions	L'utente deve aver effettuato il login;		
Success End Condition	L'utente è riuscito a fornire un riscontro sui dettagli di un itinerario;		
Failed End Condition	L'utente non è riuscito a dare un feedback;		
Primary Actor	Utente		
DESCRIPTION	Step	Utente	Sistema
	1	L'utente clicca su un Itinerario nella schermata Main-Page	
	2		<i>Il sistema mostra la schermata Itinerary-Detail dell'itinerario selezionato</i>
	3	L'utente clicca su "Send a feedback"	
	4		<i>Il sistema mostra la schermata Edit-Detail</i>
	5	L'utente inserisce una durata e/o livello di difficoltà diverso per l'itinerario	
	6	L'utente clicca sul pulsante "Salva"	
	7		<i>Il sistema ricalcola la difficoltà e la durata del percorso</i>
	8		<i>Il sistema chiude Edit-Detail e mostra la schermata Itinerary-Detail con i dati aggiornati</i>

EXTENSION #1	Step	Utente	Sistema
Errore Generico	1		<i>Il sistema mostra un messaggio di errore non riuscendo a modificare l'itinerario</i>
	2	L'utente riprova a salvare le modifiche (ritorno al punto 6 del normal flow)	
EXTENSION #2			
Durata nulla	1		<i>Il sistema mostra un messaggio di warning e un messaggio di errore accanto al widget per la selezione della durata</i>
	2	Seleziona una durata non nulla (ritorno al punto 5 del normal flow)	

3.4 Prototipazione Visuale

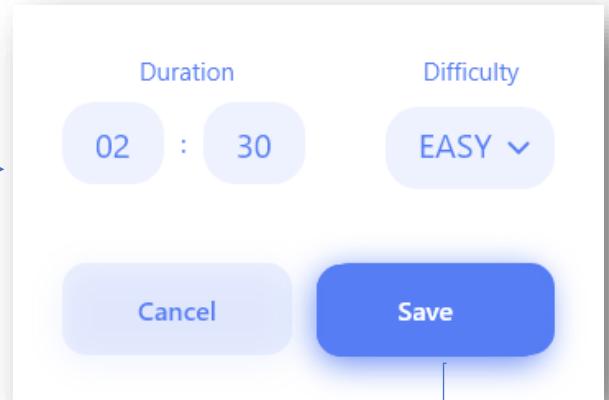


Schermata 1: Main Page



Schermata 2: Itinerary Detail

Schermata 3: Edit Detail



Schermata 4: Error

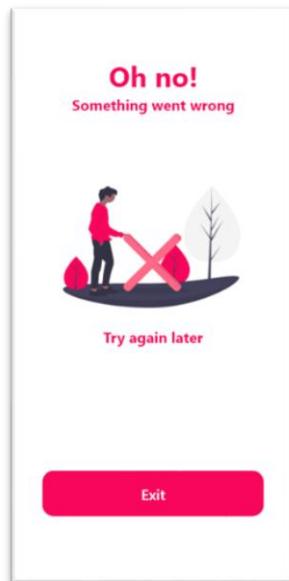


Figura 6: Prototipazione Visuale Use Case #2

3.5 Prototipazione Funzionale

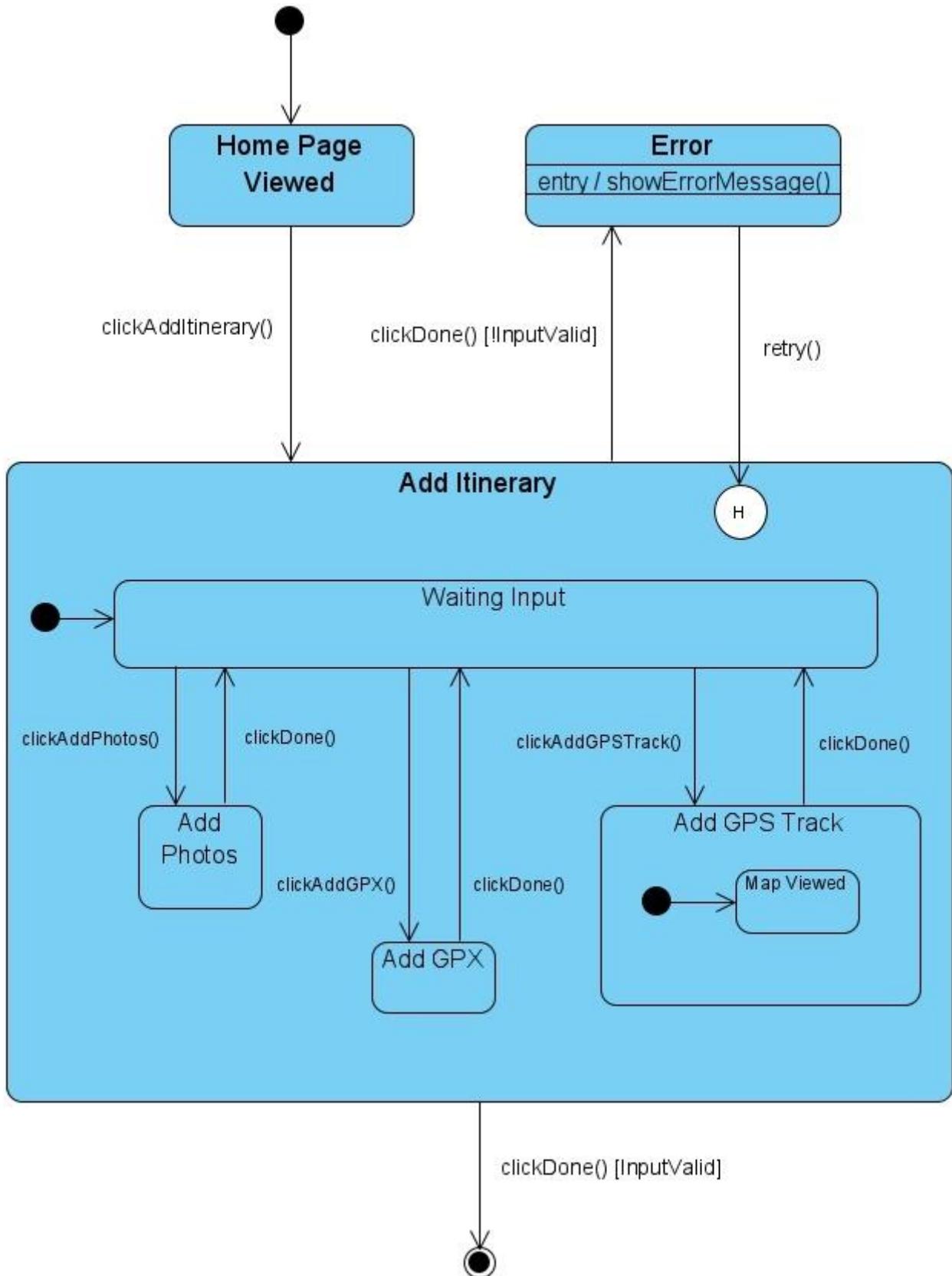


Figura 7: Prototipazione Funzionale Use Case #1

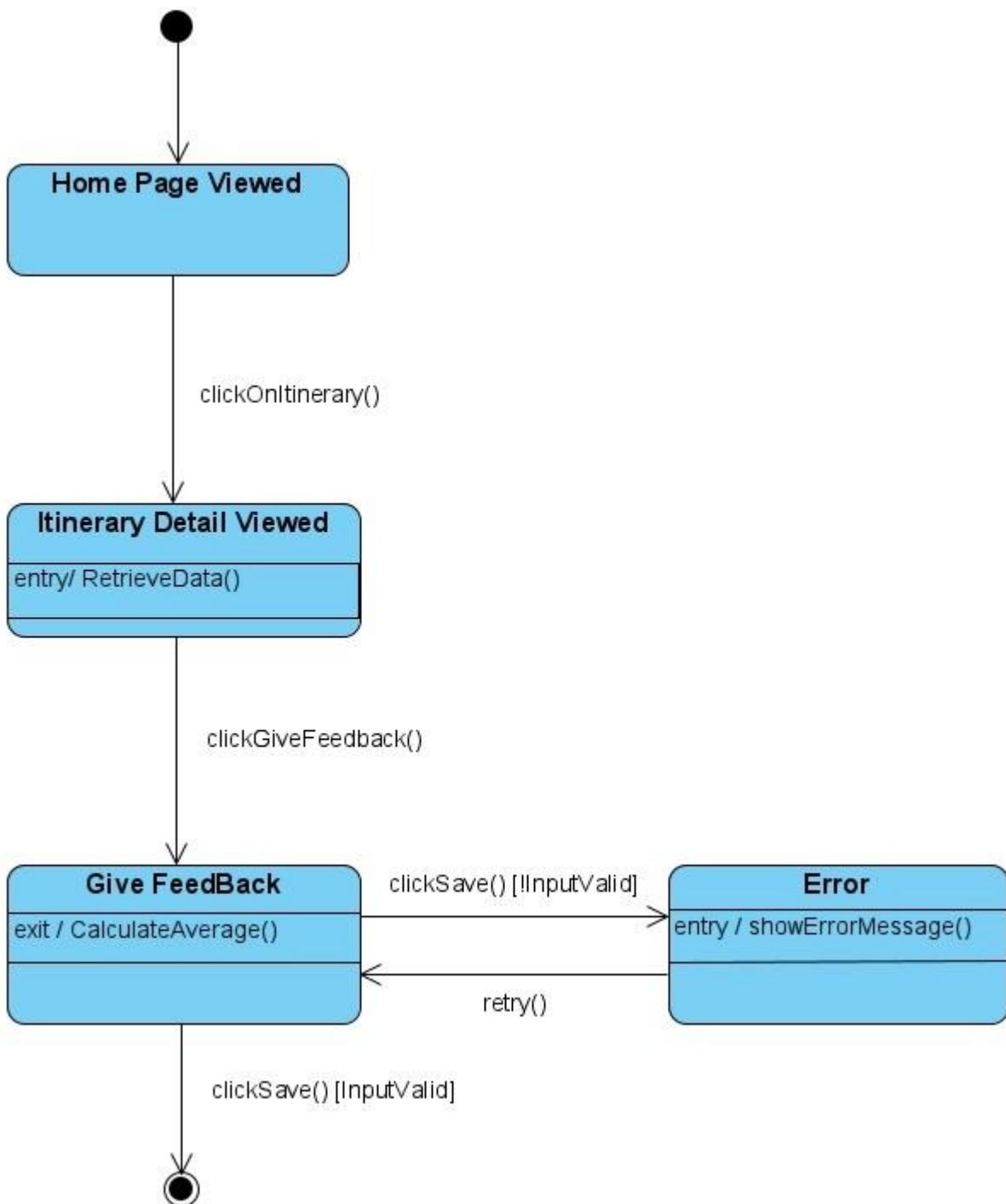


Figura 8: Prototipazione Funzionale Use Case #2

4. Analisi del Contesto

4.1 Analisi del Target degli Utenti

Secondo un'indagine svolta nel 2019, in Italia gli escursionisti sono in aumento, complice la scoperta di sempre più "Vie del Cammino"; Dai risultati del questionario effettuato dalla casa editrice di **Terre di Mezzo** emerge un ritratto del camminatore tipo in Italia: tra le motivazioni che spingono all'escursionismo troviamo in testa il "Fare Trekking" (52%), "Stare nella Natura" (50%) e "Scoprire il territorio" (46%).

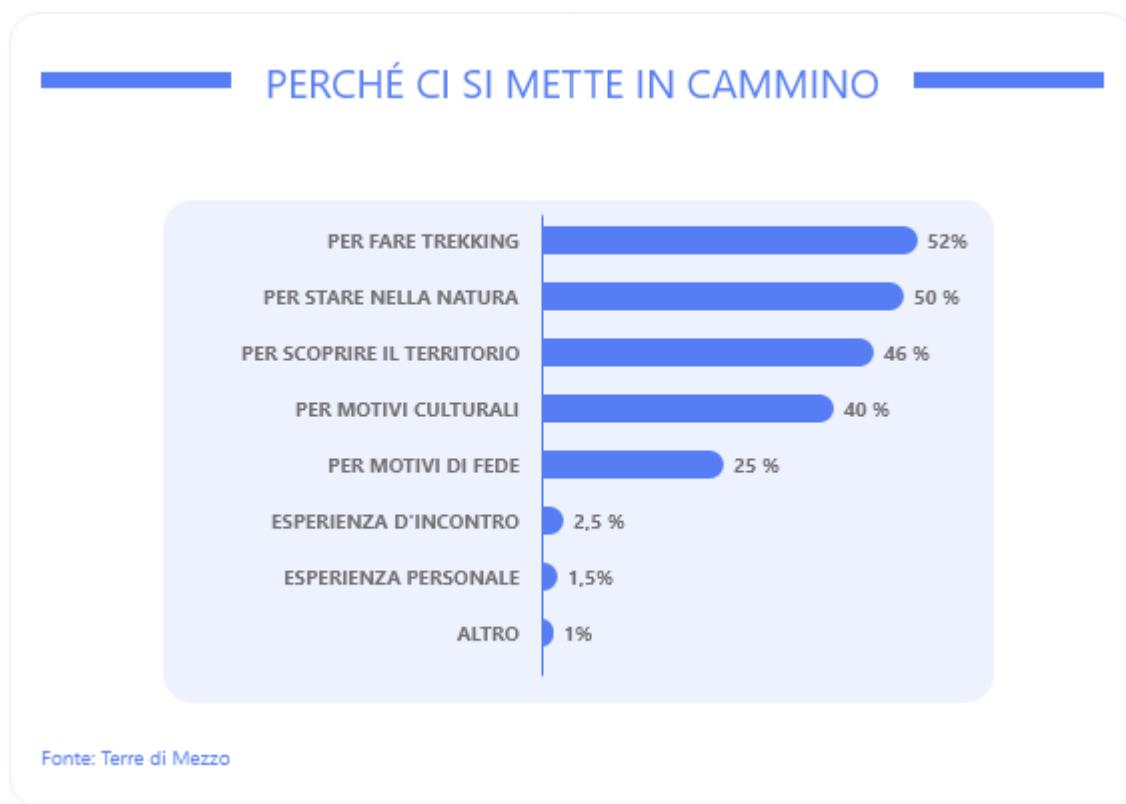


Figura 9: Sondaggio a Risposte Multiple su un campione di escursionisti

In generale uno dei motivi principali che spinge a mettersi in cammino è la voglia di conoscere i borghi ed il territorio; ed è proprio questo uno dei cardini della piattaforma: "Scoperta", conoscere luoghi grazie al contributo di altre persone e ricambiarle facendo conoscere, ampliando in questo modo l'esperienza social, la condivisione che porta allo sviluppo di una community attiva nella piattaforma. Altro dato interessante riguarda l'età: il 27,8% ha tra 51 e 60 anni, il 22,5% tra 61 e 70 anni, il 19,2% tra 41 e 50 anni.

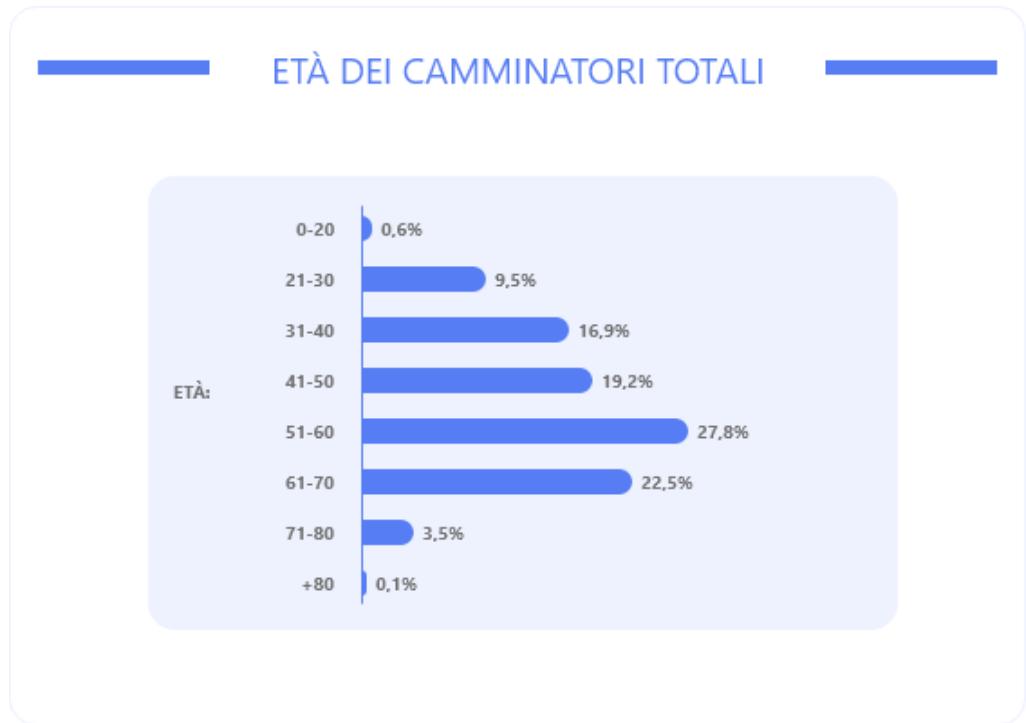


Figura 10: Rilevazione dati (età) su un campione di escursionisti

Vista l'alta percentuale di camminatori over 40, l'applicazione deve essere intuitiva, facile da usare per favorire la commercializzazione tra i meno avvezzi a nuove piattaforme. Dunque, molta enfasi è stata data all'interfaccia utente, usabile ma allo stesso tempo accattivante, per attrarre tutti ed incuriosire anche chi non appartiene al mondo delle escursioni. Secondo i dati ISTAT, la categoria delle escursioni, rilevata esplicitamente per la prima volta nel 2015, è praticata da più di 1 milione 173 mila persone, ai quali si aggiungono chi pratica sport a stretto contatto con la natura (e.g. Orienteering, Ciclismo). In questo gruppo si può identificare l'insieme dei principali utenti finali del Software.

Fonti e sitografia: -[La pratica sportiva in Italia \(istat.it\)](#); -[OUT TREKKING \(sport-press.it\)](#);
-[Terre di Mezzo Editore \(terre.it\)](#).

Ecco in dettaglio le tipologie principali di *personas* a cui si potrebbe riferire il prodotto e che, con maggiore probabilità, lo utilizzerebbero con più frequenza rispetto ad altri:



Roberto

Età: 56

Professione: Architetto

Hobby:

- Musica Classica
- Escursioni nella Natura
- Viaggi

Segni particolari:

- Gran chiacchierone
- Ama la compagnia

Rapporti con la Tecnologia:

- Buoni

Ore di utilizzo di Social Network (al giorno):

- 1



Maria

Età: 42

Professione: Commerciante

Hobby:

- Lunghe Passeggiate
- Yoga e Meditazione

Segni particolari:

- Introversa
- Mangia salutare
- Precisa in tutto quello che fa

Rapporti con la Tecnologia:

- Discreti

Ore di utilizzo di Social Network (al giorno):

- 2/3



Katia

Età: 25

Professione: Studentessa

Hobby:

- Sport tra cui Ciclismo e Corsa
- Andare al Cinema

Segni particolari:

- Ama provare cose nuove
- Divoratrice di libri

Rapporti con la Tecnologia:

- Ottimi

Ore di utilizzo di Social Network (al giorno):

- 4

4.2 Valutazione dell'usabilità a priori

Al fine della valutazione dell'usabilità del Sistema è stato ideato un piano sul testing di *prototipi* al fine di valutarne determinate caratteristiche. I prototipi sviluppati sono di tipo interattivo (permettono infatti una prova d'uso) e dispongono di una *completezza funzionale* il più *verticale* possibile, limitata ai requisiti del Committente (anche se per semplicità non abbiamo preso in considerazione le interazioni Admin-Sistema);

Inoltre, si definiscono prototipi *look&feel*, proprio in merito alla rappresentazione dell'insieme di interazioni Utente-Sistema. Nelle pagine precedenti sono state già introdotte delle schermate di mock-up, e per avere una visione di insieme più completa possibile riportiamo altre schermate utilizzate per la prototipazione:

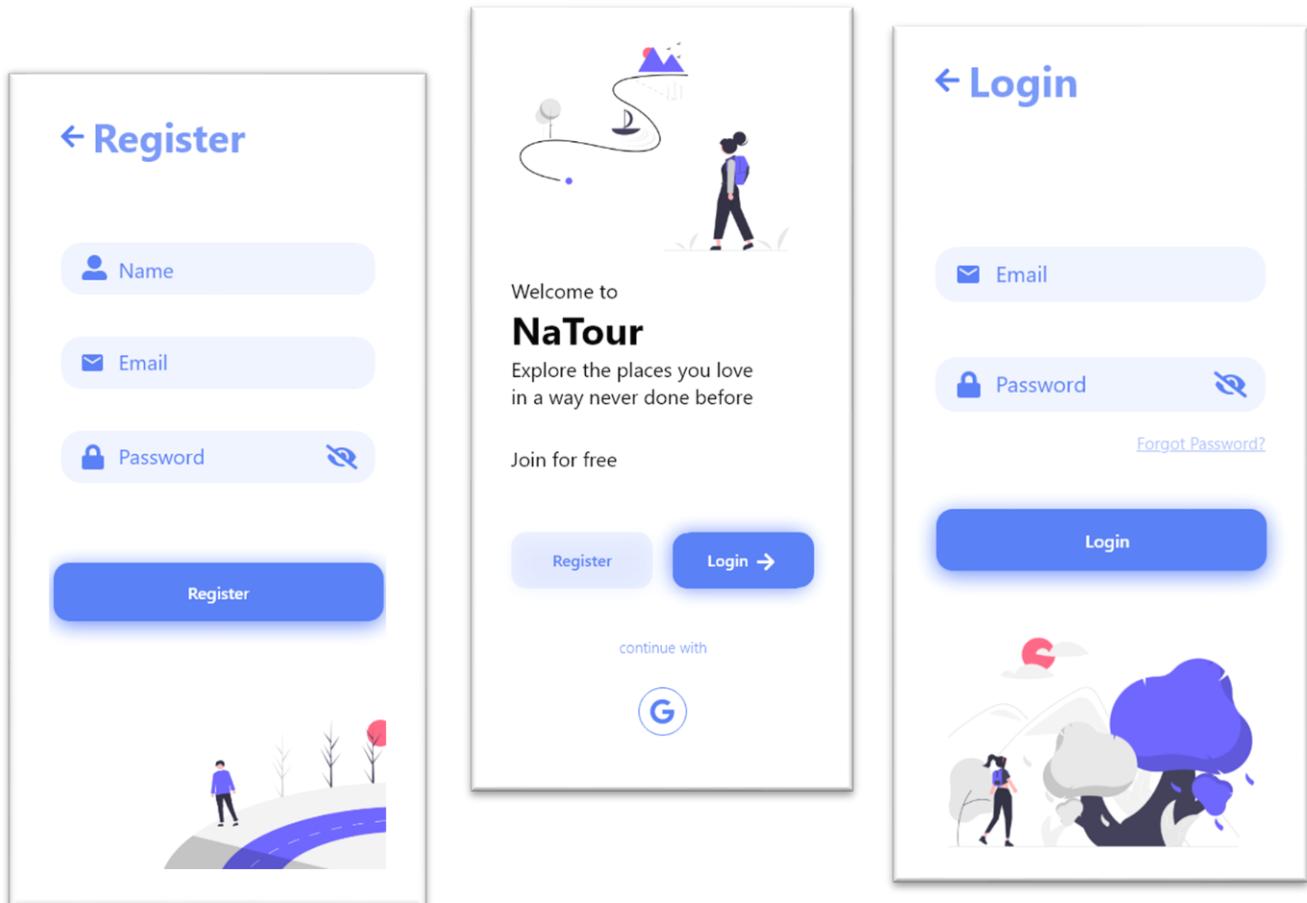


Figura11: Prototipi per l'autenticazione

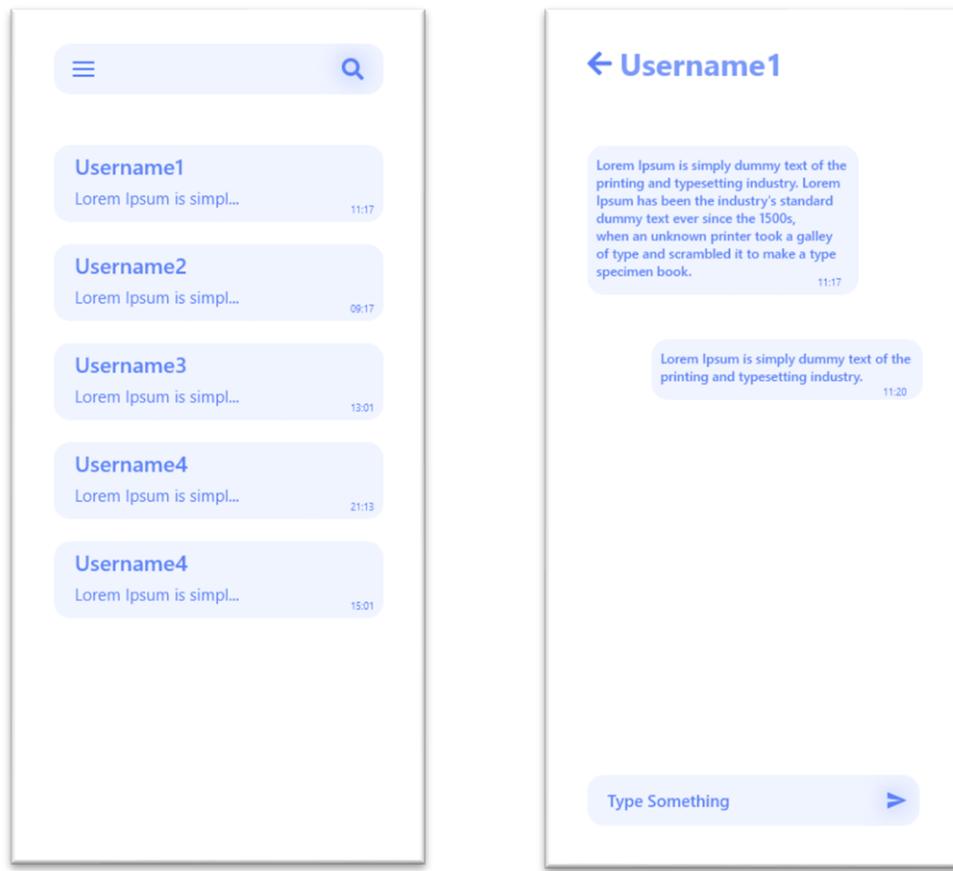


Figura 12: Prototipi per la Messaggistica

Il campione di utenti scelto è legato alla specifica delle personas del paragrafo precedente, in base alla familiarità con la tecnologia e l'utilizzo di applicazioni social; i tester hanno interagito con i prototipi grazie al metodo del *Mago di Oz*: in questo caso viene simulato un normale utilizzo dell'applicazione con l'aiuto di un software di prototipazione dietro le quinte (in particolare **Adobe XD**); quello che accadeva in genere è stato scegliere uno stakeholder che facesse da osservatore, in modo tale da prendere appunti sui pareri e sulle azioni dell'utente (riuscite o meno), mentre l'altro fungeva da guida e poneva domande in merito alle azioni che doveva compiere il tester.

