

ENTE/I: DIETI

PROTOCOLLO N.: Uni-01-2021

DATA EMISSIONE: 26/10/2021

PAG. 1/98

- PIANO ATTIVITÀ

  PIANO OPERATIVO

  REPORT ATTIVITÀ: **FINALE**

Documento dei requisiti Software, Documento di design del sistema, Documento di testing del sistema

OGGETTO:

Documentazione del progetto NaTour

SINTESI DEI CONTENUTI:

Il Piano si riferisce alle attività di progetto da effettuare nell'ambito del corso di Ingegneria del Software, e contiene le seguenti linee di attività:

A. Analisi e specifica dei requisiti mediante notazione formale
B. Definizione dell'architettura e progettazione del sistema
C. Implementazione del sistema
D. Definizione del piano di testing e di alcuni Test Automatici, e analisi dell'usabilità sul campo

EMITTENTE: (FIRMA)	**DESTINATARI:**
ELABORA: Gruppo **INGSW2122_N_24** Carmine Guarracino N86003420 Giovanni Bentivoglio N86003364 Lorenzo Bracale N86003264	A: S. Di Martino, F. Cutugno, L. L. L. Starace, M. Grazioso P.C.: n.a.

1. INDICE

Sommario

2. Obiettivi

Il sistema sviluppato ha come obiettivo la creazione di una piattaforma social per appassionati di escursioni. Tale sistema prende il nome di NaTour ed è composto da una parte mobile front-end sviluppata in Android nativo (con Java) e una parte di back-end sviluppata in PHP.

2.1 Funzionalità richieste

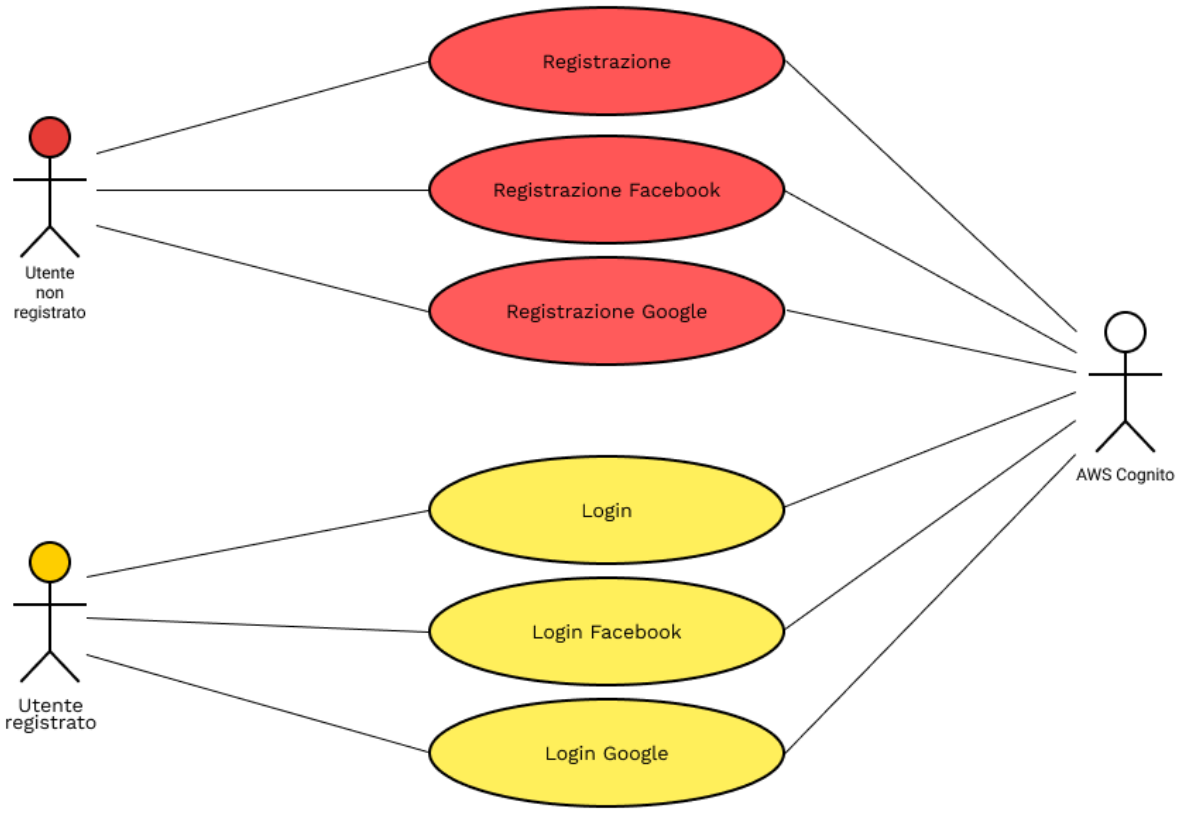
- Un utente può registrarsi/autenticarsi. È apprezzata la possibilità di autenticarsi utilizzando account su altre piattaforme come Google o Facebook
- Un utente autenticato può inserire nuovi itinerari (sentieri) in piattaforma. Un sentiero è caratterizzato da un nome, una durata, un livello di difficoltà, un punto di inizio, una descrizione (opzionale), e un tracciato geografico (opzionale) che lo rappresenta su una mappa. Il tracciato geografico deve essere inseribile manualmente (interagendo con una mappa interattiva) oppure tramite file in formato standard GPX.
- Effettuare ricerche di itinerari tra quelli presenti in piattaforma, con possibilità di filtrare i risultati per area geografica, per livello di difficoltà, per durata e per accessibilità a disabili.
- Visualizzare una schermata di dettaglio per ciascun sentiero. Questa schermata mostra tutte le informazioni note del sentiero, e visualizza su una mappa, preferibilmente interattiva, il punto di inizio e il tracciato geografico, se disponibile. Inoltre (si vedano funzionalità successive), la schermata di dettaglio mostra le eventuali recensioni degli utenti e fotografie caricate.
- Un utente può creare compilation di sentieri personalizzate, caratterizzate anche da un titolo e da una descrizione personalizzata.
- Un utente può caricare delle fotografie scattate percorrendo un sentiero. Le fotografie corrispondenti a un sentiero vengono mostrate nella pagina di dettaglio di quel sentiero. Inoltre, se la fotografia ha una posizione geografica di scatto salvata nei metadati, è apprezzata la possibilità di visualizzare un marker corrispondente alla fotografia sulla mappa, per mostrare in quale punto del sentiero è stata scattata. L'introduzione di un sistema in grado di riconoscere automaticamente (e bloccare) eventuali immagini inappropriate/offensive è facoltativa ma estremamente apprezzata.
- Un utente può inviare un messaggio privato (PM) a un altro utente, per esempio per chiedere ulteriori informazioni circa un itinerario da lui inserito. È possibile rispondere ai messaggi privati ricevuti.
- Un utente può segnalare informazioni inesatte/non aggiornate riguardo un sentiero. Una segnalazione è caratterizzata da un titolo e da una descrizione. I sentieri per cui sono presenti segnalazioni di inesattezza mostrano un warning nella schermata di dettaglio relativa.
- Gli amministratori possono visualizzare statistiche in tempo reale sul sistema (e.g. numero di utenti, numero di accessi, numero di ricerche, di recensioni, di itinerari, etc..)

3. Requisiti del software

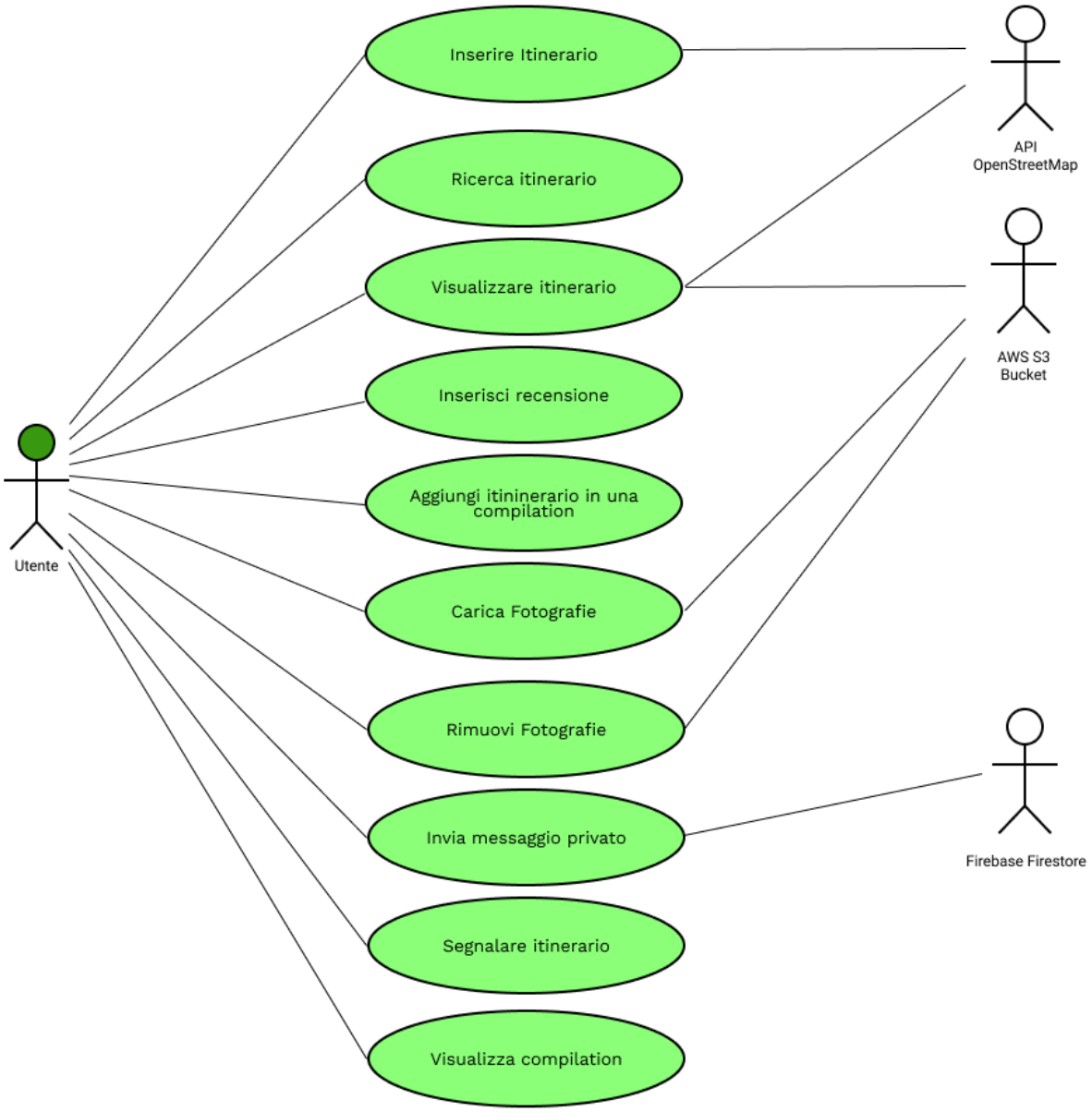
3.1 Modello Funzionale

3.1.1 Modellazione dei casi d'uso

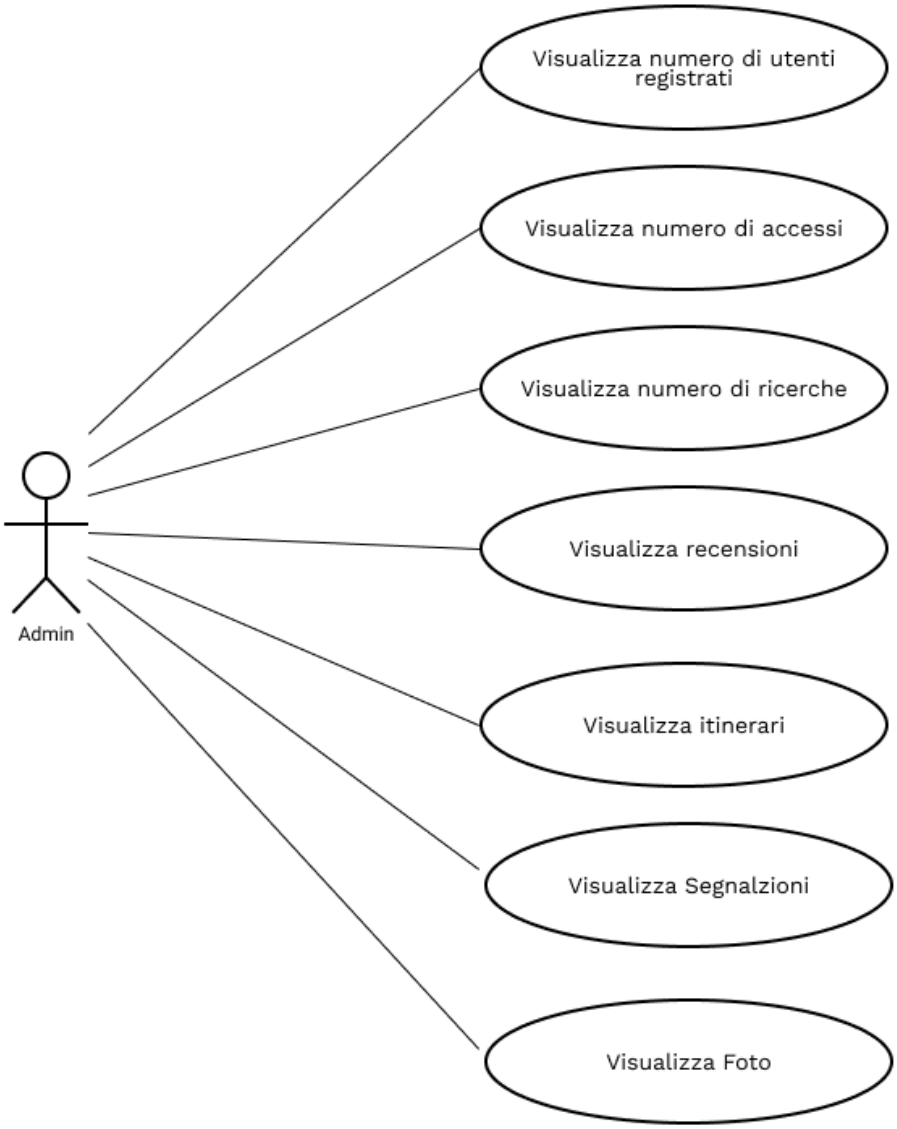
3.1.1.1 Login e registrazione



3.1.1.2 Funzionalità utente



3.1.1.3 Funzionalità admin



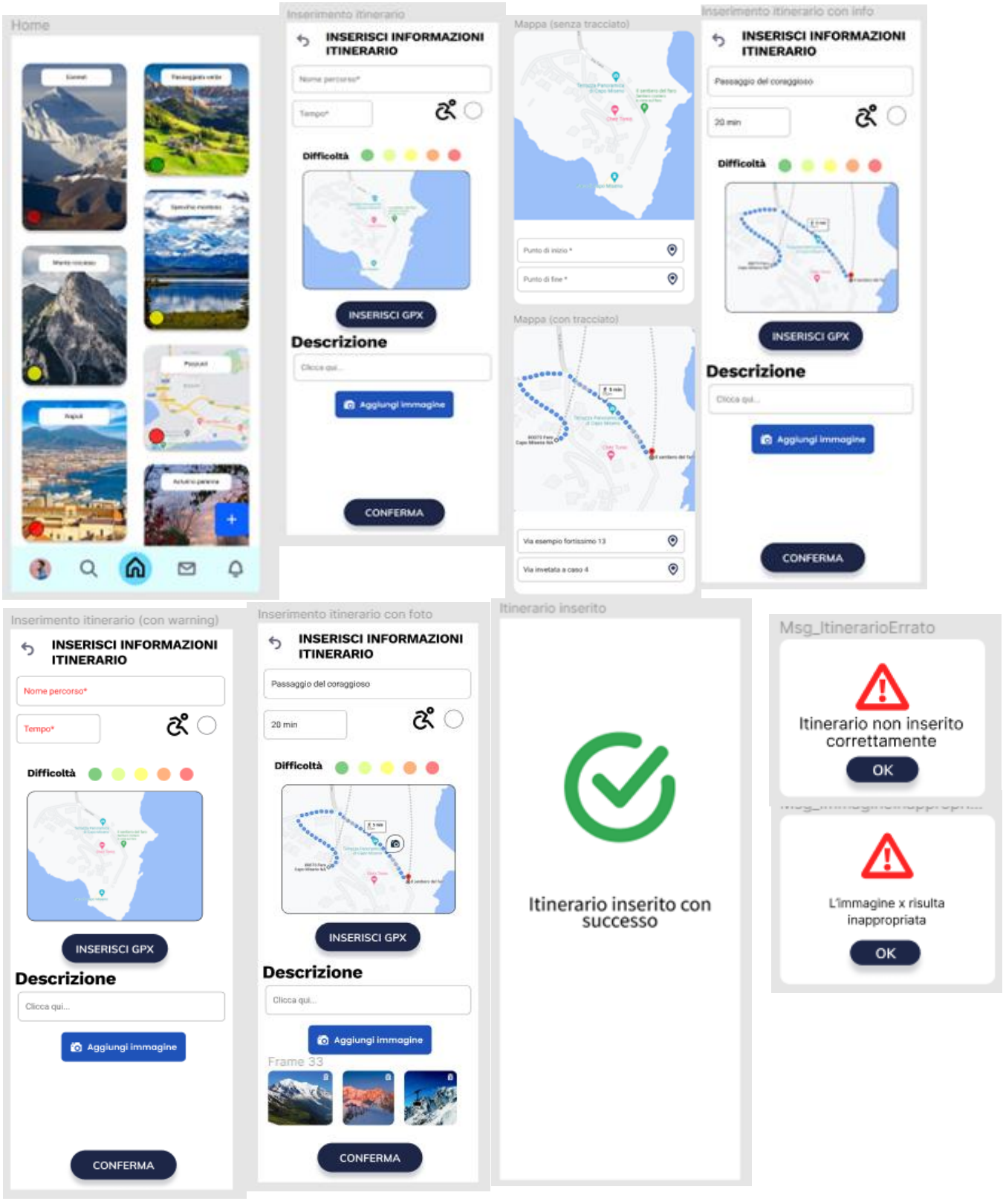
3.1.2 Tabelle di Cockburn

USE CASE #1	Inserimento di itinerario				
Goal in Context	L'utente vuole aggiungere un itinerario all'interno della piattaforma				
Preconditions	L'utente deve aver effettuato il login				
Success End Condition	L'utente riesce ad inserire l'itinerario nella piattaforma e visualizzarlo				
Failed End Condition	L'utente non riesce ad inserire l'itinerario nella piattaforma				
Primary Actor	Utente				
Trigger	Utente loggato clicca sul bottone "+" nella schermata "Home"				
Description	Step n°	Utente	Sistema	OpenStreetMap API	Image detection
	1	Preme "+" dalla schermata "Home"			
	2		Mostra "Inserimento itinerario"		
	3	Inserisce titolo, descrizione tempo, difficoltà e se è percorribile da persone con disabilità			
	4	Clicca sulla mappa			
	5		Mostra "Mappa (senza tracciato)"		
	6	Inserisce le informazioni del percorso			
	4			Traccia il percorso possibile a piedi	
	5		Mostra le informazioni dell'API sulla mappa interattiva. Esempio "Mappa (con tracciato)"		
	6	Verifica di aver inserito correttamente le informazioni. Esempio "Inserimento itinerario con info"			
	7	Clicca sul bottone "Conferma"			
	8		Mostra "Itinerario inserito" per poi mostrare successivamente "Home"		
Extension n° 1: L'utente conferma senza aver inserito tutte le info	3,1	Conferma senza aver inserito tutte le info			
	3,2		Mostra in overlay "Msg_ItinerarioErrato"		
	3,3	Clicca su "OK" o fuori dall'overlay			
	3,4		Mostra "Inserimento itinerario (con warning)"		
Extension n° 2: L'utente decide di inserire delle immagini	6,1	Clicca sul bottone "Aggiungi Immagine"			
	6,2		Permette di inserire immagine dal dispositivo		
	6,3		Manda immagine all'api per i contenuti sensibili		
	6,4				Controlla se l'immagine ha contenuti sensibili
	6,5		Mostra l'immagine al di sotto del bottone. Esempio "Inserimento itinerario con foto"		
Extension n° 3: L'utente ha inserito un immagine con contenuti sensibili	6,4,1		Mostra in overlay "Msg_ImmagineInappropriata"		
Extension n° 4: L'utente inserisce immagini con metadati	6,5,1		Mostra sulla mappa un icona che rappresenta la foto. Esempio "Inserimetno itinerario con foto"		
Extebsuib n°5: L'utente inserisce l'itinerario tramite file GPX	4,1	Clicca su "inserisci GPX" e inserisce file dal dispositivo			
	4,2			Inserisce i punti del file GPX e imposta un tracciato percorribile a piedi.	

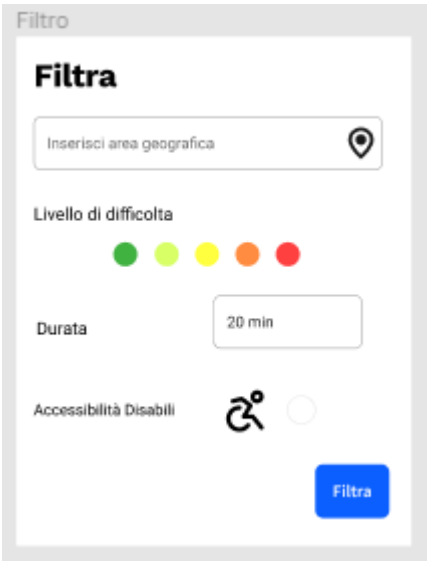
USE CASE #2	Ricerca itinerario		
Goal in Context	L'utente vuole cercare un itinerario		
Preconditions	L'utente è loggato		
Success End Condition	L'utente effettua la ricerca		
Failed End Condition	L'utente non riesce a completare la ricerca		
Primary Actor	Utente		
Trigger	Utente loggato visualizza "Ricerca Itinerari"		
Description	Step n°	Utente	Sistema
	1	Preme il box con su scritto "cerca"	
	2	dopo aver scritto qualcosa preme sull'icona lente di ingrandimento	
	3		Mostra gli itinerari associabili alla ricerca. Esempio "Ricerca Itinerari (box)"
	4	Completa la ricerca cliccando sull'itinerario cercato o scoprendone l'assenza	
Extension n°1: L'utente deceide di filtrare i risultati	4,1	Clicca sul bottone con il simbolo del filtro	
	4,2		Mostra in overlay la finestra "Filtro"
	4,3	L'utente inserisce le informazioni da filtrare	
			Mostra gli itinerari corrispondenti al filtro inserito. Esempio "Ricerca Itinerari (Box con filtro)"

3.1.3 Mockup Smartphone

3.1.3.1 Inserimento itinerario



3.1.3.2 Ricerca Itinerario



3.1.4 Presentazione dell'idea progettuale

Alla base del nostro lavoro vi è l'idea di un'applicazione social alla quale si può accedere con e-mail oppure con servizi come Google e Facebook, essa include elementi come chat, recensioni e segnalazione dei contenuti al suo interno. Parlando dei contenuti, in NaTour l'utente ha la possibilità di inserire gli itinerari con un'interfaccia pulita, la navigazione è infatti divisa in due schermate diverse, mantenendo una separazione tra inserimento delle direzioni e l'inserimento di altri dettagli dell'itinerario, in modo tale da dare all'utente un'ampia visione dell'insieme e anche di permettere di inserire i punti nel percorso senza troppe distrazioni. La presenza dell'opzione per scegliere la difficoltà serve per permettere ad altri utenti, esperti o meno, di ponderare le proprie scelte. È, inoltre, data la possibilità di inserire un itinerario tramite l'uso di un file GPX. Gli utenti sono in grado di visualizzare gli itinerari presenti sulla piattaforma nella homepage dove si può visualizzare immediatamente il nome e la difficoltà e cliccando su di essi è possibile visualizzare altri dettagli. È, inoltre data la possibilità di lasciare una recensione con una valutazione fino a cinque stelle oppure una segnalazione nel caso le informazioni inserite sono inesatte. Nel caso l'utente voglia salvare l'itinerario lo può includere in una compilation esistente o crearne una sul momento. Sempre nella schermata di visualizzazione dell'itinerario è possibile premere un bottone per aprire una chat di cui parleremo di seguito. L'utente è anche in grado di cercare e filtrare gli itinerari presenti sulla piattaforma in modo tale da trovare quello adatto alle sue esigenze, il modo in cui vengono visualizzati è simile a quello della homepage. La schermata del profilo dell'utente permette di vedere gli itinerari inseriti dall'utente che sta usando l'applicazione, inoltre permette di andare a rivedere gli itinerari inseriti nelle proprie compilation personali attraverso una schermata apposita raggiungibile tramite il menu del profilo a cui inoltre l'utente può eseguire il logout. L'utente ha la possibilità di inviare messaggi privati ad altri utenti della piattaforma dall'apposita schermata, i messaggi vengono scambiati attraverso una interfaccia minimale che permette un immediato scambio di informazioni.

Se l'utente che sta utilizzando l'applicazione è anche admin, in aggiunta alle funzionalità sopra elencate ha la possibilità di visualizzare le statistiche della piattaforma dall'apposita schermata raggiungibile dalla voce nel menù presente nella schermata del profilo.

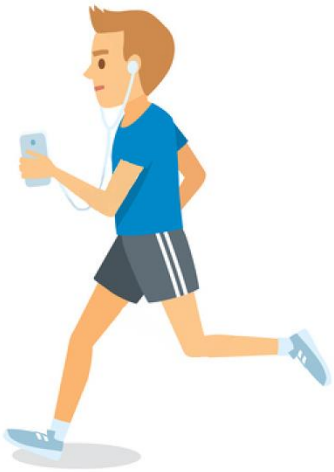
3.1.5 Individuazione del target degli utenti

Il tipo di utenti che userebbero l'applicazione sono utenti che hanno il tempo e la disponibilità economica per permettersi di spendere questi in viaggi ed escursioni. L'utente che usa la nostra applicazione può avere qualsiasi età; tuttavia, è consigliato per l'utilizzo di questa applicazione avere un'età adulta, per la precisione dai vent'anni in su. I tipi di utenti che possono usare questa applicazione spaziano tra persone che hanno la passione per le escursioni, persone sportive a cui piace correre in posti particolari, persone che vogliono per la prima volta intraprendere un'escursione. Le esigenze per una persona esperta di escursioni sono quelle di poter trovare facilmente itinerari interessanti sia per difficoltà che per luogo. Una persona sportiva deve poter scegliere luoghi facilmente accessibili rispetto a dove si trova; dunque, deve avere modo di scegliere facilmente l'itinerario in base al luogo. Una persona inesperta può avere la necessità di cercare l'itinerario

che sembra il più bello da visitare, magari attraverso dei riferimenti come foto, e che sia facile da percorrere.

Utente Esperto

- Cerca itinerari interessanti
- Esplora il mondo
- Condivide percorsi
- Aiuta i meno esperti



Utente Sportivo

- Cerca itinerari lunghi
- Corre vicino casa
- Ripercorre le solite strade

Utente Casual

- Cerca itinerari facili
- Disposto a spingersi oltre
- Non fa spesso escursioni
- Necessita aiuto dagli esperti

3.1.6 Valutazione dell'usabilità a priori

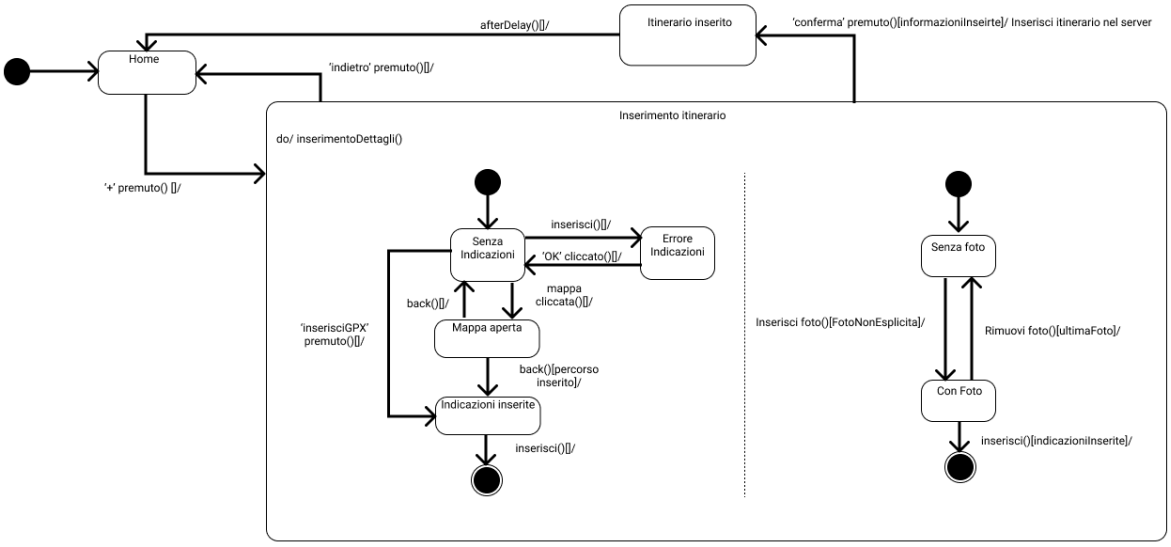
L'interfaccia dell'applicazione offre diverse affordance esplicite e altre implicite. Ad esempio, per visualizzare i dettagli dell'itinerario bisogna cliccare sull'anteprima dell'itinerario stesso sulle interfacce della homepage, del profilo e della ricerca, senza che però ci sia scritto nulla che indichi l'associazione che cliccando possa esaminare i dettagli, tuttavia è presente, nella home page, un bottone per inserire gli itinerari presenta la scritta "+ itinerario" indicando la possibilità di aggiungere qualcosa, ovvero un itinerario. Dalla schermata di inserimento ogni affordance ha una descrizione ad eccezione della mappa che vuole semplicemente che l'utente la clicchi; tuttavia, senza indicazioni l'utente dovrebbe intuire la possibilità di cliccarci sopra. Una volta capito cosa ci può essere in un itinerario l'utente sa che nella homepage vengono mostrati itinerari con le stesse informazioni e se si chiedesse come vedere le informazioni potrebbe pensare di dover cliccare le anteprime. Per navigare nelle varie zone dell'app è presente una bottom navigation bar con sopra dei simboli esplicativi delle funzioni possibili, una casetta per la home page, un omino per il profilo, una busta per lettere per i messaggi diretti e una lente di ingrandimento per la ricerca. Si è deciso di usare questi simboli in quanto sono quelli più comuni nell'ambito di applicazioni di questo genere. Nel pannello del profilo sono visualizzati immediatamente il nome dell'utente insieme agli itinerari che ha inserito, da lì può accedere al menù del profilo con un bottone a sandwitch come nelle applicazioni più recenti a cui l'utente dovrebbe essere abituato. Dal pannello della ricerca vi è una scritta "cerca qui" con icona a lente di ingrandimento che dovrebbe indicare la possibilità di inserire dei criteri di ricerca, inoltre vi è un bottone con icona a filtro che non occupa troppo la vista dell'utente rispetto a quello che cerca. Nel pannello della chat ci sono solo le chat

aperte in precedenza e cliccando su una di esse può messaggiare, ispirato alle app di messaggistica più comuni non dovrebbe creare dubbi all'utente.