In generale, l'interfaccia grafica è stata apprezzata, anche se sono state poste critiche per quanto riguarda la schermata di inserimento di un itinerario (un po' troppo confusionaria). Sebbene ci siano state difficoltà (alcuni task hanno richiesto il nostro aiuto) ci riteniamo soddisfatti di questa prima fase di feedback dagli utenti.

	Registrazione	Accesso	Aggiunta di itinerari	Visualizzare caratteristiche di un itinerario	Visualizzare su Mappa l'itinerario	Aggiunta foto ad Itinerario	Mandare Messaggio al creatore	Visualizzare Chat con Utenti
#1 K	✓	✓	✓	✓	✓	✓	✓	✓
#2 M	✓	✓	✓	✓	✓	✓		
#3 R	✓	✓		P ✓	✓	✓		✓
#4 R	✓	✓	✓		✓	p ✓		✓
#5 M	✓	✓	✓	✓	✓	✓	✓	

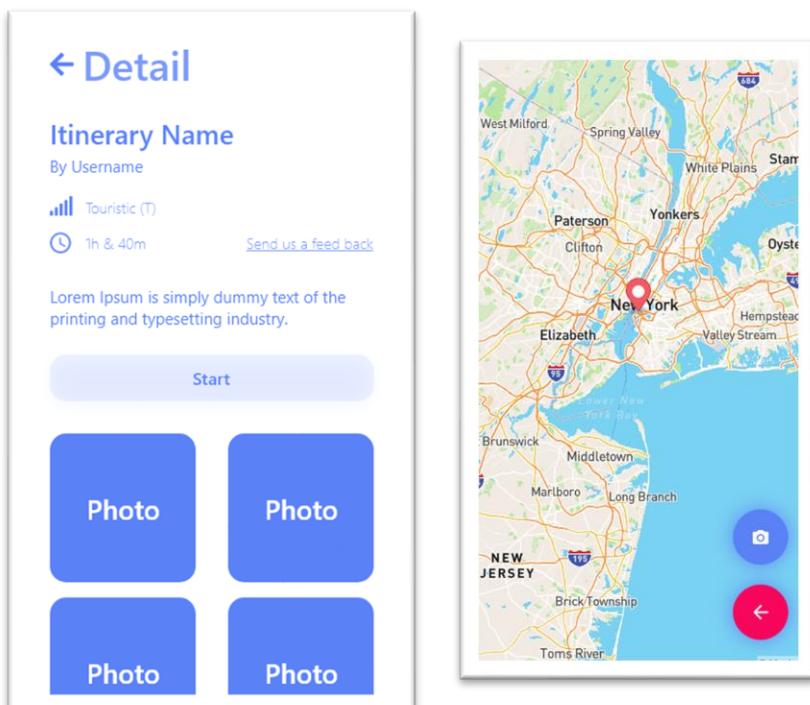


Figura 13: Prototipi per la visualizzazione di un itinerario

I prototipi per la visualizzazione di un Itinerario sono stati modificati rispetto a quelli iniziali, per permettere accesso a funzionalità come l'invio di un messaggio al creatore del sentiero o aggiungere foto al percorso. Dalla tabella dei task dati ai tester si può notare che il compito più difficoltoso sia quello legato alla visualizzazione delle chats e l'invio di messaggi ad un creatore di un Itinerario (il consiglio dato per facilitare le azioni è stato inserire qualche sorta di icona o messaggio per rendere i task più comprensibili da eseguire).

5. Modelli di Dominio

5.1 Oggetti e Relazioni del Dominio

L'entità principale del dominio è l'**<Itinerario>**, caratterizzato da un nome, una durata, una difficoltà, una descrizione e una data di creazione; ogni itinerario può avere più **<POI>** punti di interesse con una eventuale posizione geografica e una fotografia del punto. Ad ogni Itinerario inoltre corrisponde biunivocamente un **<Tracciato Geografico>**, composto da un punto iniziale, uno finale e una serie di **<Punti Intermedi>**. Si noti la scelta di distinguere tra Itinerario e Tracciato corrispondente, per manipolare al meglio la parte descrittiva e quella meramente geografica di un Percorso.

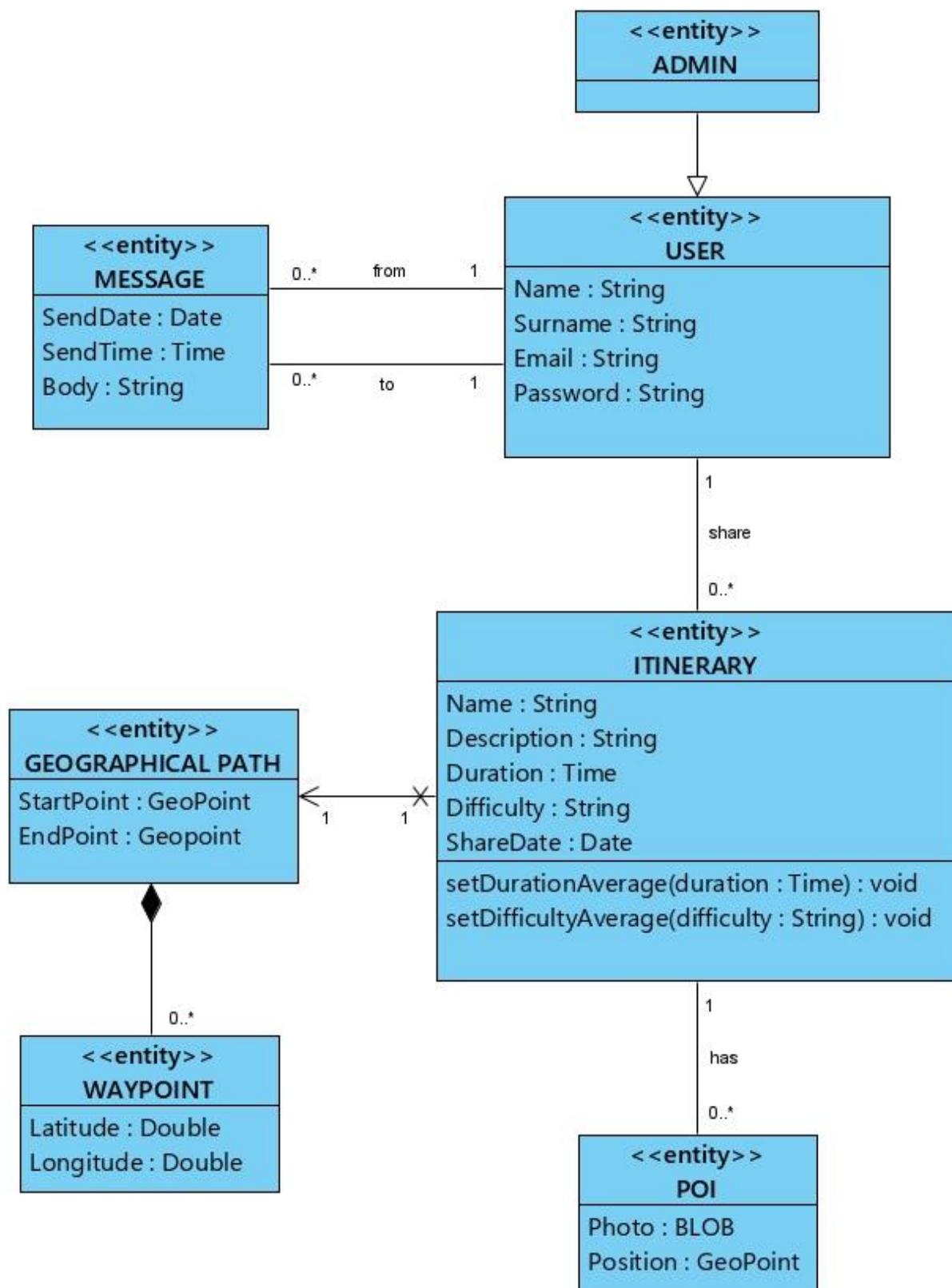
L'**<Utente>** si occupa dell'inserimento di Itinerari e può eventualmente fornire feedback per un Itinerario (per il calcolo della media di durata e/o difficoltà).

Gli Utenti possono scambiarsi **<Messaggi>** in formato testuale, con data e ora di invio.

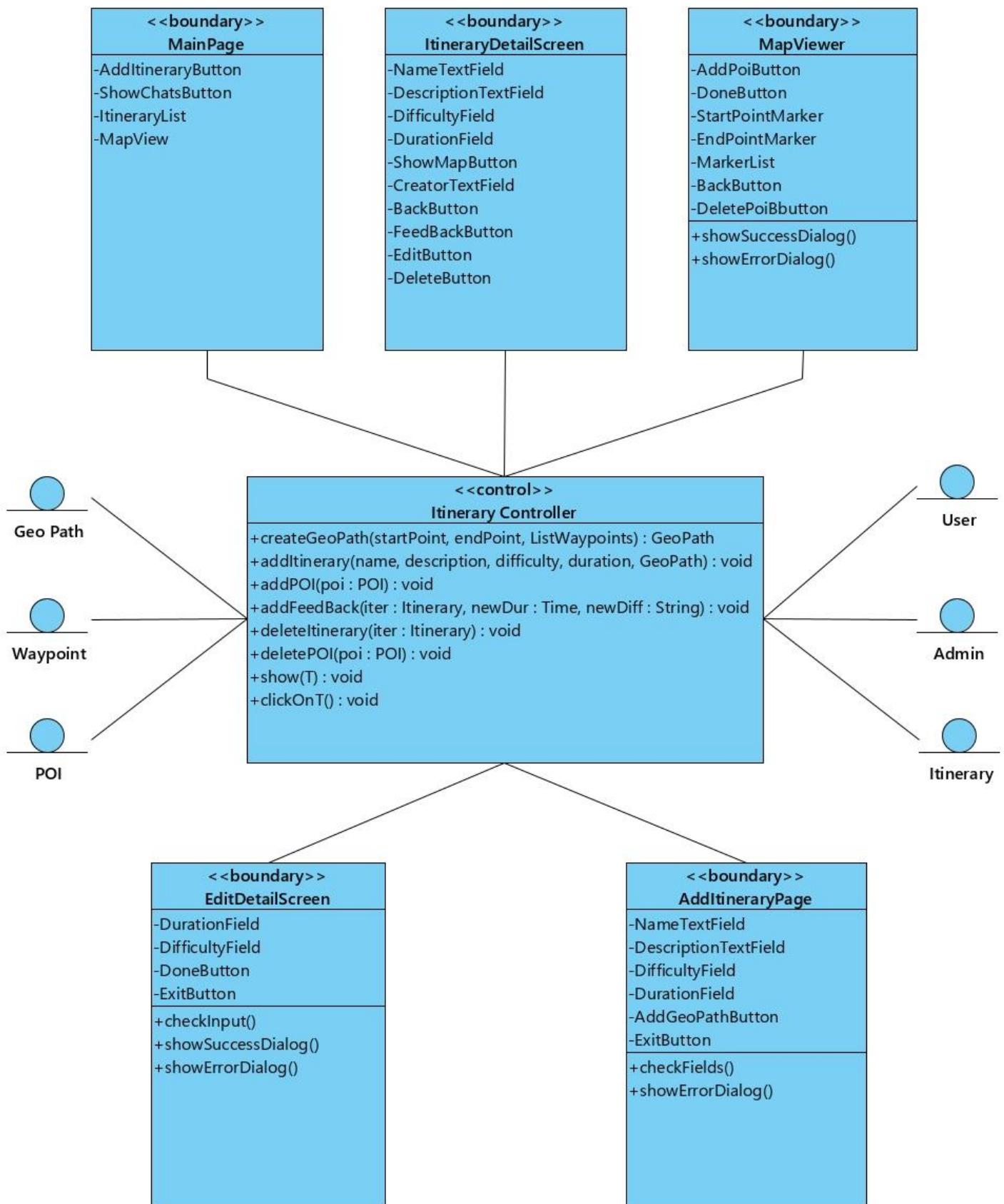
Infine, **<Admin>** è una specializzazione di Utente, che possiede dei privilegi rispetto ad un normale utilizzatore del software: rimozione e modifica di itinerari nella piattaforma.

Nelle pagine successive si troveranno in formato Class Diagram UML gli oggetti di analisi, divisi secondo l'euristica three-object-type che prevede un raggruppamento in tre layer: Entity (i concetti del Dominio), Boundary (l'interfaccia al Sistema) e Control (la logica del Sistema); sono inoltre descritti i comportamenti delle classi di analisi attraverso Sequence Diagrams (avvalendosi anche dei Data Access Objects) e Activity Diagrams.

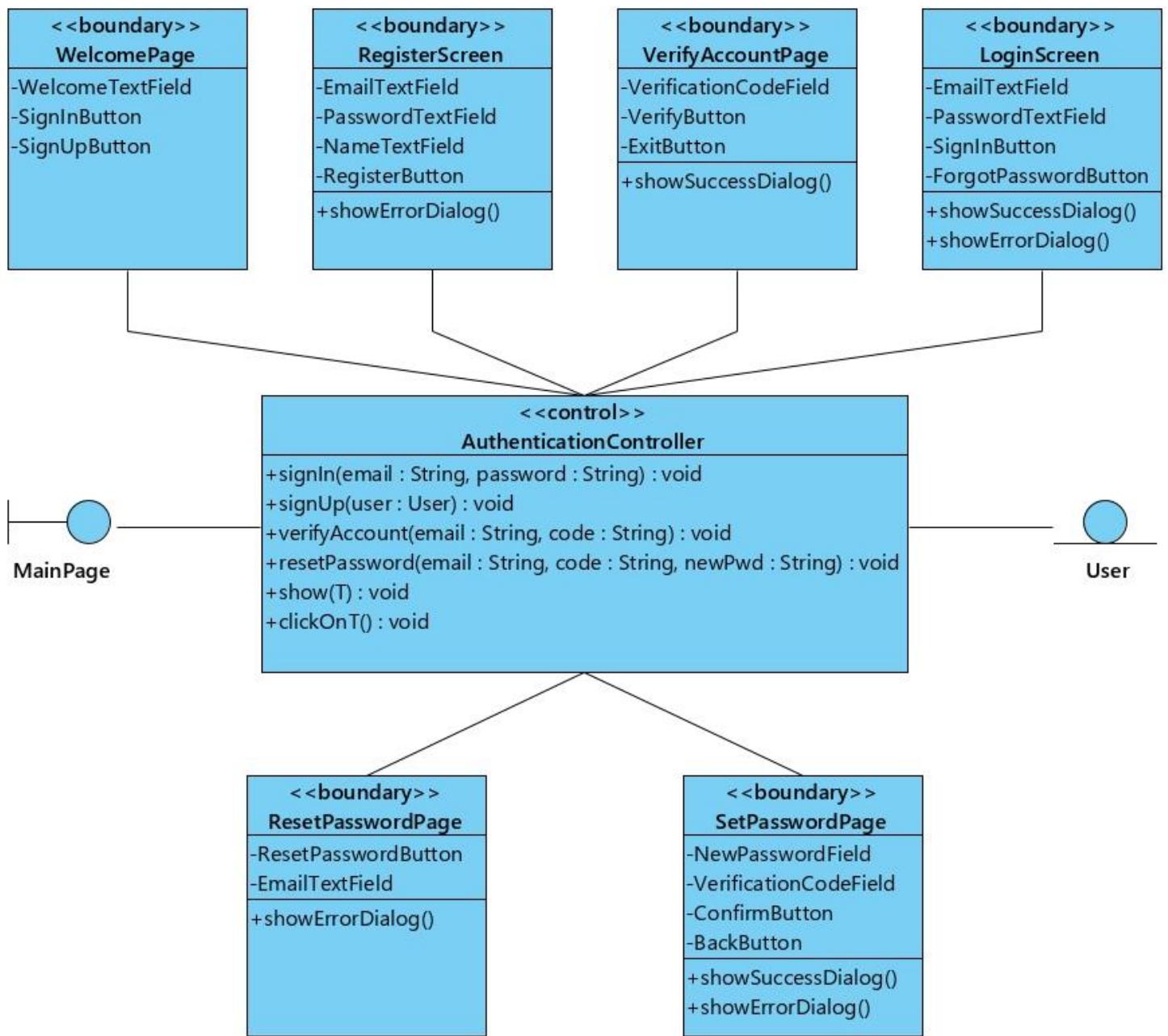
5.2 Class Diagrams di Analisi



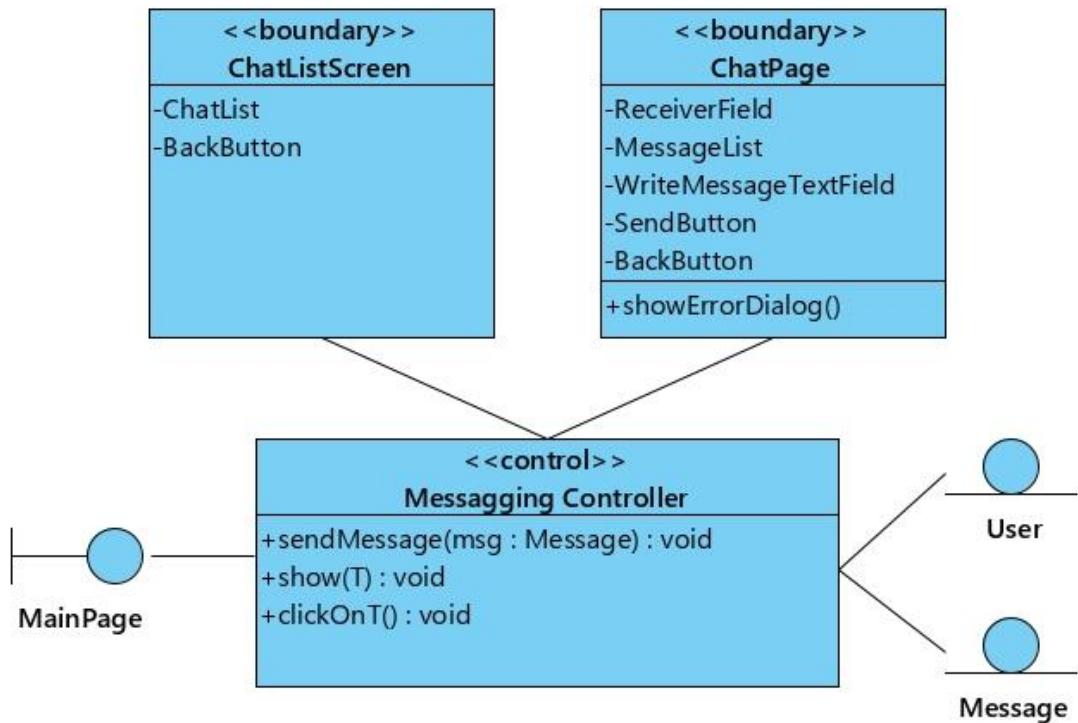
Class Diagram 1: Entity Layer (implicit getters e setters)



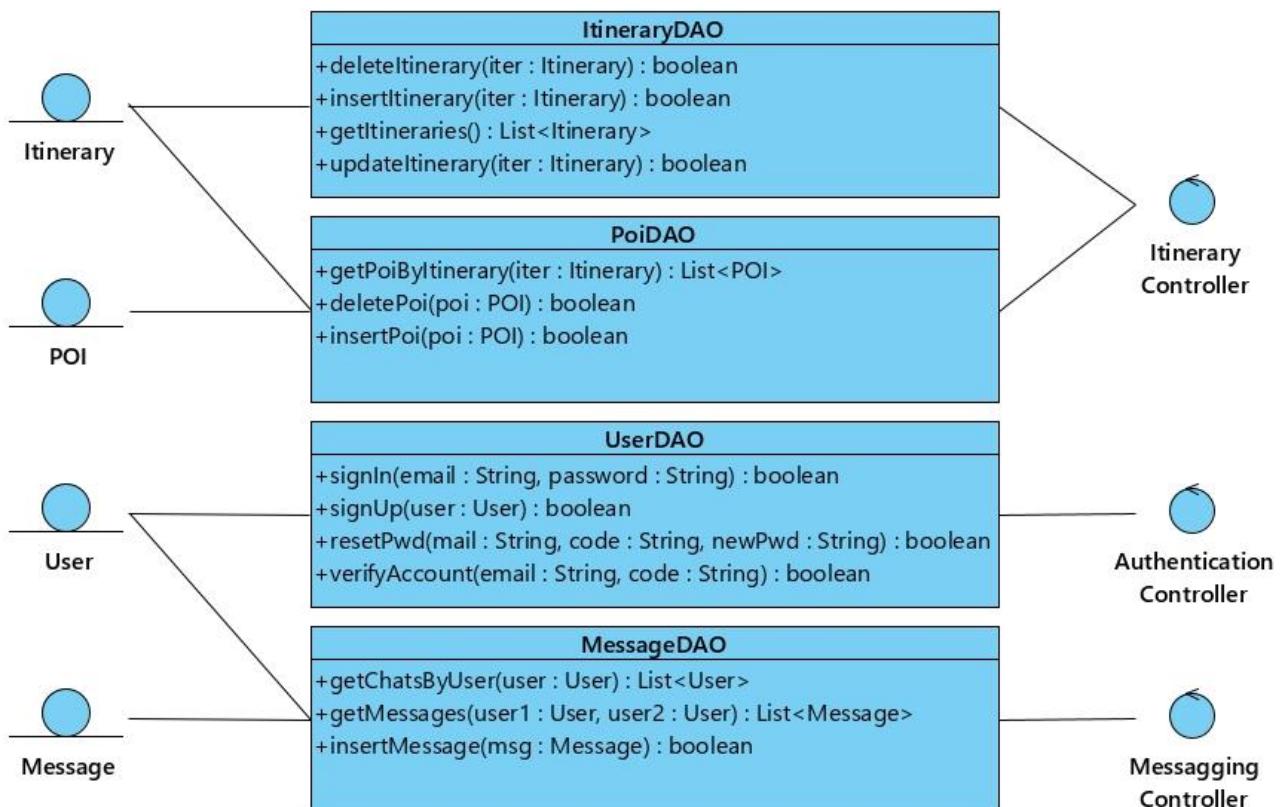
Class Diagram 2: Itinerary Management



Class Diagram 3: Authentication

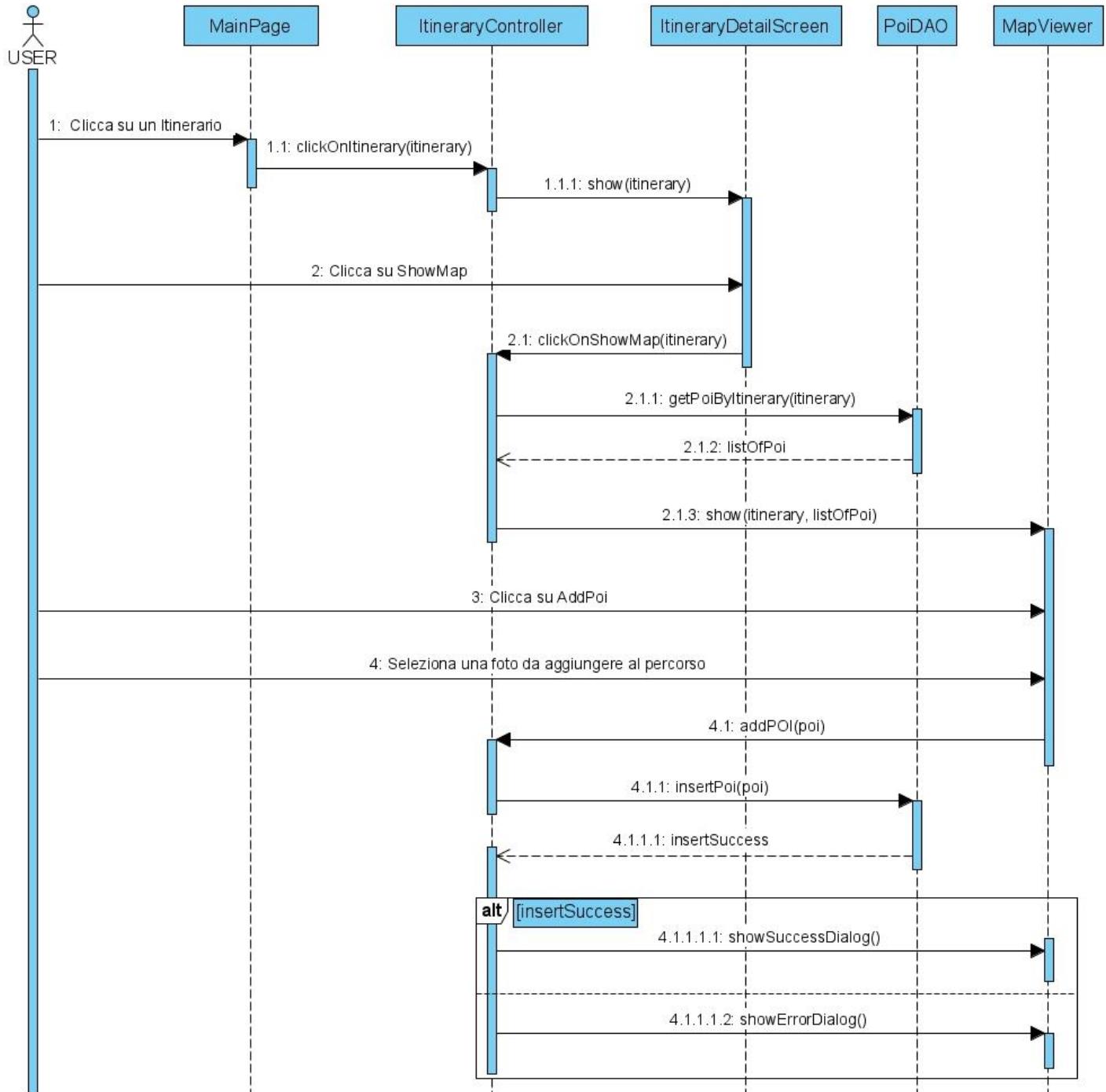


Class Diagram 4: Messaging

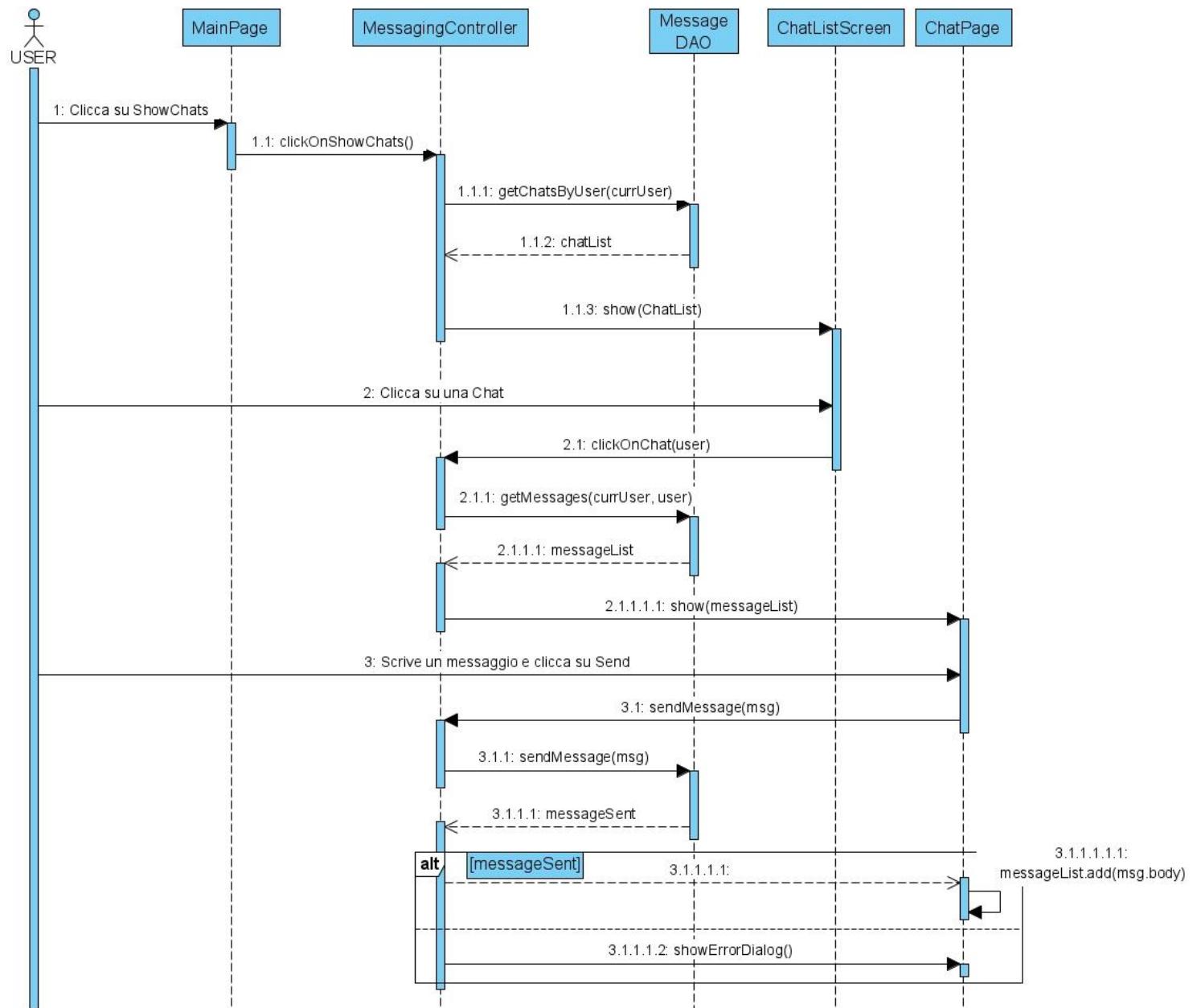


Class Diagram 5: DAO Layer

5.3 Sequence Diagrams di Analisi

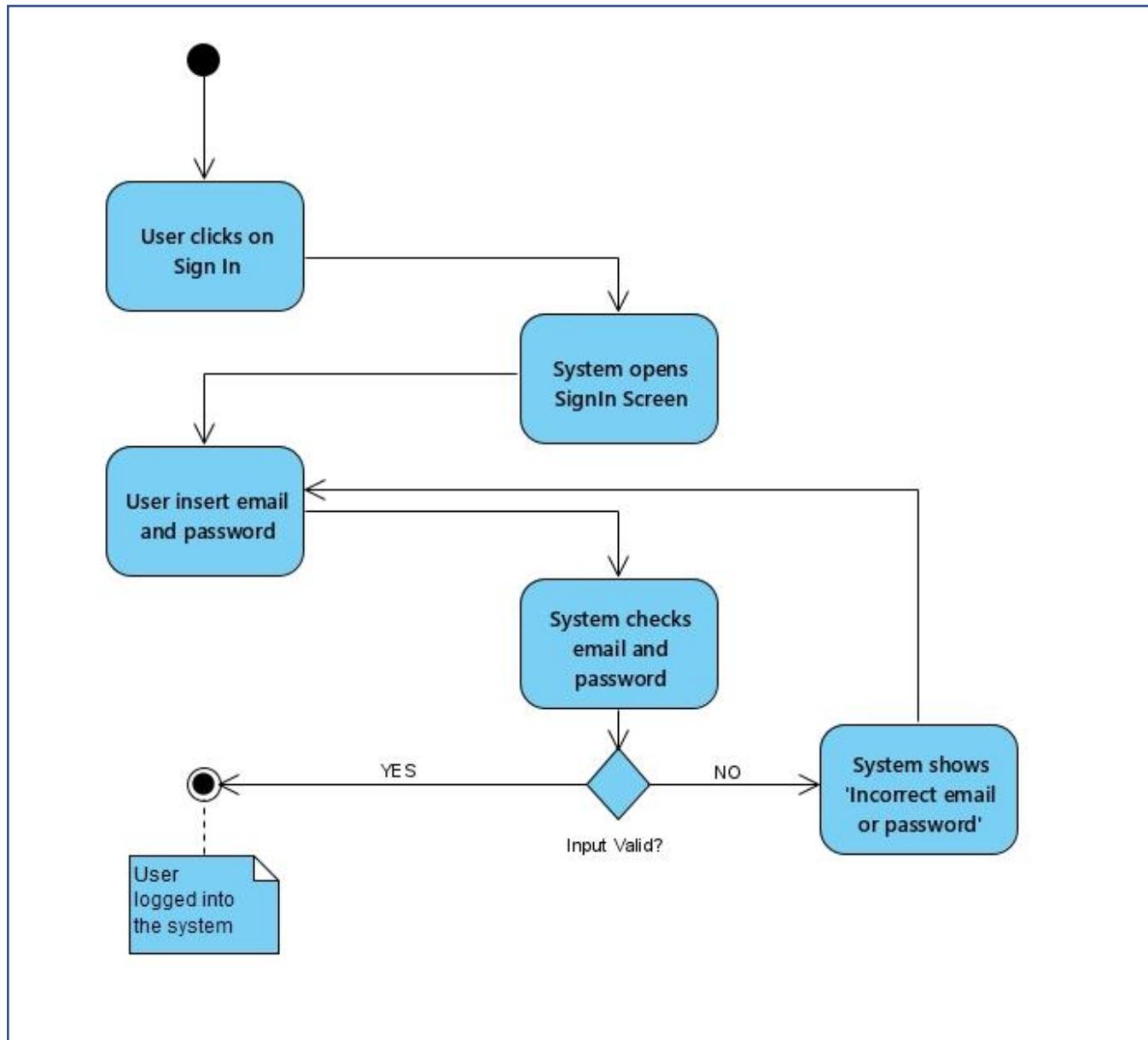


Sequence Diagram 1: Add Point of Interest

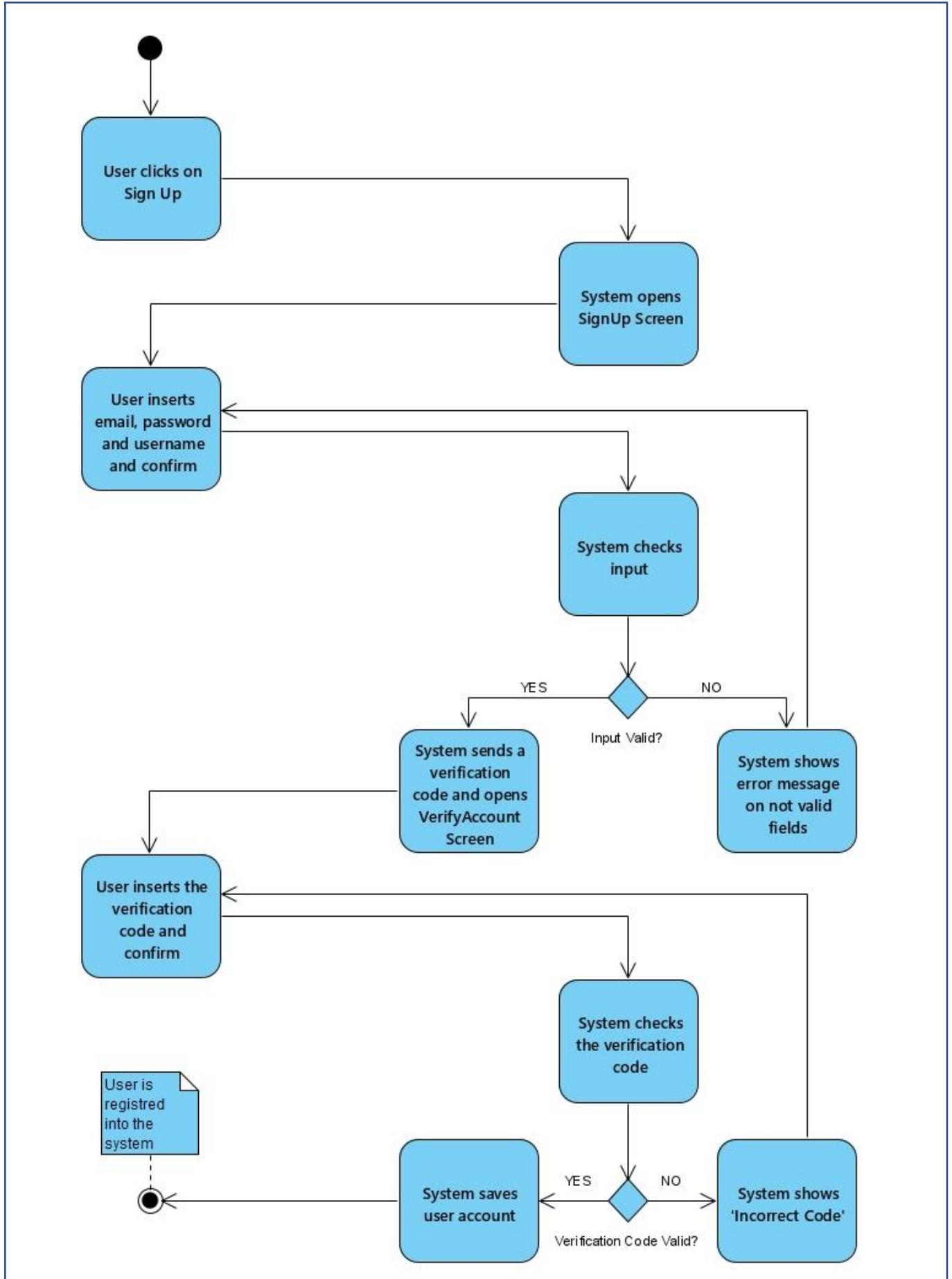


Sequence Diagram 2: Reply to a message

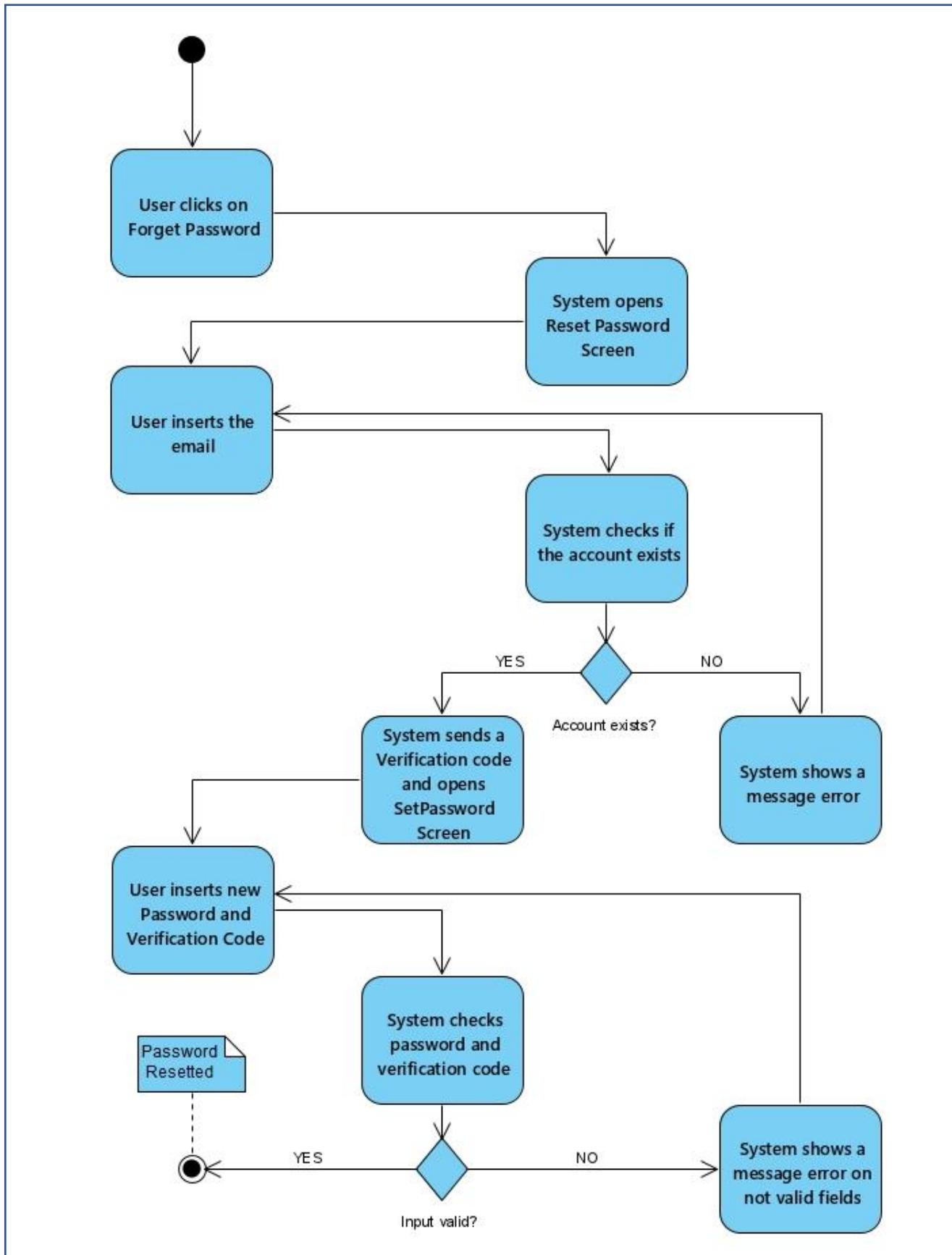
5.4 Activity Diagrams



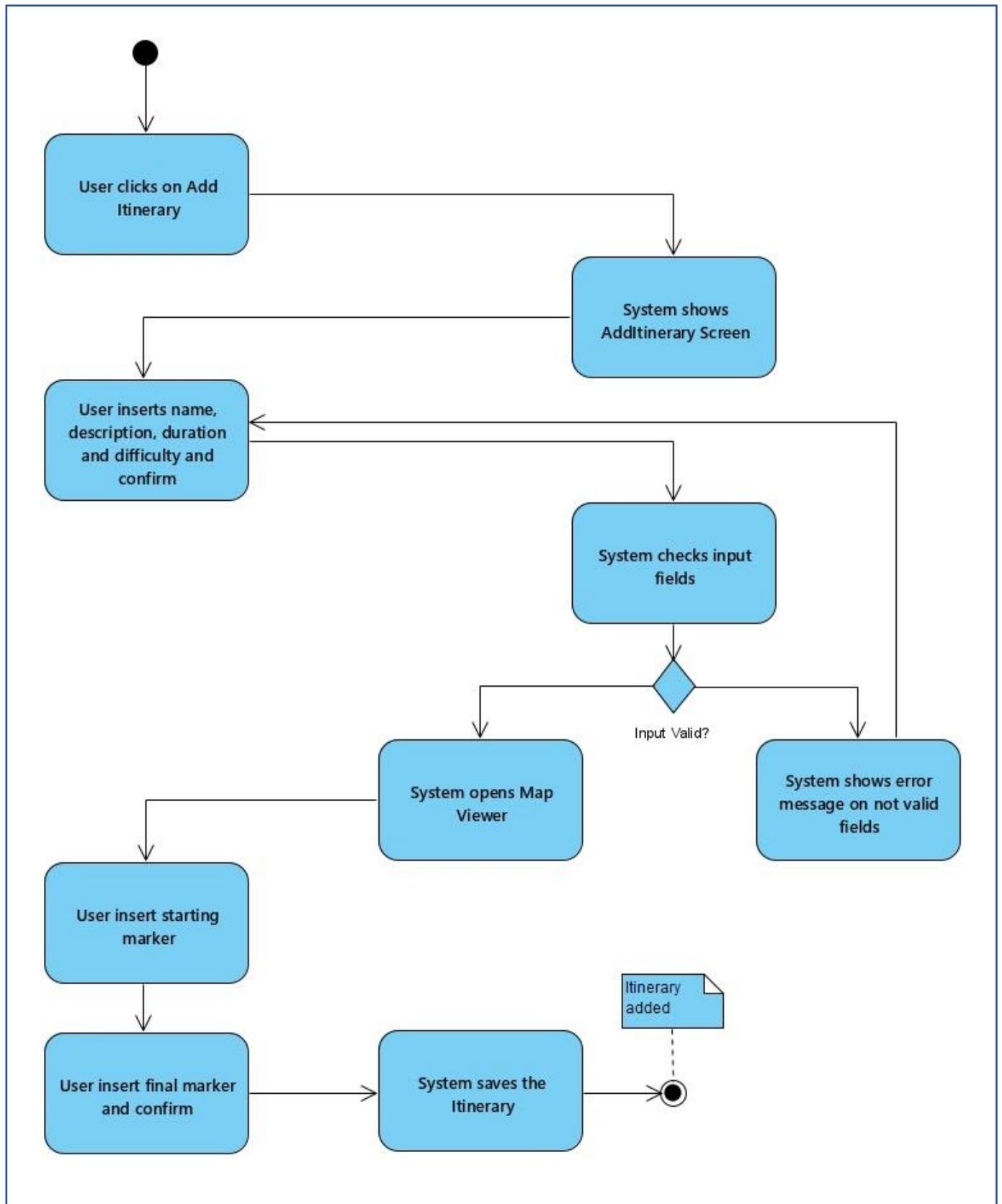
Activity Diagram 1: Login



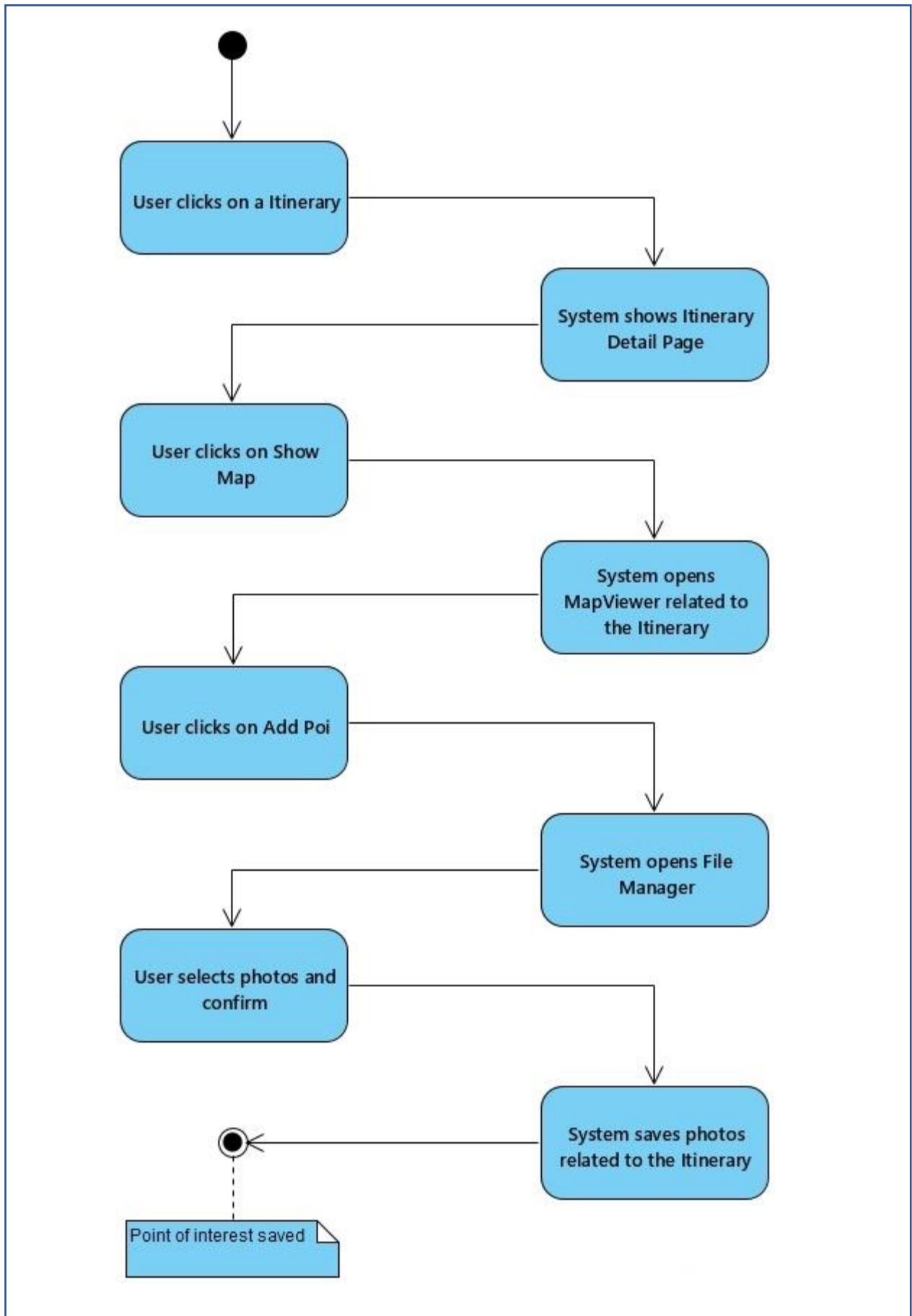
Activity Diagram 2: Registration



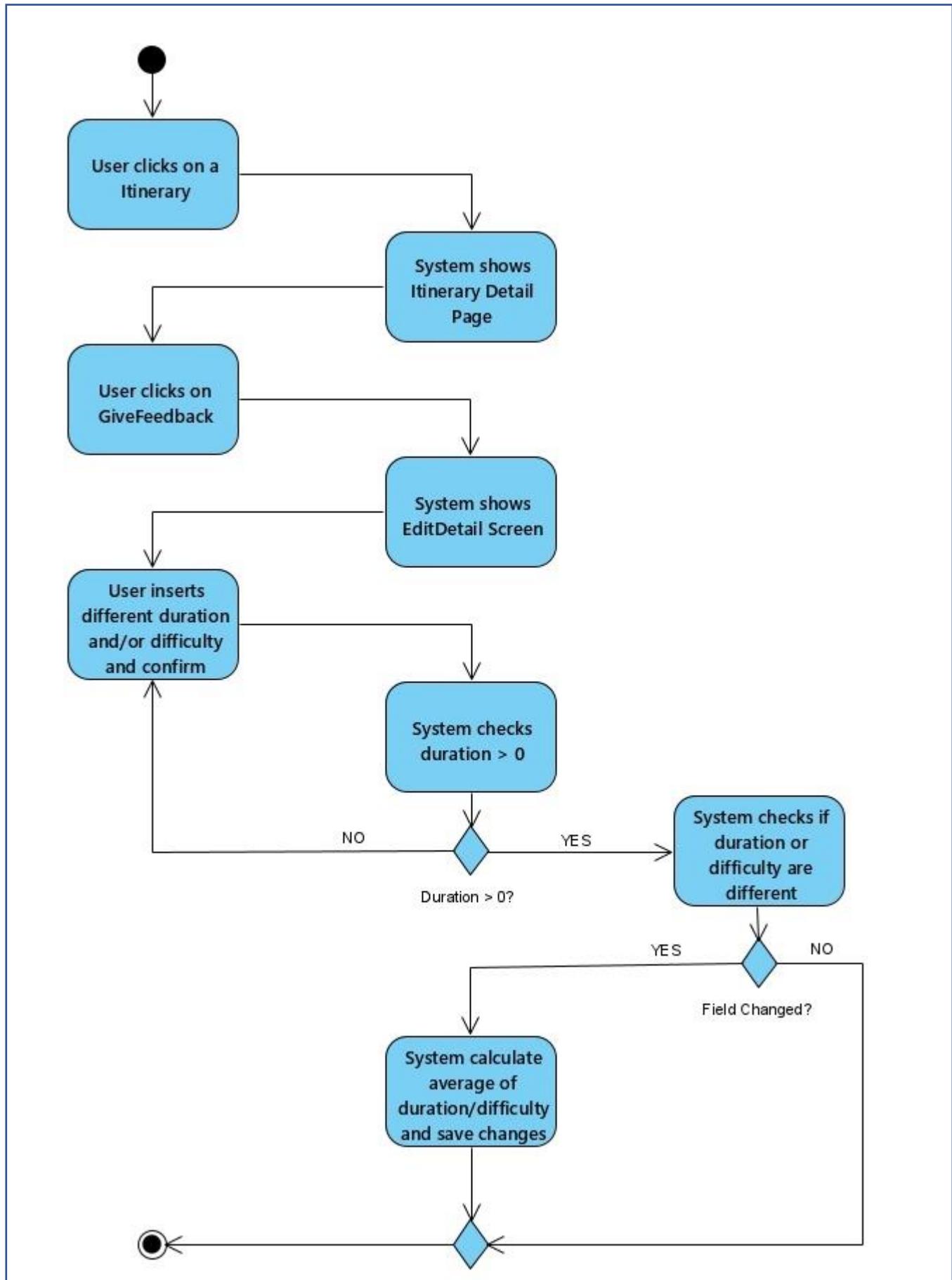
Activity Diagram 3: Reset Password



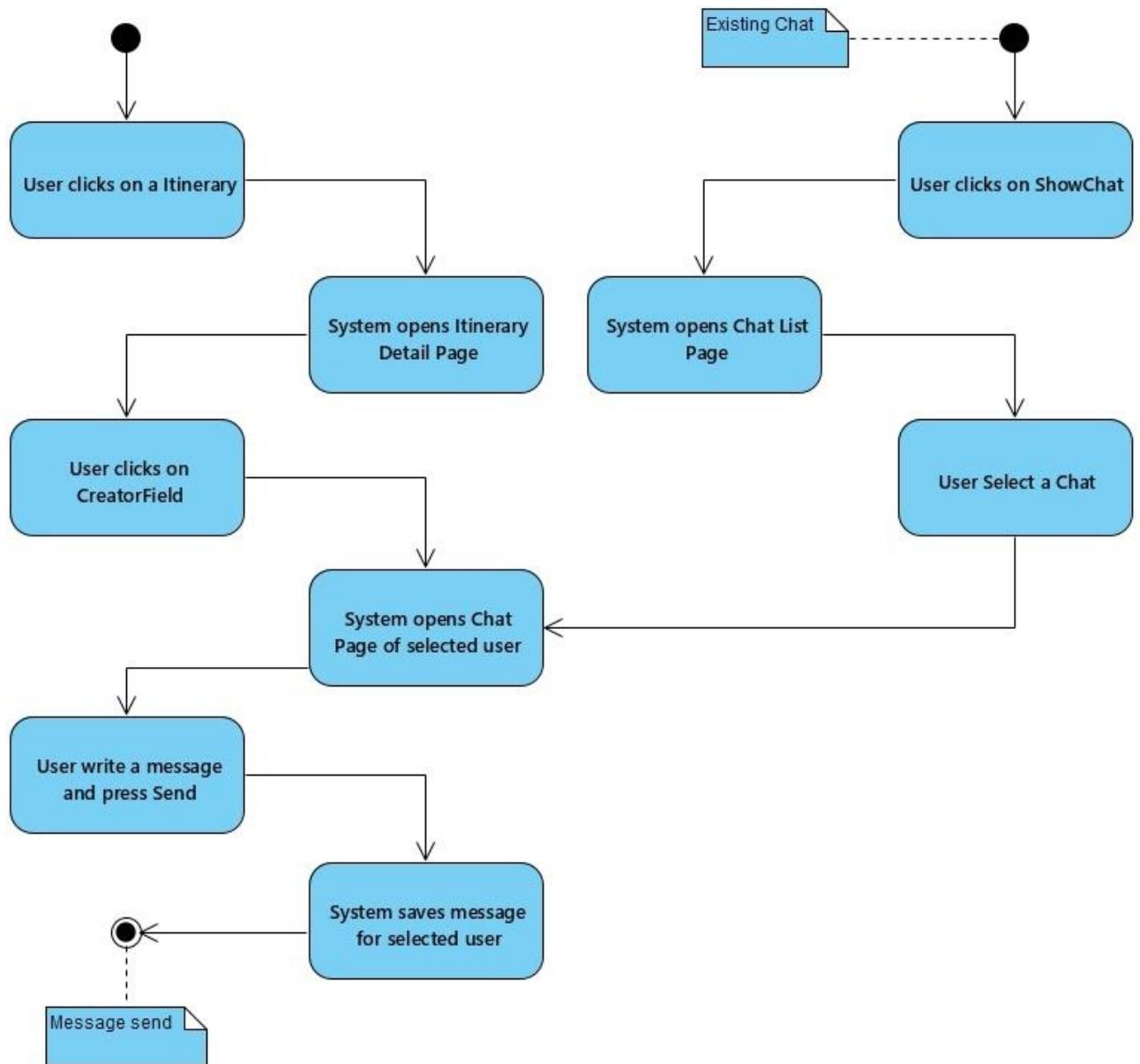
Activity Diagram 4: Add Itinerary



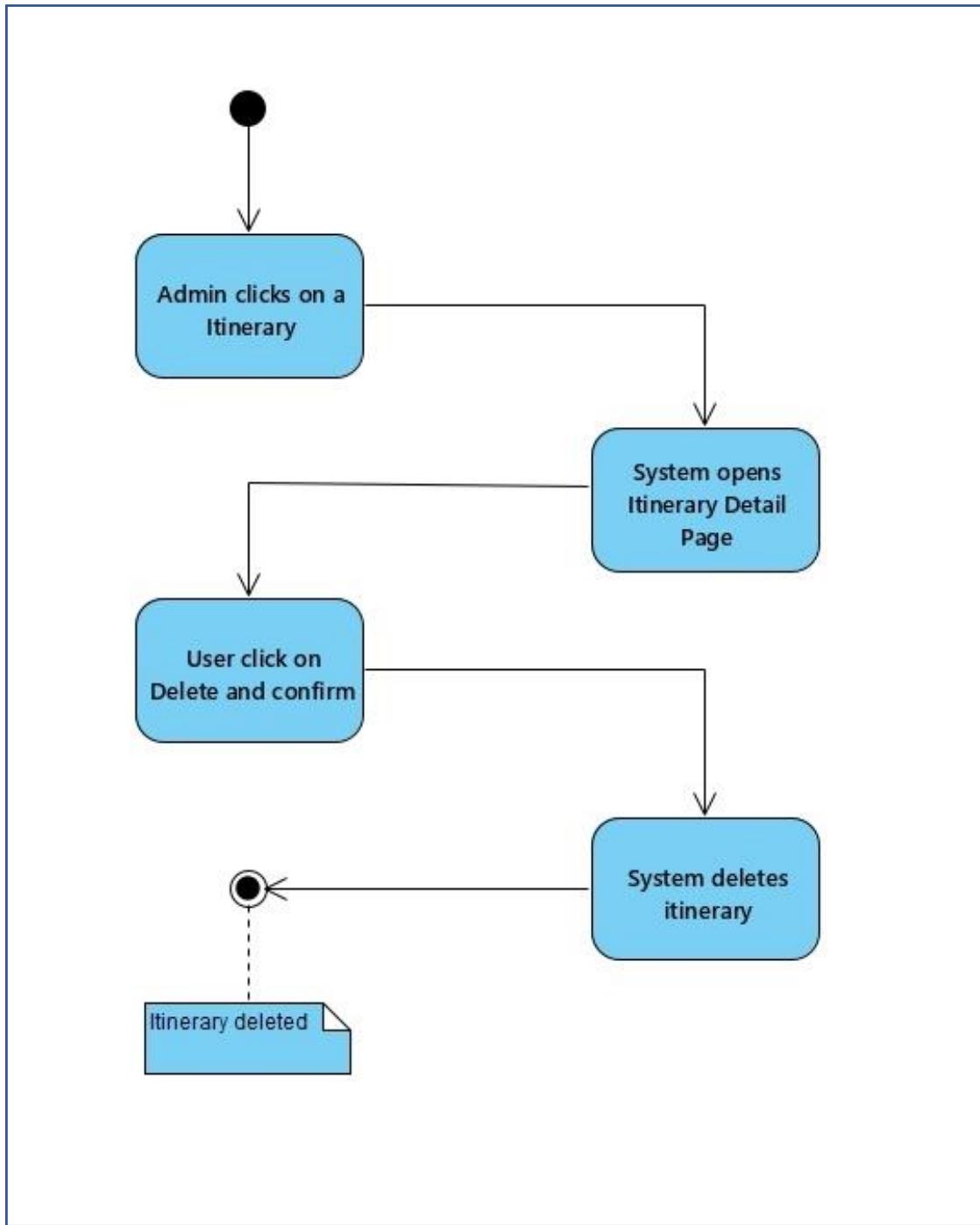
Activity Diagram 5: Add Photo



Activity Diagram 6: Feedback



Activity Diagram 7: Messaging



Activity Diagram 8: Delete Itinerary

PARTE II: Design del Sistema

6. Architettura del Sistema

6.1 Analisi dell'Architettura Esterna

Per quanto concerne l'architettura esterna del Sistema, attraverso una suddivisione in layer, in cui ogni livello comunica direttamente solo con quello sottostante, è stato possibile costruire un'architettura chiusa e sicura.

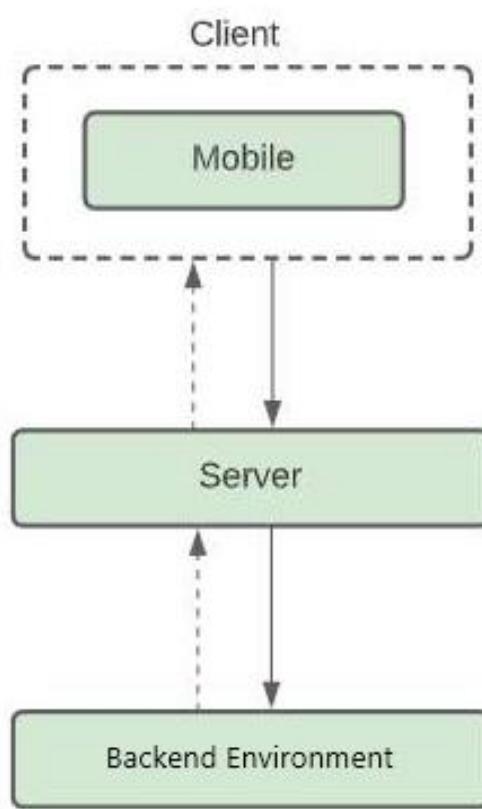


Figura 1: Three-Tier External Architecture

Il client fa affidamento ai servizi offerti dal Server, come ad esempio rispondere alle richieste di una risorsa, caricare una risorsa da parte del client e così via; il server a sua volta propaga le richieste ai servizi che gestiscono i dati per poi mandare un responso al client. Dunque, nessuno può inserire, modificare o eliminare risorse a parte il server che funge da “amministratore” del Backend Environment.

6.2 Architettura Backend

L'architettura del Backend è stata realizzata usufruendo dei servizi offerti da Amazon Web Services. Il server è stato realizzato con un approccio serverless, tale da eliminare completamente le attività di gestione di una macchina in rete, quali distribuzione di carico, manutenzione o aggiornamenti: semplicemente è solo necessario creare pacchetti di codice per il deployment in cloud, dato che in genere i servizi di cloud computing sono scalati automaticamente in base all'uso e vengono aggiornati continuamente dal service provider.

Come già affermato, il client non ha la possibilità di accedere direttamente ai servizi se non dopo l'autenticazione ed inoltre tali servizi non sono richiesti direttamente dal client, ma devono necessariamente passare per la logica serverless; ad esempio, per l'upload di una foto, il client invia la richiesta di upload al server, il quale risponde con un URI (fornito dal bucket) adempito per l'operazione. I servizi sono offerti attraverso delle API proprietarie, che offrono un'interfaccia sicura e adattabile per l'architettura.

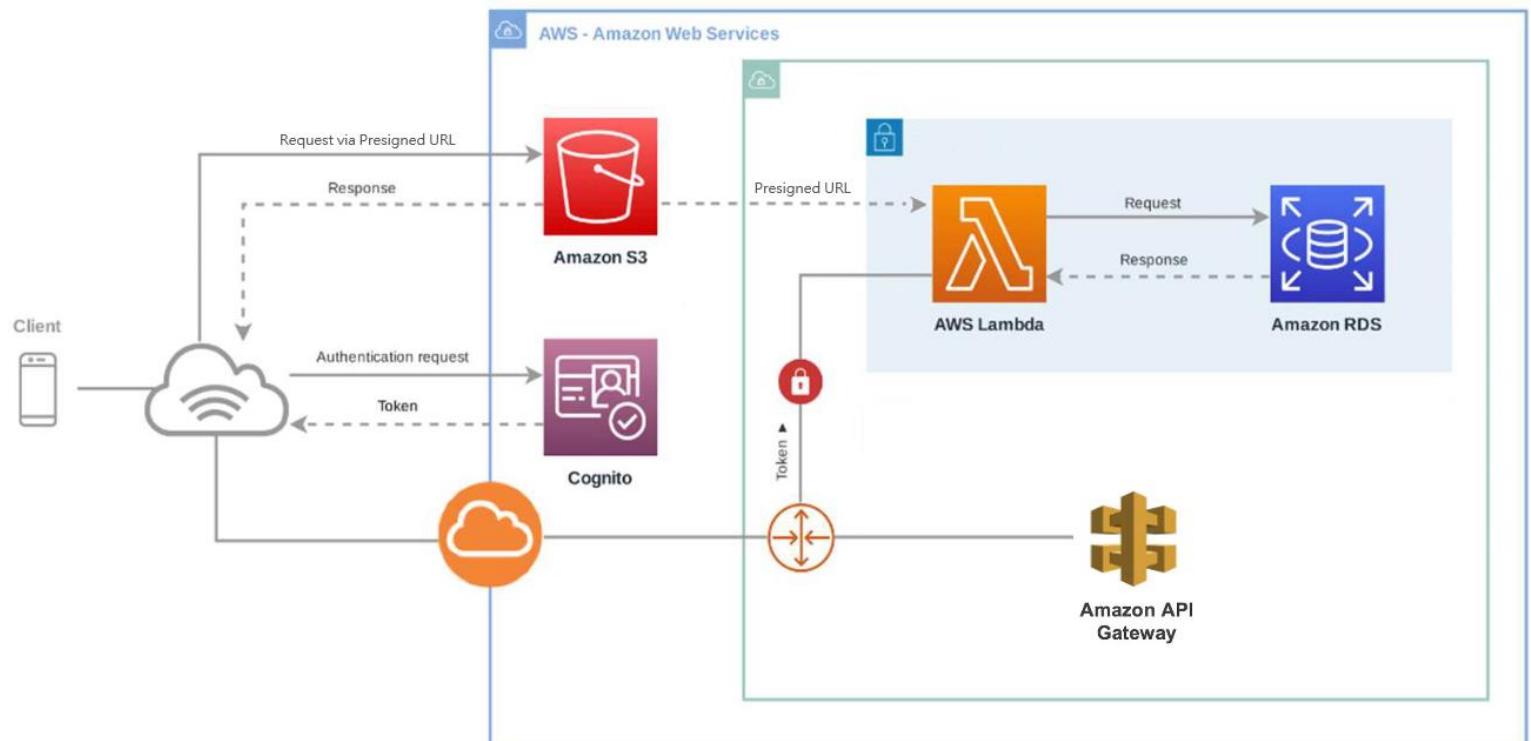


Figura 2: Backend in cloud

Le API realizzate sono di due tipi:

- 1) RESTful (REpresentational State Transfer) API**
- 2) Web Socket API**

1) Le prime (***RESTful API***) definiscono il protocollo di comunicazione tra il client, che richiede una risorsa o un servizio, e il server che la fornisce. Le risorse vengono scambiate tra le due entità tramite una rappresentazione in formato JSON (JavaScript Object Notation), uno tra i più diffusi nonché consigliabile se si lavora con un linguaggio di programmazione server-side basato su JavaScript. Infatti, per il server si è adottato il linguaggio Node.js con l'ausilio del framework Express e il protocollo di comunicazione HTTP per gestire le richieste. La comunicazione tra client e server è stateless, in quanto nessuna delle due entità mantiene informazioni sulle precedenti interazioni, rendendo ogni richiesta distinta con le altre. Di seguito è riportato l'elenco delle funzionalità offerte dalla API:

METODO HTTP	PERCORSO	DESCRIZIONE
GET	/itineraries	Permette il recupero di itinerari dal database
	/photos	Permette il recupero di presigned-url per il download di foto dal bucket S3
	/chats	Permette il recupero delle chats di un particolare utente

METODO HTTP	PERCORSO	DESCRIZIONE
<i>POST</i>	/itineraries	Permette di aggiungere un itinerario creato da un utente nel database

METODO HTTP	PERCORSO	DESCRIZIONE
<i>PUT</i>	/itineraries	Permette di sostituire un itinerario aggiornato modificato dall'Admin
	/feedback	Permette il sostituire un itinerario attraverso i feedback degli utenti