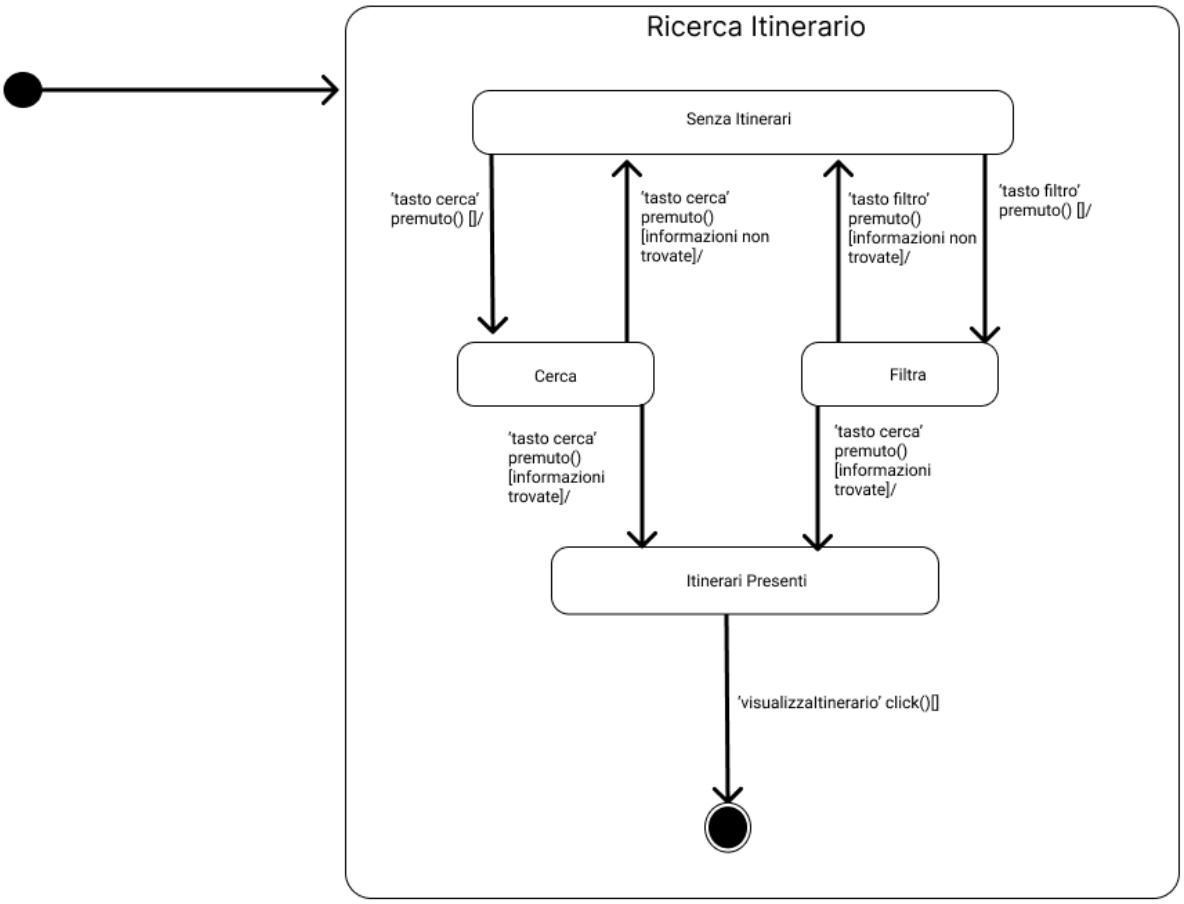
Registrazione	Login	Inserire itinerario	Ricerca itinerario	Visualizzare itinerario	Inserisci recensione	Aggiungi itinerario in una compilation	Carica fotografie	Rimuovi fotografie	Invia messaggio privato	Segnalare itinerario	Visualizza compilation	Visualizza dati sistema
Prototipo1 (figma): Non essendoci un sistema di conservazione dei dati né di inserimento da tastiera mostra solo l'interfaccia	Prototipo1 (figma): Non potendo ritirare alcun dato permette solo di poter accedere al resto del prototipo	Prototipo1 (figma): Si può interagire minimamente con le affordance, mostrando una mappa con un solo percorso possibile e delle informazioni precompilate, idem per le foto. È possibile simulare il cambio della difficoltà	Prototipo1 (figma): È possibile effettuare una sola ricerca dunque gli itinerari mostrati sono sempre gli stessi, la ricerca è solo un cambio di schermata. Il filtro disponibile permette di simulare un filtro non attualmente iteragibile eccezion fatta per la difficoltà selezionabile.	Prototipo1 (figma): può essere visualizzato solo un itinerario presente nella schermata della home.	Prototipo1 (figma): non è possibile inserire una recensione tuttavia viene mostrata la schermata di inserimento e delle stelle con cui si può effettivamente interagire.	Prototipo1 (figma): non è possibile inserire itinerari all'interno di alcuna compilation tuttavia ne è simulata l'interfaccia.	Prototipo1 (figma): È possibile inserire un solo gruppo di fotografie già preimpostate	Prototipo1 (figma): È possibile rimuovere le fotografie insieme ma non singolarmente solo dopo che sono state inserite	Prototipo1 (figma): non vi è la possibilità di mandare messaggi, vi è però la possibilità di esaminare l'interfaccia con delle chat e utenti di esempio.	Prototipo1 (figma): è possibile visualizzare le segnalazioni preimpostate dell'unico itinerario visualizzabile, allo stesso modo si può vedere l'interfaccia di inserimento senza poter aggiungere alcuna segnalazione	Prototipo1 (figma): Le compilation presenti non contengono alcun itinerario, ad eccezione della prima compilation che contiene l'unico itinerario visualizzabile.	Prototipo1 (figma): è presente una schermata accessibile indipendentemente, rappresentando l'utilizzo limitato agli admin, la schermata mostra le statistiche che non cambiano in tempo reale.
Prototipo2 (Android Studio): è possibile registrarsi accedendo alla schermata di registrazione dal login inserendo i dati dell'utente e confermando il codice inviato sulla mail inserita.	Prototipo2 (Android Studio): il login è funzionante in quanto ritira le informazioni dell'utente registrato e impedisce l'accesso a utenti che non sono ancora registrati.	Prototipo2 (Android Studio): l'inserimento dell'itinerario permette l'inserimento di tutte le informazioni dell'itinerario e delle posizioni del percorso.	Prototipo2 (Android Studio): la ricerca dell'itinerario funziona attraverso la ricerca del nome dell'itinerario oppure attraverso l'uso di un filtro che permette di cercare un itinerario che rispetta determinati parametri.	Prototipo2 (Android Studio): è possibile visualizzare ogni informazione presente di ogni itinerario presente in piattaforma.	Prototipo2 (Android Studio): le recensioni possono essere inserite e vengono visualizzate per l'itinerario a cui sono state inserite.	Prototipo2 (Android Studio): è possibile inserire un itinerario in una compilation già creata. È inoltre possibile creare una compilation dando il nome e la descrizione di quest'ultima.	Prototipo2 (Android Studio): è possibile inserire una foto alla volta. La foto sarà visualizzabile nella schermata di dettaglio dell'itinerario e anche nella schermata di inserimento prima della conferma.	Prototipo2 (Android Studio): è possibile eliminare una foto qualora ci trovassimo nella schermata di inserimento itinerario	Prototipo2 (Android Studio): Per inviare messaggi privati bisogna aprire preventivamente la chat dalla schermata di dettaglio del proprietario dell'itinerario con cui si vuole messaggiare per poi andare nella schermata adatta per lo scambio di messaggi.	Prototipo2 (Android Studio): è possibile segnalare un itinerario dalla sua schermata di dettaglio. Se un itinerario presenta una segnalazione ci sarà un warning sul bottone di inserimento. Cliccandolo sarà possibile esaminare le segnalazioni.	Prototipo2 (Android Studio): ogni utente potrà visualizzare le proprie compilation dal proprio pannello utente. È inoltre possibile esaminare l'itinerario salvato nella compilation.	Prototipo2 (Android Studio): Se l'utente è un admin può visualizzare le statistiche accedendo all'apposito pannello dalla schermata del profilo. Le statistiche si aggiorneranno in tempo reale.

3.1.7 Statechart delle funzionalità

3.1.7.1 Inserimento itinerario



3.1.7.2 Ricerca itinerario



3.1.8 Glossario

Piattaforma social: piattaforma in cui più utenti possono comunicare e scambiare informazioni.

Front-end: Lato del sistema che si occupa dell'applicazione destinata agli utenti finali.

Back-end: Lato del sistema destinato alla gestione delle informazioni.

Java: Linguaggio di programmazione a oggetti usato per lo sviluppo del front-end.

Android studio: IDE utile per lo sviluppo di applicazioni su dispositivi Android.

IDE: editor per lo sviluppo di codice.

PHP: Linguaggio di programmazione usato soprattutto per i servizi sul web.

Admin: Amministratore dell'applicazione.

Homepage: detta anche "Home" è la prima pagina dell'applicazione, offre le funzionalità principali.

Overlay: finestra di dialogo che si mostra al di sopra dell'interfaccia principale

Loggare: Indica l'operazione di effettuare il log-in, ad esempio "l'utente è loggato" ovvero "l'utente ha effettuato il login.

Chat: servizio di messaggi privati.

GPX: file contente i punti per creare una percorso sulla mappa.

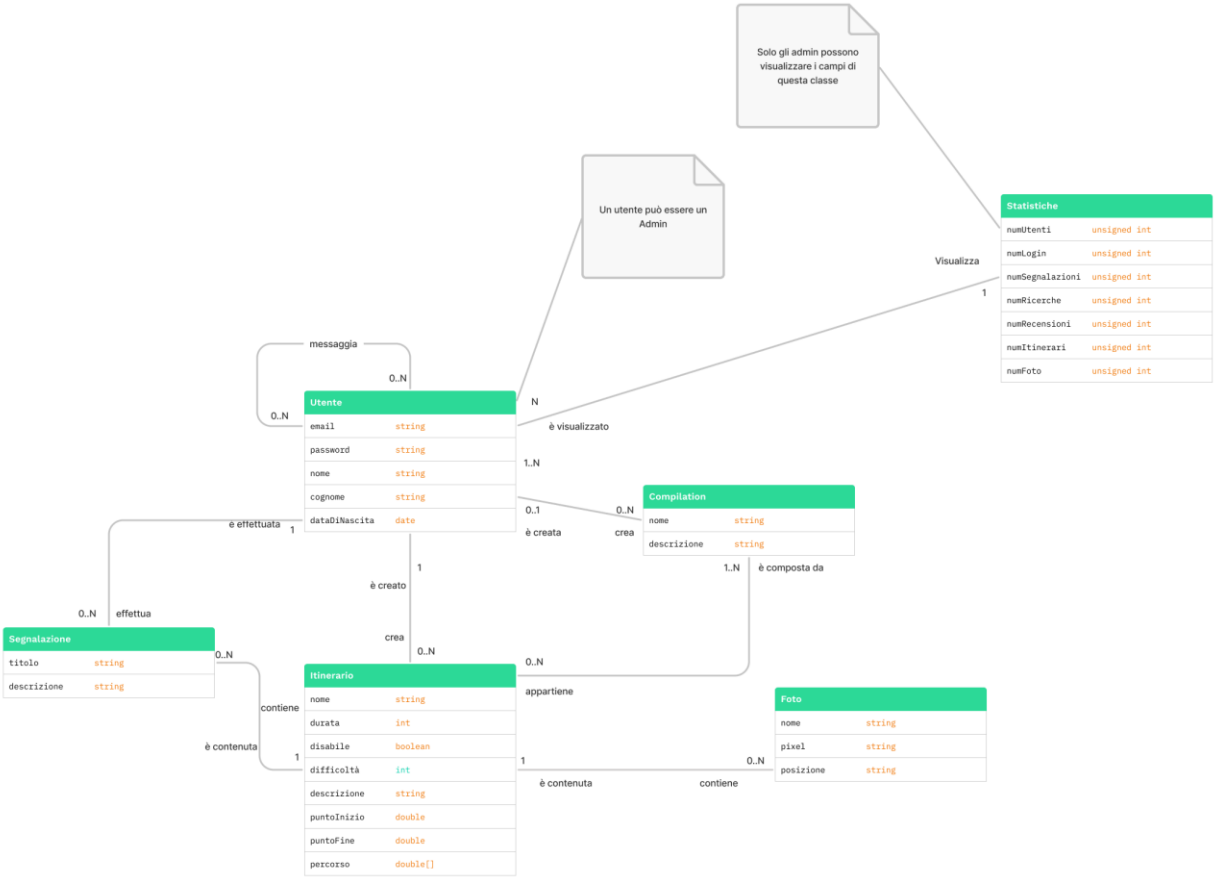
Logout: Uscire dalla applicazione e resettare la sessione attuale.

Bottom Navigation Bar: Barra che si trova nella parte inferiore della schermata permette di navigare tra le varie funzioni dell'applicazione.

Bottone Sandwitch: classico bottone con icona a tre barre, come gli strati di un panino (o sandwitch)

3.2 *Modello* DI DOMINIO

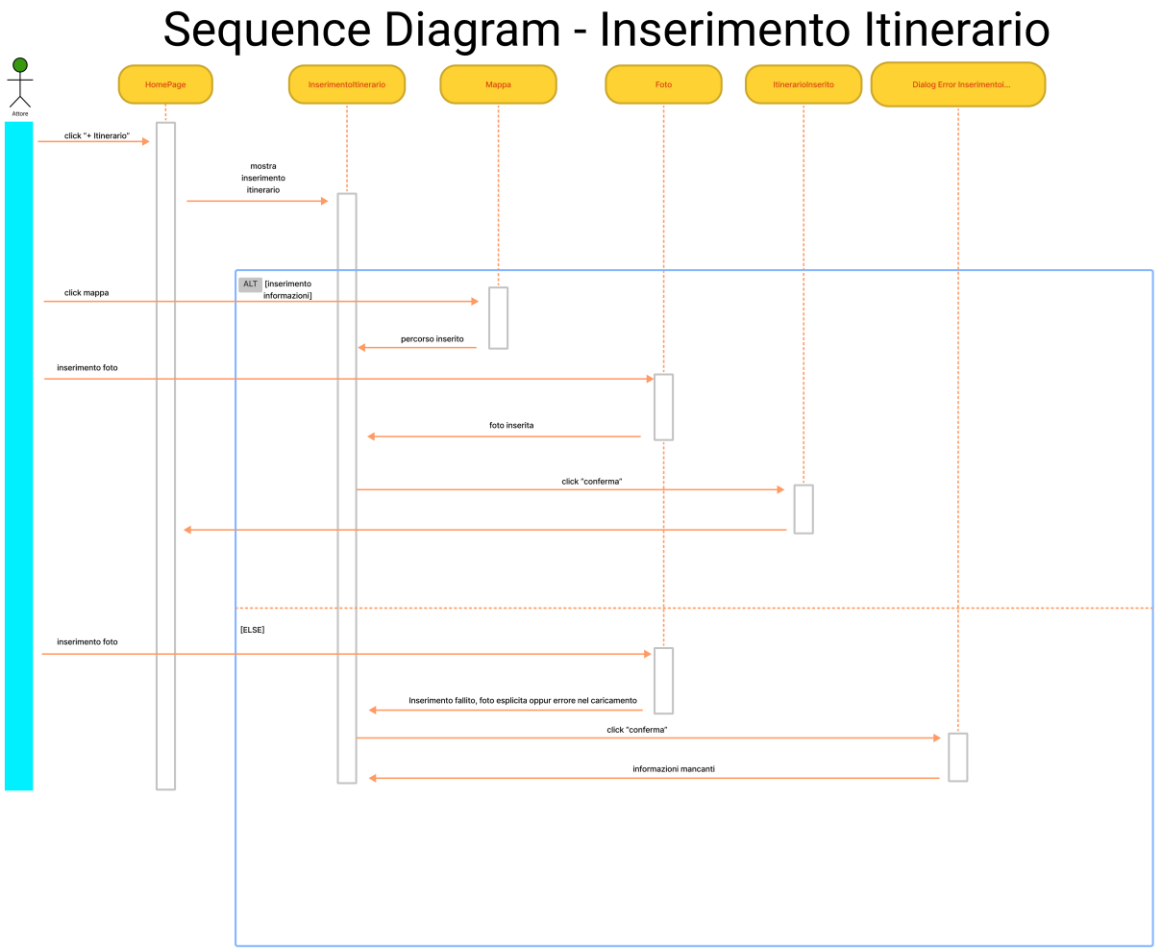
3.2.1 DIAGRAMMA DELLE CLASSI DI ANALISI



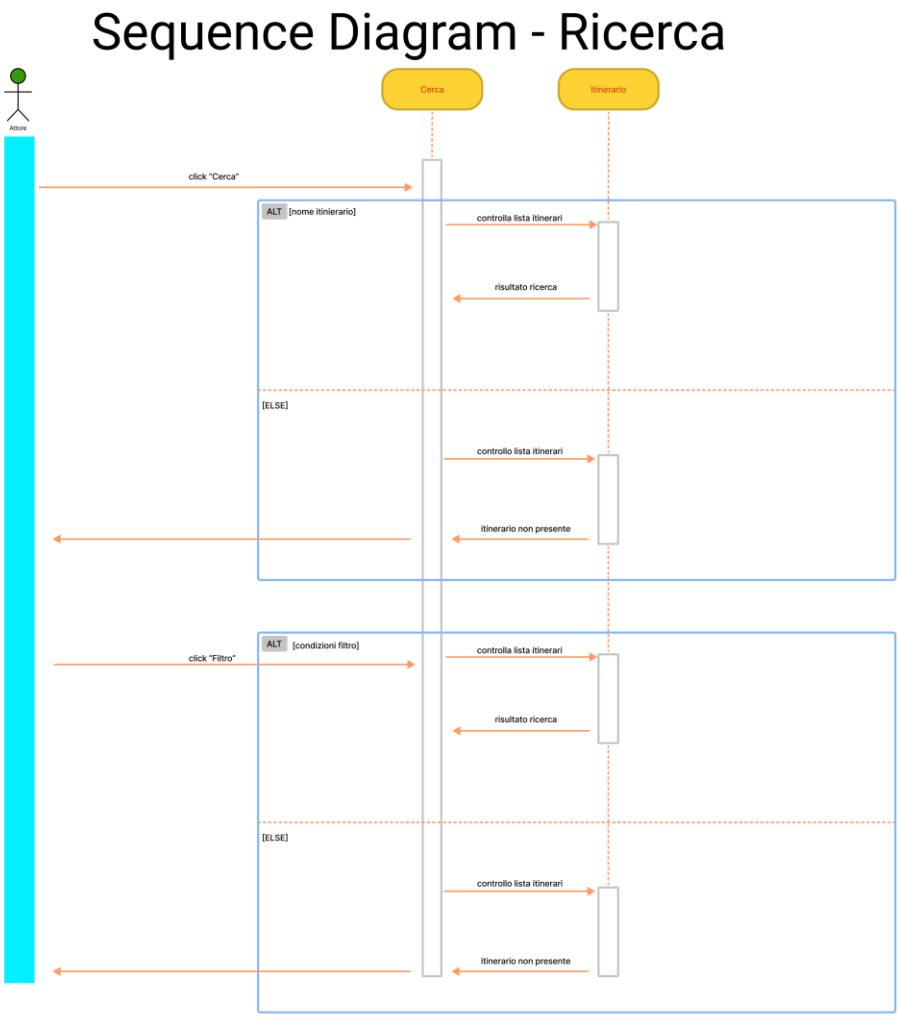
NDR. Questo è un diagramma di analisi e non rappresenta specifiche del prodotto finito ma solo una fase di progettazione degli oggetti del sistema.

3.2.2 DIAGRAMMI DI SEQUENZA DI ANALISI

3.2.2.1 Inserimento itinerario



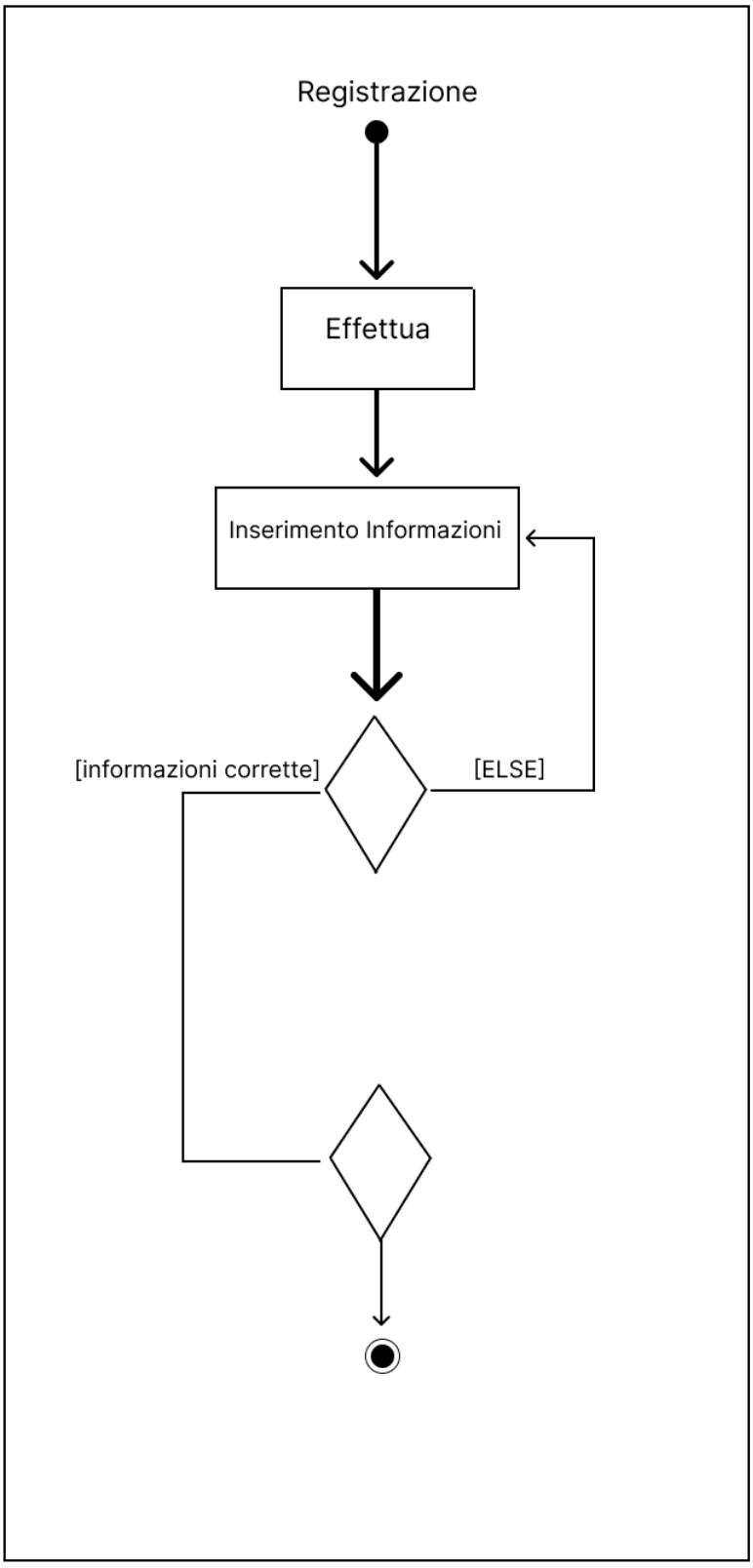
3.2.2.1 Ricerca itinerario



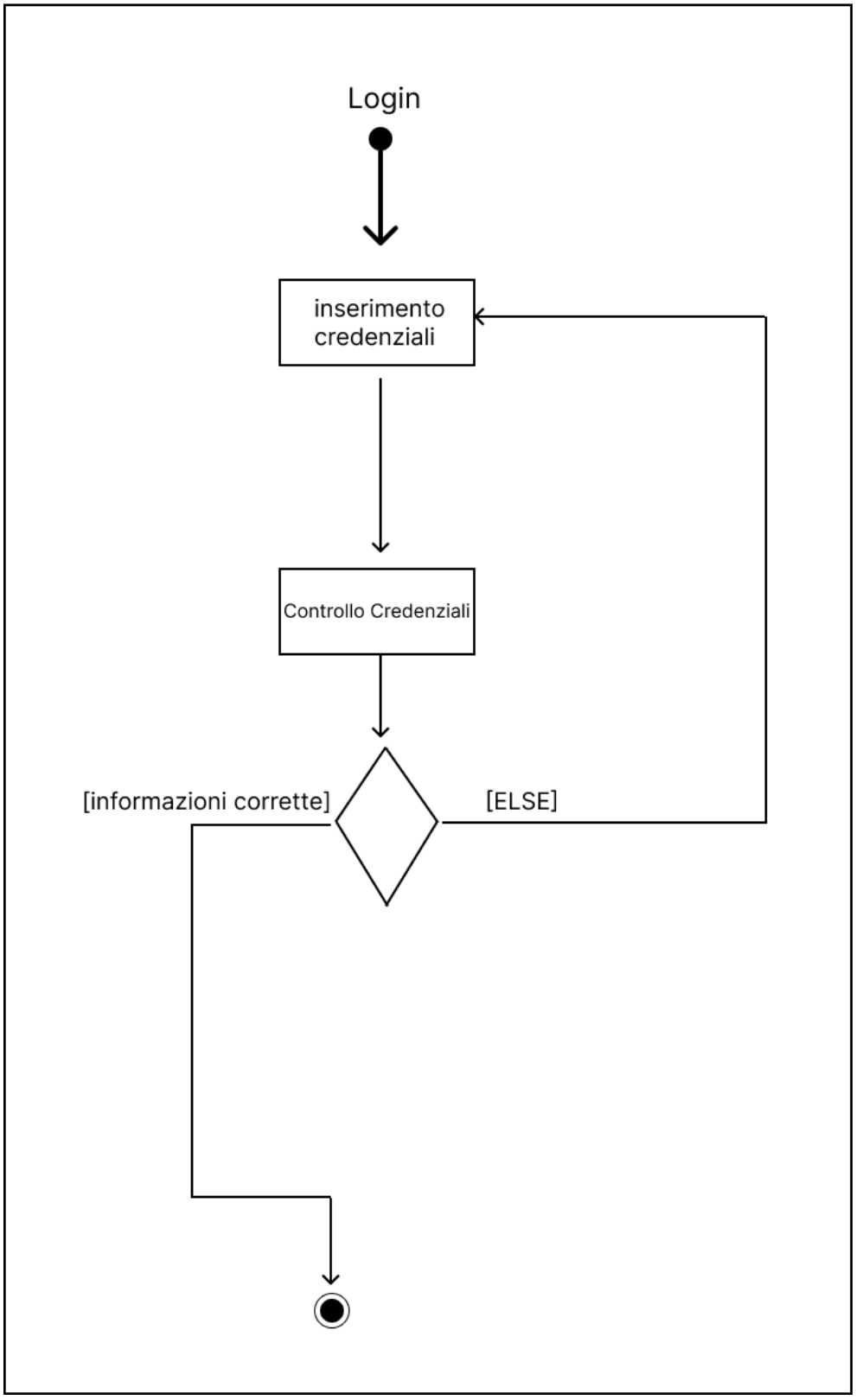
NDR. Questi sono diagrammi di analisi e non rappresentano quello che sarà il prodotto finale. Sono solo una fase di progettazione adatta alla comprensione del problema.

3.2.3 DIAGRAMMI DI ATTIVITÀ

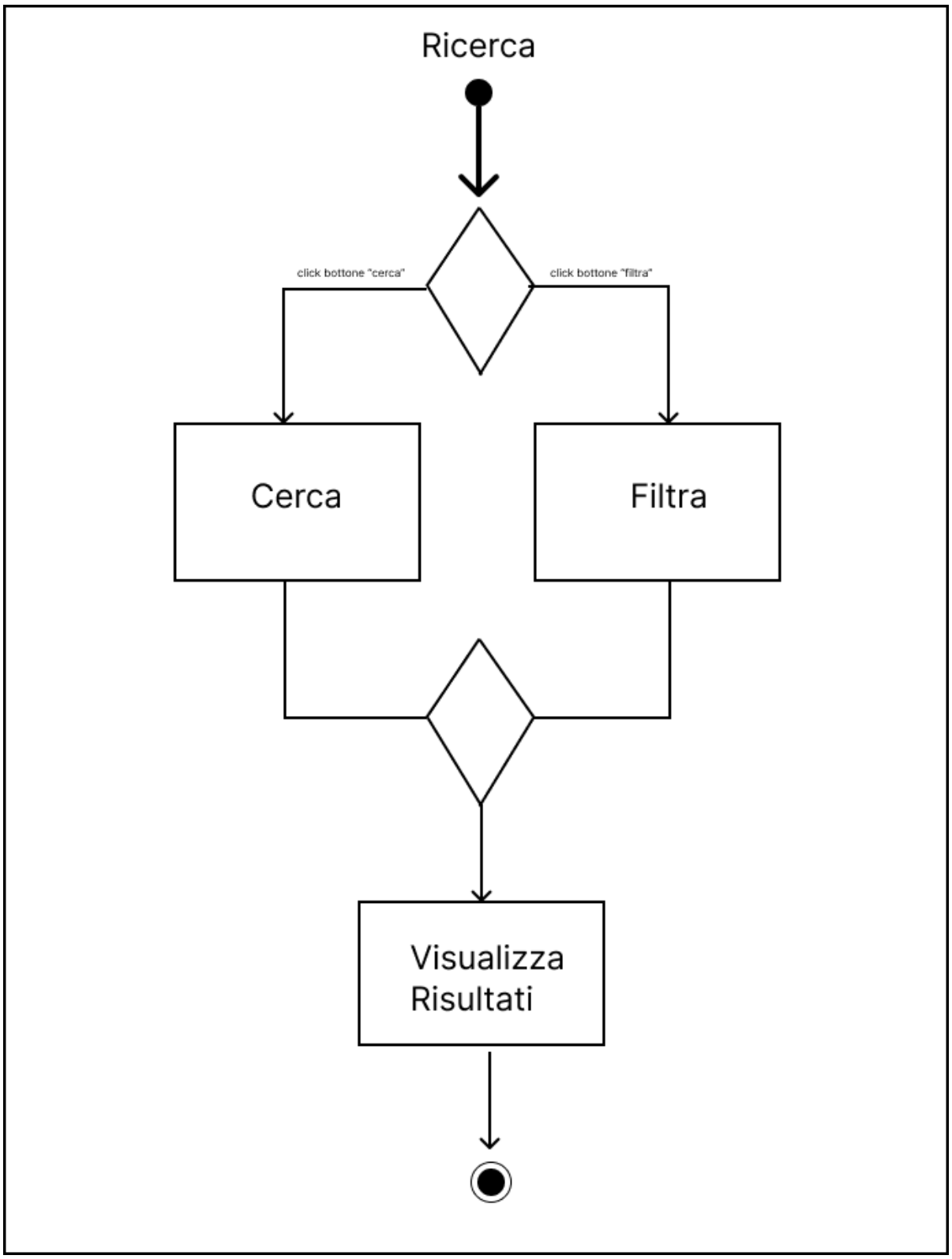
3.2.2.1 registrazione utente



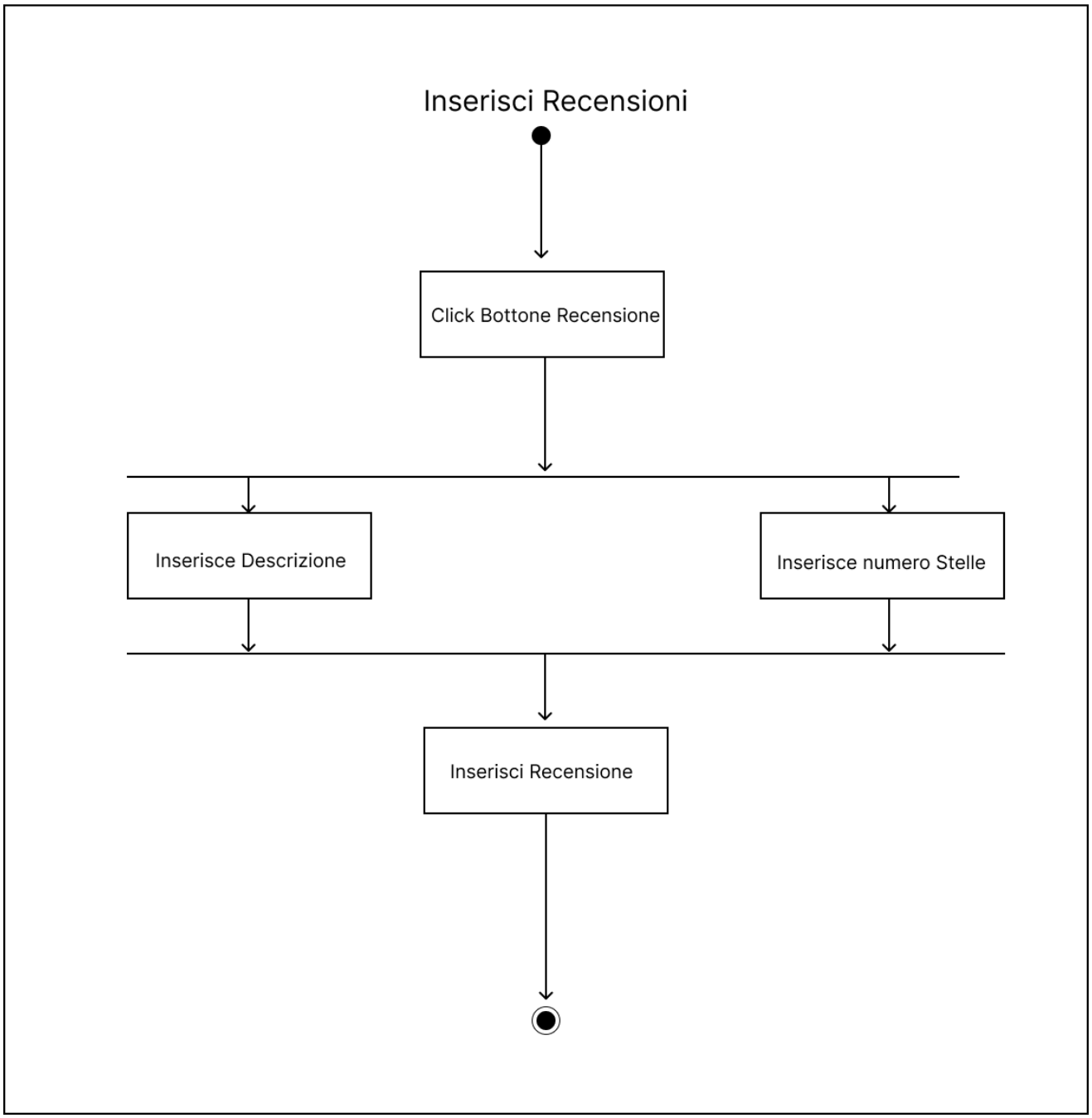
3.2.2.2 Login


Login
inserimento credenziali
Controllo Credenziali
[informazioni corrette]
[ELSE]