	/photos	Permette il recupero di presigned-url per l'upload di foto nel bucket S3

METODO HTTP	PERCORSO	DESCRIZIONE
<i>DELETE</i>	/itineraries	Permette di rimuovere un itinerario nel Database da parte dell'Admin

2) Le seconde (**Web Socket API**) basate su Socket TCP full-duplex (che permettono lo scambio di dati in entrata e in uscita contemporaneamente) sono risultate indispensabili al fine di fornire un servizio di chatting real-time; Quando un client apre la Socket, viene registrata la sua sessione in modo tale che altri client possano mandare direttamente un messaggio sulla Socket. Anche in questo caso, il server che gestisce le sessioni degli utenti online è di tipo serverless.

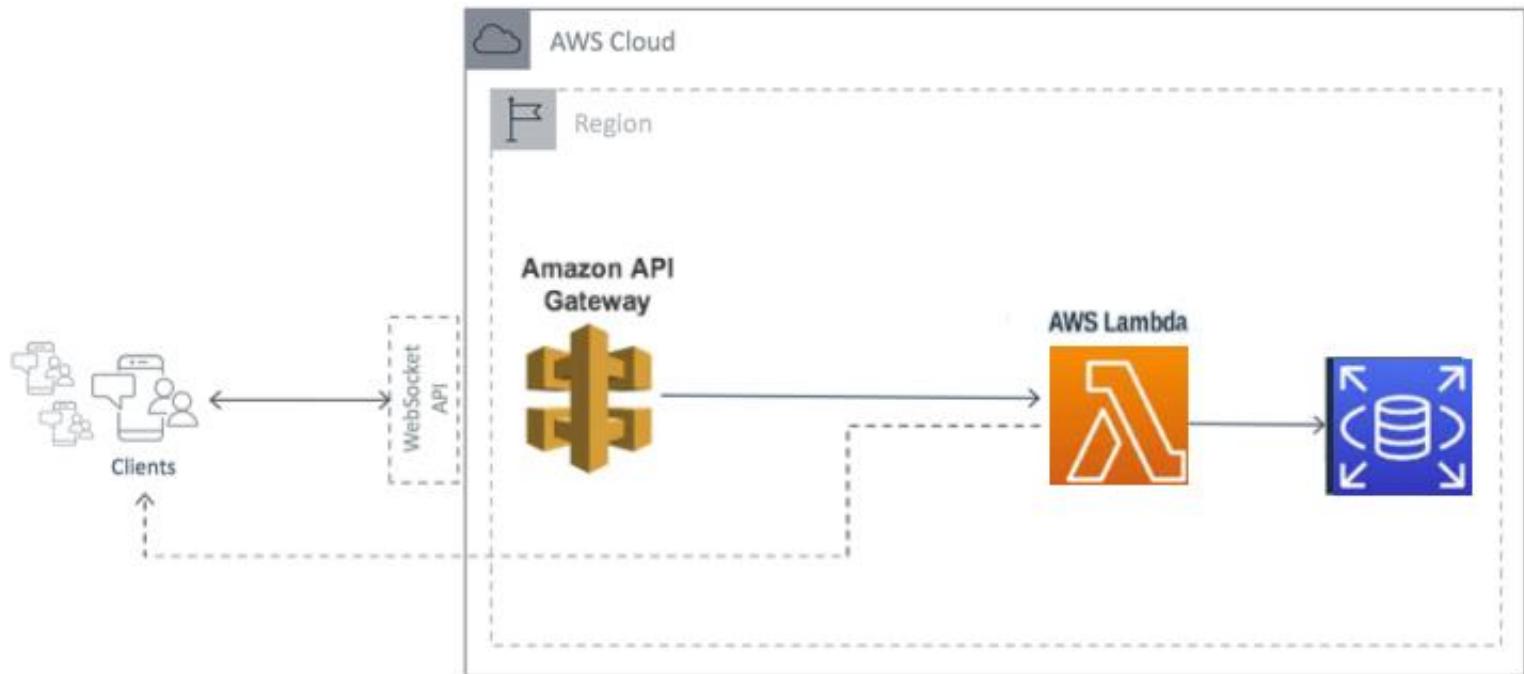


Figura 3: Web Socket API

6.3 Servizi cloud utilizzati

- [API Gateway](#)

è un servizio completamente gestito che semplifica la creazione, la pubblicazione, la manutenzione, il monitoraggio e la protezione delle API su qualsiasi scala. Le API fungono da “porta di entrata” per consentire l’accesso delle applicazioni alle funzionalità dai servizi back-end.

- [Lambda](#)

è una piattaforma di calcolo serverless ed event-driven.

Lo scopo di Lambda, comparato ad AWS EC2 e altri server tradizionali, è di semplificare la costruzione di applicazioni on-demand:

infatti, esegue automaticamente il codice senza dover effettuare il provisioning né gestire server. Inoltre, ridimensiona automaticamente le risorse dell'applicazione eseguendo il codice in risposta a ogni trigger.

Il codice viene eseguito in parallelo ed elabora ciascun trigger separatamente, ricalibrando le risorse in base al carico di lavoro.

- [Cognito](#)

fornisce autenticazione, autorizzazione e gestione degli utenti per le applicazioni Web e mobili. Gli utenti possono accedere (dopo la registrazione) direttamente con un'e-mail e una password, oppure tramite Providers di terze parti, come Google di cui abbiamo fornito l'implementazione. Grazie all'integrazione con API Gateway, solo gli utenti registrati (memorizzati nel pool d'utenza) possono accedere ai servizi dell'applicativo.

- [Simple Storage Service \(S3\)](#)

è un servizio di storage di oggetti che offre scalabilità, disponibilità dei dati e sicurezza. Amazon S3 offre caratteristiche di gestione semplici da utilizzare che consentono di organizzare i dati in modo ottimale.

- Relational Database Service (RDS)

Semplifica l'impostazione, il funzionamento e il dimensionamento di database relazionali nel cloud. Questo servizio fornisce una capacità ridimensionabile, automatizzando al tempo stesso le attività di amministrazione del database più dispendiose in termini di tempo, quali il provisioning di hardware, l'impostazione di database, gli aggiornamenti e i backup.

6.4 Architettura Frontend

L'applicazione su client Android si basa sul linguaggio Object-Oriented (come richiesto dal committente) Java.

Le classi sono state raggruppate in tre macro-package, rispettivamente *application* (per la logica di controllo), *model* (per la gestione della persistenza dei dati) e *presentation* (per le classi di interfaccia utente), sviluppando un'architettura di tipo **Three-Layer**, in cui ogni strato comunica con lo strato immediatamente superiore e inferiore.

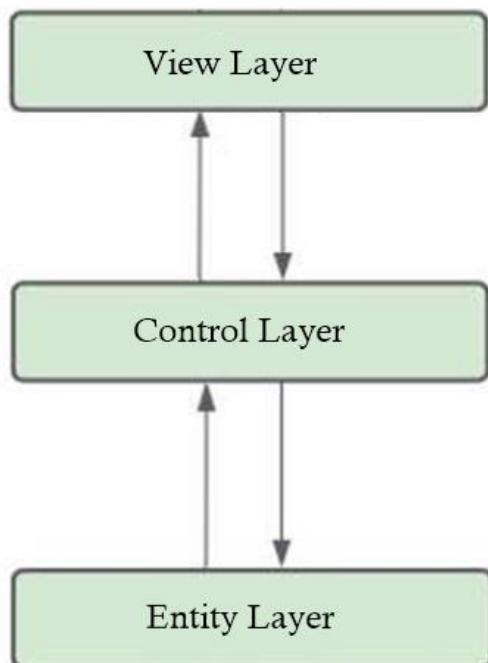


Figura 4: Frontend Three Layer Architecture

Per fare un esempio, quando l'utente interagisce con un *affordance*, dallo strato d'interfaccia grafica si propaga un'azione nel Controller che lo propaga ad una classe di livello inferiore; quando la computazione è terminata (ad esempio si è riusciti ad ottenere una response dal Backend), si ripete il procedimento in modo inverso.

La scelta di utilizzare il pattern Three-Layer deriva dalla sua semplicità, flessibilità nelle modifiche al variare dei requisiti e soprattutto dalla modularità, che ci ha consentito di lavorare contemporaneamente su moduli differenti senza intaccare sul lavoro complessivo, riuscendo ad accelerare il processo di sviluppo software.

Ecco una descrizione in dettaglio dei livelli architetturali:

[View Layer](#)

Rappresenta l'interfaccia utente, è responsabile della rappresentazione dei dati che riceve dal livello dal Control Layer e cattura ogni richiesta propagandola al livello inferiore; per facilitare la costruzione della GUI, oltre alle classi di interfaccia utente sono state create classi custom per rendere accattivante e facilitare la disposizione dei dati (chiamate *Adapter*).

[Control Layer](#)

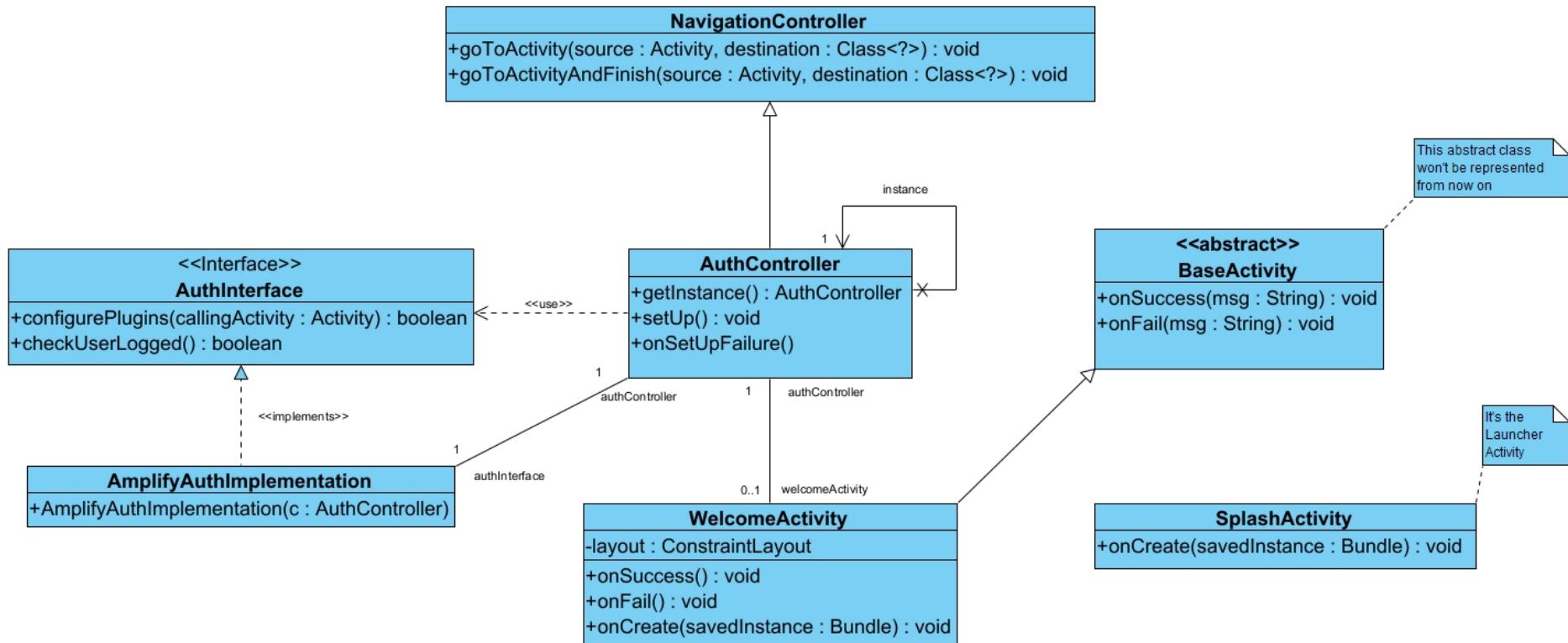
È il livello che permette la comunicazione tra le views e i model, nonché il livello che permette di interpretare le richieste degli utenti coordinando i cambiamenti di views e gli update dell'interfaccia grafica in base agli eventi.

[Entity Layer](#)

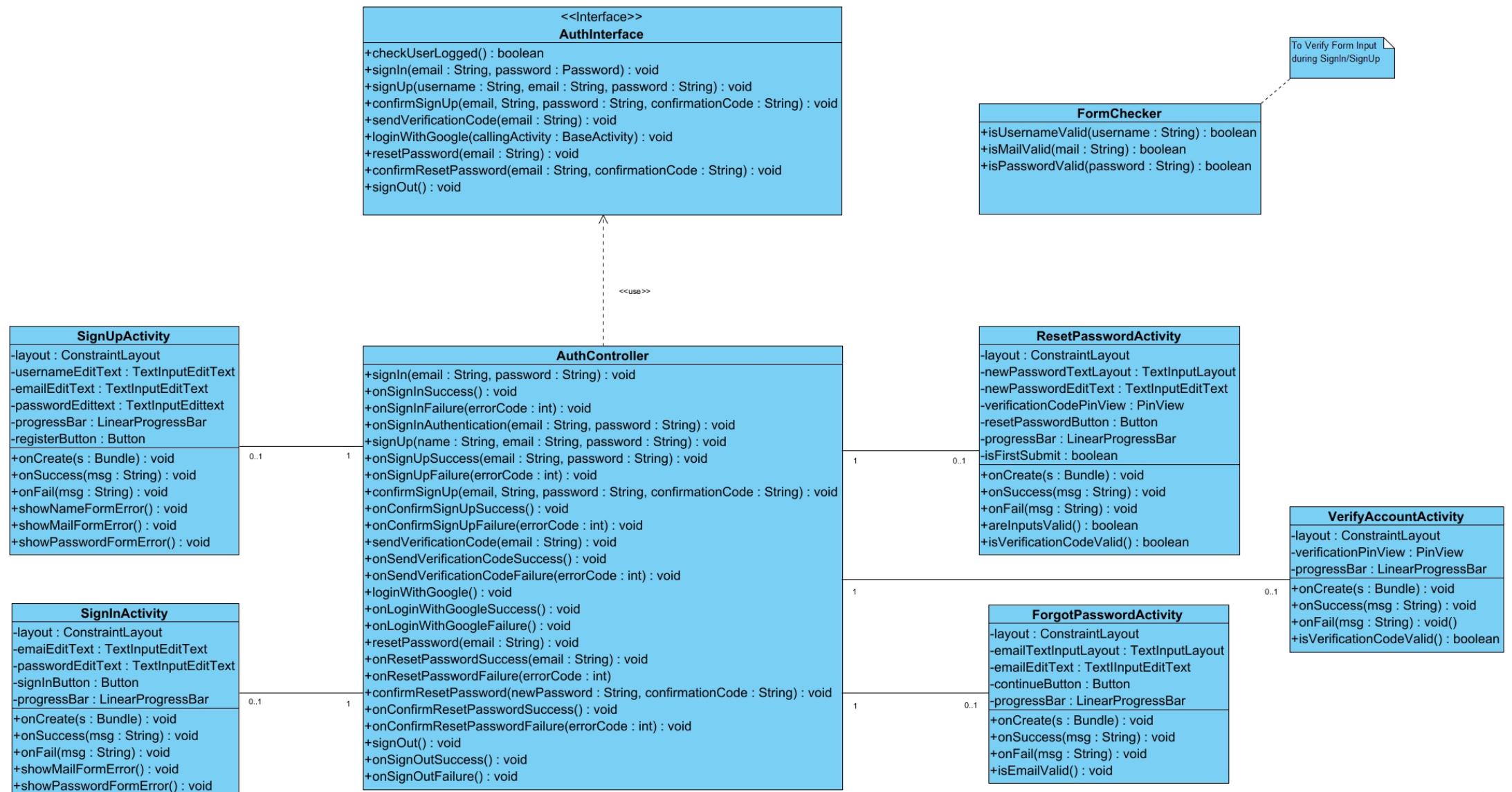
Si occupa della definizione delle classi di dominio e gestisce la persistenza dei dati: infatti a questo livello si è utilizzato il pattern *Data-Access-Object* (DAO), in modo tale da offrire al livello di controllo un'interfaccia ad una serie di funzionalità per la gestione di dati e la manipolazione degli oggetti aumentando la manutenibilità con la divisione del livello in due sub-layer.

7. Design del Sistema

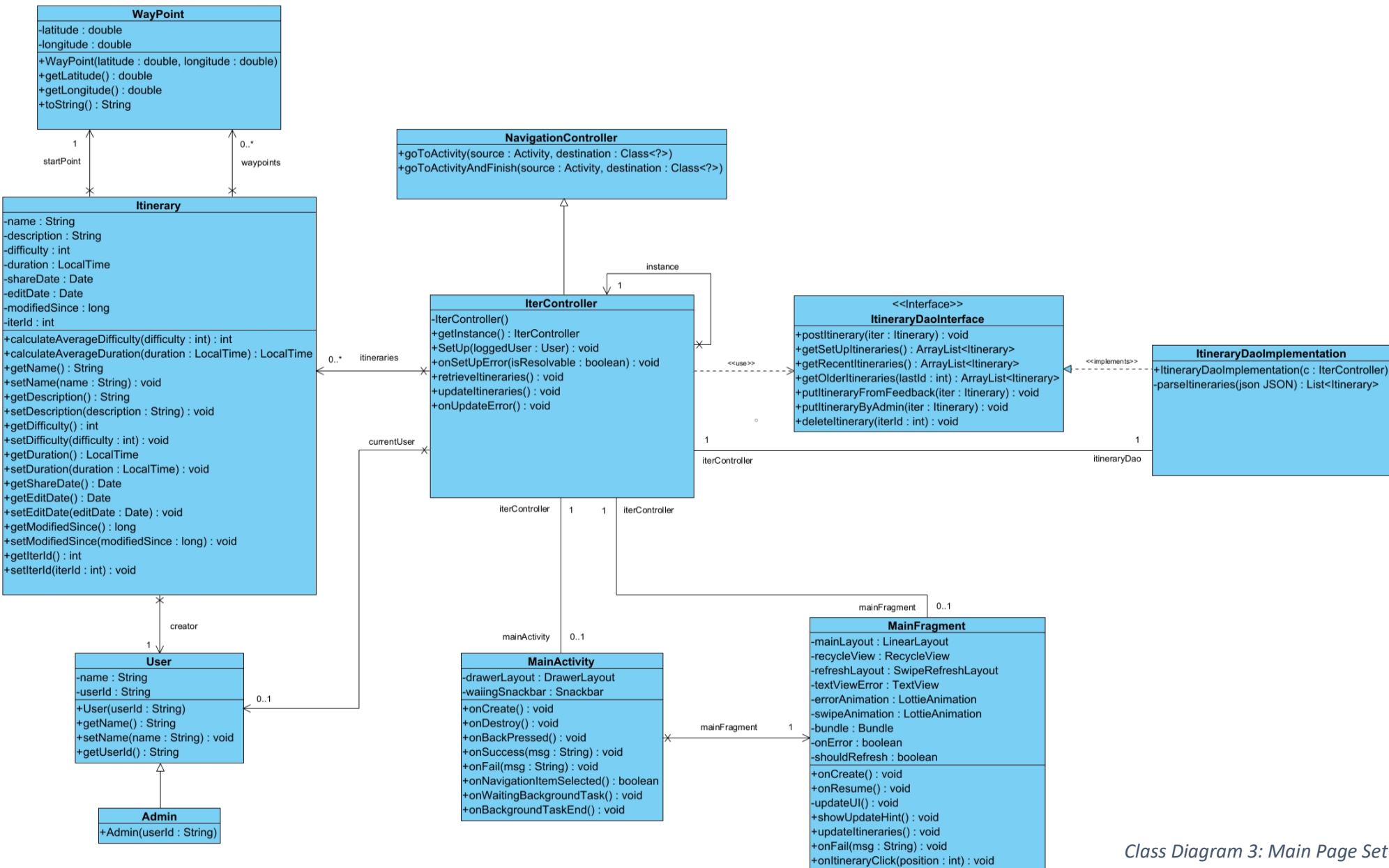
7.1 Class Diagrams di Design



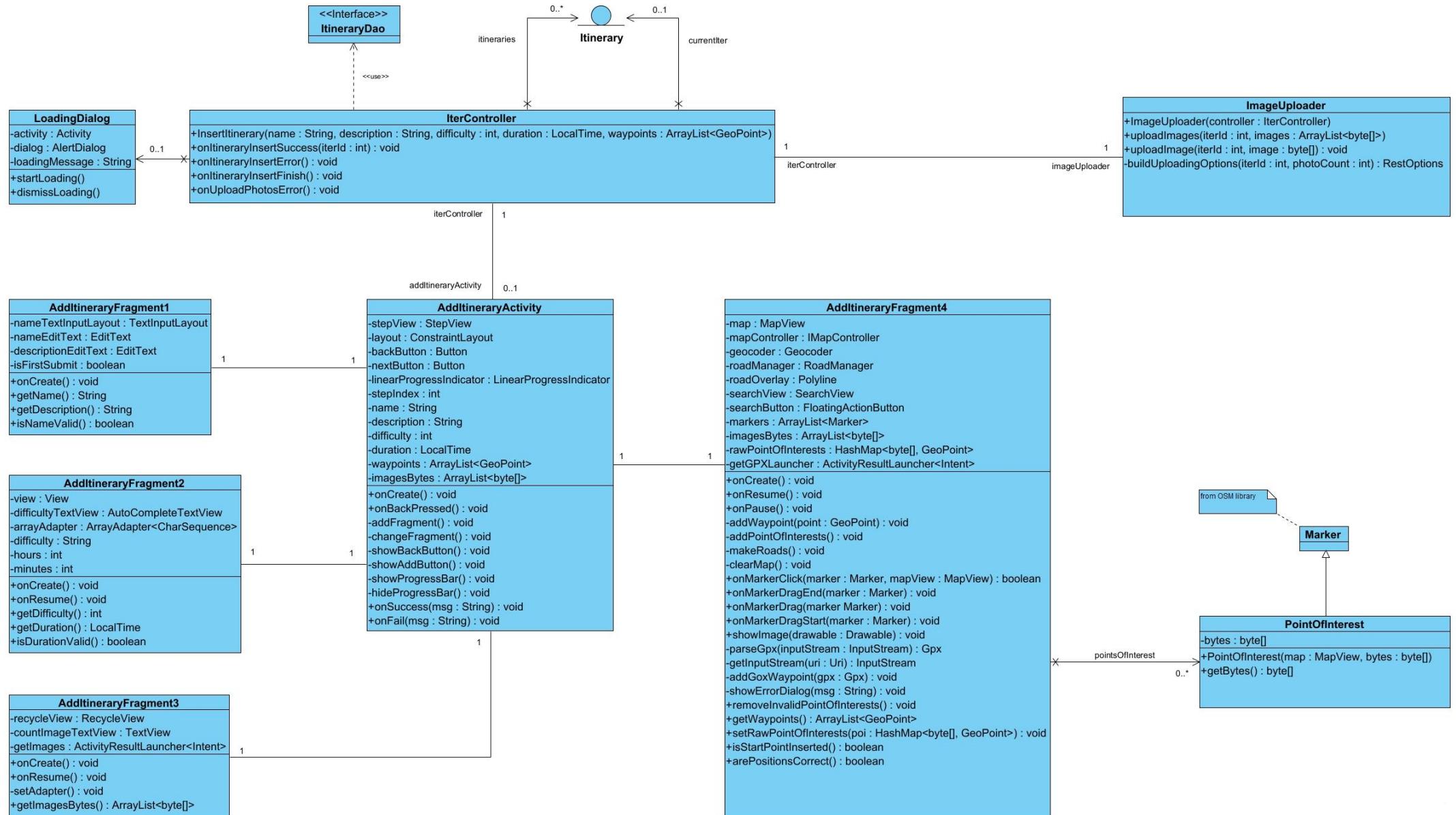
Class Diagram 1: Authentication Set Up



Class Diagram 2: Authentication Services



Class Diagram 3: Main Page Set Up

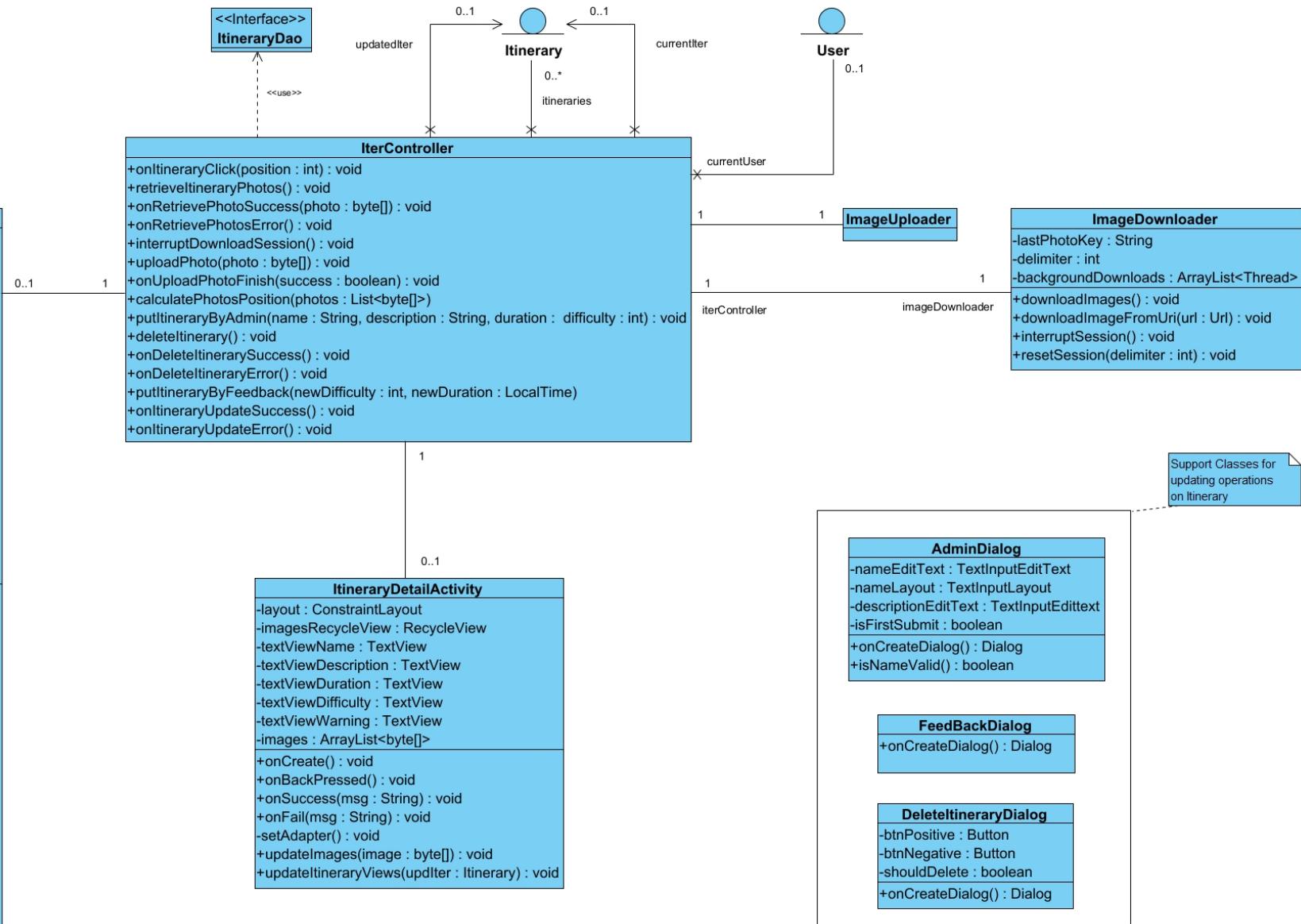


Class Diagram 4: Add Itinerary

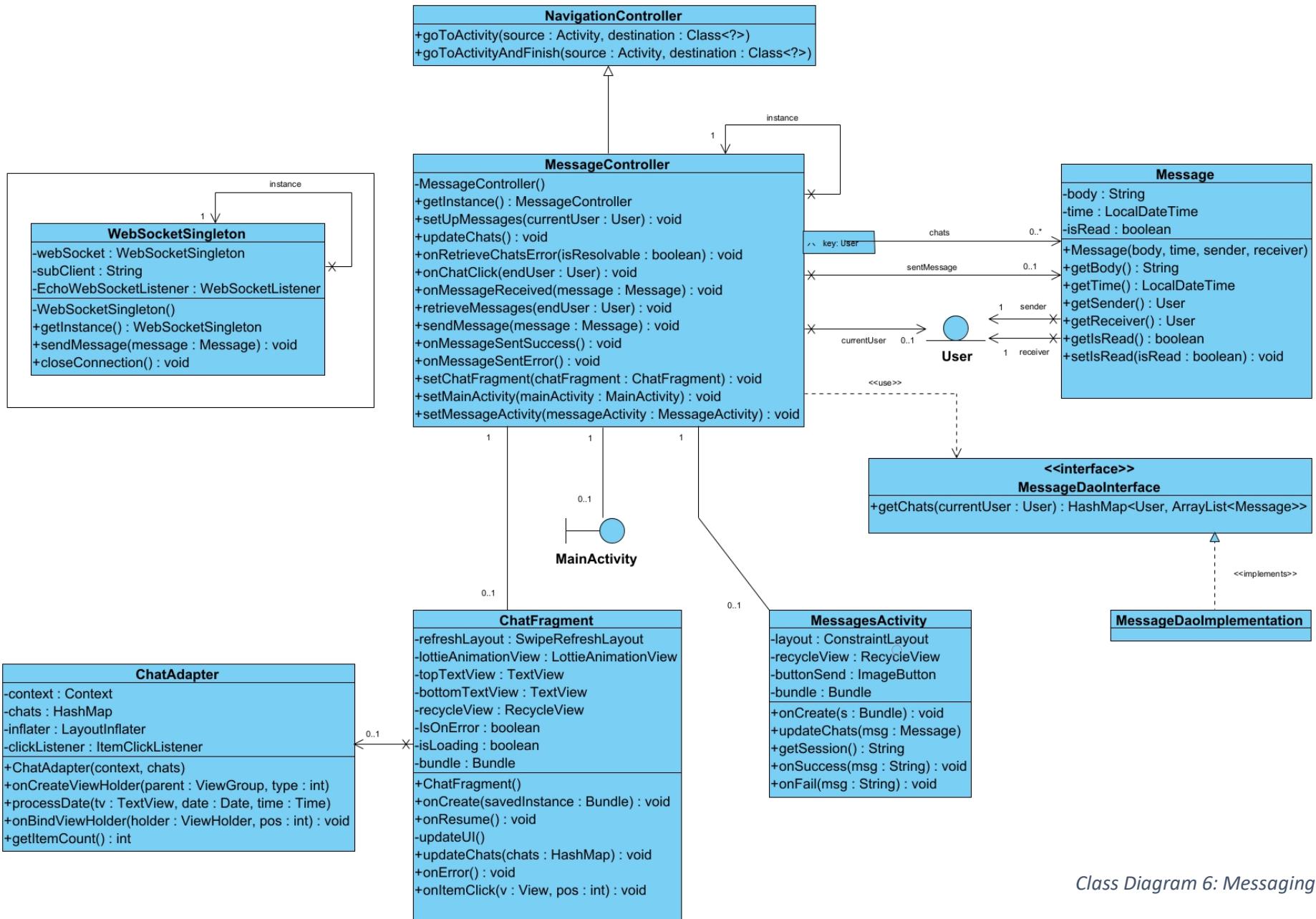
```

FollowItineraryActivity
-map : MapView
-roadManager : RoadManager
-itineraryRoadOverlay : Polyline
-myRoadOverlay : Polyline
-myLocationNewOverlay : MyLocationNewOverlay
-uploadingPhoto : HashMap<byte[], GeoPoint>
-itineraryWaypoints : ArrayList<GeoPoint>
-itineraryIndications : ArrayList<Marker>
-myRoadIndications : ArrayList<Marker>
-locationManager : LocationManager
-progressBar : ProgressBar
-cardView : CardView
-mainLayout : ConstraintLayout
-directionsLayout : LinearLayout
-toStartLayout : LinearLayout
-directionsSwitchMaterial : SwitchMaterial
-toStartSwitchMaterial : SwitchMaterial
-getImage : ActivityResultLauncher<Intent>
-requestPermissions : ActivityResultLauncher<String>
+onCreate() : void
+onResume() : void
+onPause() : void
-setUpMap() : void
-addMarker(point : GeoPoint, isFirst : boolean) : void
-addPointOfInterests(images : HashMap<byte[], GeoPoint>)
+onMarkerClick(marker : Marker, map : MapView) : boolean
-makeRoads() : void
-makeIndicationsToStartPoint() : void
+makeDirections(road : Road, indications : ArrayList<Marker>)
-checkSettingsAmdStartLocationUpdate() : void
-showAlertGpsPermissionNeeded() : void
-askLocationPermission() : void
-buildAlertMessageNoGps() : void
+arePositionsCorrect(pos : GeoPoint) : boolean
+onPhotoUploadFinish(success : boolean) : void
+onSuccess(msg : String) : void
+onFail(msg : String) : void

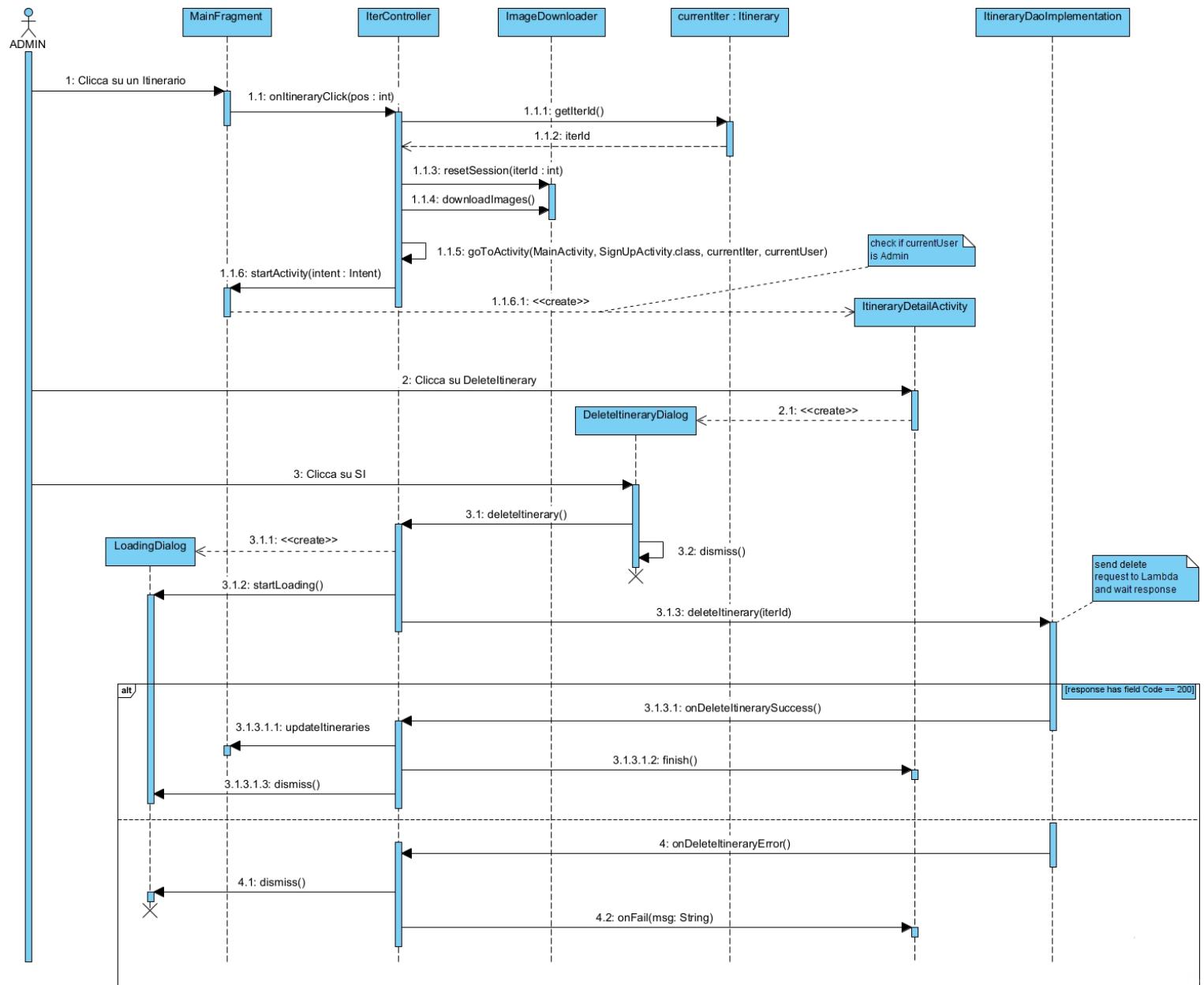
```