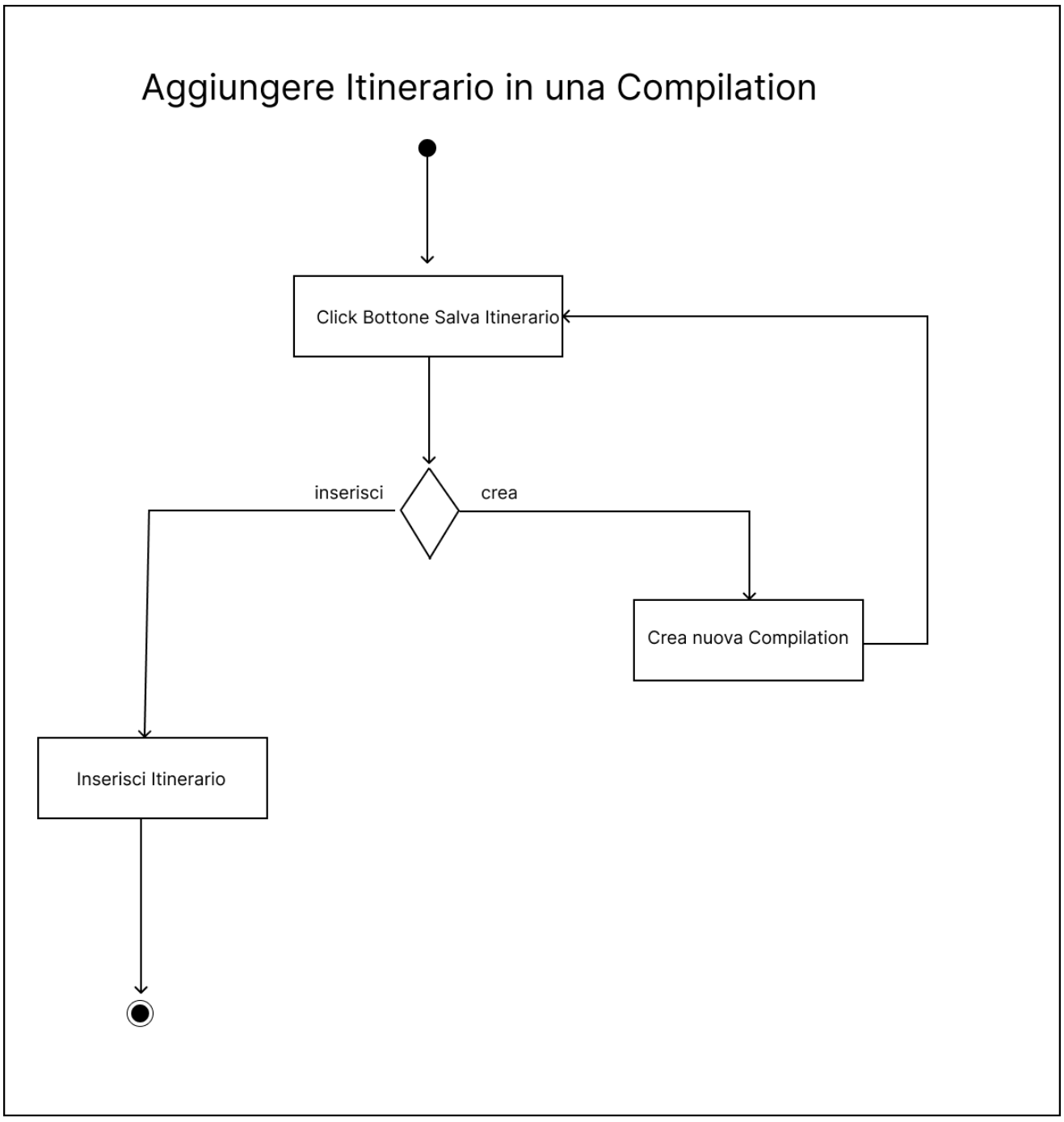
3.2.2.3 Ricerca itinerario



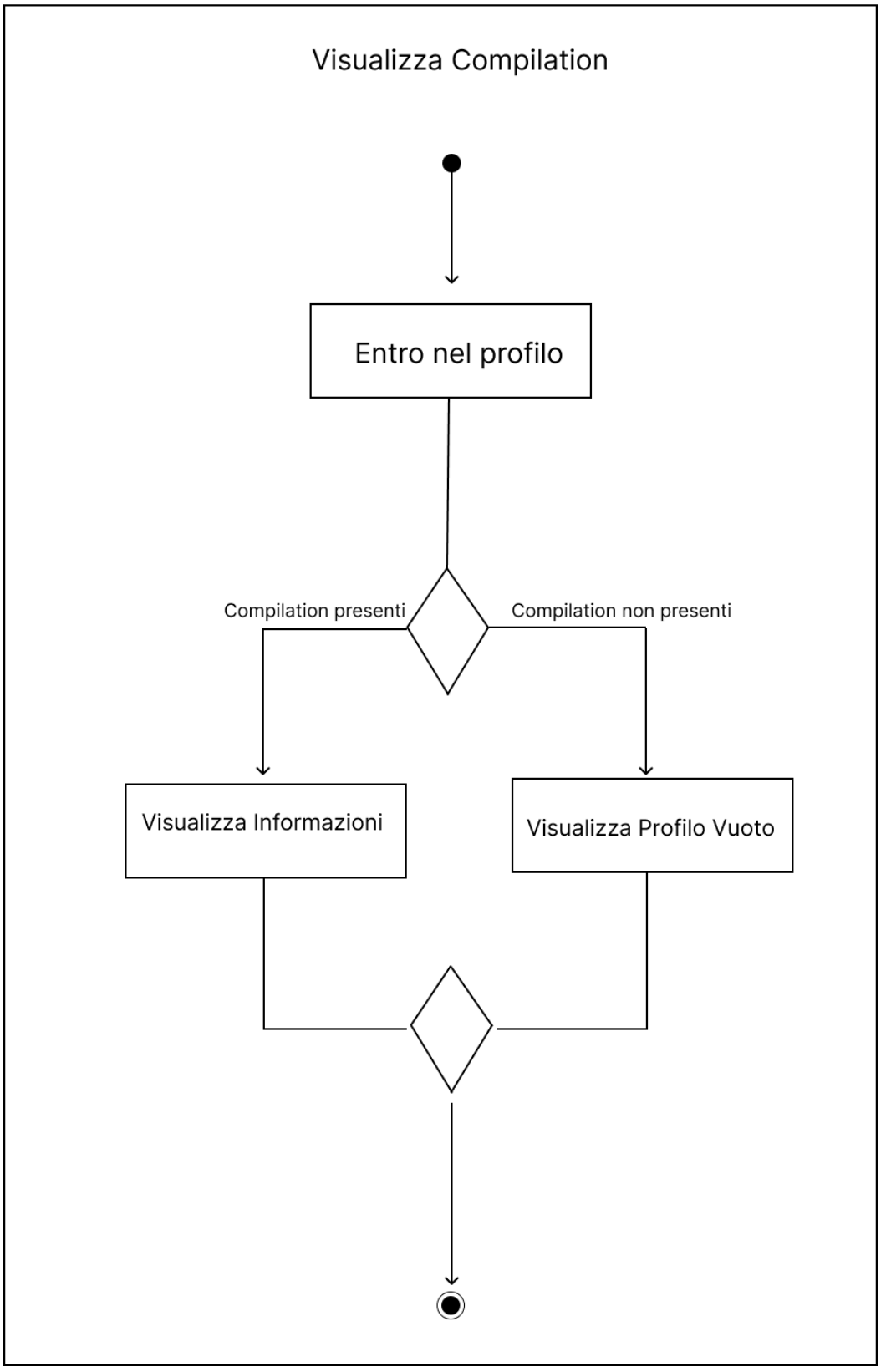
3.2.2.4 Inserire recensione



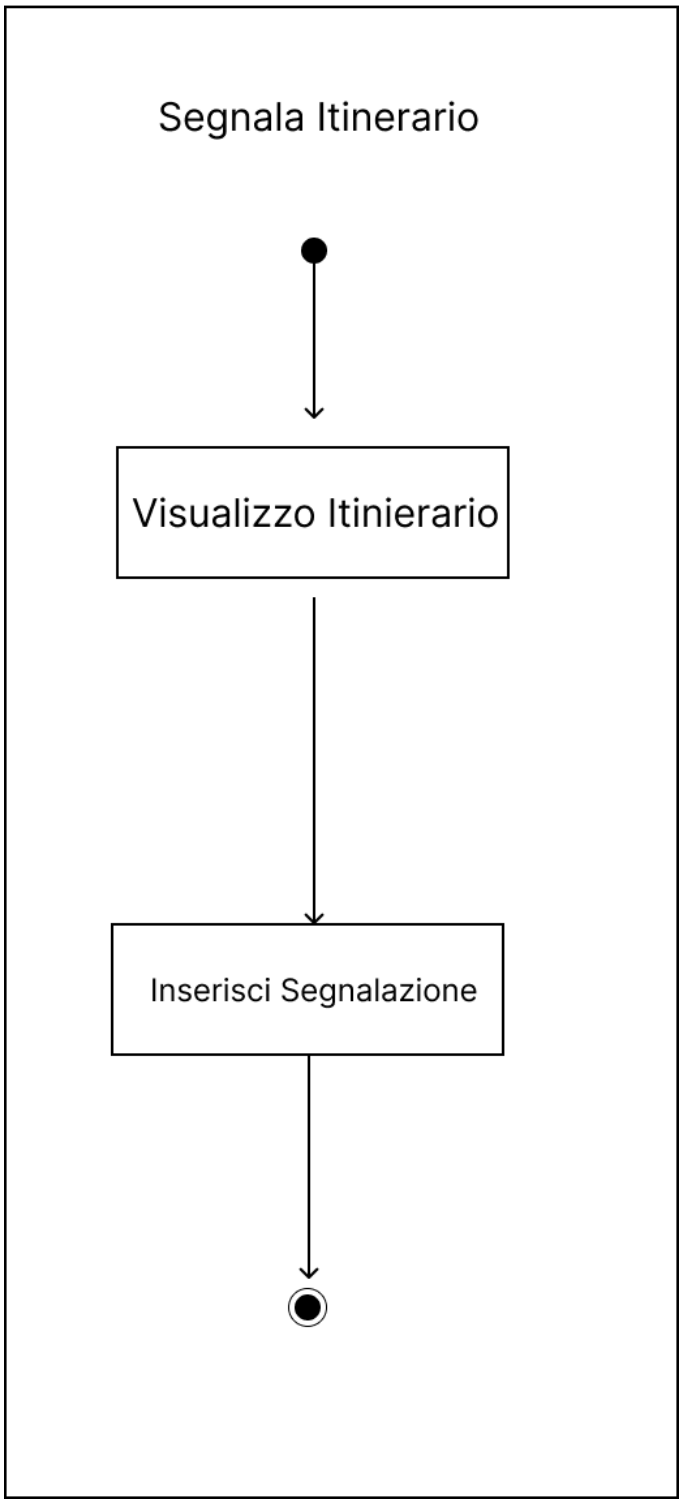
3.2.2.5 aggiungere itinerario alla compilation



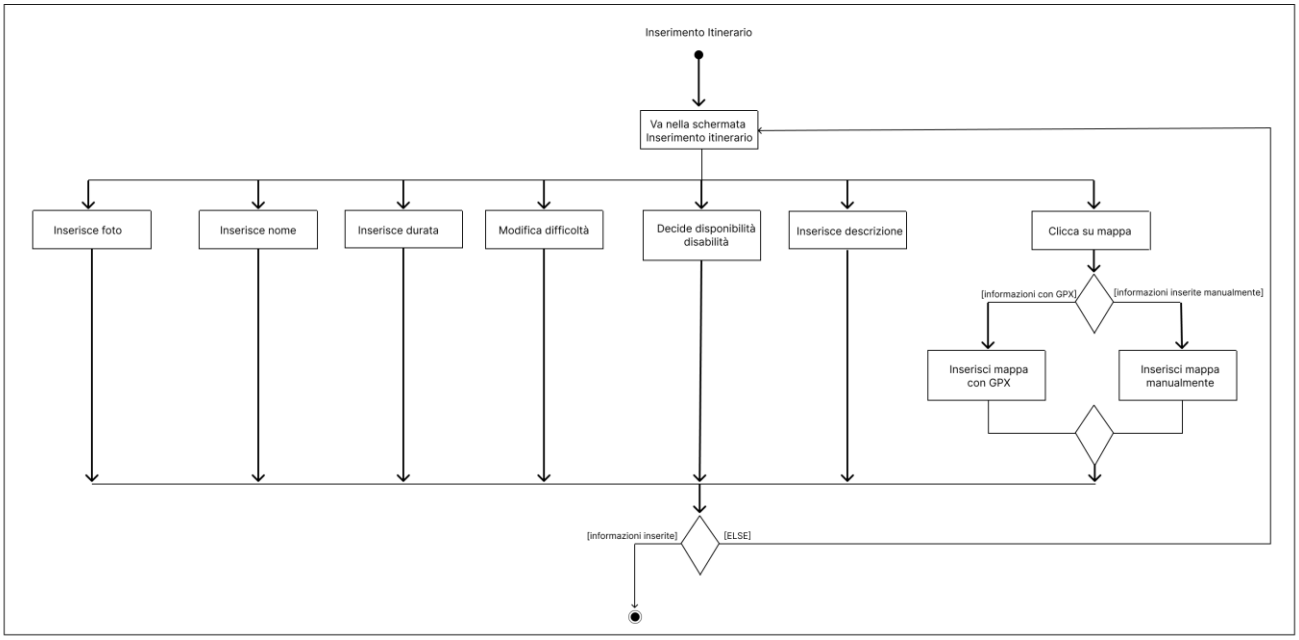
3.2.2.6 Visualizzare le compilation



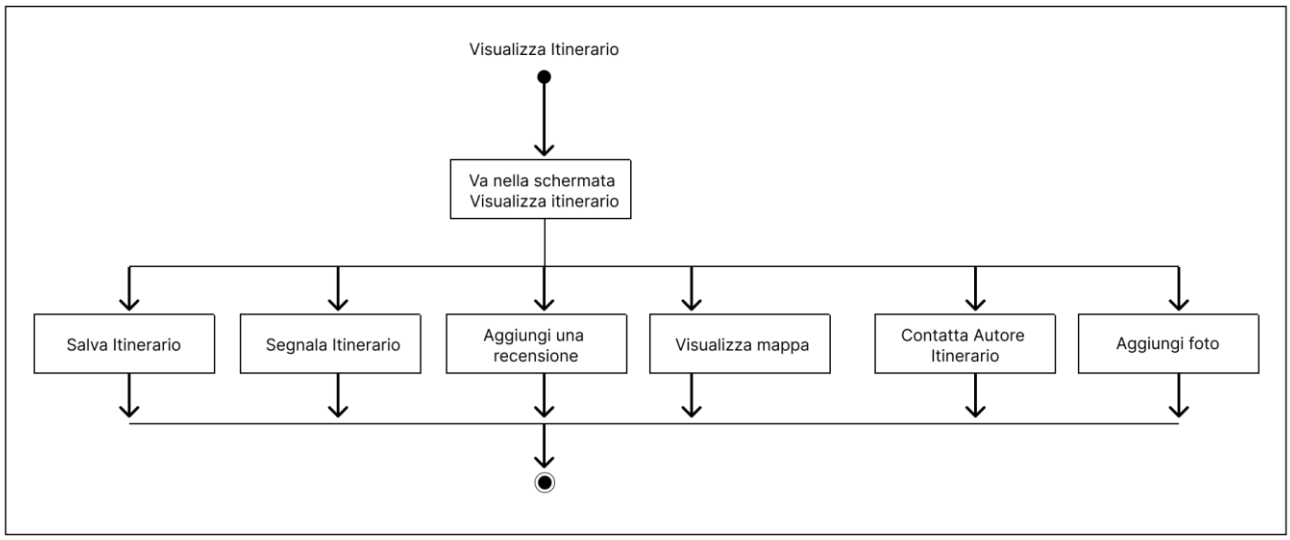
3.2.2.7 Segnalare itinerario



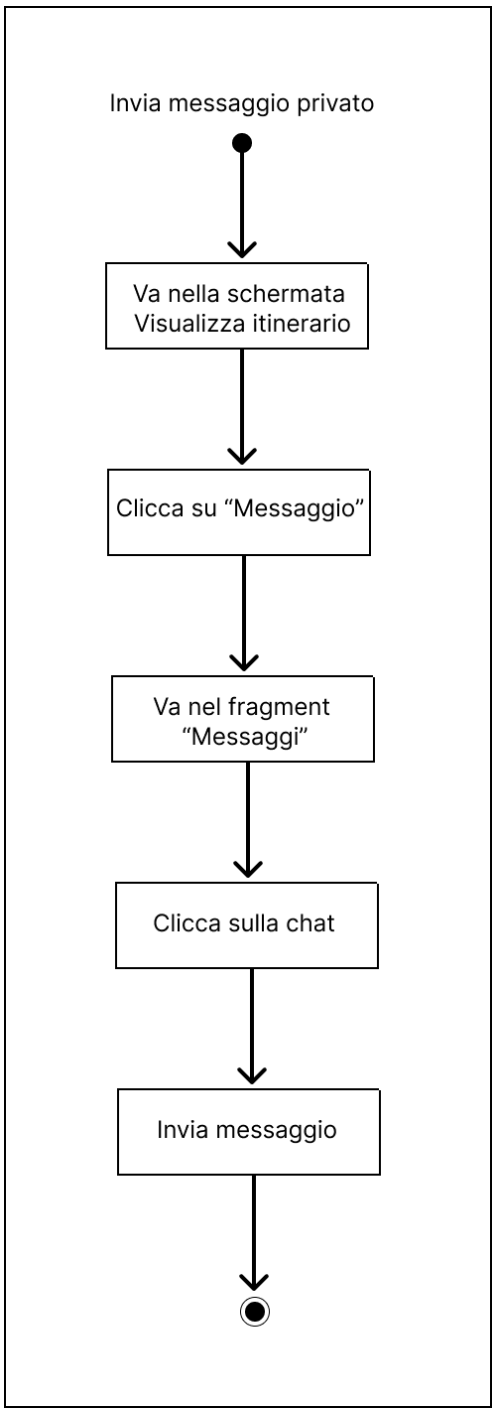
3.2.2.8 inserimento itinerario



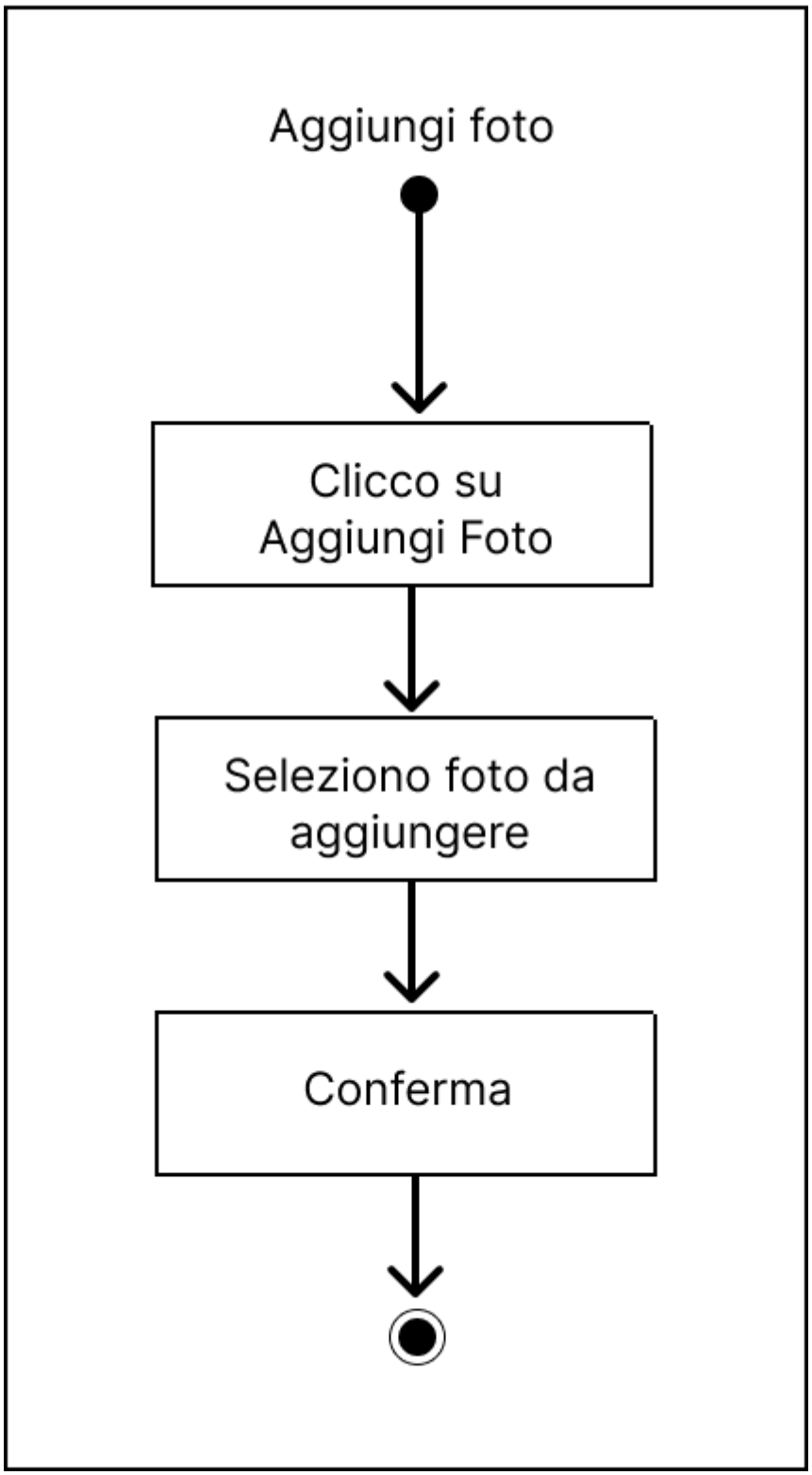
3.2.2.9 Visualizzare itinerario



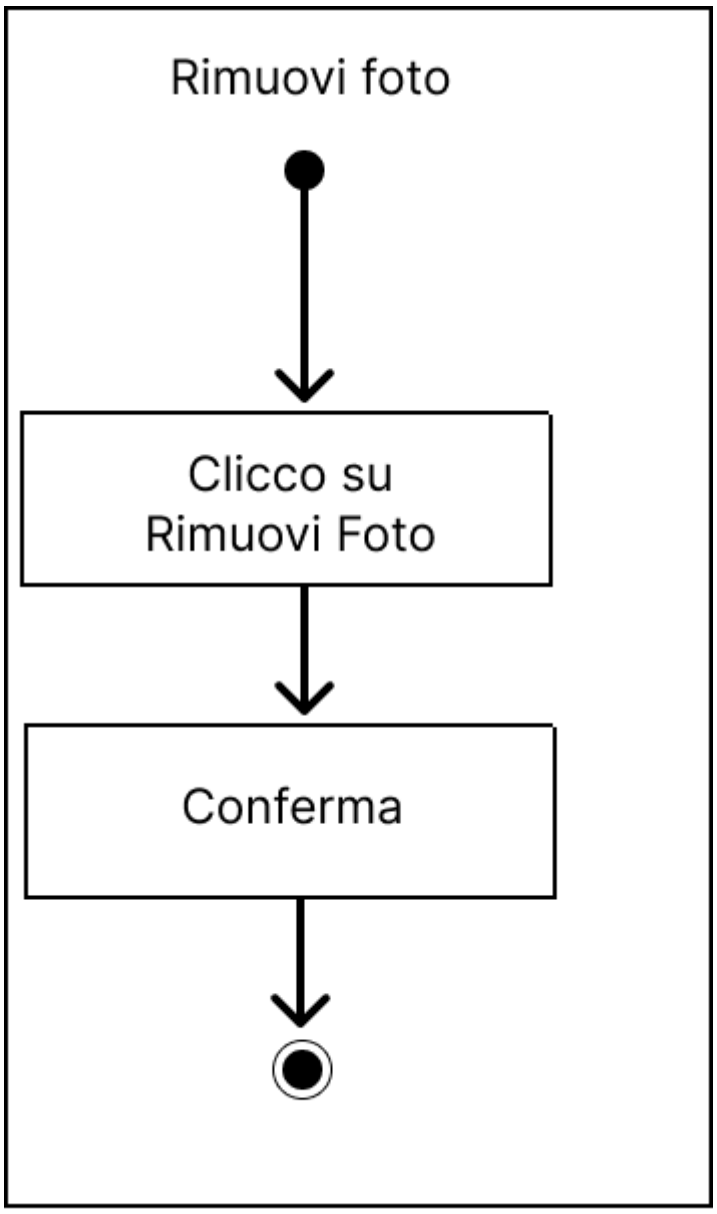
3.2.2.10 inviare messaggi privati



3.2.2.11 Aggiungere foto

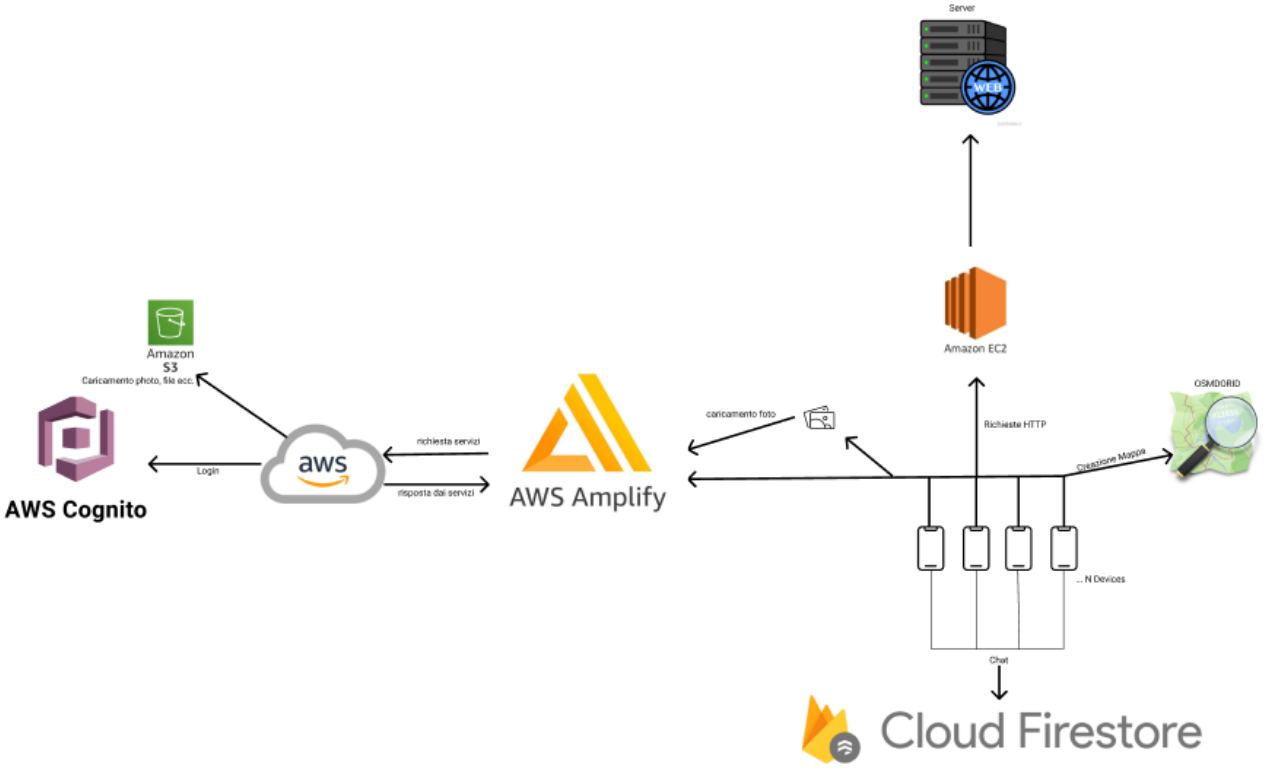


3.2.2.12 rimuovere foto



4. Design del sistema

4.1 Analisi dell'architettura e criteri di design



Il sistema sviluppato lavora principalmente su servizi CLOUD e si interfaccia al Server che risiede sulla macchina virtuale offerta da EC2.

Per la comunicazione nell'applicazione tra utenti tramite messaggi privati, si è scelto il servizio di firestore, che semplicemente permette di organizzare i messaggi inviati in modo generalizzato e in tempo reale.

Mentre per la visualizzazione della mappa e l'interazione con essa, si è usata la mappa Opensource di omsdroid, con essa è possibile disegnarci sopra con i marker, oppure inserire già un percorso esistente tramite file GPX, esso disegna il percorso dato dai punti inseriti dall'utente.