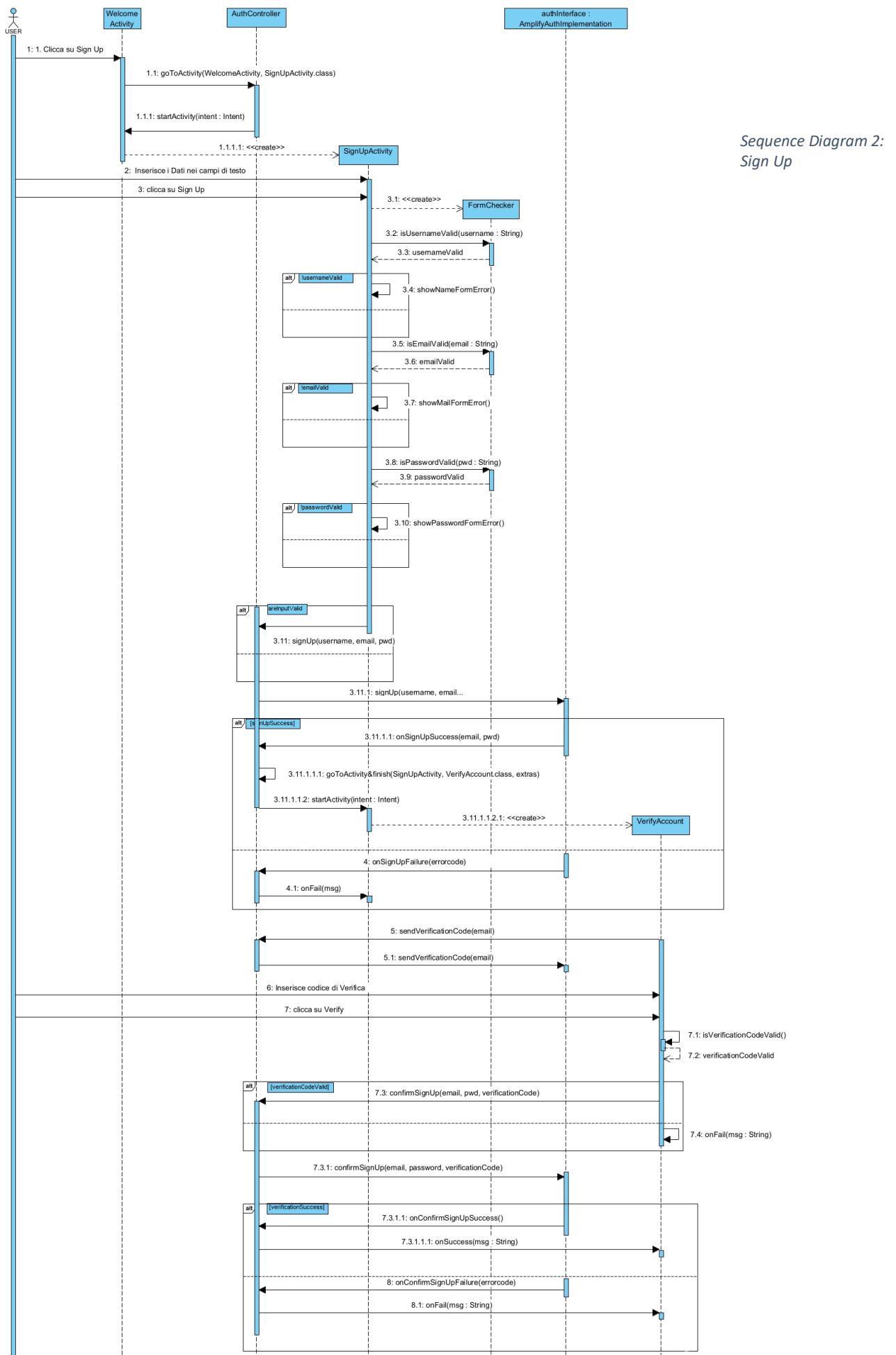
Class Diagram 5: Itinerary Operations



7.2 Sequence Diagrams di Design



Sequence Diagram 1: Admin deletes Itinerary



7.3 Gerarchie Funzionali

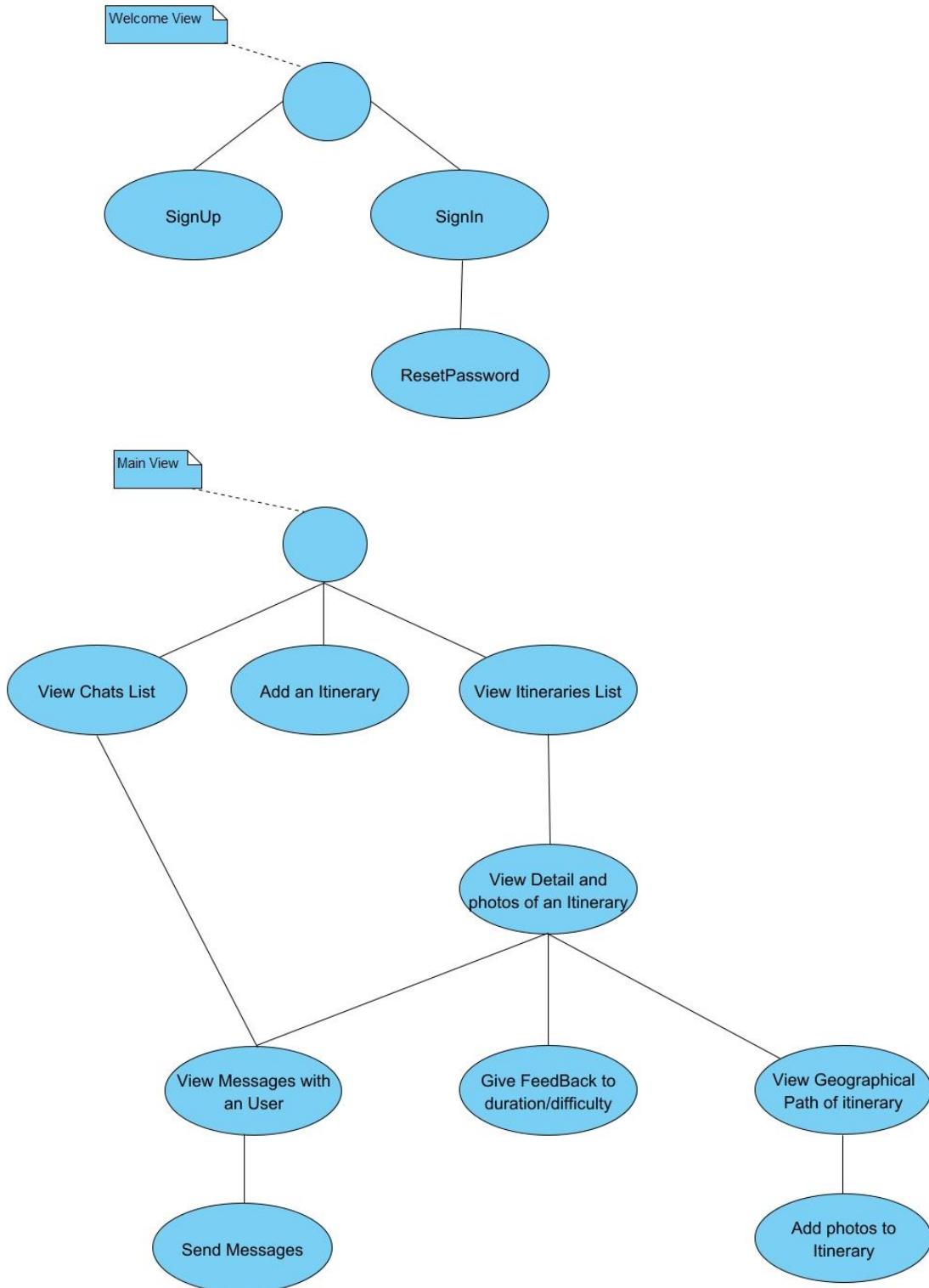


Figura 5: Gerarchia delle funzioni del software

PARTE III: Testing e Valutazioni finali

8. Unit Testing

8.1 Password Policy

Attraverso il metodo `isPasswordValid` è possibile controllare se, in fase di registrazione di un Utente, la password rispetta la policy scelta per migliorare la sicurezza di accesso ad un account.

```
17
18 @
19     public boolean isPasswordValid(String password) {
20
21         if(password.length() < 8 || password.length() > 20)
22             return false;
23
24         if(password.matches( regex: ".*\\s+.*"))
25             return false;
26
27         return password.matches( regex: ".*(?:.*[0-9])(?:.*[a-z])(?:.*[A-Z]).*" );
28
29 }
```

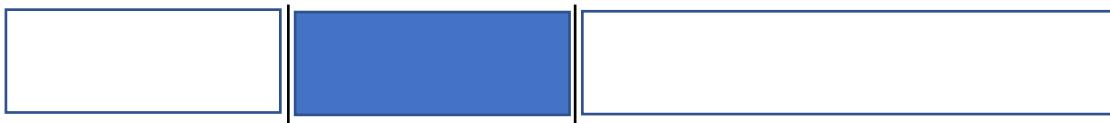
Funzione 1: isPasswordValid

Il metodo ha un solo parametro di input, la stringa da controllare (ovvero la password) e come output un booleano che indica se la stringa è o meno valida per essere usata come password. I vincoli che deve rispettare la stringa sono i seguenti:

- 1) Deve avere lunghezza compresa tra otto e venti (caratteri); {line 20}
- 2) Non deve contenere spazi (per semplificare da ora in poi chiameremo una stringa che rispetta questo vincolo ‘parola’); {line 23}
- 3) Deve contenere almeno un carattere numerico ([0-9]), un carattere minuscolo ([a-z]) e un carattere maiuscolo ([A-Z]); {line 26}

Attraverso la strategia **Black-Box** riportiamo di seguito le classi di equivalenza individuate dai vincoli di policy della password;

0 8 20



Lunghezza (x)	Classe di Equivalenza
$x > 20$	Password non valida
$x < 8$	Password non valida
$8 \leq x \leq 20$	Password valida

Tabella 1: Classi di equivalenza per la lunghezza

- 1) In relazione alla lunghezza della password abbiamo individuato sei casi di test utilizzando la tecnica del testing dei valori limite (valori medi per la classe Non Valida esclusi);

```
17  @Test
18  public void checkPasswordLengthZero() {
19      assertFalse(checker.isValid(""));
20  }
21
22  @Test
23  public void checkPasswordLengthLessThanMinimum() {
24      assertFalse(checker.isValid("LE4GTh7"));
25  }
26
27  @Test
28  public void checkPasswordLengthEight() {
29      assertTrue(checker.isValid("Ad123f-8"));
30  }
31
32  @Test
33  public void checkPasswordLengthMiddle() {
34      assertTrue(checker.isValid("Ad12345fPi_w14"));
35  }
36
37  @Test
38  public void checkPasswordLengthTwenty() {
39      assertTrue(checker.isValid("Ad12345fksJ-du469dhJ"));
40  }
41
42  @Test
43  public void checkPasswordLengthMoreThanMaximum() {
44      assertFalse(checker.isValid("STRINGS_lengthA231.21"));
45  }
```

parole

*stringhe che contengono
almeno uno spazio*



- 2) Le classi di equivalenza individuate sono due, così come il numero di casi di test utili (abbiamo aggiunto anche un caso con arbitrario numero di spazi)

Numero di spazi (y)	Classe di Equivalenza
$y = 0$	Password valida
$y \geq 1$	Password non valida

Tabella 2: Classi di equivalenza per la presenza di spazi

A screenshot of a Java code editor showing three test cases for password validation:

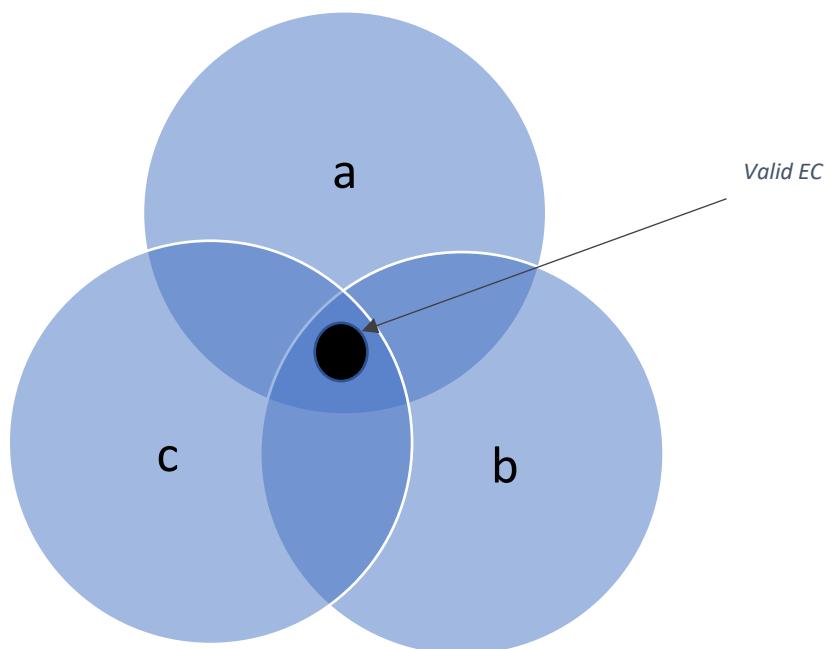
```
48     /**
49      * SPACES
50      * */
51
52     @Test
53     public void checkPasswordWithoutSpaces() {
54         assertTrue(checker.isPasswordValid("MyPassword123"));
55     }
56
57     @Test
58     public void checkPasswordWithOneSpace() {
59         assertFalse(checker.isPasswordValid("String space123"));
60     }
61
62     @Test
63     public void checkPasswordWithMultipleSpaces() {
64         assertFalse(checker.isPasswordValid(" String - space123 "));
65     }
66 }
```

The code editor highlights specific parts of the code with colored bars: the multi-line comment at the top is green, the first test case's string is green, the second test case's string is brown, and the third test case's string is green. Line numbers are visible on the left, and code navigation icons are shown on the right.

3) Infine, l'ultimo vincolo è stato quello più ostico; potremmo dividere il vincolo in tre sotto vincoli ognuno rispettivo ai caratteri che devono essere contenuti nella stringa;

Numero caratteri numerici (a)	Numero caratteri maiuscoli (b)	Numero caratteri minuscoli (c)	Classe di Equivalenza
$a = 0$	\forall	\forall	Non Valida
\forall	$b = 0$	\forall	Non Valida
\forall	\forall	$c = 0$	Non Valida
$a \geq 1$	$b \geq 1$	$c \geq 1$	Valida

Tabella 3: Classi di equivalenza per la presenza di caratteri



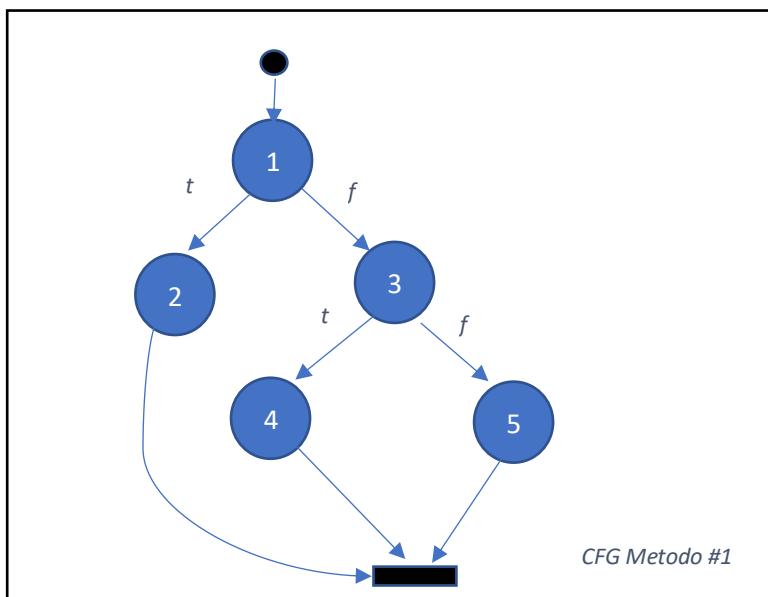
```

71
72     @Test
73     public void checkPasswordWithoutNumber() {
74         assertFalse(checker.isPasswordValid("NO-number-here"));
75     }
76
77     @Test
78     public void checkPasswordWithoutUpperCase() {
79         assertFalse(checker.isPasswordValid("no-caps-her3"));
80     }
81
82     @Test
83     public void checkPasswordWithoutLowerCase() {
84         assertFalse(checker.isPasswordValid("NO-LOWER.HER3"));
85     }
86
87
88     @Test
89     public void checkPasswordOneNumeric() { assertTrue(checker.isPasswordValid("1-NumberHere")); }
90
91
92     @Test
93     public void checkPasswordOneUpper() { assertTrue(checker.isPasswordValid("One-upper-h3r3")); }
94
95
96     @Test
97     public void checkPasswordOneLower() { assertTrue(checker.isPasswordValid("oNE-LOWER-H3R3")); }
98
99
100
101
102

```

Si noti che, affinché il risultato dell'esecuzione del metodo `isPasswordValid` sia `True`, la stringa deve rispettare contemporaneamente tutti i vincoli; per questo nei casi di test al momento della verifica di un vincolo, abbiamo preso in considerazione la validità degli input rispetto agli altri vincoli.

Per quanto riguarda il testing **White-Box** riportiamo qui sotto il Control Flow Graph del metodo; Abbiamo indicato il nodo iniziale con un pallino nero e quello finale con un rettangolo.



Partiamo dallo *statement coverage*. Affinché siano testate tutte le istruzioni, occorrono tre test:

```
9
10 ➤ public class PasswordTesterWhiteBox {
11
12     private FormChecker checker;
13
14     @Before
15     public void setUp() {
16         checker = new FormChecker();
17     }
18
19
20     @Test
21 ➤     public void TestStatementOne() {
22         assertFalse(checker.isPasswordValid("Pass1"));
23     }
24
25     @Test
26 ➤     public void TestStatementTwo() {
27         assertFalse(checker.isPasswordValid(" Password1"));
28     }
29
30     @Test
31 ➤     public void TestStatementThree() {
32         assertFalse(checker.isPasswordValid("passwordNotValid"));
33     }
34
```

- testStatementOne ha una coverage del 40%
- testStatementTwo ha una coverage del 60%
- testStatementThree ha una coverage del 60%

Per quanto riguarda il *branch coverage* non sono necessari altri test cases; i percorsi possibili dal nodo iniziale a quello finale sono tre; quindi, occorreranno tre test ognuno relativo a ciascun percorso;

8.2 Durata Media

Il secondo metodo di cui abbiamo fornito testing è `calculateAverageDuration` fornito come servizio del model `Itinerary`, che ha lo scopo di calcolare una durata media tra la durata in input e quella locale dell'entità (con tipo LocalTime); il metodo restituisce un LocalTime che rappresenta la durata media.

```
144 @ ... public LocalTime calculateAverageDuration(LocalTime duration) {  
145  
146     int newHours = duration.getHourOfDay();  
147     int newMinutes = duration.getMinuteOfHour();  
148  
149     if(newHours == 0 && newMinutes == 0)  
150         throw new IllegalArgumentException();  
151  
152     int newAccumulatedMinutes = newMinutes + (newHours * 60);  
153     int currAccumulatedMinutes = this.duration.getMinuteOfHour() + (this.duration.getHourOfDay() * 60);  
154  
155     int averageMinutes = (newAccumulatedMinutes + currAccumulatedMinutes) / 2;  
156     int averageHours = averageMinutes / 60;  
157     averageMinutes -= (60 * averageHours);  
158     return new LocalTime(averageHours, averageMinutes);  
159  
160 }  
161
```

Affinché la computazione termini correttamente, la durata in input deve essere diversa da zero (proprio perché ci aspettiamo che il percorso abbia una durata ben definita), inoltre la classe LocalTime (del package org.joda.time) non permette la costruzione di tempi per cui ora o minuti non siano nel range giusto ([0-23] e [0-59]);

Un primo sguardo riguardo la validità degli input e i casi di test per la verifica dell'input:

Durata in minuti accumulati (x)	CE
x = 0	Input non valido
x > 0	Input valido

Tabella 4: Classi di equivalenza per la verifica della durata in input

```

17     private Itinerary itinerary;
18
19     @Before
20     public void setUp() {
21         itinerary = new Itinerary(
22             name: "name", difficulty: 1, new LocalTime( i: 2, i1: 30), new WayPoint( latitude: 40.863, longitude: 14.2767),
23             );
24     }
25 }
26
27 /**
28 * INPUT VALIDATION
29 */
30 */
31
32 @Test /* (expected = IllegalArgumentException.class) */
33 public void testWithNewDurationEqualsZero() {
34     LocalTime newDuration = new LocalTime( i: 0, i1: 0);
35     assertThrows(IllegalArgumentException.class, ()-> itinerary.calculateAverageDuration(newDuration));
36 }
37
38 @Test
39 public void testAverageWithSameDuration() {
40     LocalTime currentDuration = itinerary.getDuration();
41     assertEquals(currentDuration, itinerary.calculateAverageDuration(currentDuration));
42 }

```

Essendo l'operazione principale del metodo una media, il numero di casi di test sarebbe potenzialmente grande, anche in questo caso in cui il range è limitato ($24 \times 60 \times 2 = 2880$ casi di test complessivi utilizzando SECT), per cui abbiamo considerato alcuni casi particolari, fissando la durata del percorso ad un valore fisso (2 ore e 30 minuti) e calcolando a priori l'oracolo;

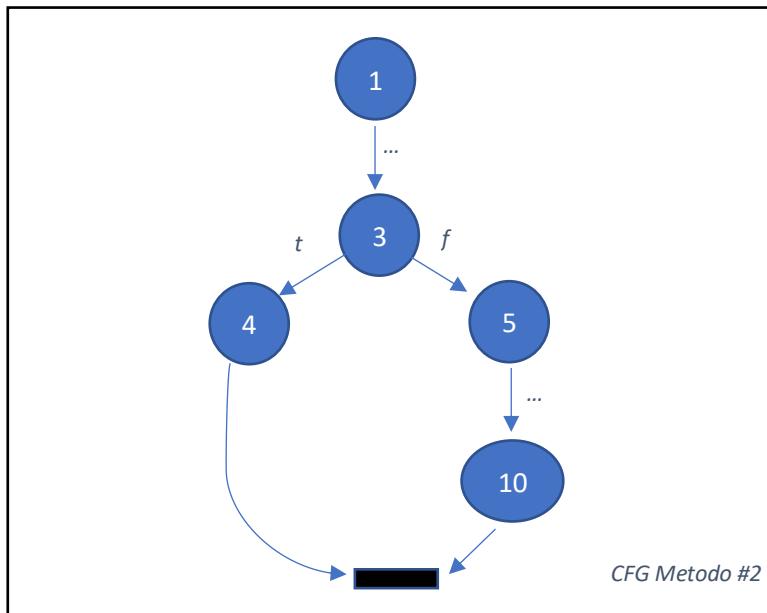
TEST CASES:

Sia y la durata fissa dell'itinerario.

- Durata minore di y
- Durata maggiore di y
- Minima durata ammessa
- Massima Durata ammessa
- Durata immediatamente precedente a y
- Durata immediatamente successiva a y

```
48
49     @Test
50     public void testAverageWithMinDuration() {
51         LocalTime newDuration = new LocalTime( 0, 15);
52         assertEquals(new LocalTime( 1, 15), itinerary.calculateAverageDuration(newDuration));
53     }
54
55
56     @Test
57     public void testAverageWithMaxDuration() {
58         LocalTime newDuration = new LocalTime( 23, 59);
59         assertEquals(new LocalTime( 13, 14), itinerary.calculateAverageDuration(newDuration));
60     }
61
62
63     @Test
64     public void testAverageWithDurationLonger() {
65         LocalTime newDuration = new LocalTime( 3, 0);
66         assertEquals(new LocalTime( 2, 45), itinerary.calculateAverageDuration(newDuration));
67     }
68
69
70     @Test
71     public void testAverageWithDurationShorter() {
72         LocalTime newDuration = new LocalTime( 2, 0);
73         assertEquals(new LocalTime( 2, 15), itinerary.calculateAverageDuration(newDuration));
74     }
75
76     @Test
77     public void testAverageWithDurationMinusOne() {
78         LocalTime newDuration = itinerary.getDuration().minusMinutes(1);
79         assertEquals(newDuration, itinerary.calculateAverageDuration(newDuration));
80     }
81
82     @Test
83     public void testAverageWithDurationPlusOne() {
84         LocalTime newDuration = itinerary.getDuration().plusMinutes(1);
85         assertEquals(itinerary.getDuration(), itinerary.calculateAverageDuration(newDuration));
86     }
87
88
```

Parlando di **White-Box**, la massima copertura della funzione ottenibile da un caso di test è del 90 %, poiché in caso in cui l'input è valido, verranno eseguite tutte le linee di codice tranne quella in cui viene lanciata l'eccezione. Parliamo invece di una coverage più bassa, 40 %, nel caso in cui la durata in input sia zero.