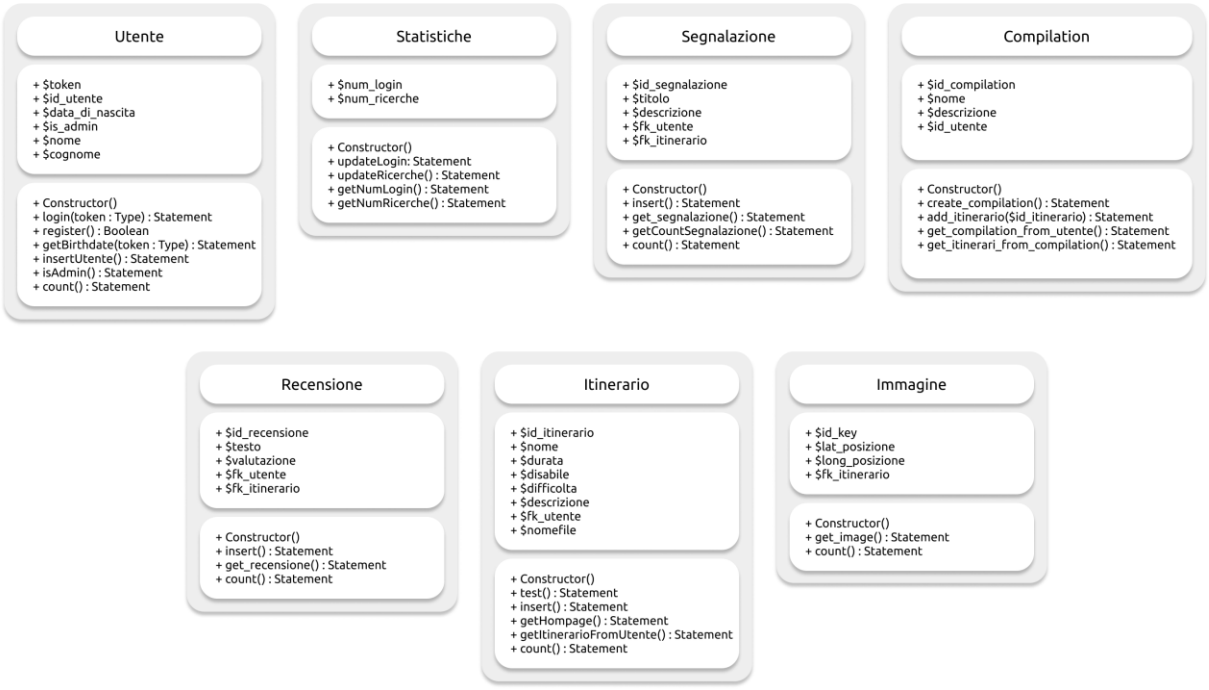
Il caricamento delle foto avviene prendendo la foto dal dispositivo, si controlla se quella foto abbia metadati riguardo la posizione, in caso li abbia, la foto viene caricata sulla mappa rappresentata da un MarkerPhoto.

Per velocizzare il caricamento delle foto abbiamo scelto il servizio che ci offre Amazon chiamato S3 Bucket.

Il Server su EC2, è organizzato in modo tale da poter ricevere richieste HTTP e inviare le risposta JSON al client. Il nostro server è basato su Apache, con un database gestito tramite PostgreSQL.

Il client invece si interfaccia al server tramite la libreria chiamata Volley che consente chiamate asincrone al Server.
Il client effettua chiamate POST quando deve inserire dati, inserendo i parametri (params) dentro il body.
Mentre quando vuole effettuare delle informazioni, invia richieste GET.

Il nostro server mette a disposizione varie API tra le quali:



Ogni API è rappresentata da un file.php (dove risiede la logica del metodo), che invia una risposta JSON al client. (TUTTO il codice del Back-end è inserito nel zip).

4.2 Class diagram di design

I diagrammi delle classi forniti di seguito sono il prodotto finale di vari cicli di sviluppo. I seguenti diagrammi sono divisi per pacchetti di codice. Una piccola spiegazione dei simboli di seguito: rispetto alla vecchia notazione UML ci sono delle differenze, le classi sono indicate con "C" prima del nome, le interfacce con "I", per la visibilità vengono usati simboli particolari come il quadrato rosso per il privato, il pallino verde per il pubblico, triangolo blu per il default e il rombo giallo per il protected. Questi ultimi sono vuoti se si tratta di attributi e pieni se trattano un metodo.

4.3.1 Class diagram "View"

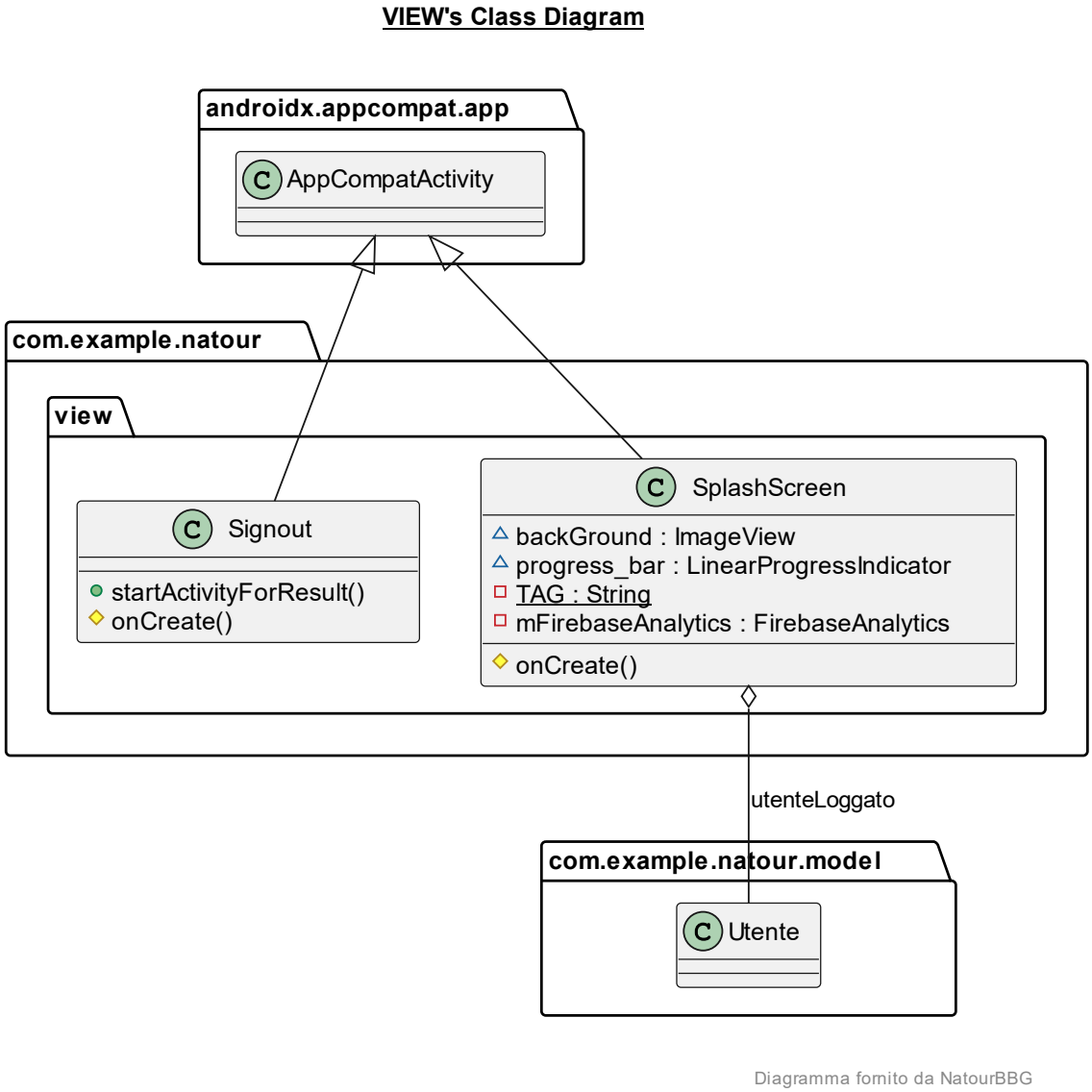


Diagramma fornito da NatourBBG

Il seguente class diagram descrive delle classi usate per l'accesso e l'uscita dal sistema, in particolare sono attività isolate che non hanno necessità di creare un pacchetto a sé in quanto non hanno bisogno di altre classi di supporto.

4.3.2 Class diagram “LoginActivity”

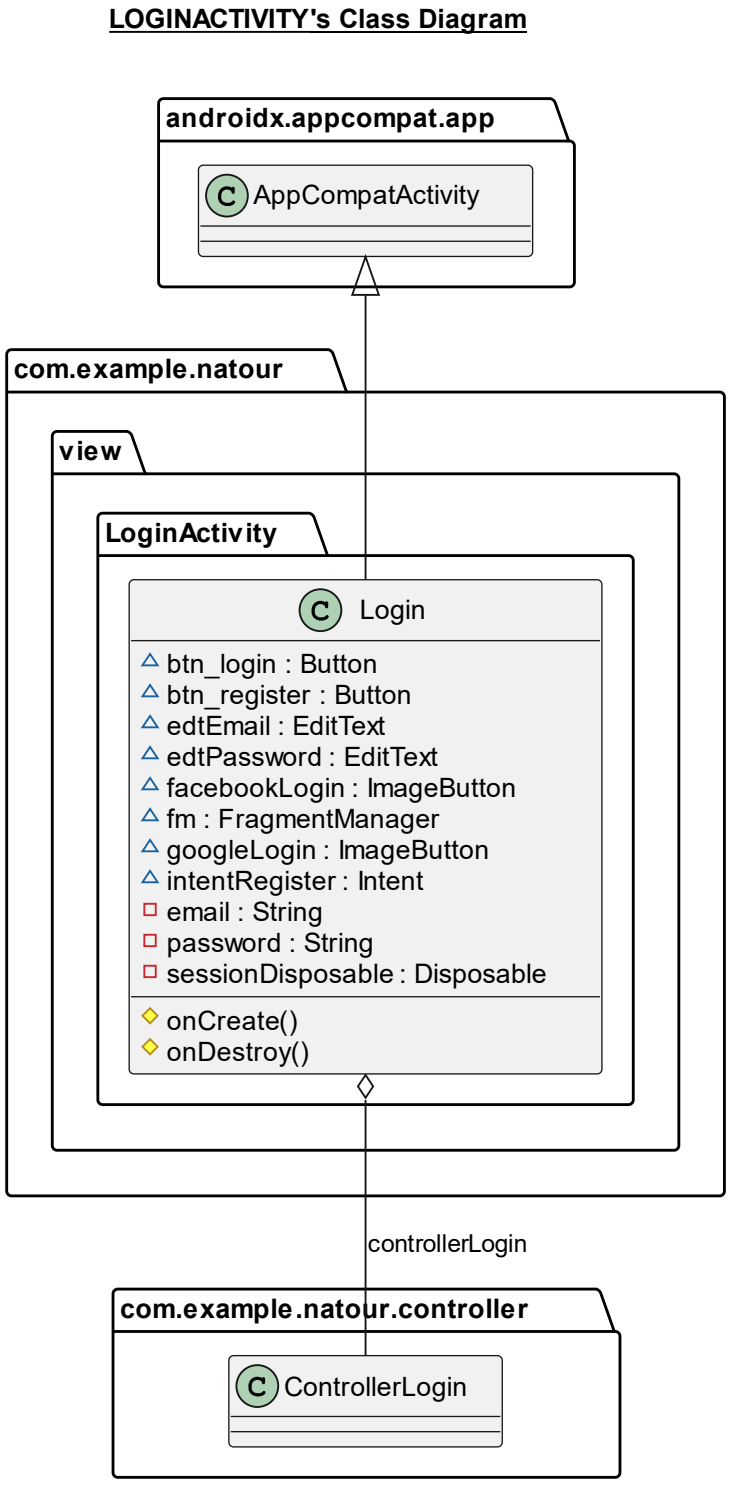
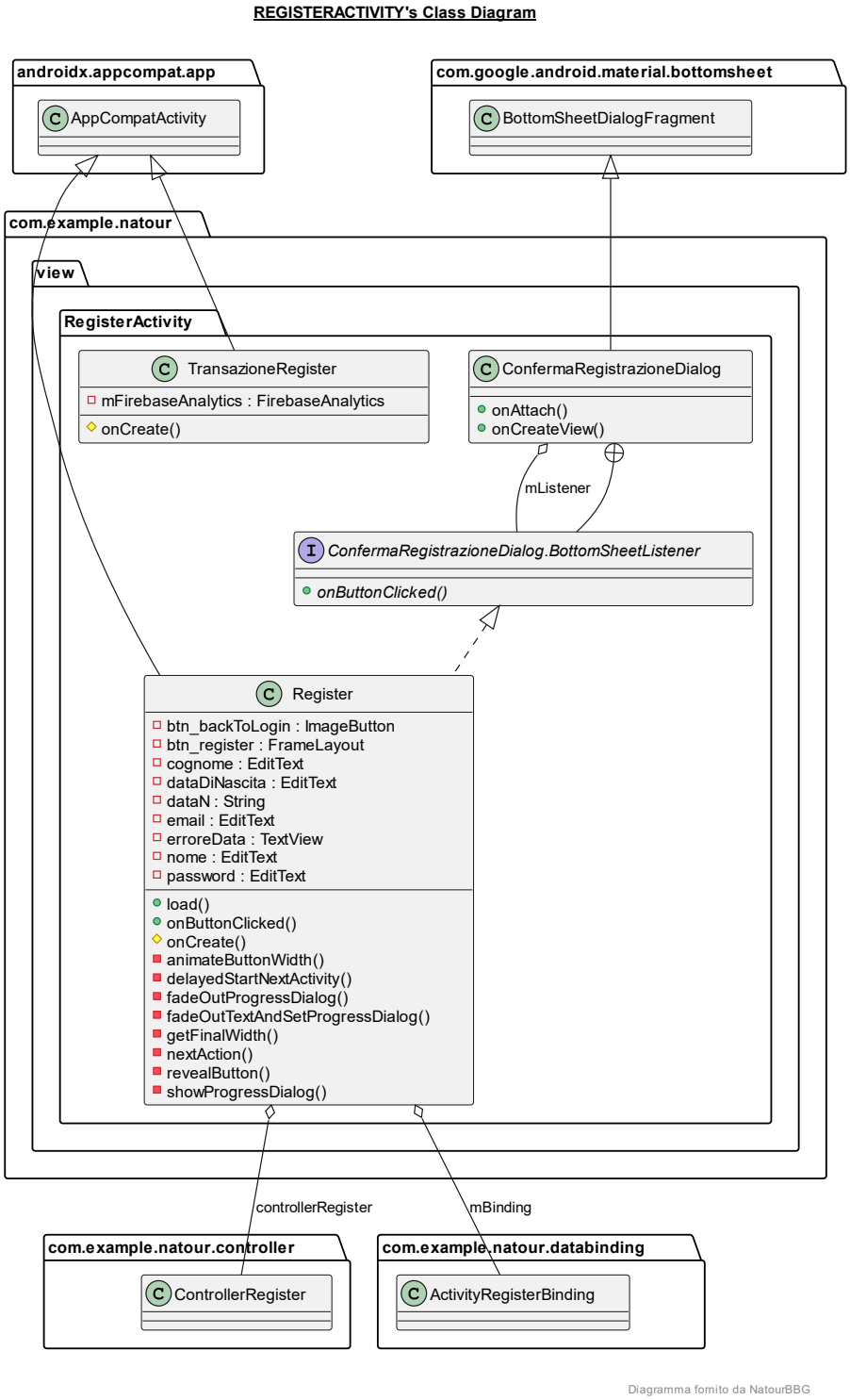
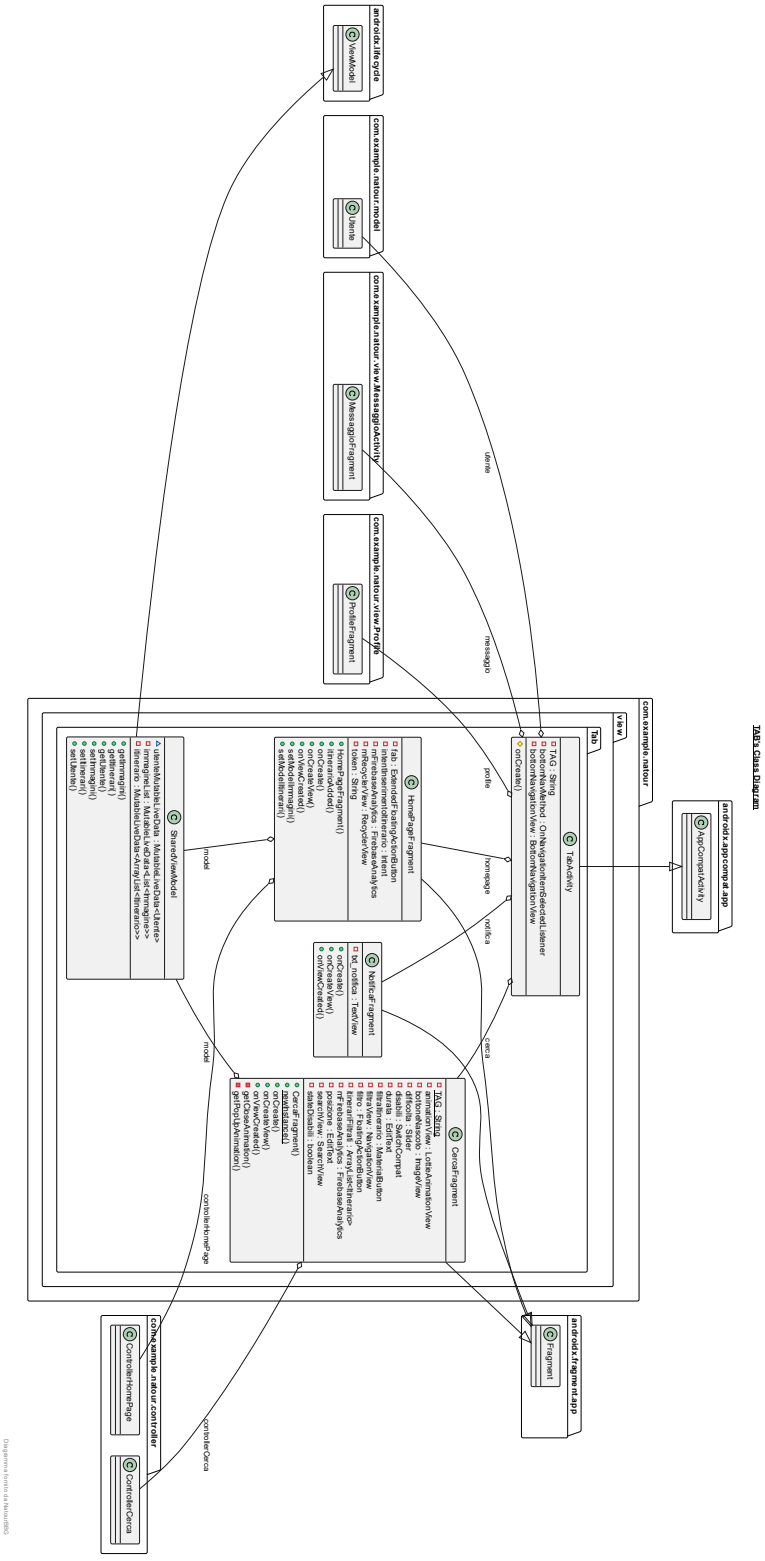


Diagramma fornito da NatourBBG

4.3.3 Class diagram “RegisterActivity”

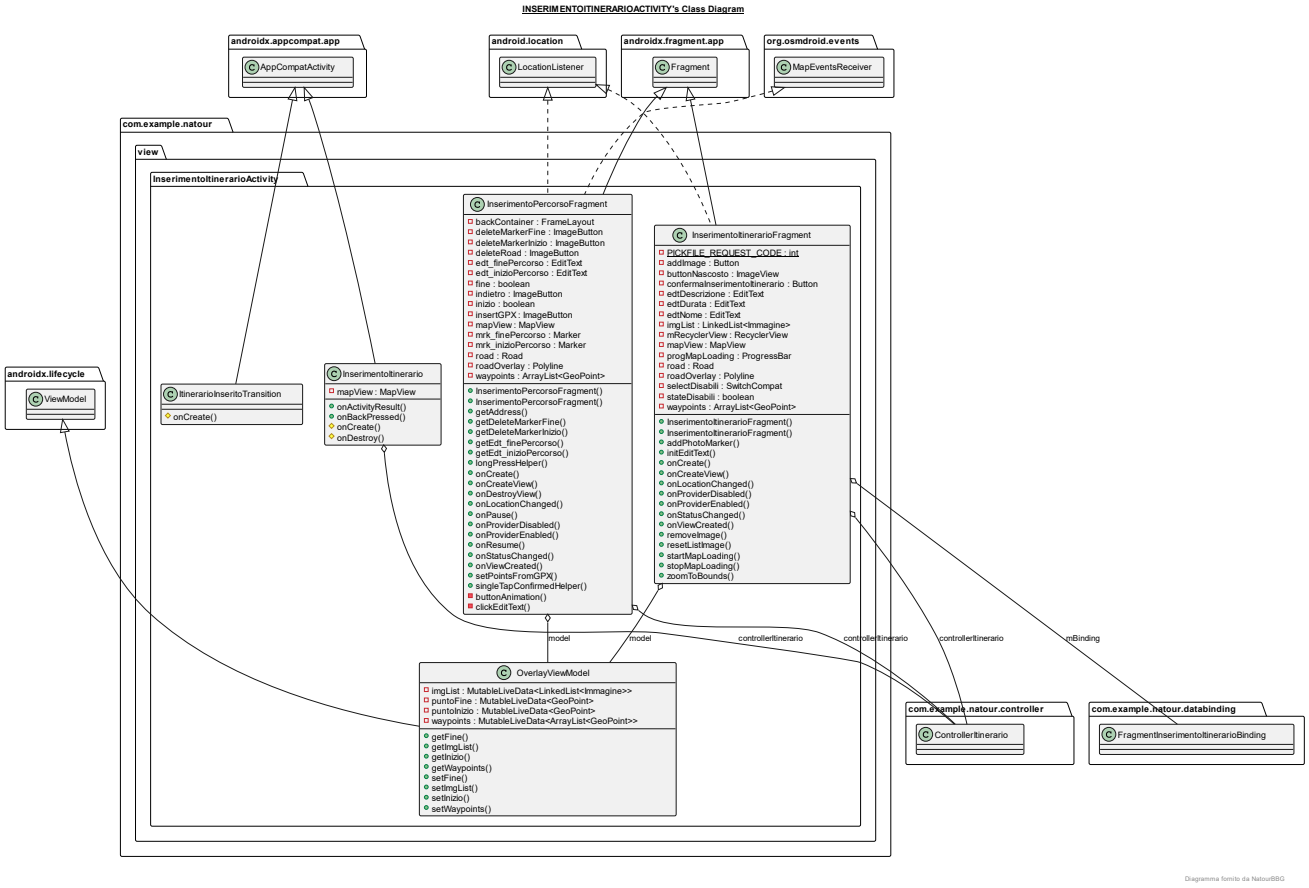


4.3.4 Class diagram “Tab”

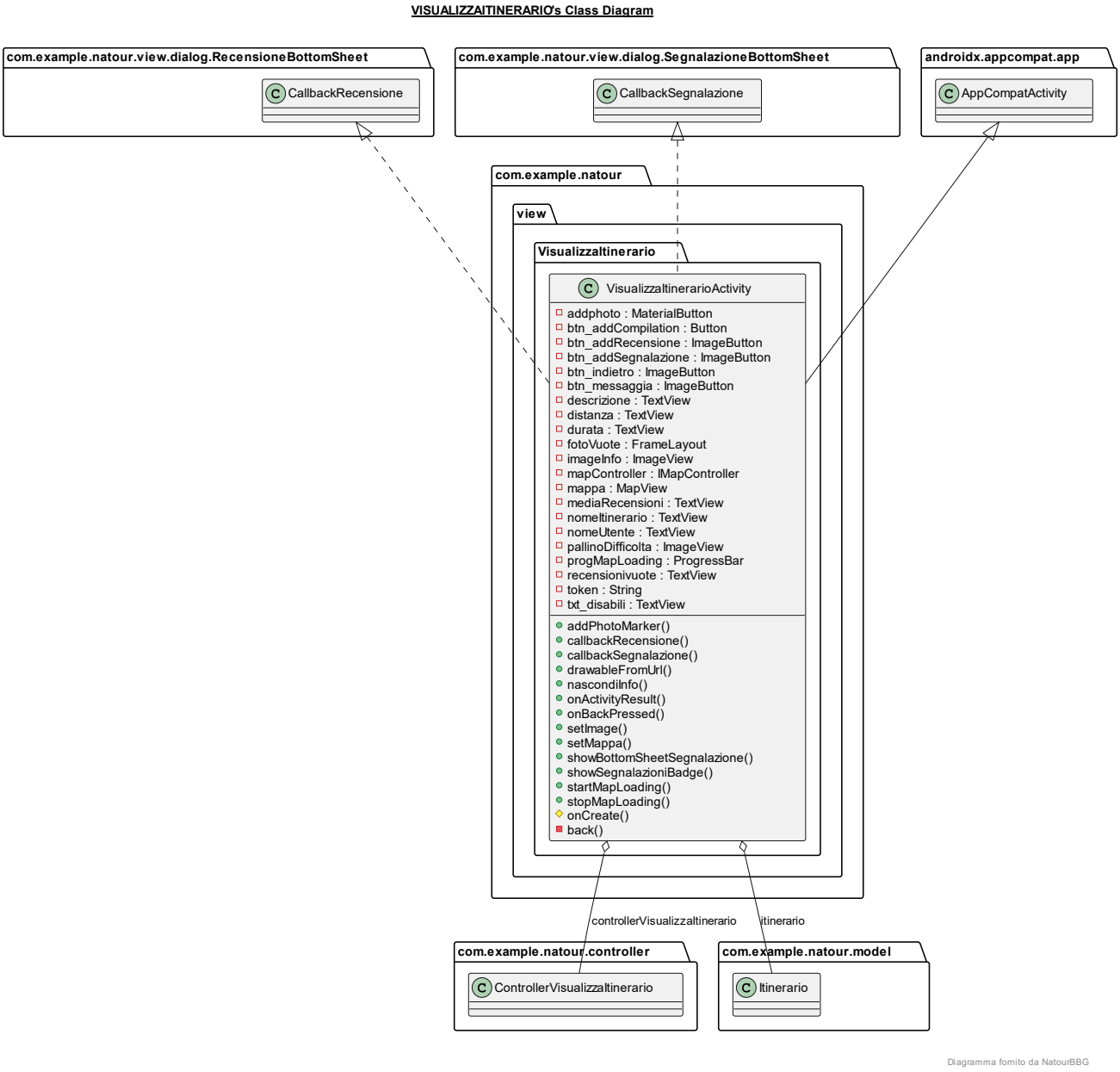


questo diagramma racchiude le attività e fragment che occupano le funzionalità dell’app.