Sono dunque necessari due casi di test:

```

25
26     @Test
27     public void testStatementOne() {
28         LocalTime newDuration = new LocalTime(0, 0);
29         assertThrows(IllegalArgumentException.class, ()-> itinerary.calculateAverageDuration(newDuration));
30     }
31
32
33     @Test
34     public void testStatementTwo() {
35         LocalTime newDuration = new LocalTime(2, 50);
36         assertEquals(new LocalTime(2, 40), itinerary.calculateAverageDuration(newDuration));
37     }
38 }
```

- `testStatementOne` ha una copertura del 40% di *statement coverage* e del 40% di *branch coverage*;
- `testStatementTwo` ha una copertura del 90% di *statement coverage* e del 60% di *branch coverage*;

8.3 Parsing degli itinerari

Il terzo metodo di cui abbiamo fornito testing è `parseItineraries` fornito come servizio del model *Itinerary*, che ha lo scopo di tradurre il response della nostra API (un JSON) in una lista di oggetti concreti localmente; Il metodo accetta in input un JSON (con tipo JSONArray) contenente una lista di n itinerari e restituisce un ArrayList di Itinerary. Per sottoporre a testing il metodo abbiamo creato la classe mock `ParseItinerariesMock`, la quale espone pubblicamente il metodo.

```
18  public class ParseItinerariesMock {
19
20     @SuppressLint("NewApi")
21     @Override
22     public ArrayList<Itinerary> parseItineraries(JSONArray result) throws JSONException {
23
24         ArrayList<Itinerary> iter = new ArrayList<>();
25
26         for(int i = 0; i < result.length(); ++i) {
27
28             JSONObject jsonObject = result.getJSONObject(i);
29
30             int id = jsonObject.getInt("iterid");
31
32             String name = jsonObject.getString("itername");
33
34             String description = null;
35             if(!jsonObject.isNull("description"))
36                 description = jsonObject.getString("description");
37
38             int difficulty = jsonObject.getInt("difficulty");
39
40             LocalTime duration = new LocalTime(jsonObject.getInt("hours"), jsonObject.getInt("minutes"));
41
42             String creatorID = jsonObject.getString("creatorid");
43             User creator = new User(creatorID);
44
45             creator.setName(jsonObject.getString("creatormame"));
46
47             Date shareDate = Date.from(Instant.from(DateTimeFormatter.ISO_DATE_TIME.parse(jsonObject.getString("sharedate"))));
48
49             Date editDate = null;
50             if(!jsonObject.isNull("updatedate"))
51
52                 long modifiedSince = jsonObject.getLong("modifiedsince");
53
54             Itinerary iter = new Itinerary(name, difficulty, duration, startPoint, creator, shareDate);
55
56             iter.setIterId(id);
57             iter.setWayPoints(iterWaypoints);
58             iter.setDescription(description);
59             iter.setEditDate(editDate);
60             iter.setModifiedSince(modifiedSince);
61
62             iterWaypoints.add(iter);
63
64         }
65
66         return iter;
67     }
68
69 }
70
71 }
```

Affinché la computazione termini correttamente, l'input deve essere un JSON vuoto oppure deve essere un JSON contenente una lista di Itinerari ben formati ovvero per ogni itinerario vengono esplicitati almeno e al più un campo:

- iterid: int
- itername: String
- description: String *[opzionale]*
- difficulty: int
- hours: int
- minutes: int
- creatorid: String
- creatorname: String
- sharedate: Date (java.util.Date)
- updatedate: Date (java.util.Date) *[opzionale]*
- startpoint: JSONObject (coppia di double)
- waypoints: JSONArray (lista di coppie di double) *[opzionale]*
- modifiedsince: long

per un totale di 13 campi, i campi dovranno essere esplicitati usando una rappresentazione che ne permetta la conversione nel tipo sopra indicato, inoltre se presenti ulteriori campi da quelli sopra elencati verranno semplicemente ignorati

# campi	Nominazione campi	Rappresentazione campi	CE
13	Corretta	Corretta	Valida
13	Corretta	Non corretta	Non valida
13	Non corretta	À	Non valida
> 13	Corretta	Corretta	Valida
> 13	Corretta	Non corretta	Non valida
> 13	Non corretta	À	Non valida
$0 < x < 13$	À	À	Non valida
0	/	/	Valida

Tabella 5: Classi di equivalenza per la verifica del JSON in input

```

19
20 public class ParseItinerariesTester {
21
22     ParseItinerariesMock parseItinerariesMock;
23
24     @Before
25     public void setUp() { parseItinerariesMock = new ParseItinerariesMock(); }
26
27
28     @Test
29     public void parseItineraryWithCorrectInput() throws JSONException{
30
31         String itineraryJson = "[{\\"iterid\\":5,\\"itername\\":\\"Roma\\",\\"description\\":null,\\"difficulty\\":0,\\"hours\\":1,\\"minutes\\":0," +
32             "\\"startpoint\\":{\\"x\\":41.91215825489158,\\"y\\":12.492356197611912},"
33             "\\"waypoints\\":[\{\\"Latitude\\":\"41.90795968255628\",\"Longitude\\\":\"12.493655406267607\"}],"
34             "\\"creatorid\\\":\"7bba5c72-7fbe-45ad-996a-686c8685a9b8\",\"sharedate\\\":\"2022-02-15T00:00:00.000Z\","
35             "\\"updatedate\\":null,\\"modifiedsince\\\":\"1644923584176\",\"creatorname\\\":\"User1\"}]";
36
37
38         Itinerary itinerary = new Itinerary(
39             name: "Roma",
40             difficulty: 0,
41             new LocalTime( i: 1, ii: 0),
42             new WayPoint( latitude: 41.91215825489158, longitude: 12.492356197611912),
43             new User( uid: "7bba5c72-7fbe-45ad-996a-686c8685a9b8"),
44             Date.from(Instant.from(DateTimeFormatter.ISO_DATE_TIME.parse("2022-02-15T00:00:00.000Z")));
45
46
47         itinerary.setIterId(5);
48         ArrayList<WayPoint> wayPoints = new ArrayList<>();
49         wayPoints.add(new WayPoint( latitude: 41.90795968255628, longitude: 12.493655406267607));
50         itinerary.setWayPoints(wayPoints);
51         itinerary.setModifiedSince(1644923584176L);
52         itinerary.getCreator().setName("User1");
53
54         ArrayList<Itinerary> itineraries = new ArrayList<>();
55         itineraries.add(itinerary);
56
57         ArrayList<Itinerary> result = parseItinerariesMock.parseItineraries(new JSONArray(itineraryJson));
58         assertEquals(itineraries, result);
59     }
60
61
62     @Test
63     public void parseItineraryWithWrongFieldRepresentation() {
64
65         String itineraryJson = "[{\\"iterid\\":5,\\"itername\\":\\"Roma\\",\\"description\\":null,\\"difficulty\\":0,\\"hours\\":1,\\"minutes\\":0," +
66             "\\"startpoint\\":{\\"x\\":41.91215825489158,\\"y\\":12.492356197611912},"
67             "\\"waypoints\\":[\{\\"Latitude\\":\"41.90795968255628\",\"Longitude\\\":\"12.493655406267607\"}],"
68             "\\"creatorid\\\":\"7bba5c72-7fbe-45ad-996a-686c8685a9b8\",\"sharedate\\\":\"2022A-02-15T00:00:00.000Z\","
69             "\\"updatedate\\":null,\\"modifiedsince\\\":\"1644923584176\",\"creatorname\\\":\"User1\"}]";
70
71         assertThrows(Exception.class, () -> parseItinerariesMock.parseItineraries(new JSONArray(itineraryJson)));
72     }
73
74     @Test
75     public void parseItineraryWithWrongNomination() {
76
77         String itineraryJson = "[{\\"iteridd\\":5,\\"itername\\":\\"Roma\\",\\"description\\":null,\\"difficulty\\":0,\\"hours\\":1,\\"minutes\\":0," +
78             "\\"startpoint\\":{\\"x\\":41.91215825489158,\\"y\\":12.492356197611912},"
79             "\\"waypoints\\":[\{\\"Latitude\\":\"41.90795968255628\",\"Longitude\\\":\"12.493655406267607\"}],"
80             "\\"creatorid\\\":\"7bba5c72-7fbe-45ad-996a-686c8685a9b8\",\"sharedate\\\":\"2022-02-15T00:00:00.000Z\","
81             "\\"updatedate\\":null,\\"modifiedsince\\\":\"1644923584176\",\"creatorname\\\":\"User1\"}]";
82
83         assertThrows(JSONException.class, () -> parseItinerariesMock.parseItineraries(new JSONArray(itineraryJson)));
84     }

```

```

85
86     @Test
87     public void parseItineraryWithMoreThan13Fields() throws JSONException{
88
89         String itineraryJson = "[{\\"iterid\":5,\\"itername\":\"Roma\",\\\"description\":null,\\\"difficulty\":0,\\\"hours\":1,\\\"minutes\":0," +
90             "\\\"startpoint\":{\"x\":41.91215825489158,\"y\":12.492356197611912},\" +
91             "\\\"waypoints\":[{\\"Latitude\":\\\"41.90795968255628\\\",\\\"Longitude\":\\\"12.493655406267607\\\"}],\" +
92             "\\\"creatorid\":\"7bba5c72-7fbe-45ad-996a-686c8685a9b8\\\",\\\"sharedate\\\":\\\"2022-02-15T00:00:00.000Z\\\",\" +
93             "\\\"updatedate\\\":null,\\\"modifiedsince\\\":\\\"1644923584176\\\",\\\"creatorname\\\":\\\"User1\\\",\\\"extraField\\\":\\\"Extra\\\"}]";
94
95         ArrayList<Itinerary> result =  parseItinerariesMock.parseItineraries(new JSONArray(itineraryJson));
96         assertFalse(result.isEmpty());
97     }
98
99     @Test
100    public void parseItineraryWithMoreThan13FieldsAndWrongRepresentation() {
101
102        String itineraryJson = "[{\\"iterid\":5,\\"itername\":\"Roma\",\\\"description\":null,\\\"difficulty\":0,\\\"hours\":1,\\\"minutes\":0," +
103            "\\\"startpoint\":{\"x\":41.91215825489158,\"y\":12.492356197611912},\" +
104            "\\\"waypoints\":[{\\"Latitude\":\\\"41.90795968255628\\\",\\\"Longitude\":\\\"12.493655406267607\\\"}],\" +
105            "\\\"creatorid\":\"7bba5c72-7fbe-45ad-996a-686c8685a9b8\\\",\\\"sharedate\\\":\\\"2022A-02-15T00:00:00.000Z\\\",\" +
106            "\\\"updatedate\\\":null,\\\"modifiedsince\\\":\\\"1644923584176\\\",\\\"creatorname\\\":\\\"User1\\\",\\\"extraField\\\":\\\"Extra\\\"}]";
107
108        assertThrows(Exception.class, () -> parseItinerariesMock.parseItineraries(new JSONArray(itineraryJson)));
109    }
110
111    @Test
112    public void parseItineraryWithMoreThan13FieldsAndWrongNomination() {
113
114        String itineraryJson = "[{\\"iteridd\\":5,\\"itername\":\"Roma\",\\\"description\":null,\\\"difficulty\":0,\\\"hours\":1,\\\"minutes\":0," +
115            "\\\"startpoint\":{\"x\":41.91215825489158,\"y\":12.492356197611912},\" +
116            "\\\"waypoints\":[{\\"Latitude\":\\\"41.90795968255628\\\",\\\"Longitude\":\\\"12.493655406267607\\\"}],\" +
117            "\\\"creatorid\":\"7bba5c72-7fbe-45ad-996a-686c8685a9b8\\\",\\\"sharedate\\\":\\\"2022-02-15T00:00:00.000Z\\\",\" +
118            "\\\"updatedate\\\":null,\\\"modifiedsince\\\":\\\"1644923584176\\\",\\\"creatorname\\\":\\\"User1\\\",\\\"extraField\\\":\\\"Extra\\\"}]";
119
120        assertThrows(JSONException.class, () -> parseItinerariesMock.parseItineraries(new JSONArray(itineraryJson)));
121    }
122
123    @Test
124    public void parseItineraryWithLessThan13Fields() {
125
126        String itineraryJson = "[{\\"iterid\":5,\\"itername\":\"Roma\",\\\"description\":null,\\\"difficulty\":0,\\\"hours\":1,\\\"minutes\":0," +
127            "\\\"startpoint\":{\"x\":41.91215825489158,\"y\":12.492356197611912},\" +
128            "\\\"waypoints\":[{\\"Latitude\":\\\"41.90795968255628\\\",\\\"Longitude\":\\\"12.493655406267607\\\"}],\" +
129            "\\\"creatorid\":\"7bba5c72-7fbe-45ad-996a-686c8685a9b8\\\",\\\"sharedate\\\":\\\"2022A-02-15T00:00:00.000Z\\\",\" +
130            "\\\"updatedate\\\":null,\\\"modifiedsince\\\":\\\"1644923584176\\\"]";
131
132        assertThrows(JSONException.class, () -> parseItinerariesMock.parseItineraries(new JSONArray(itineraryJson)));
133    }
134
135    @Test
136    public void parseItineraryWithEmptyJson() throws JSONException{
137
138        String itineraryJson = "[]";
139
140        ArrayList<Itinerary> result =  parseItinerariesMock.parseItineraries(new JSONArray(itineraryJson));
141        assertTrue(result.isEmpty());
142    }

```

Per quanto riguarda il testing **White-Box** non forniremo il GFC in quanto risulterebbe troppo carico visivamente, ci limiteremo dunque a fornire un'analisi sulla coverage del testing che sarà del:

- Statement Coverage:
 - 100% nel caso di input corretto e con campi “description”, “updatedate” e “waypoints” presenti;
- Branch Coverage:
 - 85% nel caso di input corretto e con campi “description”, “updatedate” e “waypoints” presenti;
 - 58% nel caso di input corretto e con campi “description”, “updatedate” e “waypoints” non presenti;

Complessivamente sono necessari due casi di test per avere coverage totale;

```

30      /*
31       * Parsing of 2 itineraries with all fields set
32       * - A json with 2 itineraries is created
33       * - An arrayList with 2 itineraries equivalent to those of the json is created
34       * - The correct result of parseItineraries will be an arrayList equivalent to the previous arrayList
35     */
36   @Test
37   public void parseItineraryFullFields() throws JSONException {
38
39     String itineraryJson = "[" +
40       "{" +
41         "\riterid":5,\itername\" : "Roma\", \description\" : description,\difficulty\" : 0,\hours\" : 1,\minutes\" : 0, " +
42         "\startpoint\" : {\\"x\" : 41.91215825489158,\\"y\" : 12.492356197611912}, " +
43         "\waypoints\" : [{\\"Latitude\" : \"41.90795968255628\", \\"Longitude\" : \"12.493655406267607\"}], " +
44         "\creatorid\" : \"7bba5c72-7fbe-45ad-996a-686c8685a9b8\", \\"sharedate\" : \"2022-02-15T00:00:00.000Z\", " +
45         "\updatedate\" : \"2022-02-15T00:00:00.000Z\", \\"modifiedsince\" : \"1644923584176\", \\"creatorname\" : \"User1\" " +
46       "}, " +
47       "{" +
48         "\riterid":5,\itername\" : "Roma\", \description\" : description,\difficulty\" : 0,\hours\" : 1,\minutes\" : 0, " +
49         "\startpoint\" : {\\"x\" : 41.91215825489158,\\"y\" : 12.492356197611912}, " +
50         "\waypoints\" : [{\\"Latitude\" : \"41.90795968255628\", \\"Longitude\" : \"12.493655406267607\"}], " +
51         "\creatorid\" : \"7bba5c72-7fbe-45ad-996a-686c8685a9b8\", \\"sharedate\" : \"2022-02-15T00:00:00.000Z\", " +
52         "\updatedate\" : \"2022-02-15T00:00:00.000Z\", \\"modifiedsince\" : \"1644923584176\", \\"creatorname\" : \"User1\" " +
53       "}, " +
54     "]";
55
56     Itinerary itinerary = new Itinerary(
57       name: "Roma",
58       difficulty: 0,
59       new LocalTime( 1, 0),
60       new WayPoint( latitude: 41.91215825489158, longitude: 12.492356197611912),
61       new User( uid: "7bba5c72-7fbe-45ad-996a-686c8685a9b8"),
62       Date.from(Instant.from(DateTimeFormatter.ISO_DATE_TIME.parse( text: "2022-02-15T00:00:00.000Z"))));
63   );
64
65     itinerary.setIterId(5);
66     itinerary.setDescription("description");
67     itinerary.setEditDate(Date.from(Instant.from(DateTimeFormatter.ISO_DATE_TIME.parse( text: "2022-02-15T00:00:00.000Z"))));
68
69     ArrayList<WayPoint> wayPoints = new ArrayList<>();
70     wayPoints.add(new WayPoint( latitude: 41.90795968255628, longitude: 12.493655406267607));
71     itinerary.setWayPoints(wayPoints);
72     itinerary.setModifiedSince(1644923584176L);
73     itinerary.getCreator().setName("User1");
74
75     ArrayList<Itinerary> itineraries = new ArrayList<>();
76     itineraries.add(itinerary);
77     itineraries.add(itinerary);
78
79     ArrayList<Itinerary> result = parseItinerariesMock.parseItineraries(new JSONArray(itineraryJson));
80     assertEquals(itineraries, result);
81   }

```

```

83     /*
84      * Parsing of 2 itineraries whit all the nullable fields set to null
85      * - A json with 2 itineraries is created
86      * - An arrayList with 2 itineraries equivalent to those of the json is created
87      * - The correct result of parseItineraries will be an arrayList equivalent to the previous arrayList
88     */
89     @Test
90     public void parseItineraryWithNullFields() throws JSONException {
91
92         String itineraryJson = "[" +
93             "{" +
94                 "\"iterid\":5,\"itername\":\"Roma\",\"description\":null,\"difficulty\":0,\"hours\":1,\"minutes\":0," +
95                 "\"startpoint\":{\"x\":41.91215825489158,\"y\":12.492356197611912},\"" +
96                 "\"waypoints\":null," +
97                 "\"creatorid\":\"7bba5c72-7fbe-45ad-996a-686c8685a9b8\",\"sharedate\":\"2022-02-15T00:00:00.000Z\"," +
98                 "\"updatedate\":null,\"modifiedsince\":\"1644923584176\",\"creatorname\":\"User1\"" +
99                 "},\"" +
100                "{" +
101                    "\"iterid\":5,\"itername\":\"Roma\",\"description\":null,\"difficulty\":0,\"hours\":1,\"minutes\":0," +
102                    "\"startpoint\":{\"x\":41.91215825489158,\"y\":12.492356197611912},\"" +
103                    "\"waypoints\":null," +
104                    "\"creatorid\":\"7bba5c72-7fbe-45ad-996a-686c8685a9b8\",\"sharedate\":\"2022-02-15T00:00:00.000Z\"," +
105                    "\"updatedate\":null,\"modifiedsince\":\"1644923584176\",\"creatorname\":\"User1\"" +
106                    "},\"" +
107                "]";
108
109        Itinerary itinerary = new Itinerary(
110            name: "Roma",
111            difficulty: 0,
112            new LocalTime( 1, 0),
113            new WayPoint( latitude: 41.91215825489158, longitude: 12.492356197611912),
114            new User( uid: "7bba5c72-7fbe-45ad-996a-686c8685a9b8"),
115            Date.from(Instant.from(DateTimeFormatter.ISO_DATE_TIME.parse( text: "2022-02-15T00:00:00.000Z"))));
116    );
117
118        itinerary.setIterId(5);
119        itinerary.setModifiedSince(1644923584176L);
120        itinerary.getCreator().setName("User1");
121
122        ArrayList<Itinerary> itineraries = new ArrayList<>();
123        itineraries.add(itinerary);
124        itineraries.add(itinerary);
125
126        ArrayList<Itinerary> result = parseItinerariesMock.parseItineraries(new JSONArray(itineraryJson));
127        assertEquals(itineraries, result);
128    }

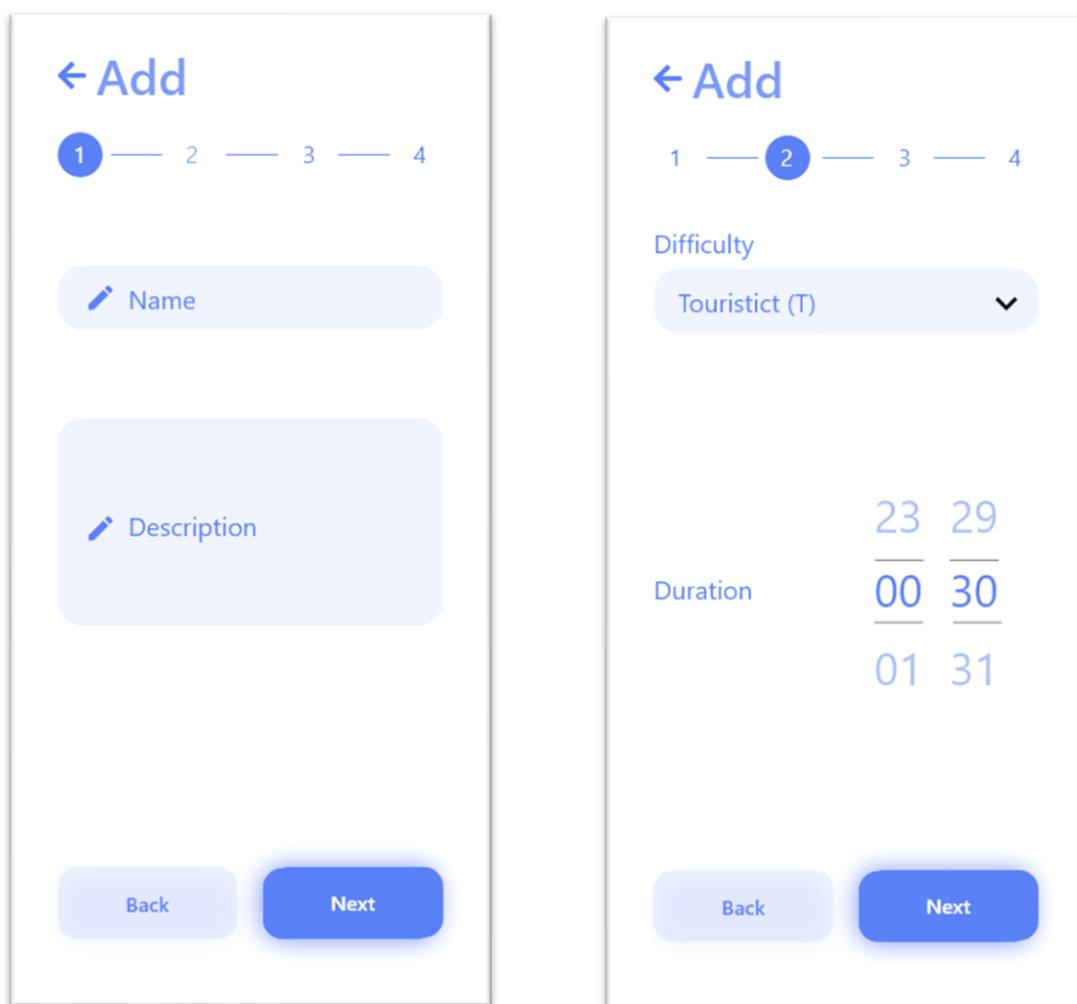
```

9. Valutazione sul campo

9.1 Prototipi finali

Rispetto ai mock-up iniziali e i prototipi e ai prototipi intermedi, grazie ai feedback accolti durante la fase di valutazione a priori, i prototipi finali (su cui si è modellata l’interfaccia grafica dell’applicazione) presentano dei miglioramenti soprattutto nelle schermate più criticate.

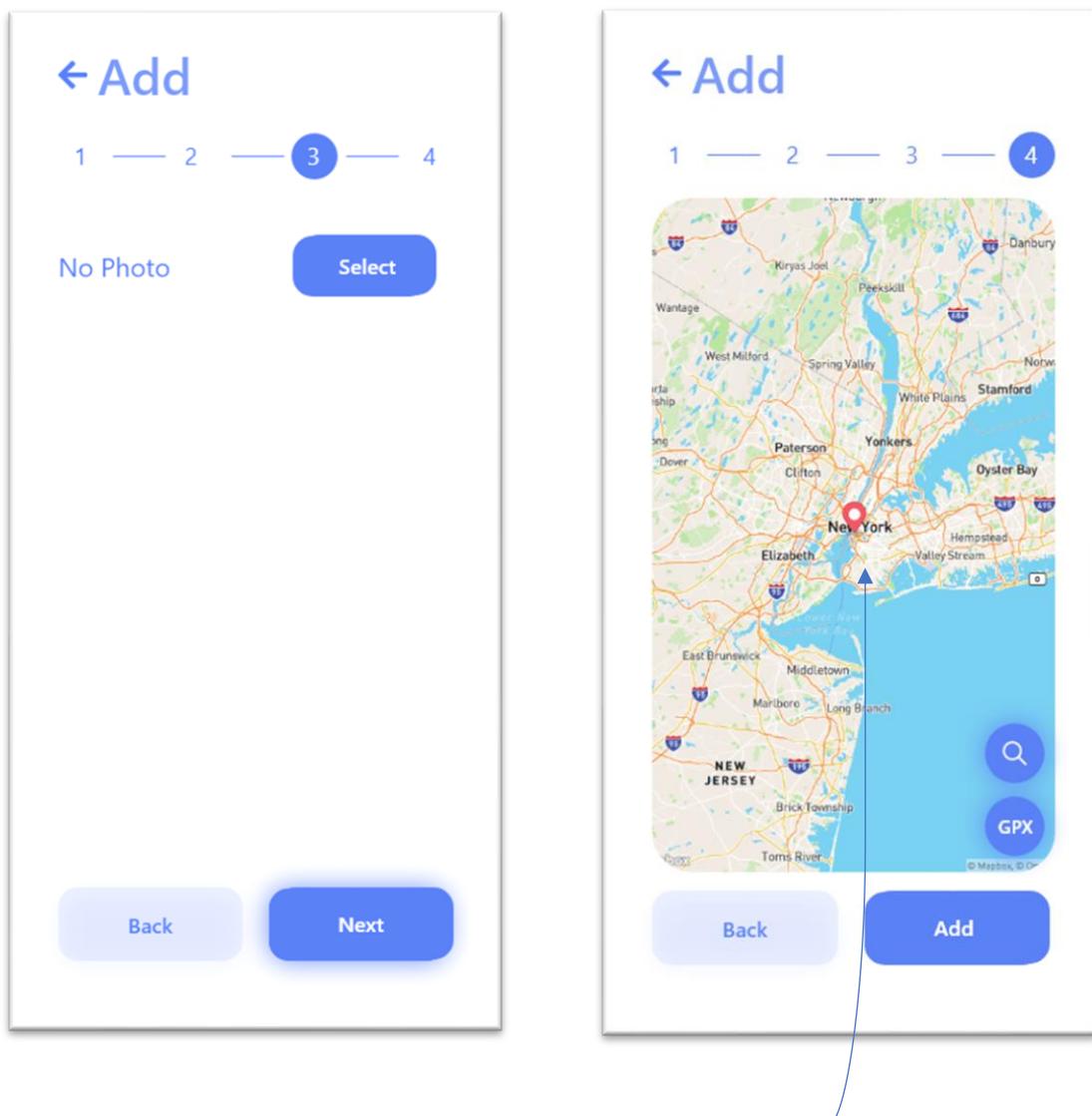
Innanzitutto, il processo di inserimento di un itinerario è stato suddiviso in task con l’introduzione di *fragment* multipli in modo da semplificare la quantità di informazioni presenti su schermo e focalizzare l’utente su un particolare aspetto del processo decisionale dell’inserimento.