4.3.5 Class diagram “InserimentoItinerarioActivity”



4.3.6 Class diagram “VisualizzaItinerario”



4.3.7 Class diagram “Profile”

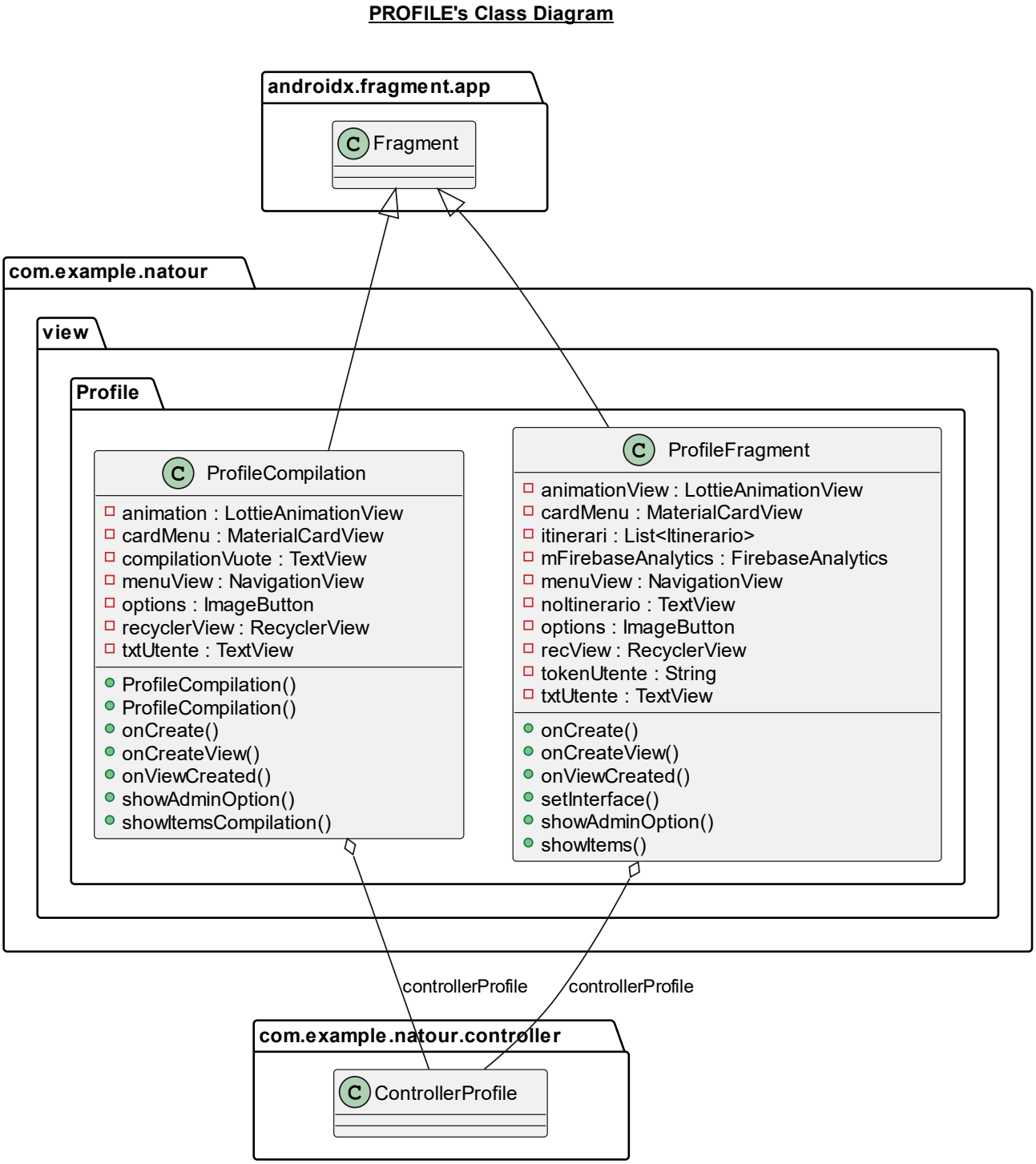


Diagramma fornito da NatourBBG

4.3.8 Class diagram “PannelloAdmin”

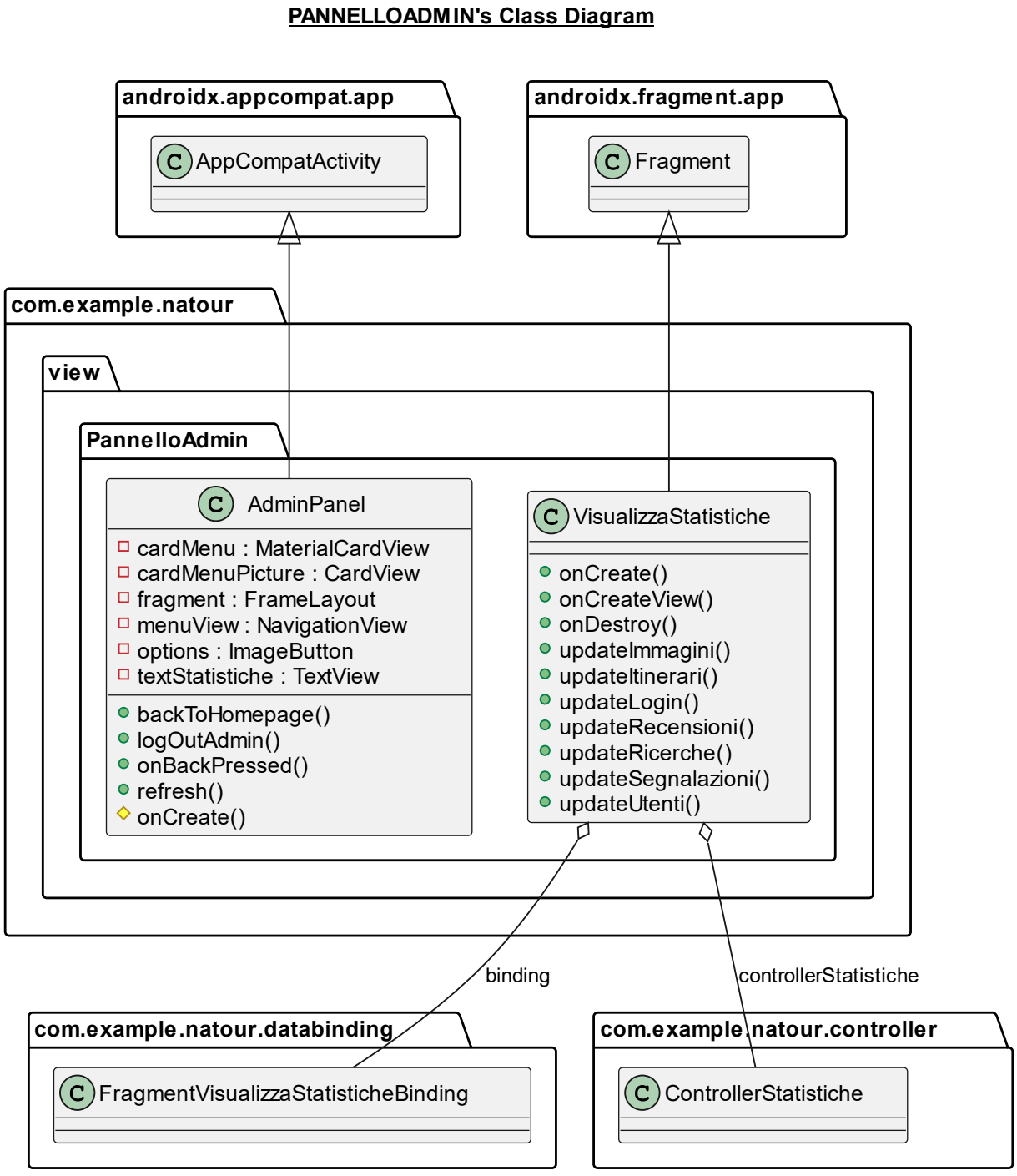
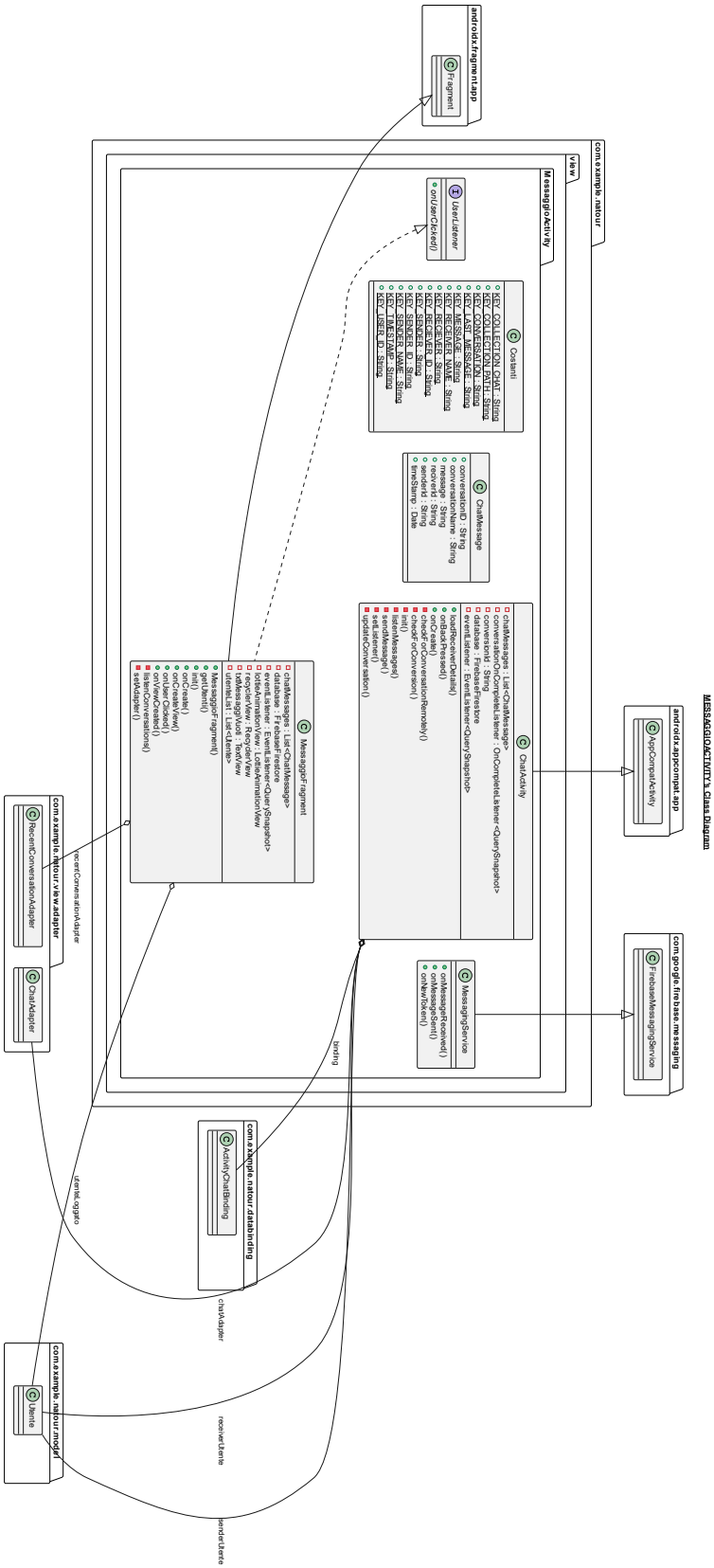


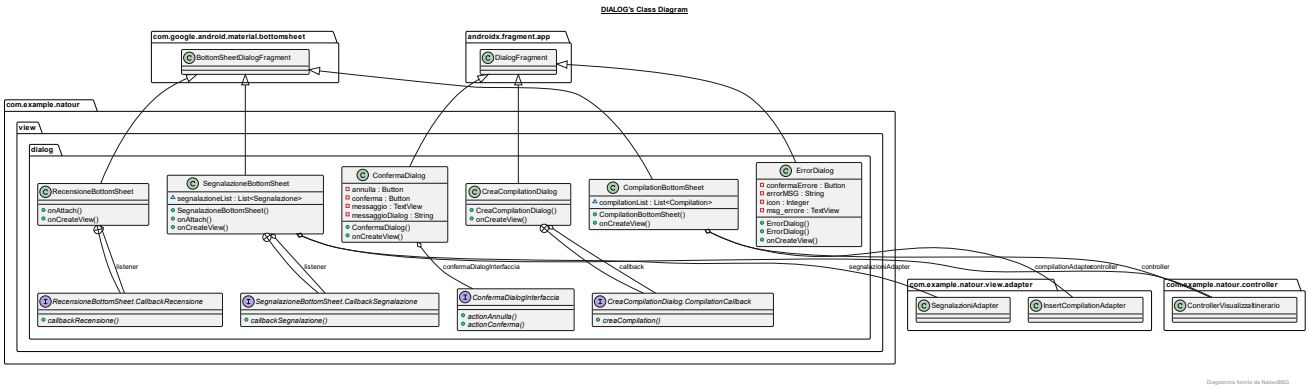
Diagramma fornito da NatourBBG

4.3.9 Class diagram “MessaggioActivity”



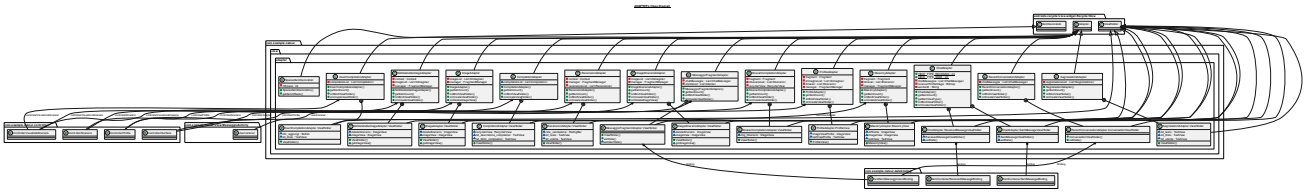
4.3.10 Class diagram “Dialog”

Il diagramma seguente contiene le classi delle dialog, ovvero finestre che appaiono al di sopra della schermata dell’utente. Esse possono avere diversi stili, ad esempio BottomSheet mostra questa finestra come interfaccia che si apre dal basso come un foglio di carta.



4.3.11 Class diagram “Adapter”

Il diagramma seguente contiene classi Adapter, ovvero delle classi che in android studio producono degli oggetti che verranno mostrati all’interno di container nelle attività. Gli oggetti che mostrano dipendono da una lista da cui recuperano le informazioni.



4.3.12 Class diagram “Util”

UTIL's Class Diagram

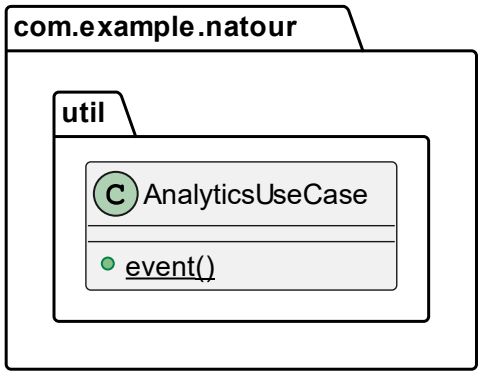
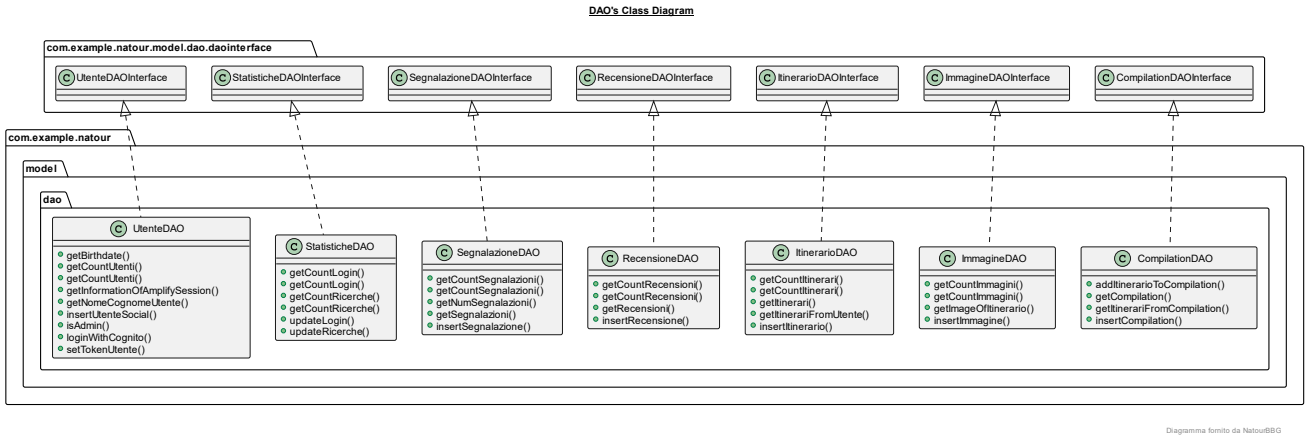


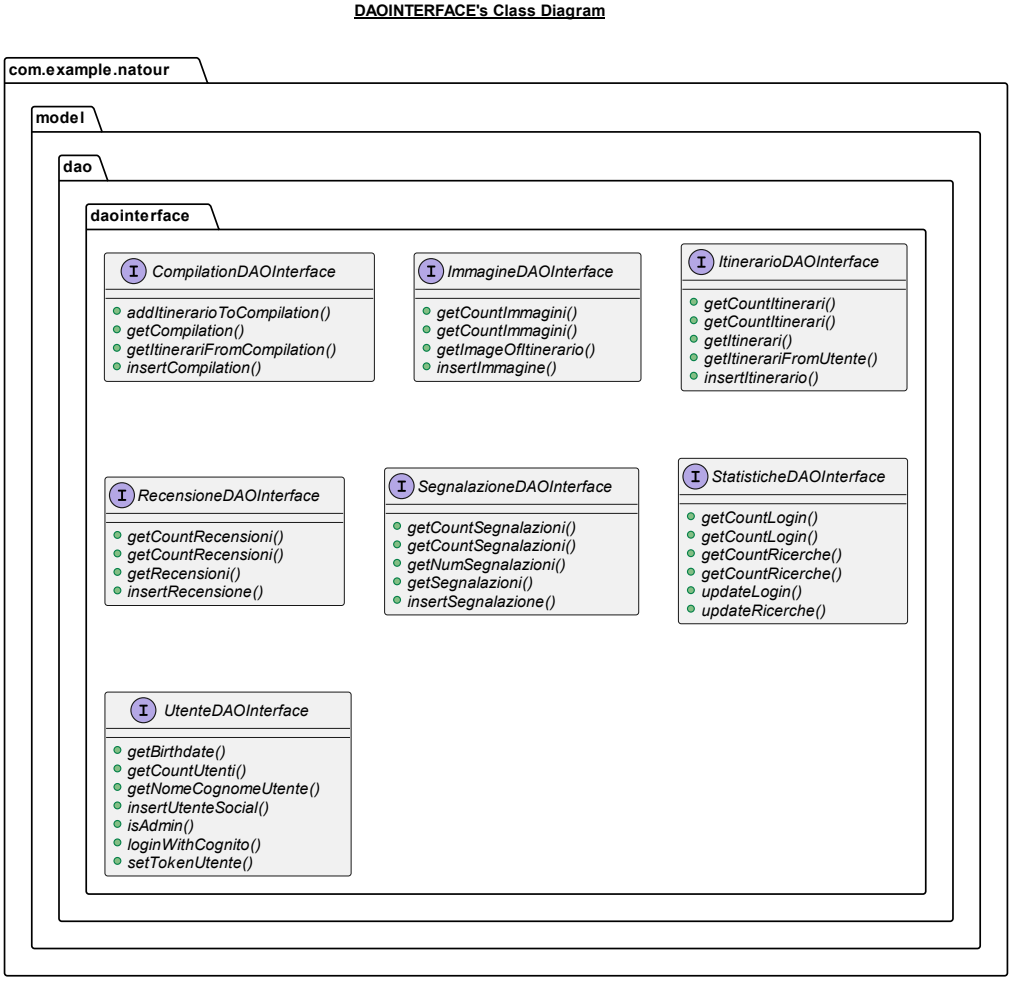
Diagramma fornito da NatourBBG

Questa classe serve per aggiungere gli eventi utili per il logging.

4.3.13 Class diagram “DAO”



4.3.14 Class diagram “DAOinterface”



4.3.15 Class diagram “Connection”

CONNECTION's Class Diagram

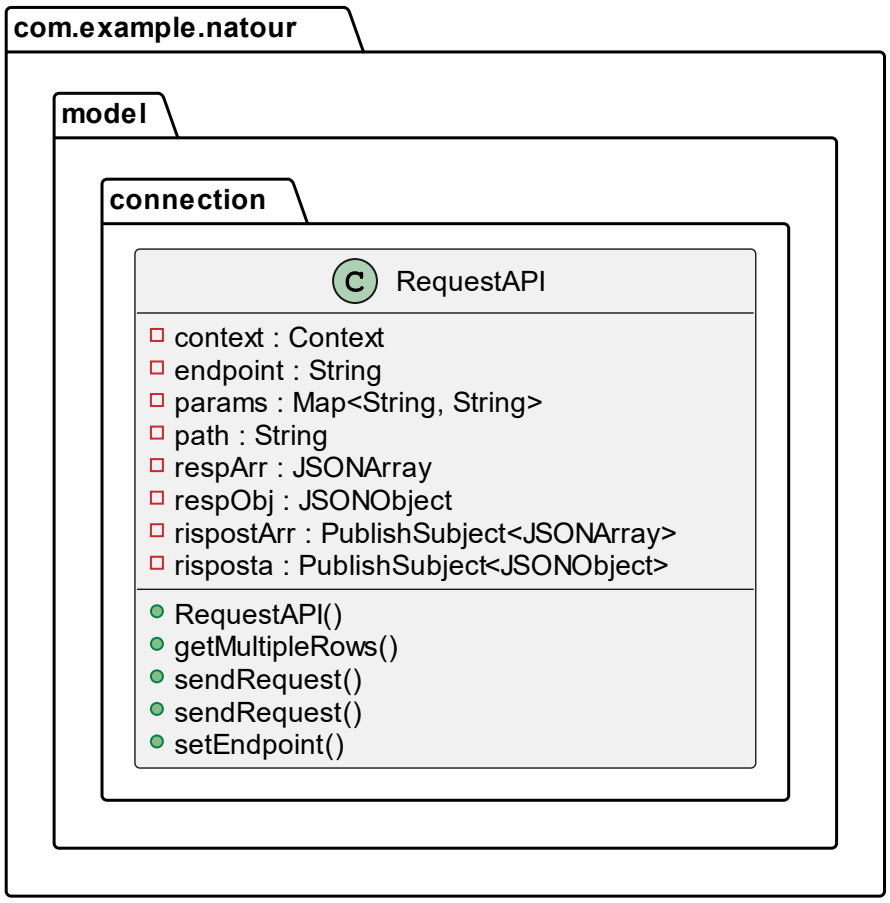
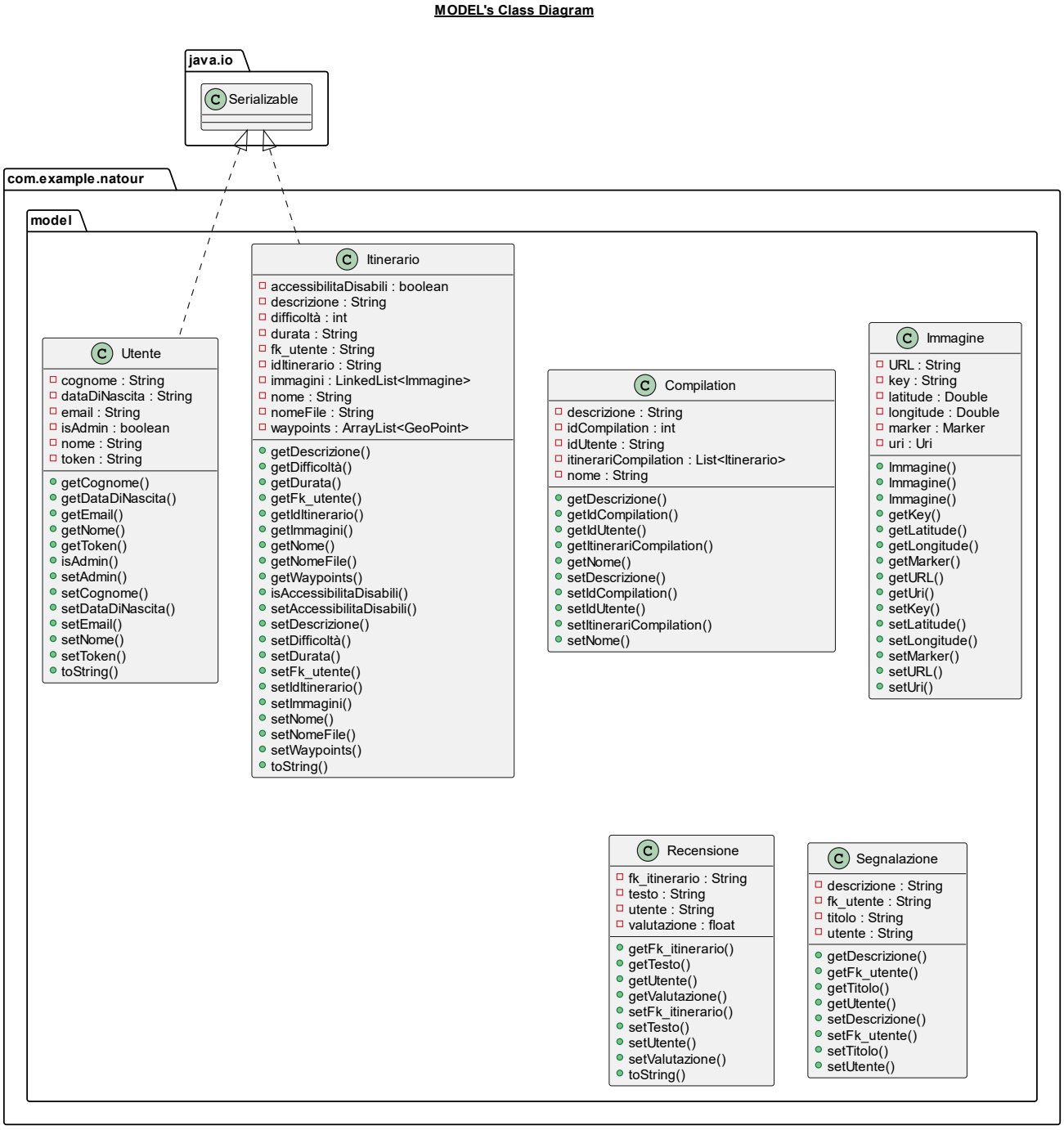


Diagramma fornito da NatourBBG

Questa classe serve per eseguire delle richieste HTTP

4.3.16 Class diagram “Model”



Questo diagramma contiene le classi del nostro dominio.

4.3.17 Class diagram “Exceptions”

Il diagramma seguente contiene eccezioni personalizzate per alcuni metodi.

EXCEPTION's Class Diagram

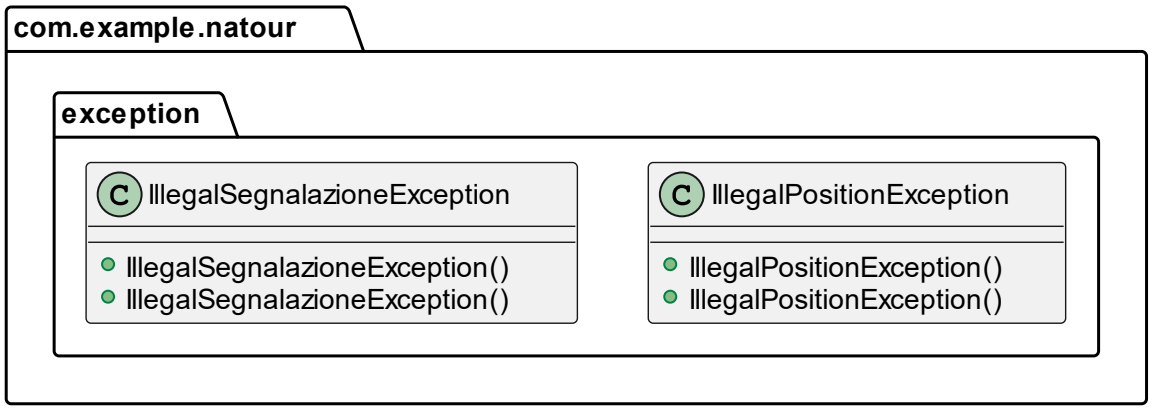
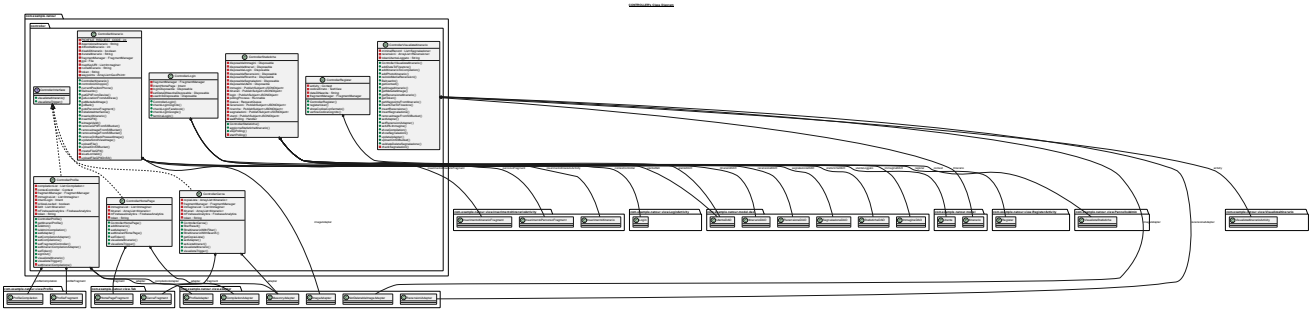


Diagramma fornito da NatourBBG

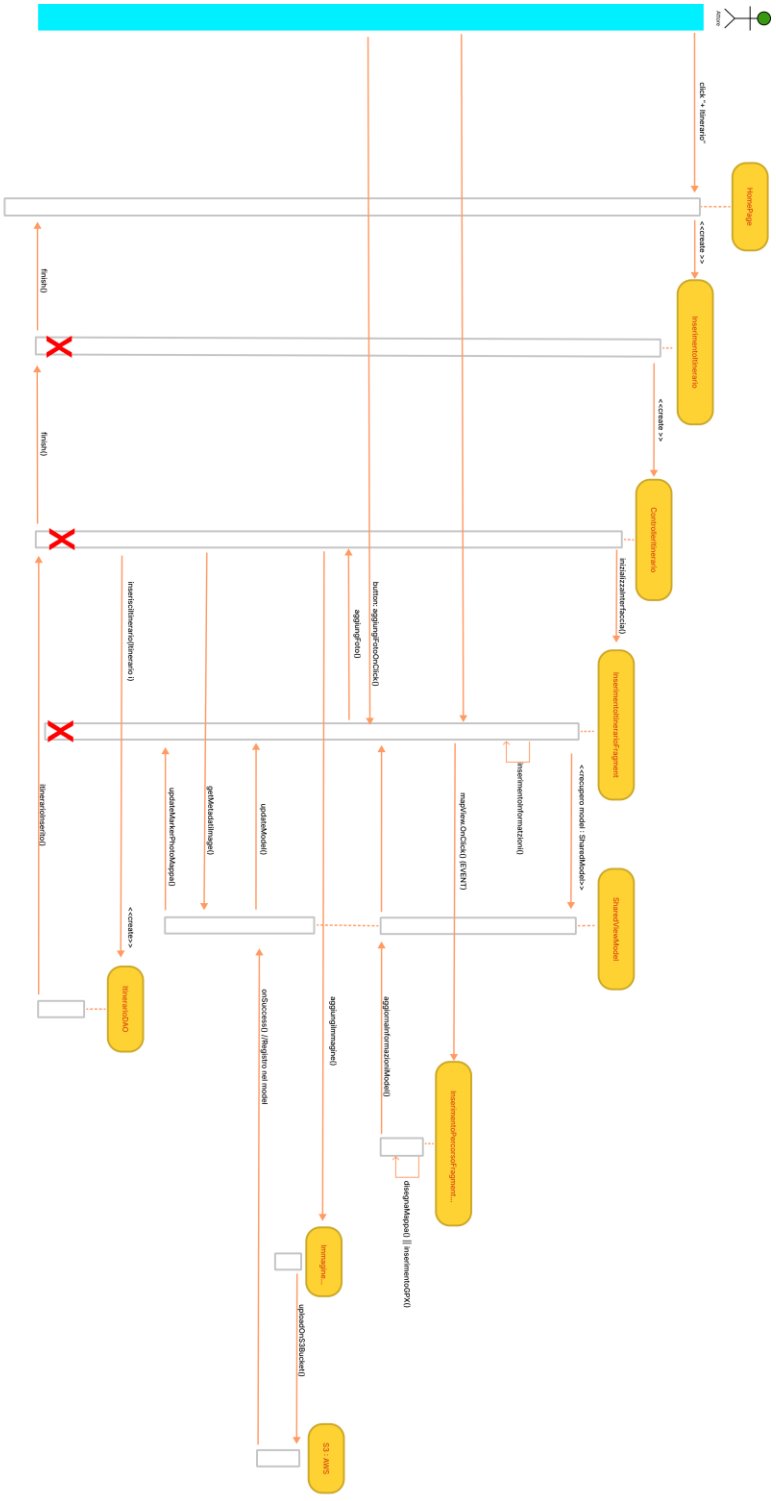
4.3.18 Class diagram “Controller”



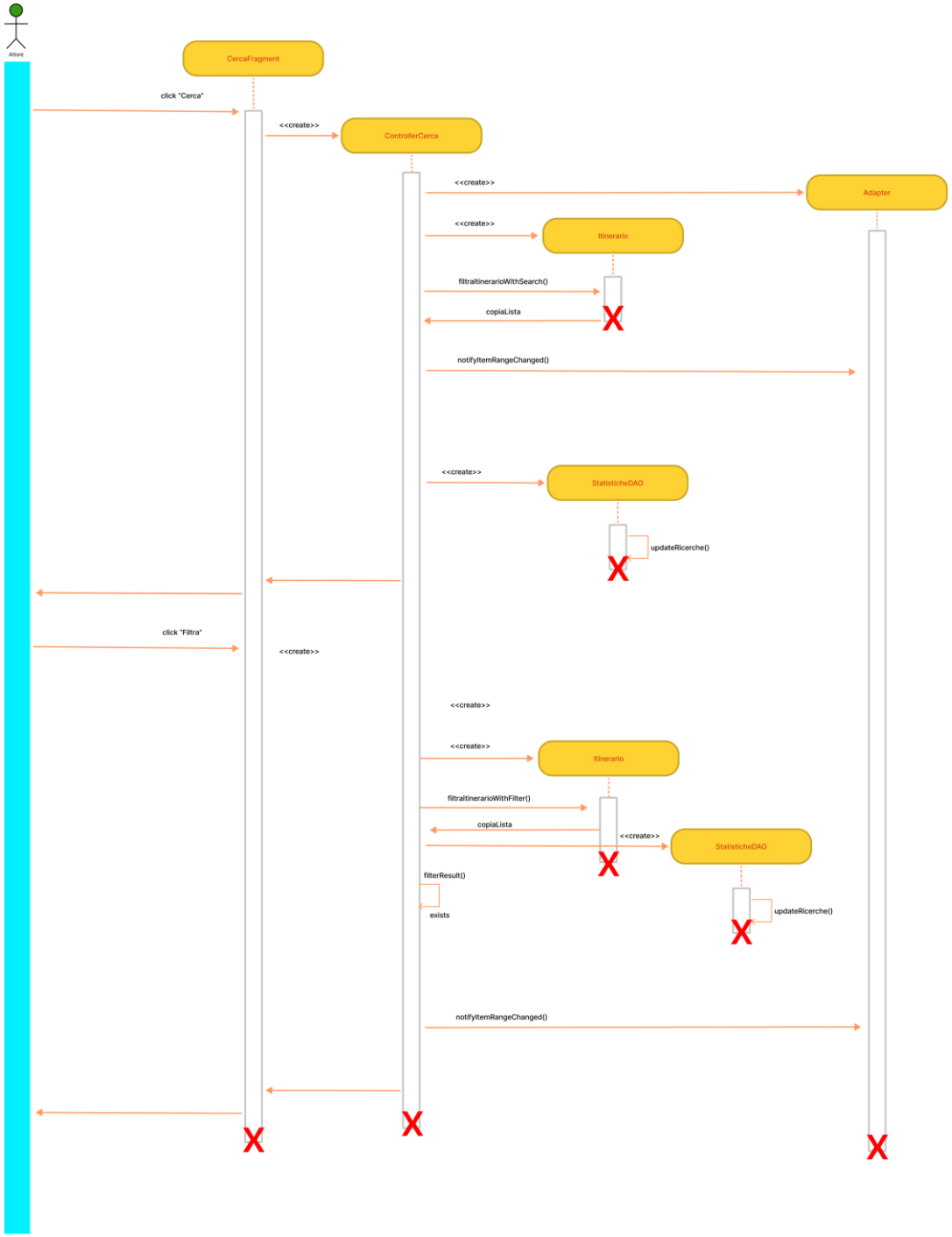
Questo diagramma contiene le classi che contengono la maggior parte della logica applicativa del nostro sistema.

4.3 Sequence Diagram di Design

4.3.1 Sequence Diagram “inserimento itinerario”



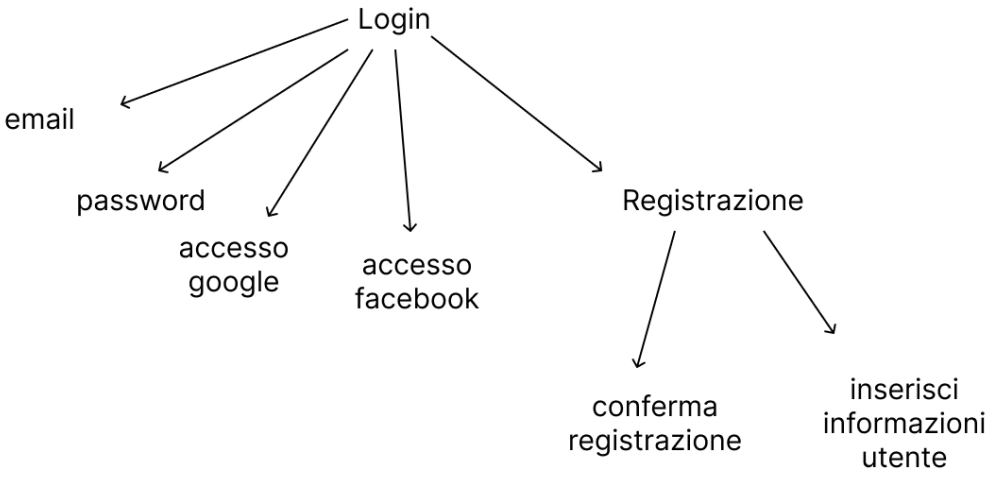
4.3.1 Sequence Diagram “Ricerca itinerario”



4.4 Definizione delle gerarchie funzionali

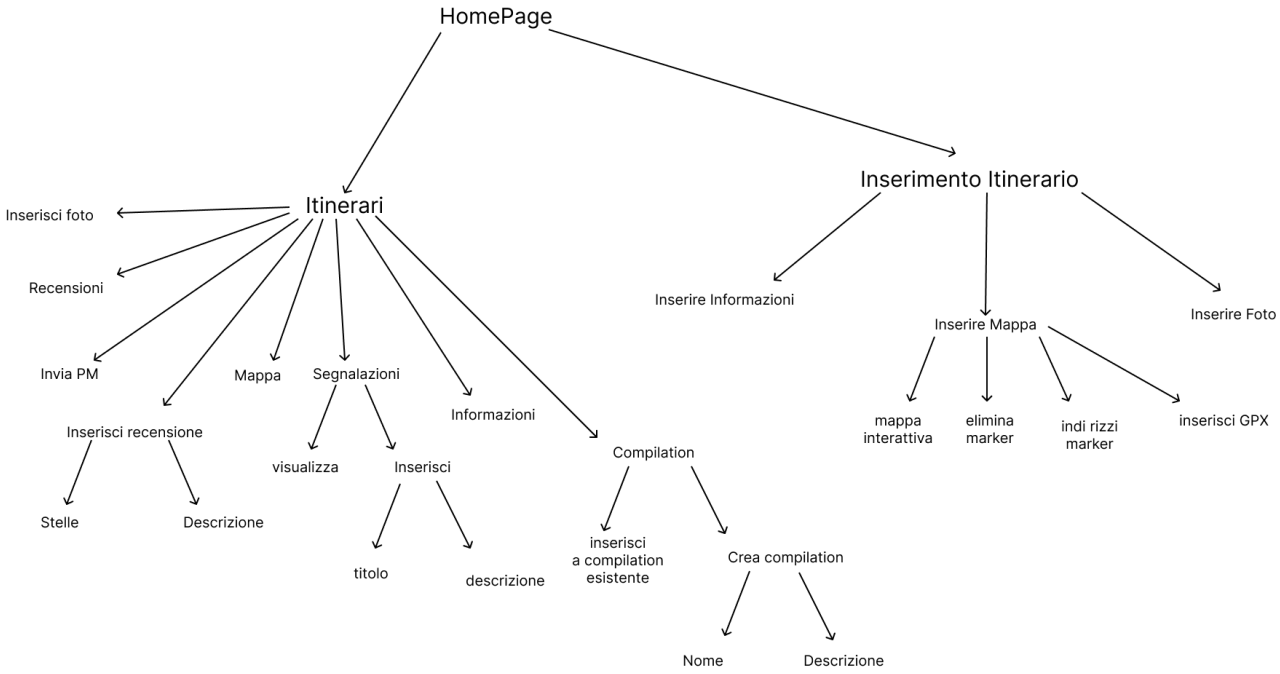
Le gerarchie funzionali dell'applicazione partono da quattro radici principali delle funzionalità dell'applicazione con in più il login. Si è deciso di separare le gerarchie in quanto rappresentano funzionalità che tra di loro non comunicano, immediatamente dopo "Login" si ha accesso a "Homepage"; tuttavia, il resto dell'applicazione sarà navigabile attraverso la bottom navigation bar.

4.4.1 Gerarchia funzionale per "Login"



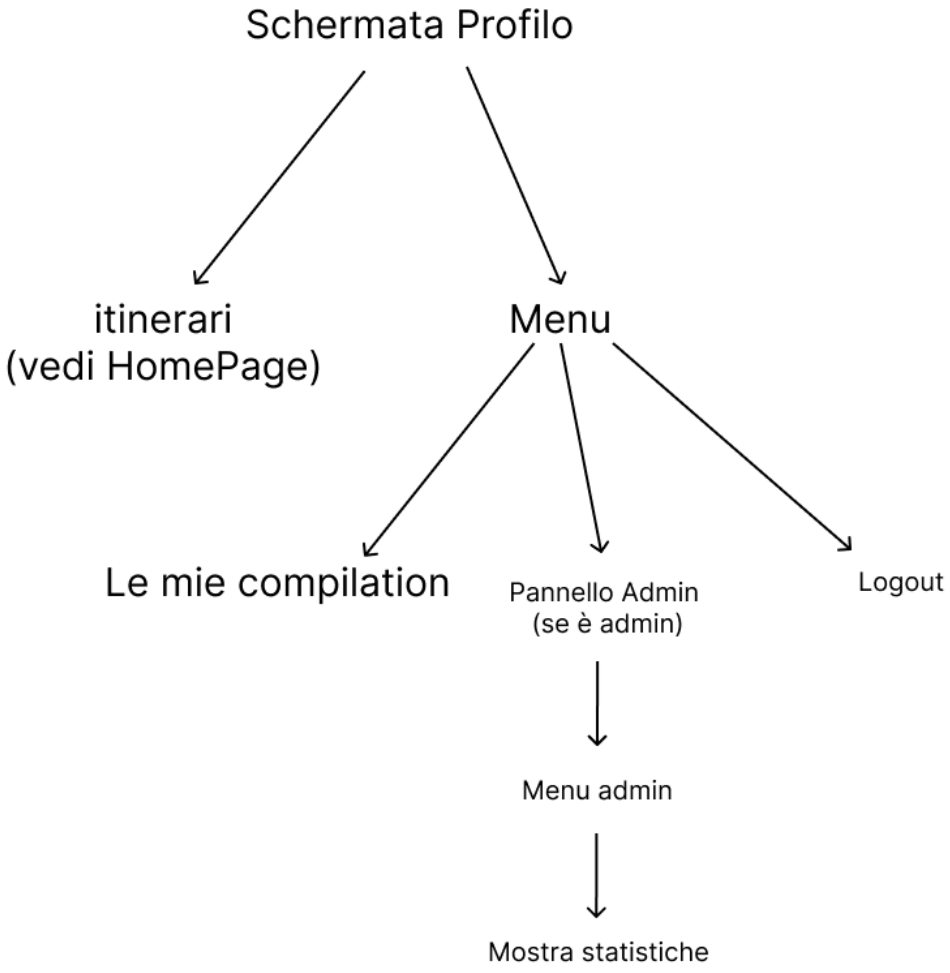
La foglia "inserisci informazione utente" racchiude tutte le affordance necessarie per raccogliere le informazioni dell'utente quali email, password, data di nascita, nome e cognome.

4.4.2 Gerarchia funzionale per “HomePage”



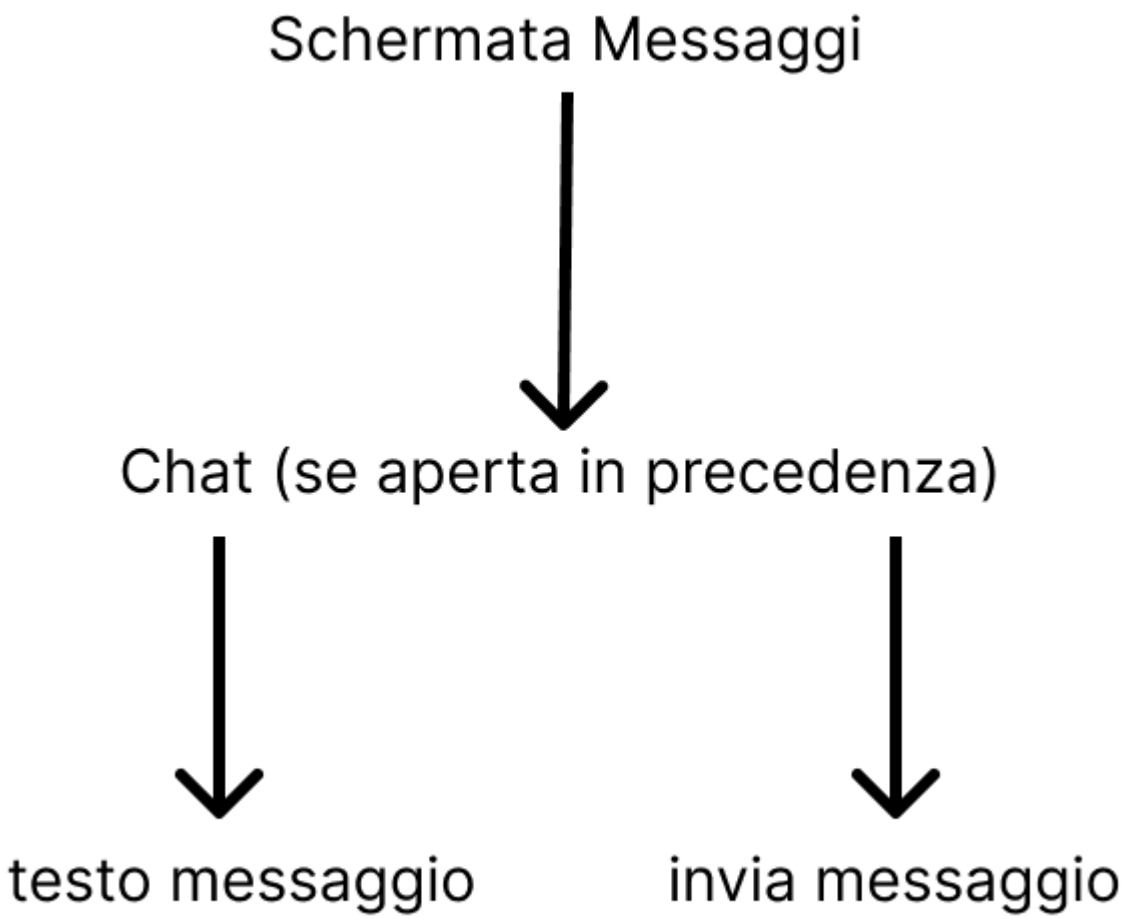
Anche in questo caso quello che rappresentano le foglie con la scritta “informazioni” intendono il gruppo di informazioni che possiede l’itinerario, nel caso di “Itinerari” vengono mostrate, nel caso di “Inserimento Itinerario” bisogna inserirle tramite apposite affordance. Il nodo Itinerari rappresenta i molteplici itinerari visualizzabili in “HomePage” se non sono presenti itinerari nel sistema allora non ci sarà alcun itinerario mostrato in “HomePage”

4.4.3 Gerarchia funzionale per “Schermata Profilo”



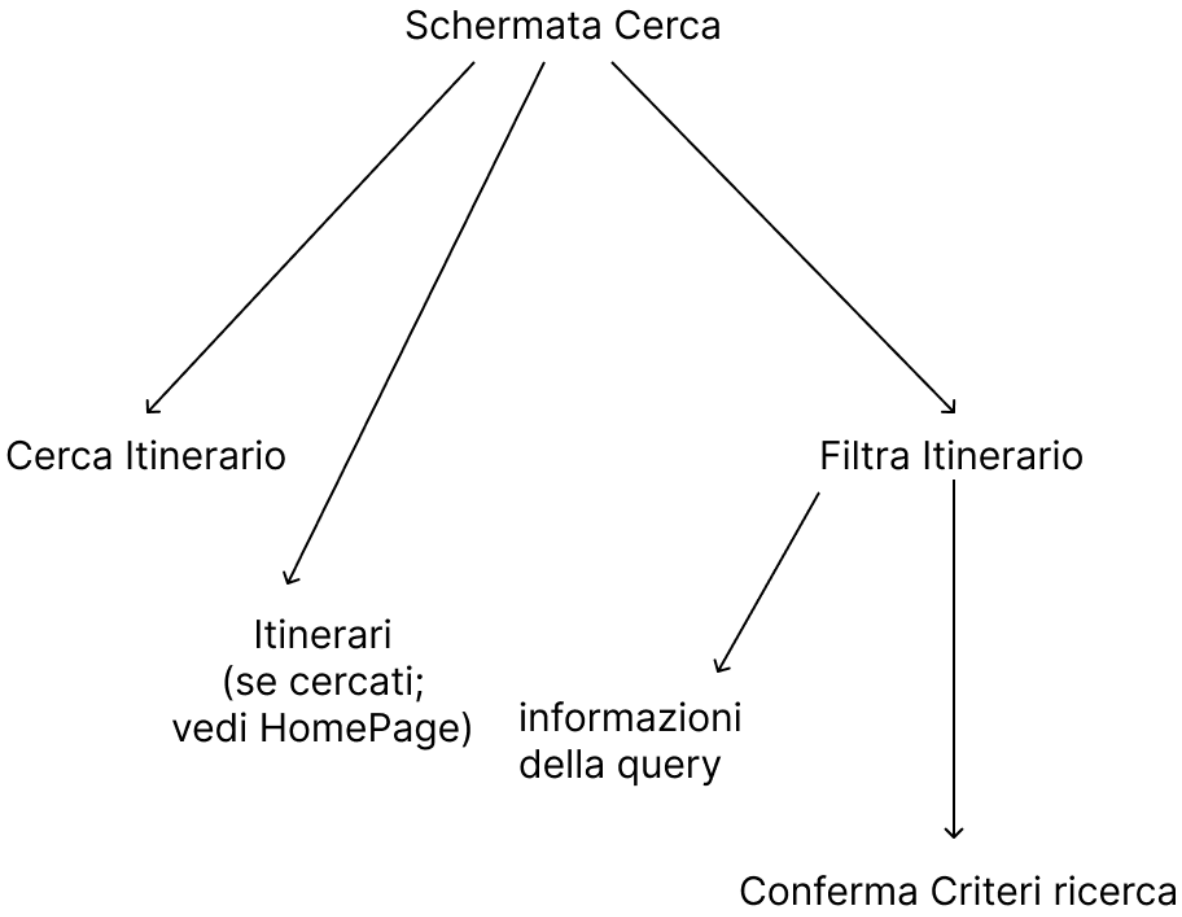
La foglia “itinerari” in realtà rappresenta il nodo già visto all’interno della gerarchia funzionale di “HomePage” nel caso della visualizzazione degli stessi. Il nodo Pannello admin è rappresentato da un affordance che è visualizzabile solamente se l’utente loggato è un amministratore.

4.4.4 Gerarchia funzionale per “Schermata Profilo”



Le affordance “Chat” sono molteplici e valgono per ogni chat aperta, di fatto, se nessuna chat è stata aperta allora non sarà mostrata alcuna chat.

4.4.5 Gerarchia funzionale per “Schermata Cerca”



La foglia itinerari rappresenta il nodo già trattato in precedenza, essi saranno mostrati solamente se l’utente compie una ricerca con una delle affordance di questa gerarchia.

5 Definizione di un piano di testing e valutazione sul campo dell'usabilità

5.1 Valutazione dell'usabilità sul campo

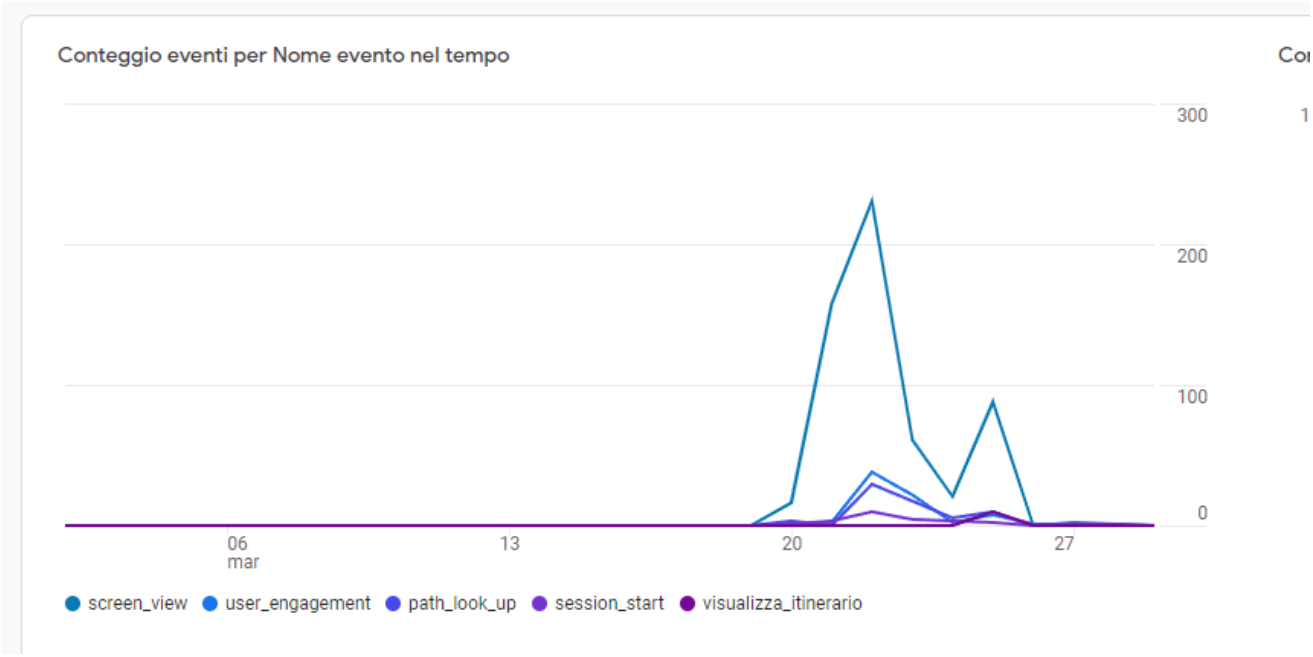
Per valutare la nostra applicazione sul campo è stato utilizzato il meccanismo di collezione di eventi di google analytics, viene distribuita una versione dell'applicazione che ha al suo interno parti di codice dedicate al lancio di eventi che verranno successivamente visualizzati sulla piattaforma. È inoltre possibile esaminare il singolo dispositivo nel dettaglio, in questo modo, con l'aiuto di un utente disposto ad effettuare la verifica, è possibile registrare le azioni che compie all'interno dell'applicazione mentre si osservano i suoi comportamenti.

È possibile ritirare le informazioni ricevute dagli utenti che testano l'applicazione in blocchi esaminabili in una pratica tabella che esprime il tipo di evento e quante volte è stato lanciato.

Column1	Column2	Column3
# ----------		
# Eventi		
# ----------		
Nome evento	Conteggio	Utenti
app_remove	4	4
cerca	2	1
cerca_con_filtro	2	1
cerca_con_searc	2	1
cerca_con_search	1	1
click	2	1
first_open	9	9
homepage	1	1
name	1	1
path_look_up	64	6
recensione_insert	1	1
screen_view	577	8
segnalazioni_itinerario	2	1
select_content	4	1
session_start	25	8
sign_up	3	2
visualizza_itinerario	10	1

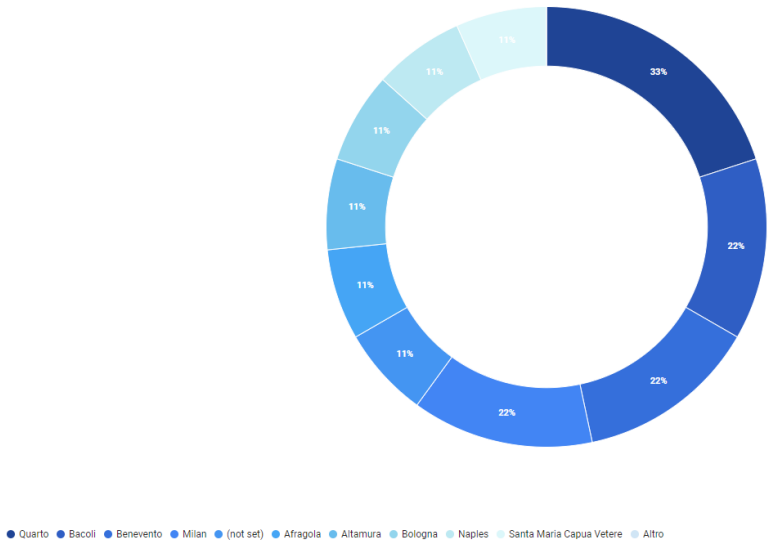
5.1.1 Metodi per la valutazione dell'usabilità

Grazie al servizio offerto da google è possibile esaminare in tempo reale diversi dati, come ad esempio quanti eventi sono lanciati in un determinato periodo di tempo:



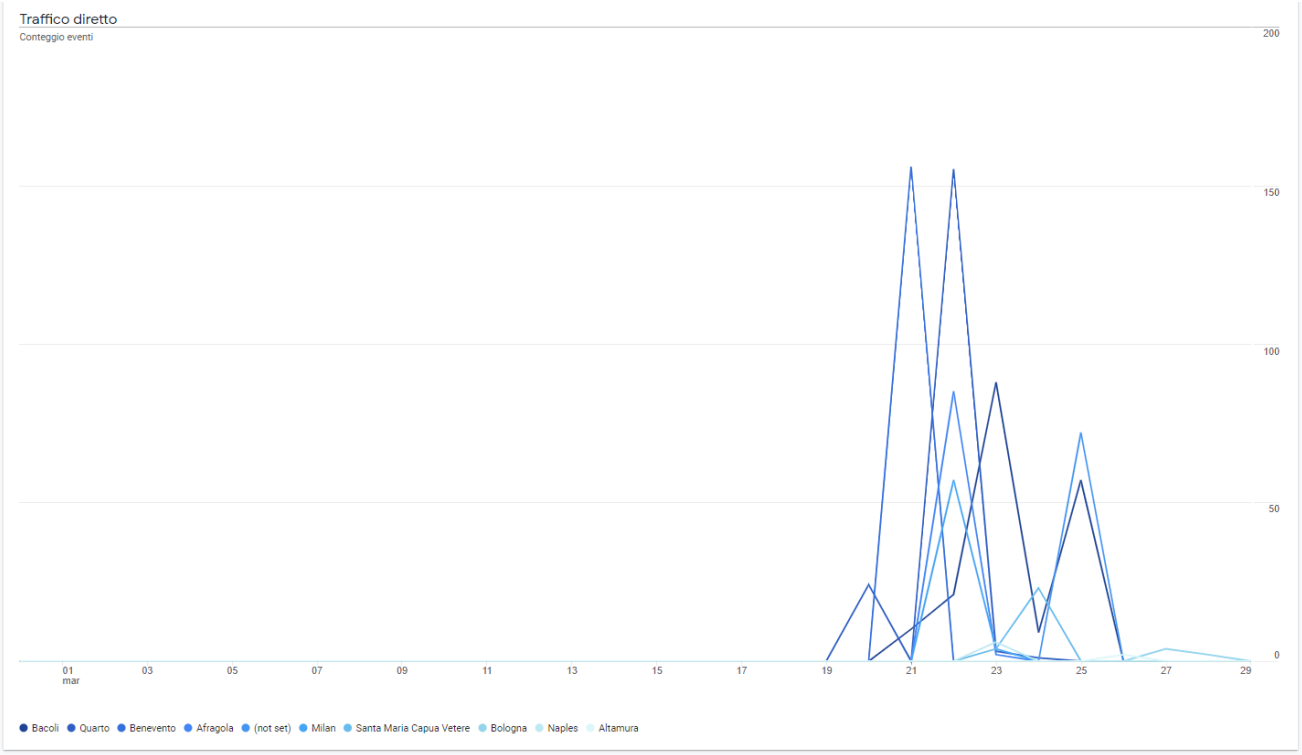
È possibile, inoltre, esaminare in quali paesi la nostra applicazione viene utilizzata:

Categoria del dispositivo		mobile	Totali
Città		Utenti attivi	↓ Utenti attivi
	Totali	9 100,0% del totale	9 100,0% del totale
1	Quarto	3	3
2	Bacoli	2	2
3	Benevento	2	2
4	Milan	2	2
5	(not set)	1	1
6	Afragola	1	1
7	Altamura	1	1
8	Bologna	1	1
9	Naples	1	1
10	Santa Maria Capua Vetere	1	1

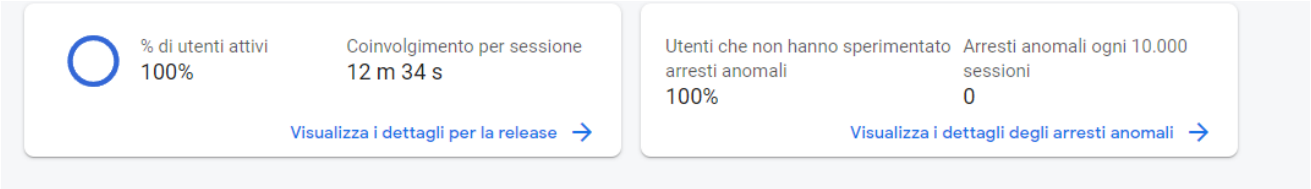


Qui sopra un'altra modalità di visualizzazione.

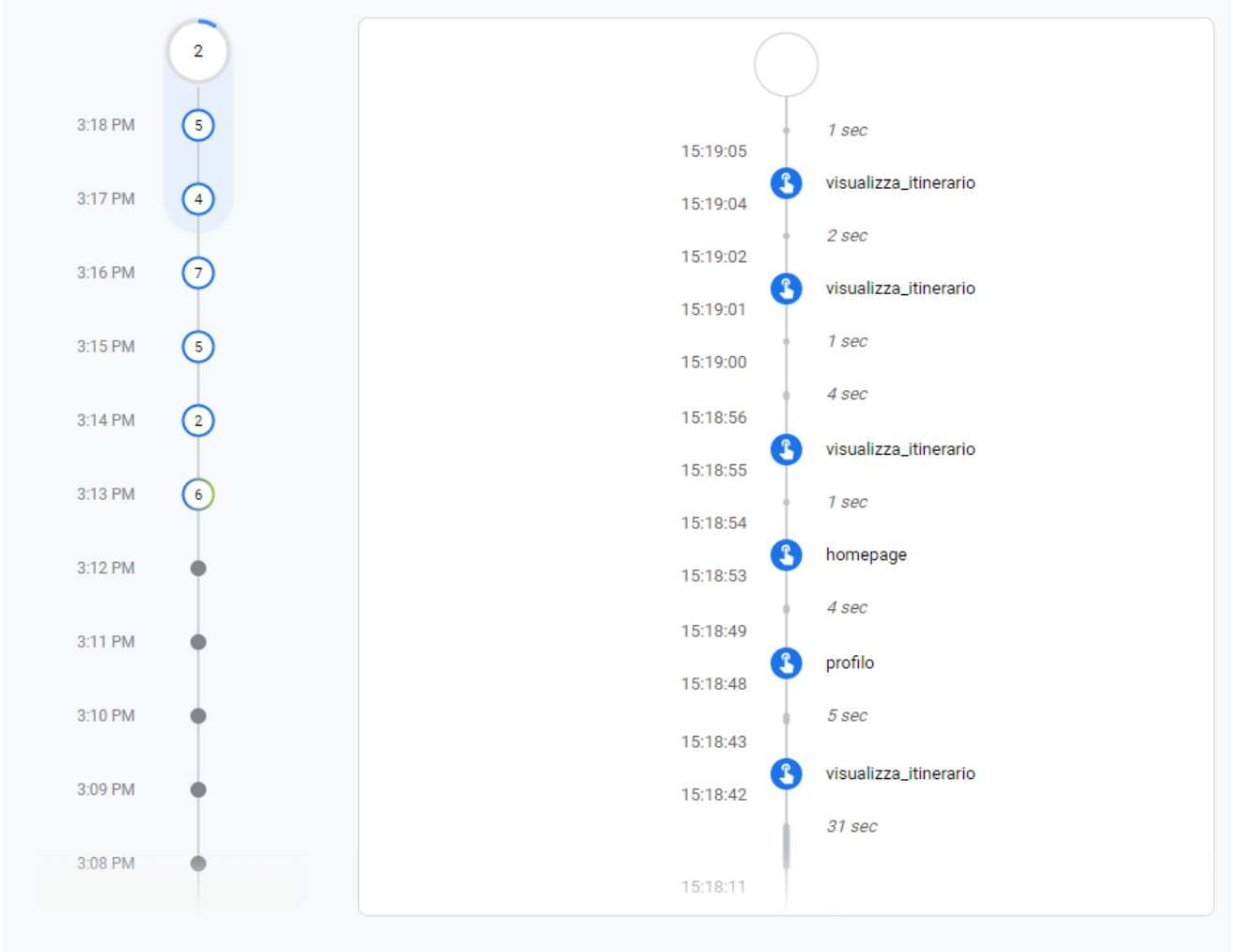
Vi è anche la possibilità di visualizzare il grafico di lancio degli eventi per i dispositivi utilizzati



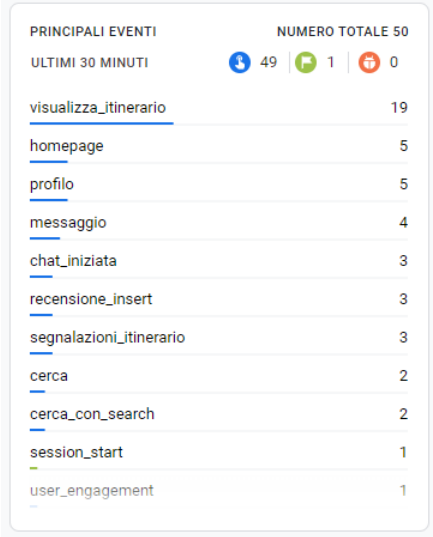
Il sistema è anche in grado di riconoscere il tempo delle sessioni degli utenti e quanti utenti hanno sperimentato arresti anomali:



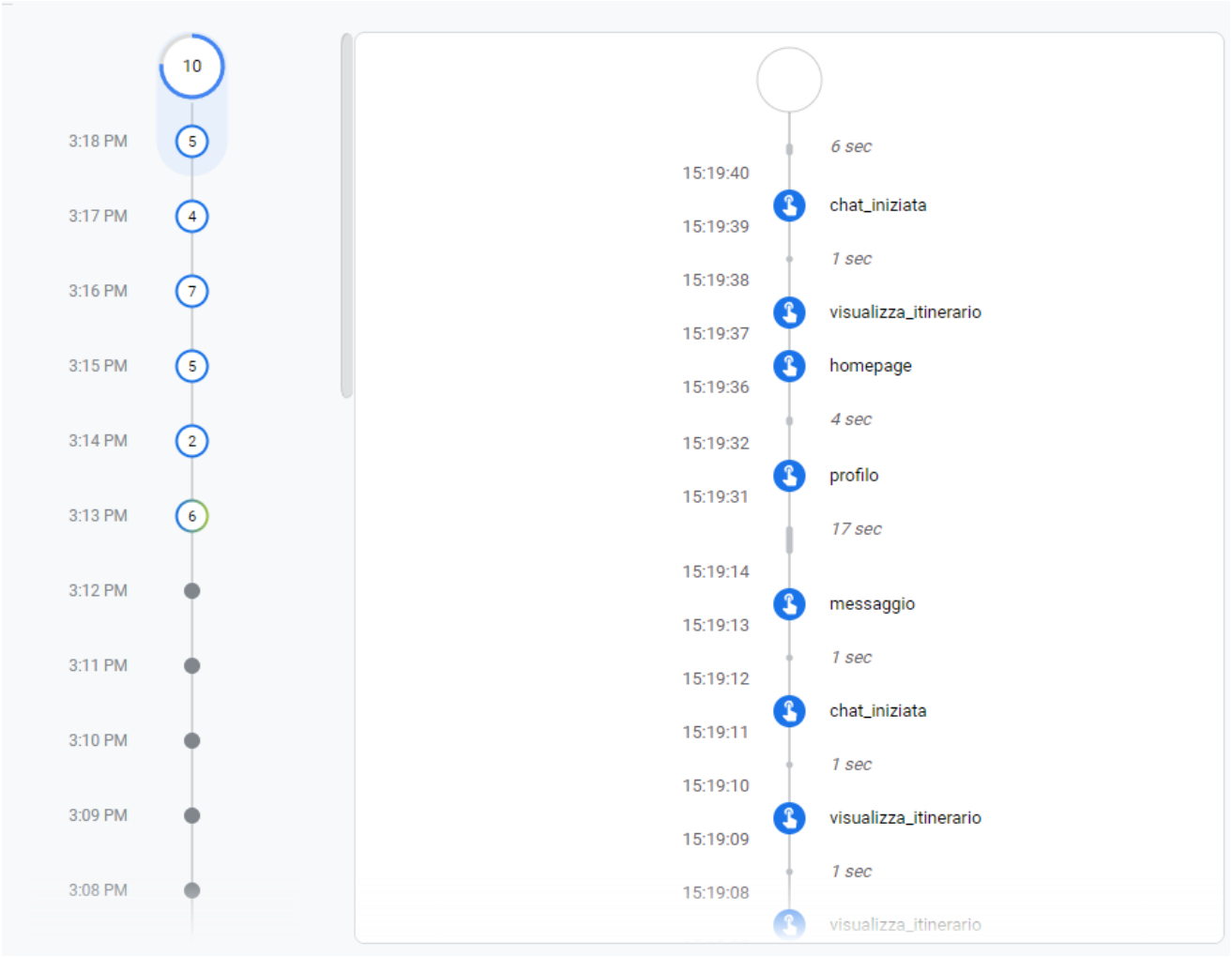
Possiamo esaminare in determinati dispositivi una schermata di debug dei singoli eventi e l'intervallo di tempo tra un evento e l'altro.