Prototipo Finale dell’Inserimento di Itinerari

Adesso l'inserimento di percorsi nella piattaforma prevede quattro steps:

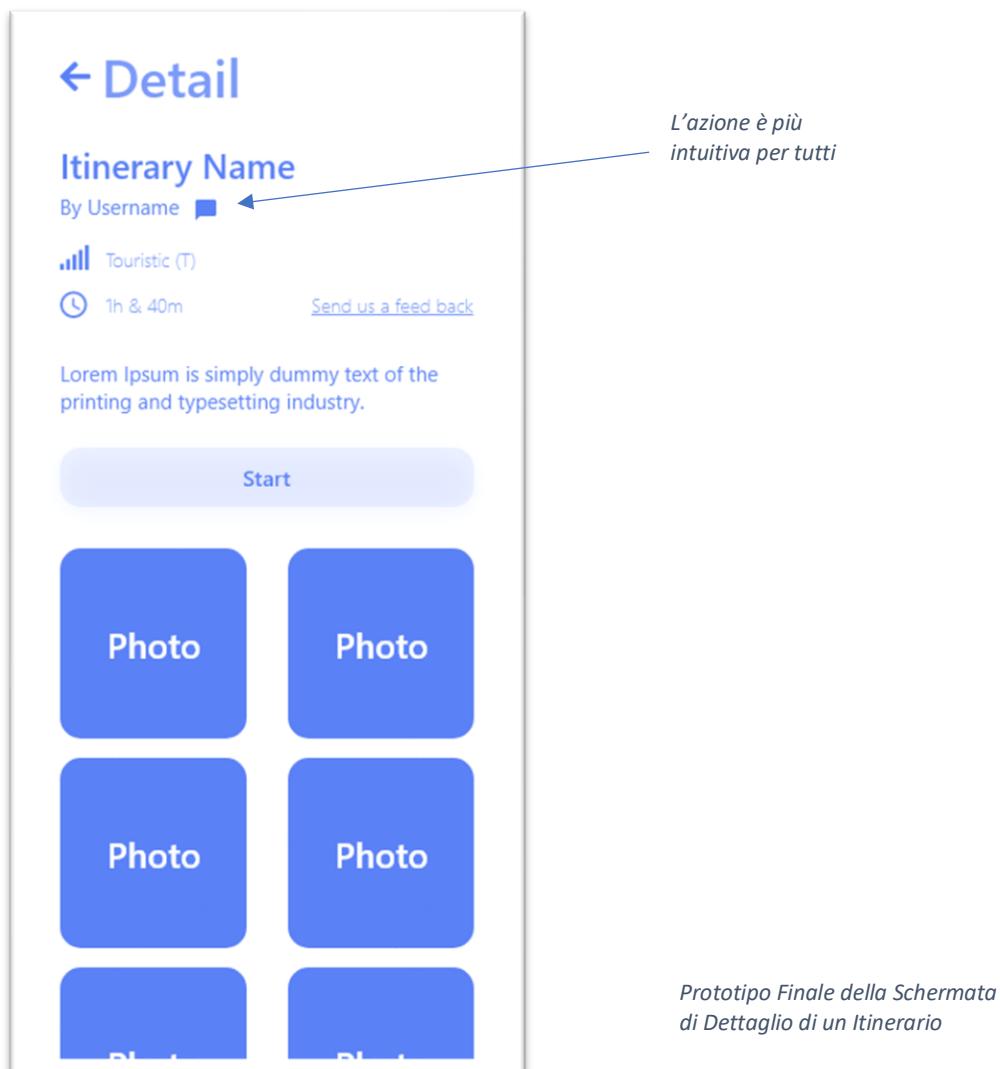
- 1) Inserimento di nome e descrizione
- 2) Inserimento di difficoltà e durata
- 3) Inserimento (facoltativo) di fotografie al percorso
- 4) Inserimento di un percorso geografico su mappa, attraverso interazioni intuitive



on Click: viene inserito sulla mappa un waypoint, ovvero un punto di passaggio; l'unione di tutti i waypoint sarà il percorso geografico; L'utente può modificare i waypoint già presenti sulla mappa, on Touch: viene eliminato il waypoint, on Long Press: il waypoint può essere spostato a piacimento sulla mappa;

Per favorire il passaggio da itinerari a chats, la pagina iniziale è stata dotata di un menù laterale da cui gli utenti possono spostarsi tra la lista di itinerari presenti in piattaforma e la lista delle proprie chats con altri utenti;

Per rendere intuitiva l'azione di poter mandare un messaggio al creatore di un percorso, abbiamo pensato di inserire un'icona di facile interpretazione. Per rendere i messaggi di feedback del sistema accattivanti, abbiamo creato delle *custom dialogs* con icone e animazioni coerenti con il messaggio di feedback (ad esempio nell'atto di eliminazione di un Itinerario il Sistema mostra una schermata di conferma con l'animazione di un cestino della spazzatura che si apre).



9.2 Usabilità sul campo

Per valutare l'usabilità del prodotto abbiamo installato l'applicazione sui dispositivi di otto (8) tester (il numero di tester anche se apparentemente esiguo sarà sufficiente ad individuare la quasi totalità dei problemi di usabilità vista la regola di Nielsen la quale afferma che basteranno da 3 a 5 tester per individuare oltre il 75% delle problematiche), tra cui alcuni provenienti dalla fase di usabilità a priori già a conoscenza dell'interfaccia e delle funzionalità del prodotto ma esponendoli ai cambiamenti visivi che si sono rivelati opportuni durante la progettazione, altri invece, sono stati esposti per la prima volta al prodotto.

Non abbiamo assegnato a ciascun utente dei task, ma semplicemente ci siamo limitati a spronare l'utilizzo dell'applicazione e di costruire un diario sulle perplessità riscontrate allegando screenshots, descrizioni brevi e registrazioni audio e video dei dispositivi.

Per fornire al committente strumenti di analisi sull'utilizzo e la fidelity degli utenti abbiamo integrato il servizio di analytics Amazon Pinpoint per effettuare il tracking degli eventi in-app generati dagli utenti.

Amazon Pinpoint oltre a fornire dettagli sulla user-base permette la creazione e il tracking di eventi custom;

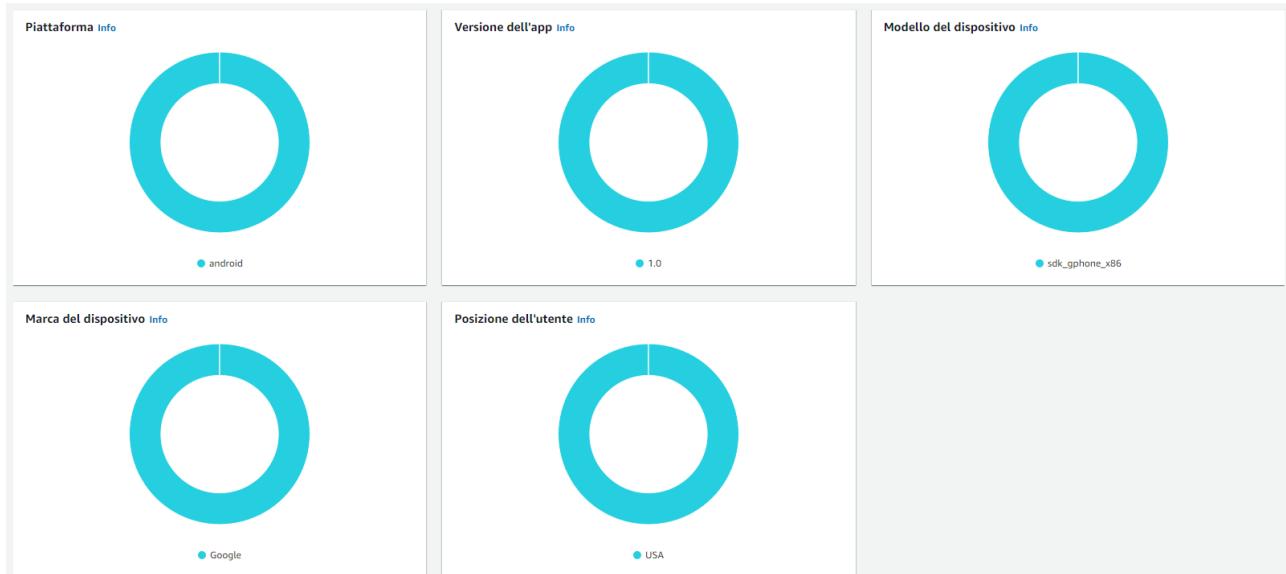
In particolare, andremo ad eseguire il tracking dei seguenti eventi:

- Session start
 - *Verrà generato ogni qualvolta un utente avvia l'app*
- Session stop
 - *Verrà generato ogni qualvolta un utente chiude l'app*
- FollowItinerary
 - *Verrà generato ogni qualvolta un utente deciderà di seguire un itinerario postato da un altro utente*
- InsertItinerary
 - *Verrà generato ogni qualvolta un utente deciderà di creare un nuovo itinerario*

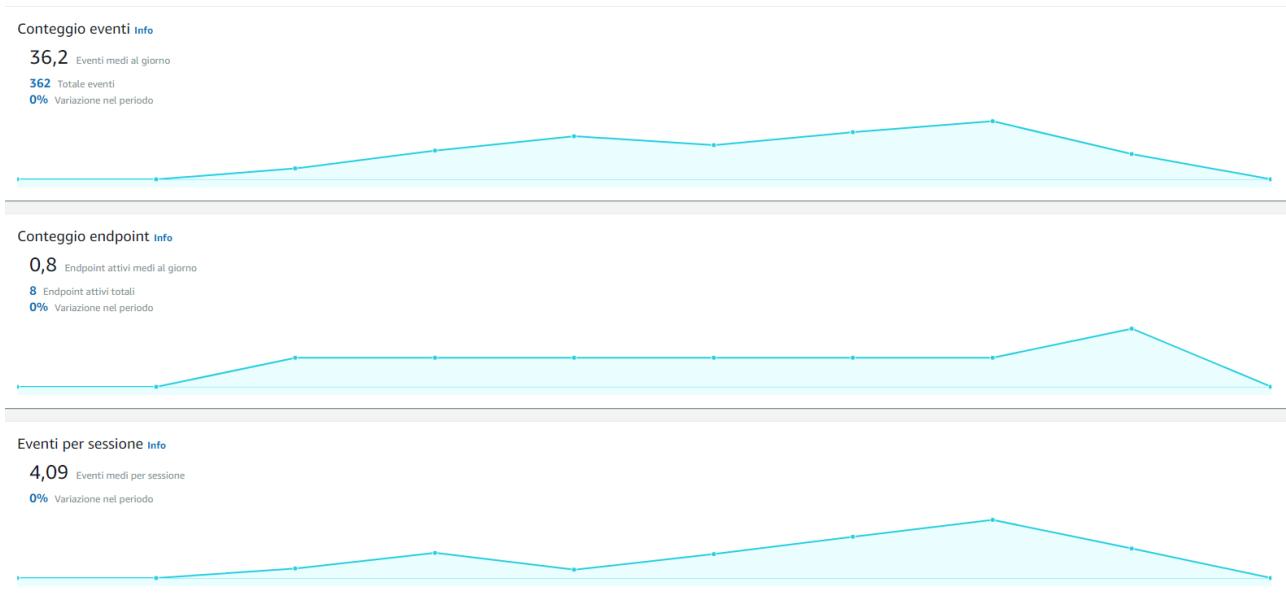
- SendMessage
 - *Verrà generato ogni qualvolta un utente invia un messaggio*
- SignIn
 - *Verrà generato ogni qualvolta un utente esegue il SignIn*
- UploadPhoto
 - *Verrà generato ogni qualvolta un utente che sta seguendo un itinerario decide di caricarvi una nuova foto*

Di seguito alcuni dei grafici a cui avrà accesso il committente per visualizzare l'andamento del prodotto (grafici di esempio dimostrativi a cui avrò accesso il committente):

- Dettagli sulla user-base:

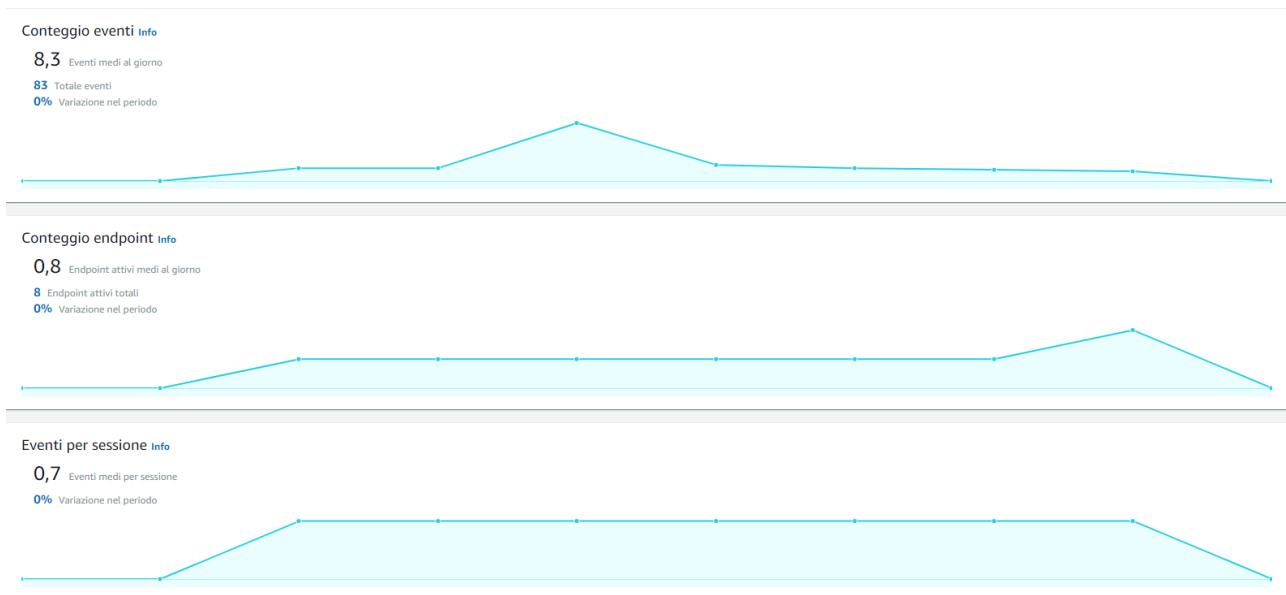


- Eventi totali / endpoint totali / numero di eventi per sessione totali:

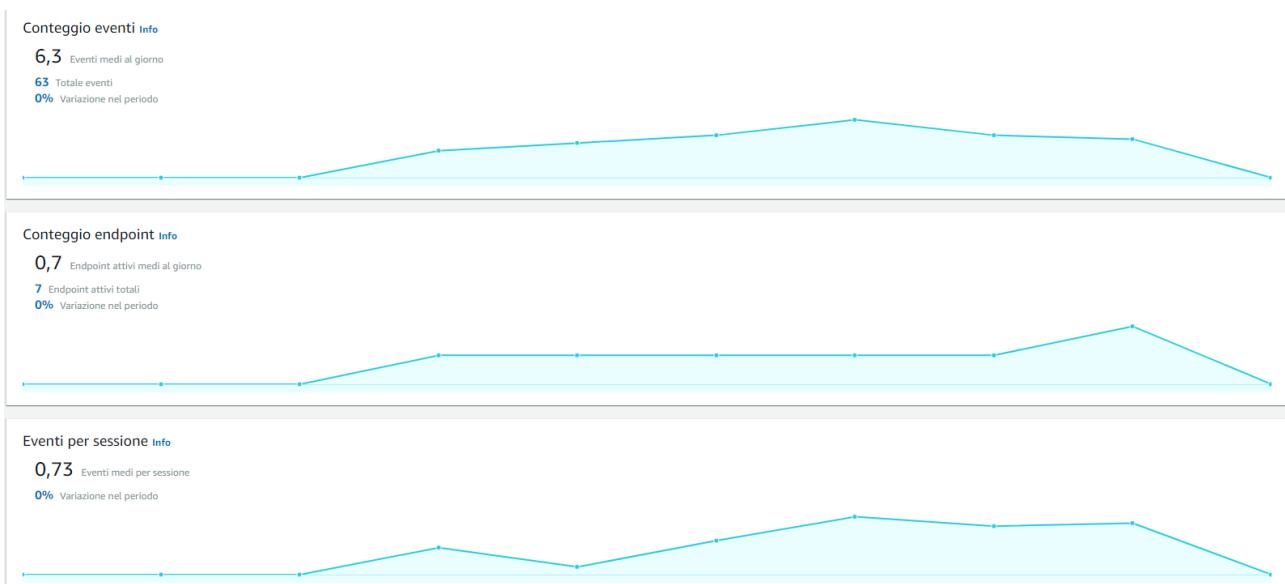


Gli eventi sono filtrabili per nome, data, attributi e per gli endpoint che li hanno generati ed i loro attributi (piattaforma, fuso orario, paese, modello, marca...) di seguito i grafici di alcuni eventi:

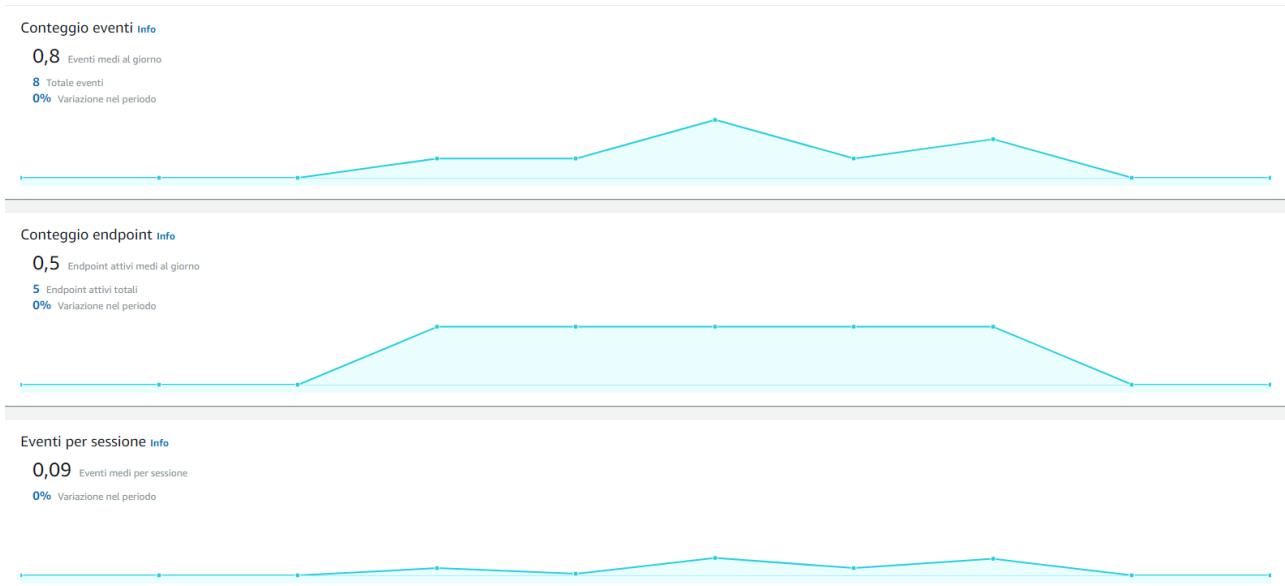
- SessionStart



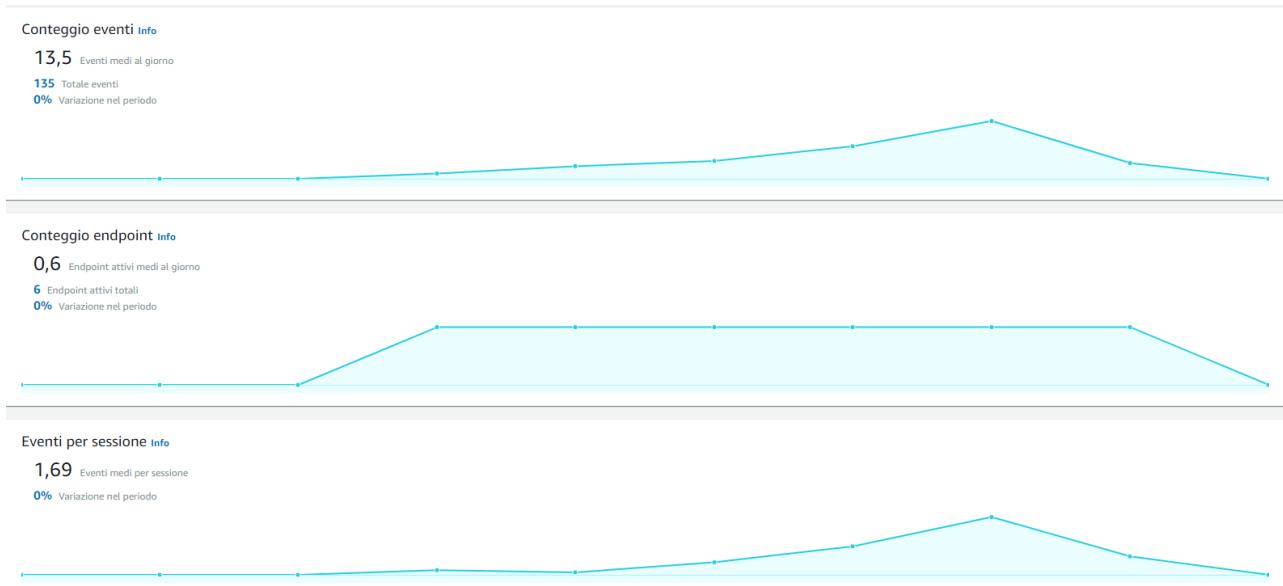
- FollowItinerary



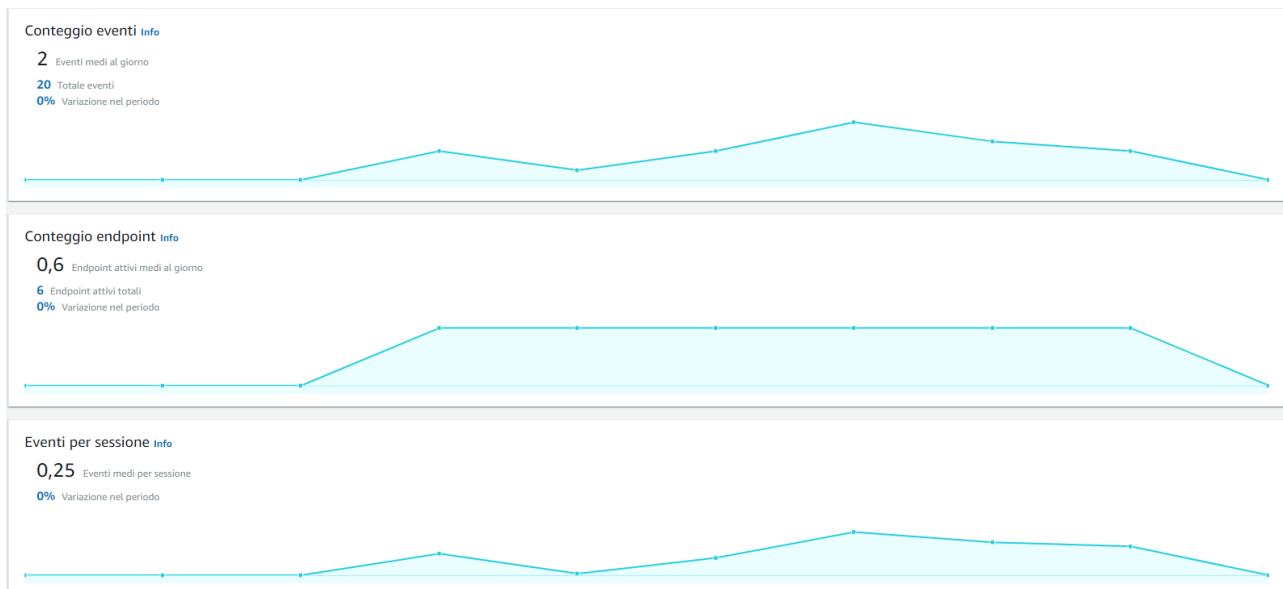
- InsertItinerary



● SendMessage



● UploadPhoto



Amazon Pinpoint oltre a rendere immediata la creazione di nuovi eventi da tracciare permetterà in caso si renda necessario il set-up di campagne pubblicitarie (e-mail, messaggi e notifiche-push) basate su eventi, questo permetterà al committente di aumentare la fidelity degli utenti, ad esempio, consigliando itinerari simili a quelli già seguiti.