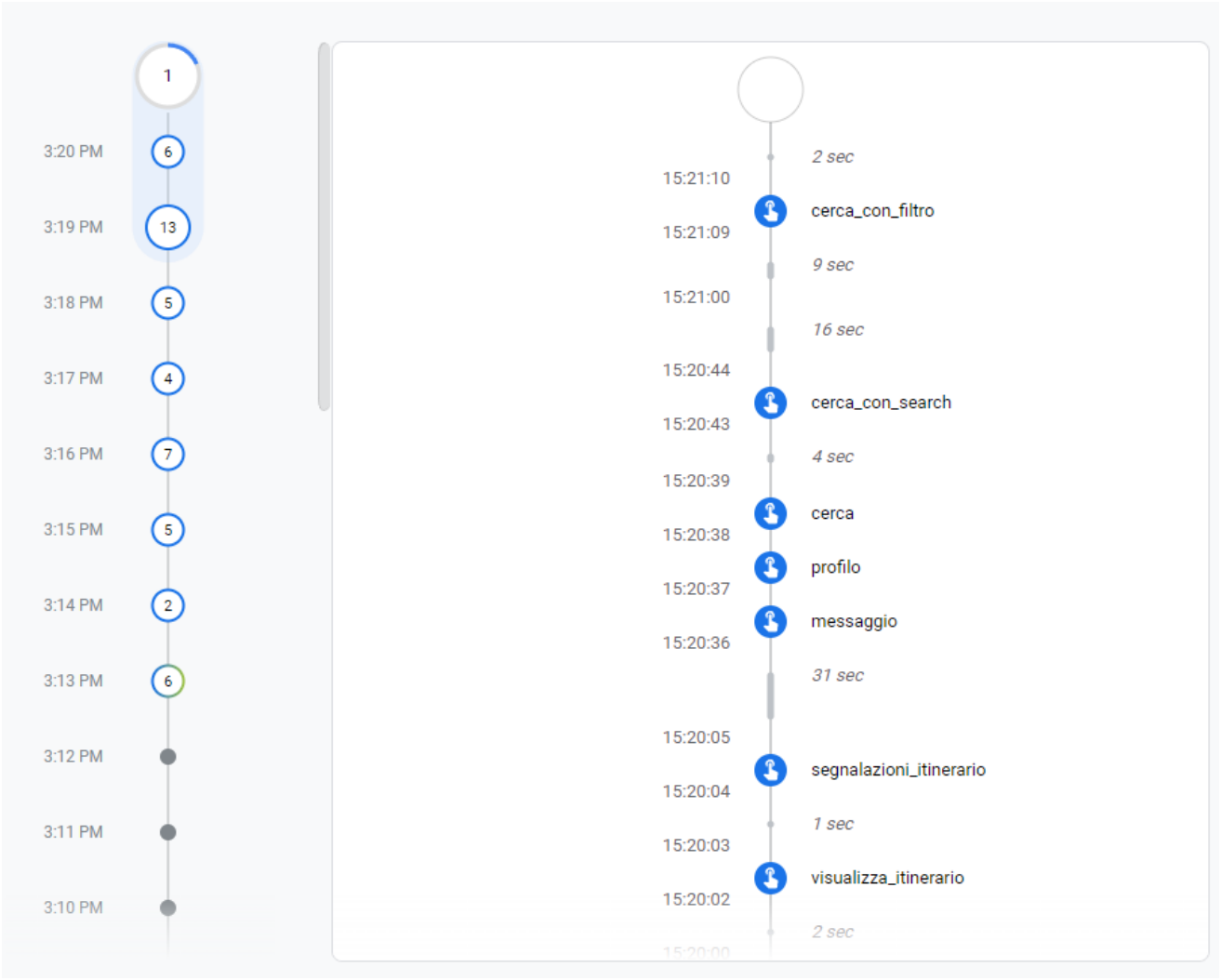


Insieme a questo possiamo vedere un resoconto degli eventi lanciati nel dispositivo



Qui di seguito altri esempi della schermata di debug:




1
3:20 PM
6
3:19 PM
13
3:18 PM
5
3:17 PM
4
3:16 PM
7
3:15 PM
5
3:14 PM
2
3:13 PM
6
3:12 PM
3:11 PM
3:10 PM
2 sec
15:21:10
cerca_con_filtro
15:21:09
9 sec
15:21:00
16 sec
15:20:44
cerca_con_search
15:20:43
4 sec
15:20:39
cerca
15:20:38
profilo
15:20:37
messaggio
15:20:36
31 sec
15:20:05
segnalazioni_itinerario
15:20:04
1 sec
15:20:03
visualizza_itinerario
15:20:02
2 sec

5.2 Codice JUnit per unit testing

Per il testing si è deciso di adottare la strategia black box per un metodo che controlla se un utente è abilitato per la cancellazione di una segnalazione o meno. La strategia white box con tecnica di statement coverage per un metodo che applica un filtro data una lista di itinerari. Infine, si è deciso di usare la strategia white box con tecnica di branch coverage per un metodo che data un'immagine e una lista di punti della mappa riconosce se questa immagine si trova abbastanza vicino o meno all'itinerario.

5.2.1 Testing del metodo "validateSegnalazione"

Per questo metodo si è deciso di usare la strategia blackbox. Il metodo vuole in input gli oggetti utente, itinerario e segnalazione e sappiamo che al loro interno vuole solo il token poiché saranno oggetto di confronti. Le classi di equivalenza scelte sono quelle dei possibili risultati dei confronti tra i vari token più dei casi per cui i token possono essere null, inoltre ci si aspetta un comportamento diverso quando l'utente che tenta di eliminare la cancellazione è admin. Se la segnalazione è stata scritta dall'autore dell'itinerario si verifica un errore da controllare. Infine, sappiamo che i risultati del metodo sono: vero se l'utente corrente può eliminare la segnalazione quindi se è admin oppure se è autore della segnalazione, falso altrimenti.
Qui di seguito il codice junit usato per il testing, nei commenti sono descritte le diverse classi di equivalenza. La strategia per il testing delle classi di equivalenze è WECT perché le classi di equivalenza identificate comprendono al loro interno valori essenziali, ad esempio se l'utente loggato è lo stesso dell'utente proprietario dell'itinerario: i valori di questa relazione possono essere solo VERO o FALSO, allo stesso modo le classi di equivalenza. La stessa cosa vale per l'utente loggato che può essere admin o meno. Caso particolare che comunque è stato preso in considerazione è quello in cui i vari token degli utenti in gioco possono essere null. Non ha senso dunque prendere in considerazione casi limite o fuori dai domini dei valori poiché comunque impossibili nei casi d'uso dell'applicazione.

```
package com.example.natour.controller;

import static org.junit.Assert.*;

import com.example.natour.exception.IllegalSegnalazioneException;
import com.example.natour.model.Itinerario;
import com.example.natour.model.Segnalazione;
import com.example.natour.model.Utente;

import org.junit.Test;

public class ControllerVisualizzaItinerarioTest
{

    //il metodo testato esamina i token dell'utente dei vari oggetti
    //esamina anche se l'utente loggato sia l'admin

    /*
    * Le classi di equivalenza prese in esame sono le possibili uguaglianze tra
    * l'utente loggato, l'utente dell'itinerario e l'utente della segnalazione,
    * si sono inoltre considerati i casi in cui l'utente è admin oppure no
    * */

    /** Per il metodo seguente:
     * utente = utente itinerario = utente segnalazione
```

```
     * utente non è admin **/
    @Test
    public void validateSegnalazione1()
    {
        Utente utente = new Utente();
        utente.setToken("token1");
        Itinerario itinerario = new Itinerario();
        itinerario.setFk_utente(utente.getToken());
        Segnalazione segnalazione = new Segnalazione();
        segnalazione.setFk_utente(utente.getToken());
        ControllerVisualizzaItinerario controllerVisualizzaItinerario =

                new ControllerVisualizzaItinerario(null
                        , itinerario, utente.getToken());
        try{
            controllerVisualizzaItinerario

                    .validateDeleteSegnalazione(utente,itinerario,segnalazione);
            fail();
        }catch (IllegalSegnalazioneException e){
            assertTrue(true);
        }catch (Exception e){
            fail();
        }
    }

    /** Per il metodo seguente:
     * utente = utente itinerario != utente segnalazione
     * utente non è admin **/
    @Test
    public void validateSegnalazione2()
    {
        Utente utente = new Utente();
        utente.setToken("token1");
        Itinerario itinerario = new Itinerario();
        itinerario.setFk_utente(utente.getToken());
        Segnalazione segnalazione = new Segnalazione();
        segnalazione.setFk_utente("token2");
        ControllerVisualizzaItinerario controllerVisualizzaItinerario =
                new ControllerVisualizzaItinerario(null
                        , itinerario, utente.getToken());
        assertFalse(controllerVisualizzaItinerario
                .validateDeleteSegnalazione(utente,itinerario,segnalazione));
    }

    /** Per il metodo seguente:
     * utente != utente itinerario = utente segnalazione
     * utente non è admin **/
    @Test
    public void validateSegnalazione3()
    {
        Utente utente = new Utente();
        utente.setToken("token1");
        Itinerario itinerario = new Itinerario();
        itinerario.setFk_utente("token2");
        Segnalazione segnalazione = new Segnalazione();
        segnalazione.setFk_utente(itinerario.getFk_utente());
```

```
        ControllerVisualizzaItinerario controllerVisualizzaItinerario =
                new ControllerVisualizzaItinerario(null
                        , itinerario, utente.getToken());
        try{
            controllerVisualizzaItinerario
                    .validateDeleteSegnalazione(utente,itinerario,segnalazione);
            fail();
        }catch (IllegalSegnalazioneException e){
            assertTrue(true);
        }catch (Exception e){
            fail();
        }    }

    /** Per il metodo seguente:
     * utente != utente itinerario != utente segnalazione
     * utente non è admin **/
    @Test
    public void validateSegnalazione4()
    {
        Utente utente = new Utente();
        utente.setToken("token1");
        Itinerario itinerario = new Itinerario();
        itinerario.setFk_utente("token2");
        Segnalazione segnalazione = new Segnalazione();
        segnalazione.setFk_utente("token3");
        ControllerVisualizzaItinerario controllerVisualizzaItinerario =
                new ControllerVisualizzaItinerario(null
                        , itinerario, utente.getToken());
        assertFalse(controllerVisualizzaItinerario
                .validateDeleteSegnalazione(utente,itinerario,segnalazione));
    }

    /** Per il metodo seguente:
     * utente = utente segnalazione != utente itinerario
     * utente non è admin **/
    @Test
    public void validateSegnalazione5()
    {
        Utente utente = new Utente();
        utente.setToken("token1");
        Itinerario itinerario = new Itinerario();
        itinerario.setFk_utente("token2");
        Segnalazione segnalazione = new Segnalazione();
        segnalazione.setFk_utente(utente.getToken());
        ControllerVisualizzaItinerario controllerVisualizzaItinerario =
                new ControllerVisualizzaItinerario(null
                        , itinerario, utente.getToken());
        assertTrue(controllerVisualizzaItinerario
                .validateDeleteSegnalazione(utente,itinerario,segnalazione));
    }

    /** Per il metodo seguente:
     * utente = utente itinerario = utente segnalazione
     * utente è admin **/
    @Test
    public void validateSegnalazione6()
    {
```

```
        Utente utente = new Utente();
        utente.setToken("token1");
        utente.setAdmin(true);
        Itinerario itinerario = new Itinerario();
        itinerario.setFk_utente(utente.getToken());
        Segnalazione segnalazione = new Segnalazione();
        segnalazione.setFk_utente(utente.getToken());
        ControllerVisualizzaItinerario controllerVisualizzaItinerario =
                new ControllerVisualizzaItinerario(null
                        , itinerario, utente.getToken());
        try{
            controllerVisualizzaItinerario
                    .validateDeleteSegnalazione(utente,itinerario,segnalazione);
            fail();
        }catch (IllegalSegnalazioneException e){
            assertTrue(true);
        }catch (Exception e){
            fail();
        }
    }

    /** Per il metodo seguente:
     * utente = utente itinerario != utente segnalazione
     * utente è admin **/
    @Test
    public void validateSegnalazione7()
    {
        Utente utente = new Utente();
        utente.setToken("token1");
        utente.setAdmin(true);
        Itinerario itinerario = new Itinerario();
        itinerario.setFk_utente(utente.getToken());
        Segnalazione segnalazione = new Segnalazione();
        segnalazione.setFk_utente("token2");
        ControllerVisualizzaItinerario controllerVisualizzaItinerario =
                new ControllerVisualizzaItinerario(null
                        , itinerario, utente.getToken());
        assertTrue(controllerVisualizzaItinerario
                .validateDeleteSegnalazione(utente,itinerario,segnalazione));
    }

    /** Per il metodo seguente:
     * utente != utente itinerario = utente segnalazione
     * utente è admin **/
    @Test
    public void validateSegnalazione8()
    {
        Utente utente = new Utente();
        utente.setToken("token1");
        utente.setAdmin(true);
        Itinerario itinerario = new Itinerario();
        itinerario.setFk_utente("token2");
        Segnalazione segnalazione = new Segnalazione();
        segnalazione.setFk_utente(itinerario.getFk_utente());
        ControllerVisualizzaItinerario controllerVisualizzaItinerario =
                new ControllerVisualizzaItinerario(null
                        , itinerario, utente.getToken());
```

```
        try{
            controllerVisualizzaItinerario
                    .validateDeleteSegnalazione(utente,itinerario,segnalazione);
            fail();
        }catch (IllegalSegnalazioneException e){
            assertTrue(true);
        }catch (Exception e){
            fail();
        }    }

    /** Per il metodo seguente:
     * utente != utente itinerario != utente segnalazione
     * utente è admin **/
    @Test
    public void validateSegnalazione9()
    {
        Utente utente = new Utente();
        utente.setToken("token1");
        utente.setAdmin(true);
        Itinerario itinerario = new Itinerario();
        itinerario.setFk_utente("token2");
        Segnalazione segnalazione = new Segnalazione();
        segnalazione.setFk_utente("token3");
        ControllerVisualizzaItinerario controllerVisualizzaItinerario =
                new ControllerVisualizzaItinerario(null
                        , itinerario, utente.getToken());
        assertTrue(controllerVisualizzaItinerario
                .validateDeleteSegnalazione(utente,itinerario,segnalazione));
    }

    /** Per il metodo seguente:
     * utente = utente segnalazione != utente itinerario
     * utente è admin **/
    @Test
    public void validateSegnalazione10()
    {
        Utente utente = new Utente();
        utente.setToken("token1");
        utente.setAdmin(true);
        Itinerario itinerario = new Itinerario();
        itinerario.setFk_utente("token2");
        Segnalazione segnalazione = new Segnalazione();
        segnalazione.setFk_utente(utente.getToken());
        ControllerVisualizzaItinerario controllerVisualizzaItinerario =
                new ControllerVisualizzaItinerario(null
                        , itinerario, utente.getToken());
        assertTrue(controllerVisualizzaItinerario
                .validateDeleteSegnalazione(utente,itinerario,segnalazione));
    }

    /** Per il metodo seguente:
     * utente = utente itinerario = utente segnalazione
     * utente non è admin
     * il suo token è null**/
    @Test
    public void tokenUtenteNull1()
    {
```

```
        Utente utente = new Utente();
        Itinerario itinerario = new Itinerario();
        itinerario.setFk_utente(utente.getToken());
        Segnalazione segnalazione = new Segnalazione();
        segnalazione.setFk_utente(utente.getToken());
        ControllerVisualizzaItinerario controllerVisualizzaItinerario =
                new ControllerVisualizzaItinerario(null
                        , itinerario, utente.getToken());
        try{
            controllerVisualizzaItinerario
                    .validateDeleteSegnalazione(utente,itinerario,segnalazione);
            fail();
        }catch (NullPointerException e){
            assertTrue(true);
        }catch (Exception e){
            fail();
        }
    }

    /** Per il metodo seguente:
     * utente = utente itinerario != utente segnalazione
     * utente non è admin
     * il suo token è null**/
    @Test
    public void tokenUtenteNull2()
    {
        Utente utente = new Utente();
        Itinerario itinerario = new Itinerario();
        itinerario.setFk_utente(utente.getToken());
        Segnalazione segnalazione = new Segnalazione();
        segnalazione.setFk_utente("token2");
        ControllerVisualizzaItinerario controllerVisualizzaItinerario =
                new ControllerVisualizzaItinerario(null
                        , itinerario, utente.getToken());
        try{
            controllerVisualizzaItinerario
                    .validateDeleteSegnalazione(utente,itinerario,segnalazione);
            fail();
        }catch (NullPointerException e){
            assertTrue(true);
        }catch (Exception e){
            fail();
        }
    }

    /** Per il metodo seguente:
     * utente != utente itinerario = utente segnalazione
     * utente non è admin
     * il suo token è null**/
    @Test
    public void tokenUtenteNull3()
    {
        Utente utente = new Utente();
        Itinerario itinerario = new Itinerario();
        itinerario.setFk_utente("token2");
        Segnalazione segnalazione = new Segnalazione();
        segnalazione.setFk_utente(itinerario.getFk_utente());
```

```
        ControllerVisualizzaItinerario controllerVisualizzaItinerario =
                new ControllerVisualizzaItinerario(null
                        , itinerario, utente.getToken());
        try{
            controllerVisualizzaItinerario
                    .validateDeleteSegnalazione(utente,itinerario,segnalazione);
            fail();
        }catch (NullPointerException e){
            assertTrue(true);
        }catch (Exception e){
            fail();
        }
    }

    /** Per il metodo seguente:
     * utente != utente itinerario != utente segnalazione
     * utente non è admin
     * il suo token è null**/
    @Test
    public void tokenUtenteNull4()
    {
        Utente utente = new Utente();
        Itinerario itinerario = new Itinerario();
        itinerario.setFk_utente("token2");
        Segnalazione segnalazione = new Segnalazione();
        segnalazione.setFk_utente("token3");
        ControllerVisualizzaItinerario controllerVisualizzaItinerario =
                new ControllerVisualizzaItinerario(null
                        , itinerario, utente.getToken());
        try{
            controllerVisualizzaItinerario
                    .validateDeleteSegnalazione(utente,itinerario,segnalazione);
            fail();
        }catch (NullPointerException e){
            assertTrue(true);
        }catch (Exception e){
            fail();
        }
    }

    /** Per il metodo seguente:
     * utente = utente segnalazione != utente itinerario
     * utente non è admin
     * il suo token è null**/
    @Test
    public void tokenUtenteNull5()
    {
        Utente utente = new Utente();
        Itinerario itinerario = new Itinerario();
        itinerario.setFk_utente("token2");
        Segnalazione segnalazione = new Segnalazione();
        segnalazione.setFk_utente(utente.getToken());
        ControllerVisualizzaItinerario controllerVisualizzaItinerario =
                new ControllerVisualizzaItinerario(null
                        , itinerario, utente.getToken());
        try{
            controllerVisualizzaItinerario
```

```
                    .validateDeleteSegnalazione(utente,itinerario,segnalazione);
            fail();
        }catch (NullPointerException e){
            assertTrue(true);
        }catch (Exception e){
            fail();
        }
    }

    /** Per il metodo seguente:
     * utente = utente itinerario = utente segnalazione
     * utente è admin
     * il suo token è null**/
    @Test
    public void tokenUtenteNull6()
    {
        Utente utente = new Utente();
        utente.setAdmin(true);
        Itinerario itinerario = new Itinerario();
        itinerario.setFk_utente(utente.getToken());
        Segnalazione segnalazione = new Segnalazione();
        segnalazione.setFk_utente(utente.getToken());
        ControllerVisualizzaItinerario controllerVisualizzaItinerario =
                new ControllerVisualizzaItinerario(null
                        , itinerario, utente.getToken());
        try{
            controllerVisualizzaItinerario
                    .validateDeleteSegnalazione(utente,itinerario,segnalazione);
            fail();
        }catch (NullPointerException e){
            assertTrue(true);
        }catch (Exception e){
            fail();
        }
    }

    /** Per il metodo seguente:
     * utente = utente itinerario != utente segnalazione
     * utente è admin
     * il suo token è null**/
    @Test
    public void tokenUtenteNull7()
    {
        Utente utente = new Utente();
        utente.setAdmin(true);
        Itinerario itinerario = new Itinerario();
        itinerario.setFk_utente(utente.getToken());
        Segnalazione segnalazione = new Segnalazione();
        segnalazione.setFk_utente("token2");
        ControllerVisualizzaItinerario controllerVisualizzaItinerario =
                new ControllerVisualizzaItinerario(null
                        , itinerario, utente.getToken());
        try{
            controllerVisualizzaItinerario
                    .validateDeleteSegnalazione(utente,itinerario,segnalazione);
            fail();
        }catch (NullPointerException e){
```

```
            assertTrue(true);
        }catch (Exception e){
            fail();
        }
    }

    /** Per il metodo seguente:
     * utente != utente itinerario = utente segnalazione
     * utente è admin
     * il suo token è null**/
    @Test
    public void tokenUtenteNull8()
    {
        Utente utente = new Utente();
        utente.setAdmin(true);
        Itinerario itinerario = new Itinerario();
        itinerario.setFk_utente("token2");
        Segnalazione segnalazione = new Segnalazione();
        segnalazione.setFk_utente(itinerario.getFk_utente());
        ControllerVisualizzaItinerario controllerVisualizzaItinerario =
                new ControllerVisualizzaItinerario(null
                        , itinerario, utente.getToken());
        try{
            controllerVisualizzaItinerario
                    .validateDeleteSegnalazione(utente,itinerario,segnalazione);
            fail();
        }catch (NullPointerException e){
            assertTrue(true);
        }catch (Exception e){
            fail();
        }
    }

    /** Per il metodo seguente:
     * utente != utente itinerario != utente segnalazione
     * utente è admin
     * il suo token è null**/
    @Test
    public void tokenUtenteNull9()
    {
        Utente utente = new Utente();
        utente.setAdmin(true);
        Itinerario itinerario = new Itinerario();
        itinerario.setFk_utente("token2");
        Segnalazione segnalazione = new Segnalazione();
        segnalazione.setFk_utente("token3");
        ControllerVisualizzaItinerario controllerVisualizzaItinerario =
                new ControllerVisualizzaItinerario(null
                        , itinerario, utente.getToken());
        try{
            controllerVisualizzaItinerario
                    .validateDeleteSegnalazione(utente,itinerario,segnalazione);
            fail();
        }catch (NullPointerException e){
            assertTrue(true);
        }catch (Exception e){
            fail();
```

```
        }
    }

    /** Per il metodo seguente:
     * utente = utente segnalazione != utente itinerario
     * utente è admin
     * il suo token è null**/
    @Test
    public void tokenUtenteNull10()
    {
        Utente utente = new Utente();
        utente.setAdmin(true);
        Itinerario itinerario = new Itinerario();
        itinerario.setFk_utente("token2");
        Segnalazione segnalazione = new Segnalazione();
        segnalazione.setFk_utente(utente.getToken());
        ControllerVisualizzaItinerario controllerVisualizzaItinerario =
                new ControllerVisualizzaItinerario(null
                        , itinerario, utente.getToken());
        try{
            controllerVisualizzaItinerario
                    .validateDeleteSegnalazione(utente,itinerario,segnalazione);
            fail();
        }catch (NullPointerException e){
            assertTrue(true);
        }catch (Exception e){
            fail();
        }
    }


    /** Per il metodo seguente:
     * utente != utente itinerario = utente segnalazione
     * utente non è admin
     * fk utente itinerario è null **/
    @Test
    public void tokenUtenteItinerarioNull1()
    {
        Utente utente = new Utente();
        utente.setToken("token1");
        Itinerario itinerario = new Itinerario();
        Segnalazione segnalazione = new Segnalazione();
        segnalazione.setFk_utente(itinerario.getFk_utente());
        ControllerVisualizzaItinerario controllerVisualizzaItinerario =
                new ControllerVisualizzaItinerario(null
                        , itinerario, utente.getToken());
        try{
            controllerVisualizzaItinerario
                    .validateDeleteSegnalazione(utente,itinerario,segnalazione);
            fail();
        }catch (NullPointerException e){
            assertTrue(true);
        }catch (Exception e){
            fail();
        }
    }
```

```
    /** Per il metodo seguente:
     * utente != utente itinerario != utente segnalazione
     * utente non è admin
     * fk utente itinerario è null **/
    @Test
    public void tokenUtenteItinerarioNull2()
    {
        Utente utente = new Utente();
        utente.setToken("token1");
        Itinerario itinerario = new Itinerario();
        Segnalazione segnalazione = new Segnalazione();
        segnalazione.setFk_utente("token3");
        ControllerVisualizzaItinerario controllerVisualizzaItinerario =
                new ControllerVisualizzaItinerario(null
                        , itinerario, utente.getToken());
        try{
            controllerVisualizzaItinerario
                    .validateDeleteSegnalazione(utente,itinerario,segnalazione);
            fail();
        }catch (NullPointerException e){
            assertTrue(true);
        }catch (Exception e){
            fail();
        }
    }

    /** Per il metodo seguente:
     * utente = utente segnalazione != utente itinerario
     * utente non è admin
     * fk utente itinerario è null **/
    @Test
    public void tokenUtenteItinerarioNull3()
    {
        Utente utente = new Utente();
        utente.setToken("token1");
        Itinerario itinerario = new Itinerario();
        itinerario.setFk_utente(null);
        Segnalazione segnalazione = new Segnalazione();
        segnalazione.setFk_utente(utente.getToken());
        ControllerVisualizzaItinerario controllerVisualizzaItinerario =
                new ControllerVisualizzaItinerario(null
                        , itinerario, utente.getToken());
        try{
            controllerVisualizzaItinerario
                    .validateDeleteSegnalazione(utente,itinerario,segnalazione);
            fail();
        }catch (NullPointerException e){
            assertTrue(true);
        }catch (Exception e){
            fail();
        }
    }


    /** Per il metodo seguente:
     * utente != utente itinerario = utente segnalazione
     * utente è admin
```

```
     * fk utente itinerario è null **/
    @Test
    public void tokenUtenteItinerarioNull4()
    {
        Utente utente = new Utente();
        utente.setToken("token1");
        utente.setAdmin(true);
        Itinerario itinerario = new Itinerario();
        Segnalazione segnalazione = new Segnalazione();
        segnalazione.setFk_utente(itinerario.getFk_utente());
        ControllerVisualizzaItinerario controllerVisualizzaItinerario =
                new ControllerVisualizzaItinerario(null
                        , itinerario, utente.getToken());
        try{
            controllerVisualizzaItinerario
                    .validateDeleteSegnalazione(utente,itinerario,segnalazione);
            fail();
        }catch (NullPointerException e){
            assertTrue(true);
        }catch (Exception e){
            fail();
        }
    }

    /** Per il metodo seguente:
     * utente != utente itinerario != utente segnalazione
     * utente è admin
     * fk utente itinerario è null **/
    @Test
    public void tokenUtenteItinerarioNull5()
    {
        Utente utente = new Utente();
        utente.setAdmin(true);
        utente.setToken("token1");
        Itinerario itinerario = new Itinerario();
        Segnalazione segnalazione = new Segnalazione();
        segnalazione.setFk_utente("token3");
        ControllerVisualizzaItinerario controllerVisualizzaItinerario =
                new ControllerVisualizzaItinerario(null
                        , itinerario, utente.getToken());
        try{
            controllerVisualizzaItinerario
                    .validateDeleteSegnalazione(utente,itinerario,segnalazione);
            fail();
        }catch (NullPointerException e){
            assertTrue(true);
        }catch (Exception e){
            fail();
        }
    }

    /** Per il metodo seguente:
     * utente = utente segnalazione != utente itinerario
     * utente è admin
     * fk utente itinerario è null **/
    @Test
    public void tokenUtenteItinerarioNull6()
```

```
    {
        Utente utente = new Utente();
        utente.setAdmin(true);
        utente.setToken("token1");
        Itinerario itinerario = new Itinerario();
        Segnalazione segnalazione = new Segnalazione();
        segnalazione.setFk_utente(utente.getToken());
        ControllerVisualizzaItinerario controllerVisualizzaItinerario =
                new ControllerVisualizzaItinerario(null
                        , itinerario, utente.getToken());
        try{
            controllerVisualizzaItinerario
                    .validateDeleteSegnalazione(utente,itinerario,segnalazione);
            fail();
        }catch (NullPointerException e){
            assertTrue(true);
        }catch (Exception e){
            fail();
        }
    }

    /** Per il metodo seguente:
     * utente = utente itinerario != utente segnalazione
     * utente non è admin
     * fk utente di segnalazione è null **/
    @Test
    public void fkUtenteSegnalazioneNull1()
    {
        Utente utente = new Utente();
        utente.setToken("token1");
        Itinerario itinerario = new Itinerario();
        itinerario.setFk_utente(utente.getToken());
        Segnalazione segnalazione = new Segnalazione();
        ControllerVisualizzaItinerario controllerVisualizzaItinerario =
                new ControllerVisualizzaItinerario(null
                        , itinerario, utente.getToken());
        try{
            controllerVisualizzaItinerario
                    .validateDeleteSegnalazione(utente,itinerario,segnalazione);
            fail();
        }catch (NullPointerException e){
            assertTrue(true);
        }catch (Exception e){
            fail();
        }
    }

    /** Per il metodo seguente:
     * utente != utente itinerario != utente segnalazione
     * utente non è admin
     * fk utente di segnalazione è null **/
    @Test
    public void fkUtenteSegnalazioneNull2()
    {
        Utente utente = new Utente();
        utente.setToken("token1");
        Itinerario itinerario = new Itinerario();
```

```
        itinerario.setFk_utente("token2");
        Segnalazione segnalazione = new Segnalazione();
        ControllerVisualizzaItinerario controllerVisualizzaItinerario =
                new ControllerVisualizzaItinerario(null
                        , itinerario, utente.getToken());
        try{
            controllerVisualizzaItinerario
                    .validateDeleteSegnalazione(utente,itinerario,segnalazione);
            fail();
        }catch (NullPointerException e){
            assertTrue(true);
        }catch (Exception e){
            fail();
        }
    }

    /** Per il metodo seguente:
     * utente = utente itinerario != utente segnalazione
     * utente è admin
     * fk utente di segnalazione è null **/
    @Test
    public void fkUtenteSegnalazioneNull3()
    {
        Utente utente = new Utente();
        utente.setAdmin(true);
        utente.setToken("token1");
        Itinerario itinerario = new Itinerario();
        itinerario.setFk_utente(utente.getToken());
        Segnalazione segnalazione = new Segnalazione();
        ControllerVisualizzaItinerario controllerVisualizzaItinerario =
                new ControllerVisualizzaItinerario(null
                        , itinerario, utente.getToken());
        try{
            controllerVisualizzaItinerario
                    .validateDeleteSegnalazione(utente,itinerario,segnalazione);
            fail();
        }catch (NullPointerException e){
            assertTrue(true);
        }catch (Exception e){
            fail();
        }
    }


    /** Per il metodo seguente:
     * utente != utente itinerario != utente segnalazione
     * utente è admin
     * fk utente di segnalazione è null **/
    @Test
    public void fkUtenteSegnalazioneNull4()
    {
        Utente utente = new Utente();
        utente.setAdmin(true);
        utente.setToken("token1");
        Itinerario itinerario = new Itinerario();
        itinerario.setFk_utente("token2");
        Segnalazione segnalazione = new Segnalazione();
```

```
        ControllerVisualizzaItinerario controllerVisualizzaItinerario =
                new ControllerVisualizzaItinerario(null
                        , itinerario, utente.getToken());
        try{
            controllerVisualizzaItinerario
                    .validateDeleteSegnalazione(utente,itinerario,segnalazione);
            fail();
        }catch (NullPointerException e){
            assertTrue(true);
        }catch (Exception e){
            fail();
        }
    }
}
```

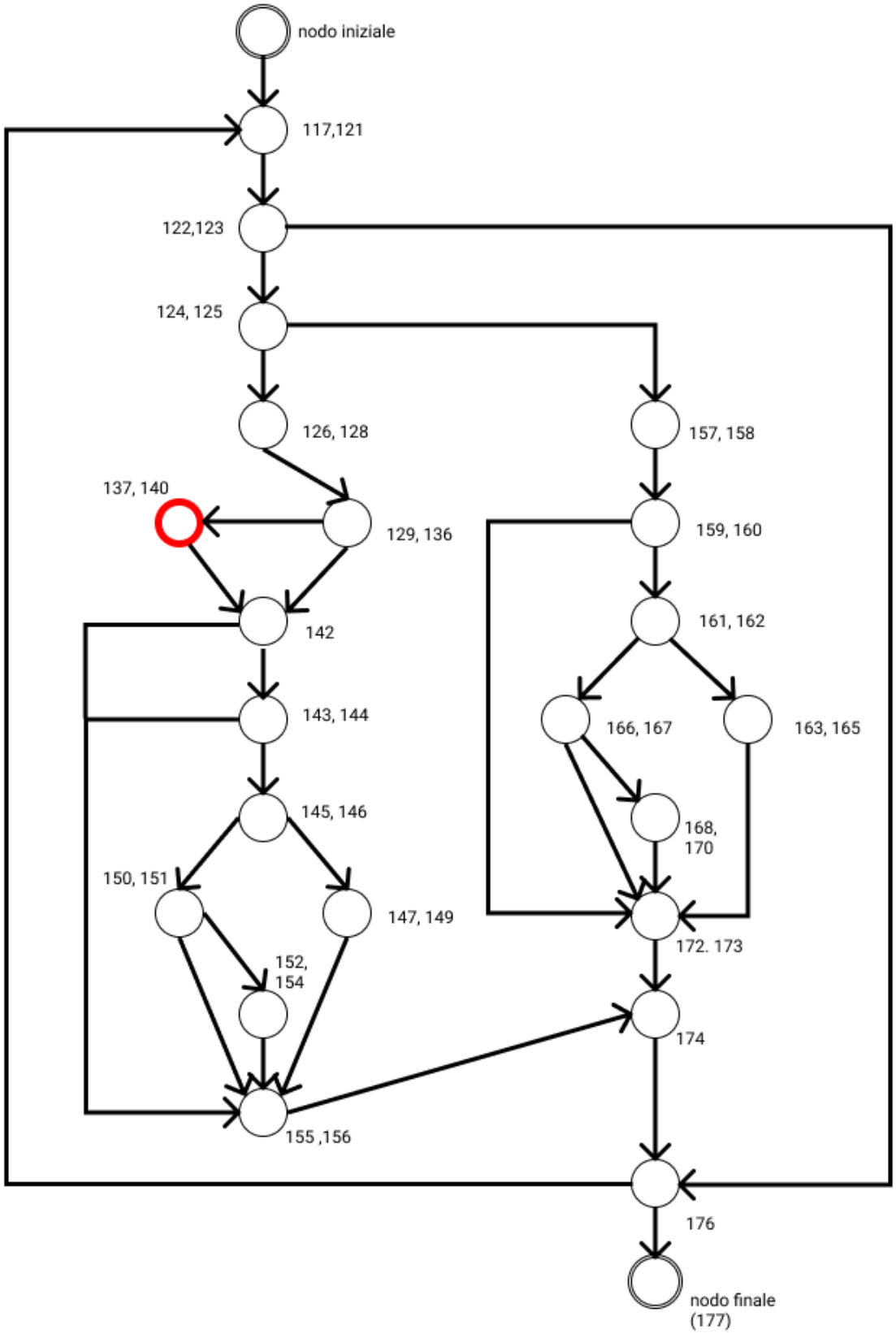
5.2.2 Testing del metodo “filterResult”

Per questo metodo si è deciso di adottare la strategia white box con tecnica di statement coverage, dunque, dato il codice si è deciso di fare testing in modo tale che ogni istruzione fosse eseguita. Prima vediamo il codice del metodo in questione:

```
public Boolean filterResult(String address, int difficolta, String durata,
boolean disabili, Context context) throws NullPointerException
{
    boolean exists = Boolean.FALSE;
    copiaLista.clear();
    List<Address> addressList = new ArrayList<>();
    for (Itinerario i : itinerari)
    {
        if(i.isAccessibilitaDisabili() == disabili && i.getDifficoltà() ==
difficolta)
        {
            if(!address.isEmpty())
            {
                Geocoder coder = new Geocoder(context);
                Address location = null;
                Address currLocation = null;
                try
                {
                    addressList = coder.getFromLocationName(address, 5);
                    location = addressList.get(0);
                    addressList = coder.getFromLocation(i.getWaypoints()
                                    .get(0).getLatitude()
                            , i.getWaypoints().get(0)
                                    .getLongitude(), 1);
                    currLocation = addressList.get(0);
                }
                catch (IOException | IndexOutOfBoundsException e)
                {
                    e.printStackTrace();
                }

                if(currLocation != null && currLocation.getCountryName()
                        .equals(location.getCountryName()))
                    if(!copiaLista.contains(i))
                    {
                        if(durata.isEmpty())
                        {
                            copiaLista.add(i);
                            exists = true;
                        }
                        else if(i.getDurata().contains(durata))
                        {
                            copiaLista.add(i);
                            exists = true;
                        }
                    }
            }
            else
            {
                if(!copiaLista.contains(i))
                {
```

```
                    if(durata.isEmpty())
                    {
                        copiaLista.add(i);
                        exists = true;
                    }
                    else if(i.getDurata().contains(durata))
                    {
                        copiaLista.add(i);
                        exists = true;
                    }
                }
            }

        }

    }
    return exists;
}
```

Per questo metodo sono stati costruiti dei grafi del flusso di controllo per esaminare il comportamento del metodo (i numeri dei nodi rispecchiano il numero di righe del codice nell'editor utilizzato) qui di seguito il grafo:


nodo iniziale
117,121
122,123
124, 125
126, 128
157, 158
137, 140
129, 136
159, 160
142
161, 162
143, 144
166, 167
163, 165
145, 146
168,
170
150, 151
147, 149
172. 173
152,
154
174
155 ,156
176
nodo finale
(177)

Per ogni test eseguito ci sarà il grafo del flusso di controllo con i nodi esaminati colorati in blu.

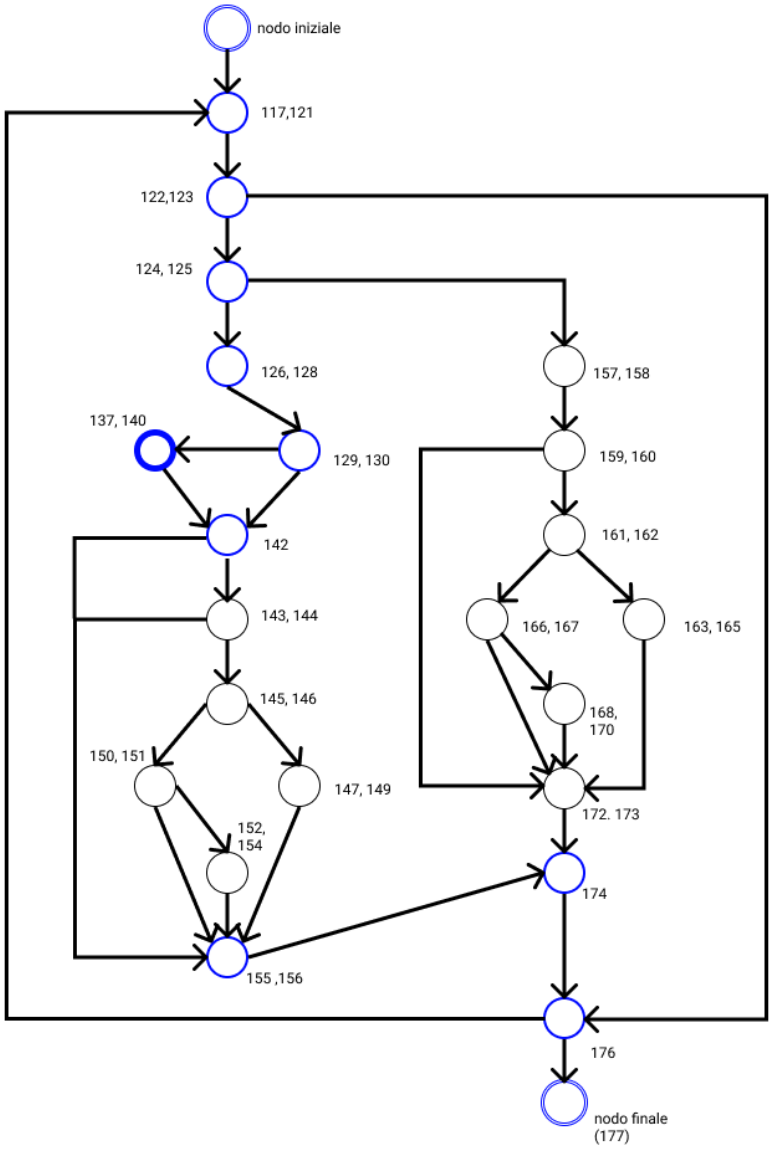
```
package com.example.natour.controller;

import static org.junit.Assert.*;

import android.content.Context;

import androidx.test.platform.app.InstrumentationRegistry;

import com.example.natour.model.Itinerario;

import org.junit.Test;
import org.osmdroid.util.GeoPoint;

import java.util.ArrayList;

public class ControllerCercaTest
{

    /*
    * Il branch del catch (IOException) non può essere verificato in
    * quanto sarebbe necessario imporre al geocoder di non funzionare
    * oppure mettere la modalità aereo,
    * cosa non possibile in fase di test.
    * Nel caso in cui il geocoder dovrebbe fallire si comporta come
    * se non ci fossero itinerari corrispondenti
    * all'address ricercato
    * */

    /* il punto 40.85157508957582, 14.26479657279487 è stato preso
    * con servizi esterni e indica la posizione di napoli
    * */

    /*
    * I test seguenti sono stati effettuati mediante tecnica whitebox
    * con strategia di statement coverage.
    * */


    @Test
    public void testfilterResultCorretto()
    {
        Context appContext = InstrumentationRegistry
                .getInstrumentation().getTargetContext();
        ArrayList<Itinerario> itinerari = new ArrayList<>();
        Itinerario itinerario = new Itinerario();
        itinerario.setNome("Napoli che bella");
        itinerario.setDifficoltà(3);
        itinerario.setDescrizione("una bella passeggiata a napoli");
        itinerario.setDurata("13");
        itinerario.setAccessibilitaDisabili(false);
        GeoPoint punto = new GeoPoint(40.85157508957582,
                14.26479657279487);
        ArrayList<GeoPoint> waypoints = new ArrayList<>();
        waypoints.add(punto);
        itinerario.setWaypoints(waypoints);
        itinerari.add(itinerario);
```

```
        ControllerCerca controllerCerca = new ControllerCerca(null,
                null, itinerari, null);
        Boolean exist;
        exist = controllerCerca.filterResult("napoli",
                3,"13",false, appContext);
        assertTrue(exist);

    }

    @Test
    public void testfilterResultWithoutAddress()
    {
        Context appContext = InstrumentationRegistry
                .getInstrumentation().getTargetContext();
        ArrayList<Itinerario> itinerari = new ArrayList<>();
        Itinerario itinerario = new Itinerario();
        itinerario.setNome("Napoli che bella");
        itinerario.setDifficoltà(3);
        itinerario.setDescrizione("una bella passeggiata a napoli");
        itinerario.setDurata("13");
        itinerario.setAccessibilitaDisabili(false);
        GeoPoint punto = new GeoPoint(40.85157508957582,
                14.26479657279487);
        ArrayList<GeoPoint> waypoints = new ArrayList<>();
        waypoints.add(punto);
        itinerario.setWaypoints(waypoints);
        itinerari.add(itinerario);
        ControllerCerca controllerCerca = new ControllerCerca(null,
                null, itinerari, null);
        Boolean exist;
        exist = controllerCerca.filterResult("",
                3,"13",false, appContext);
        assertTrue(exist);

    }



    @Test
    public void testfilterResultWrongAddress()
    {
        Context appContext = InstrumentationRegistry
                .getInstrumentation().getTargetContext();
        ArrayList<Itinerario> itinerari = new ArrayList<>();
        Itinerario itinerario = new Itinerario();
        itinerario.setNome("Napoli che bella");
        itinerario.setDifficoltà(3);
        itinerario.setDescrizione("una bella passeggiata a napoli");
        itinerario.setDurata("13");
        itinerario.setAccessibilitaDisabili(false);
        GeoPoint punto = new GeoPoint(40.85157508957582,
                14.26479657279487);
        ArrayList<GeoPoint> waypoints = new ArrayList<>();
        waypoints.add(punto);
        itinerario.setWaypoints(waypoints);
        itinerari.add(itinerario);
        ControllerCerca controllerCerca = new ControllerCerca(null,
```

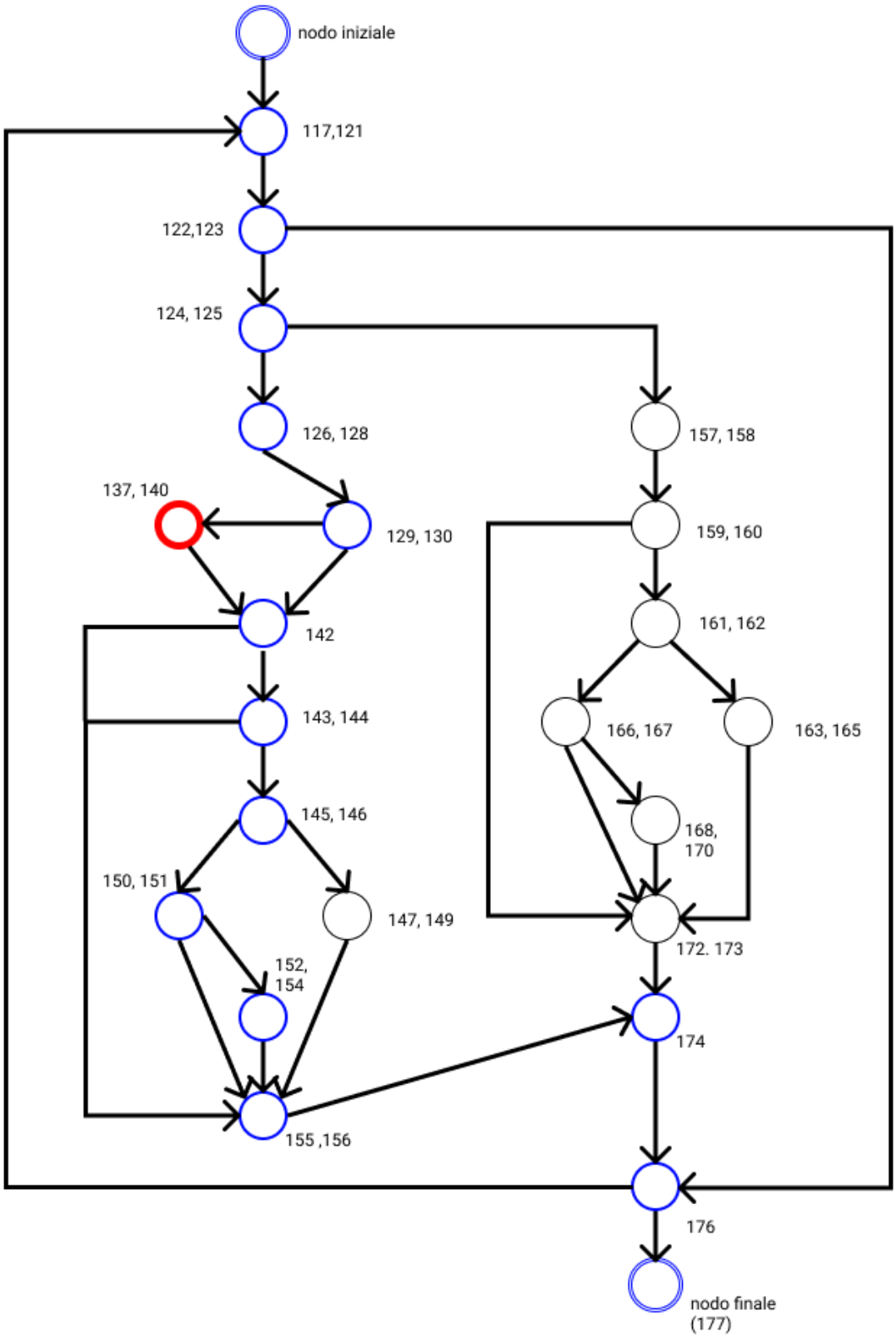
```
                null, itinerari, null);
        Boolean exist;
        exist = controllerCerca.filterResult("xxxxxxxx",
                3,"13",false, appContext);
        assertFalse(exist);

    }

    @Test
    public void testfilterResultWithoutDurata()
    {
        Context appContext = InstrumentationRegistry
                .getInstrumentation().getTargetContext();
        ArrayList<Itinerario> itinerari = new ArrayList<>();
        Itinerario itinerario = new Itinerario();
        itinerario.setNome("Napoli che bella");
        itinerario.setDifficoltà(3);
        itinerario.setDescrizione("una bella passeggiata a napoli");
        itinerario.setDurata("13");
        itinerario.setAccessibilitaDisabili(false);
        GeoPoint punto = new GeoPoint(40.85157508957582, 14.26479657279487);
        ArrayList<GeoPoint> waypoints = new ArrayList<>();
        waypoints.add(punto);
        itinerario.setWaypoints(waypoints);
        itinerari.add(itinerario);
        ControllerCerca controllerCerca = new ControllerCerca(null,
                null, itinerari, null);
        Boolean exist;
        exist = controllerCerca.filterResult("napoli",
                3,"",false, appContext);
        assertTrue(exist);

    }

    @Test
    public void testfilterResultWithoutDurataNorAddress()
    {
        Context appContext = InstrumentationRegistry
                .getInstrumentation().getTargetContext();
        ArrayList<Itinerario> itinerari = new ArrayList<>();
        Itinerario itinerario = new Itinerario();
        itinerario.setNome("Napoli che bella");
        itinerario.setDifficoltà(3);
        itinerario.setDescrizione("una bella passeggiata a napoli");
        itinerario.setDurata("13");
        itinerario.setAccessibilitaDisabili(false);
        GeoPoint punto = new GeoPoint(40.85157508957582,
                14.26479657279487);
        ArrayList<GeoPoint> waypoints = new ArrayList<>();
        waypoints.add(punto);
        itinerario.setWaypoints(waypoints);
        itinerari.add(itinerario);
        ControllerCerca controllerCerca = new ControllerCerca(null,
                null, itinerari, null);
        Boolean exist;
        exist = controllerCerca.filterResult("",
                3,"",false, appContext);
```

```
        assertTrue(exist);

    }

}
```

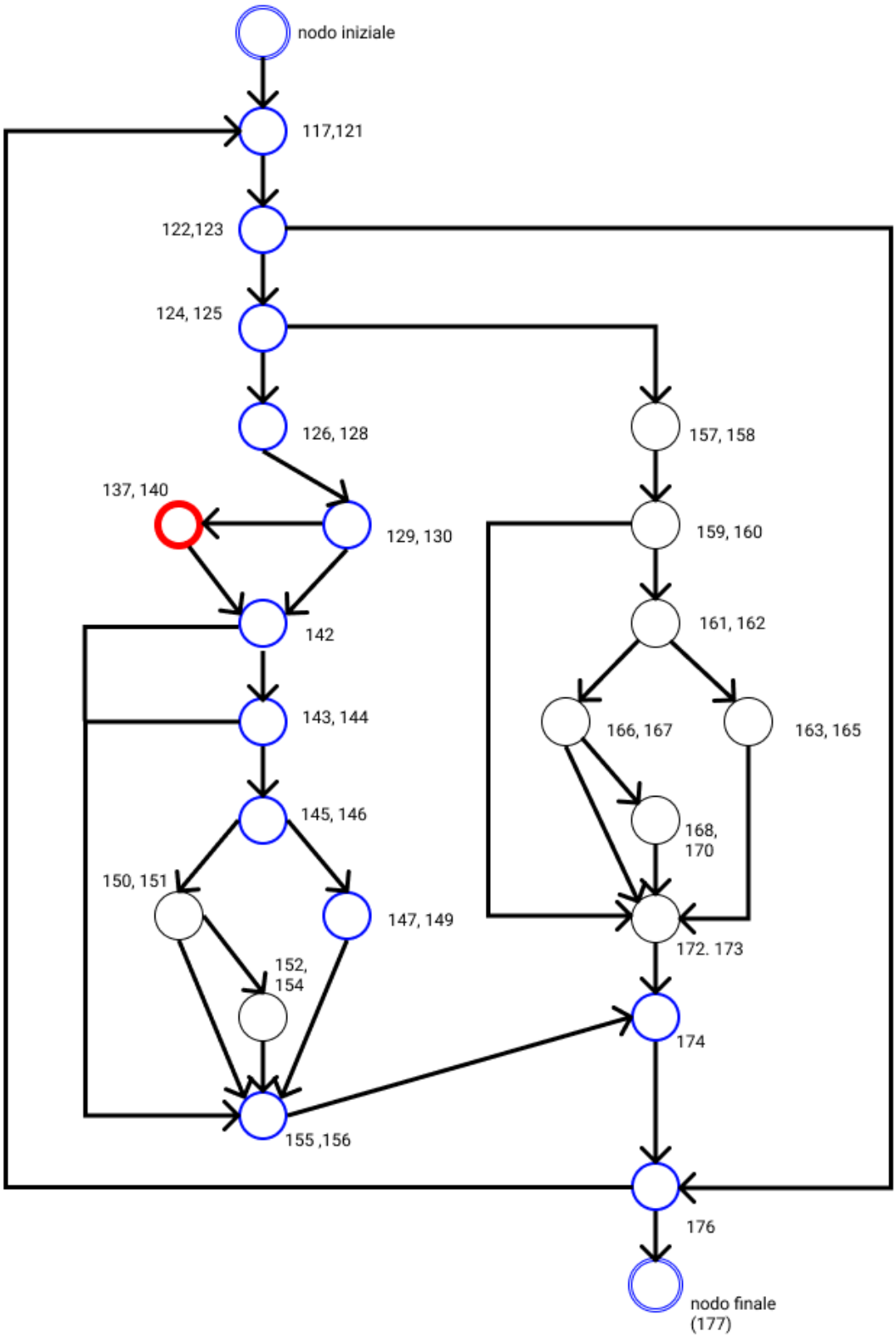
GFC (Grafo del Flusso di Controllo) per il test “testfilterResultWrongAddress”



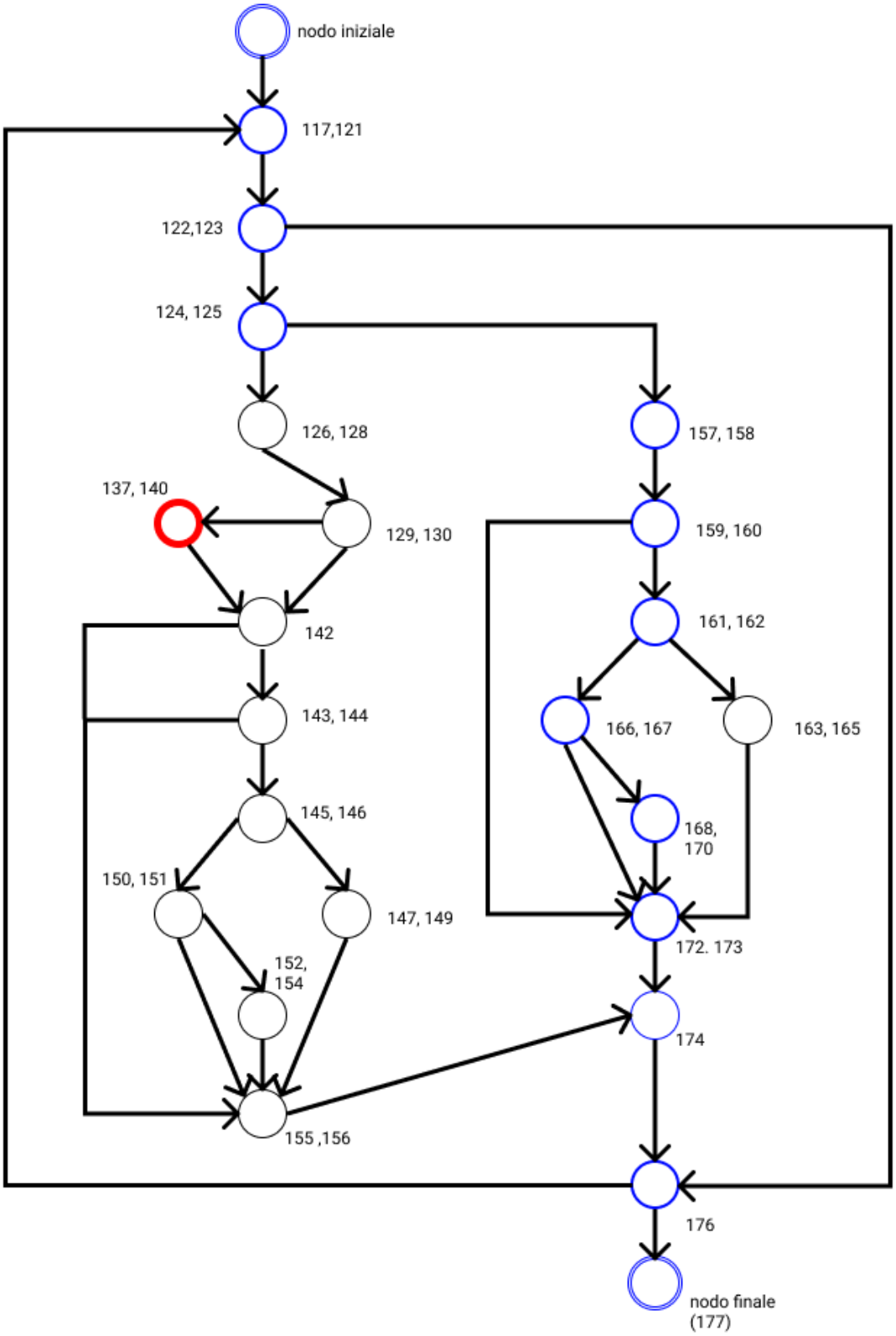
GFC per il test “testfilterResultCorretto”



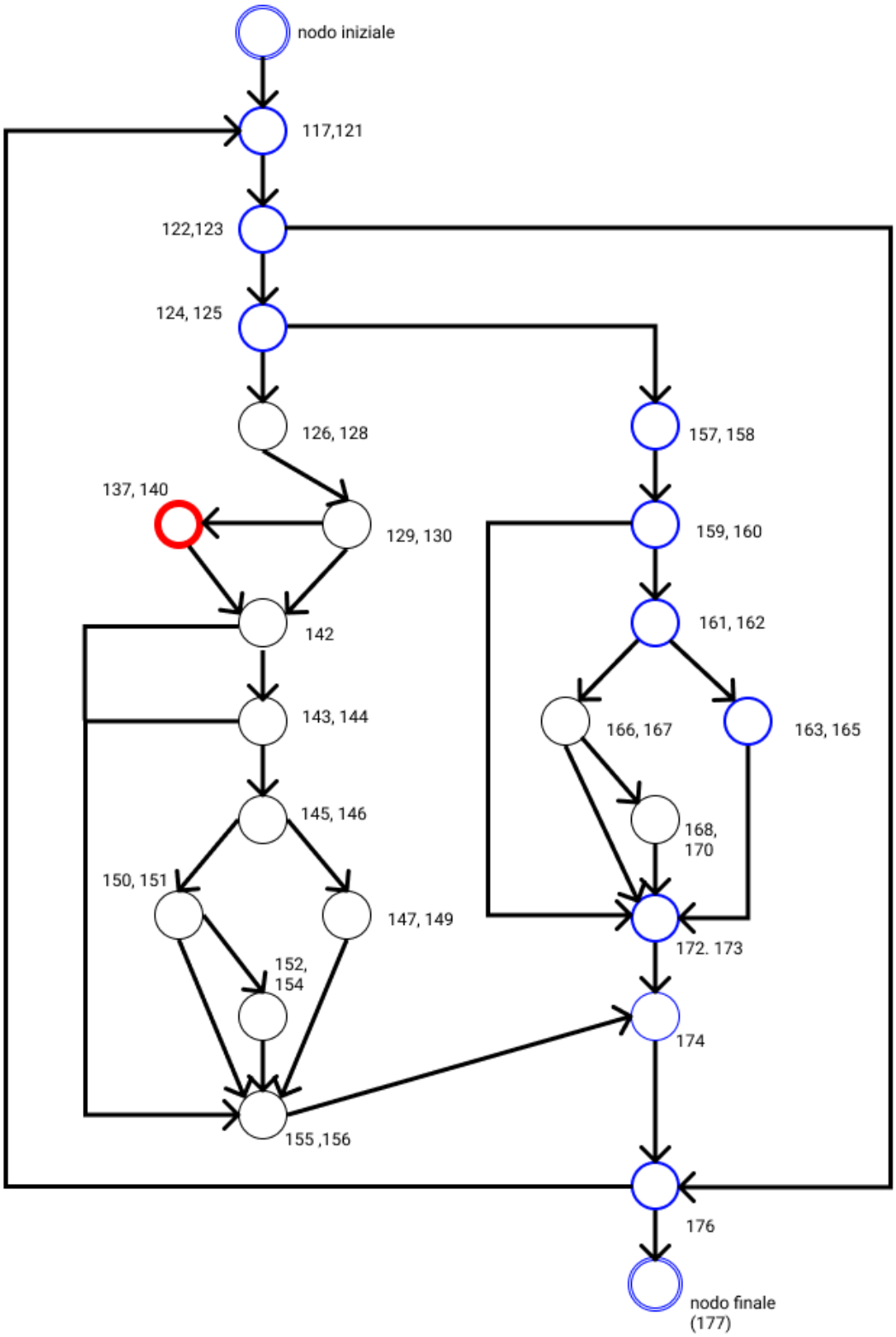
GFC per il test “testfilterResultWithoutDurata”



GFC per il test “testfilterResultWithoutAddress”



GFC per il test “testfilterResultWithoutDurataNorAddress”



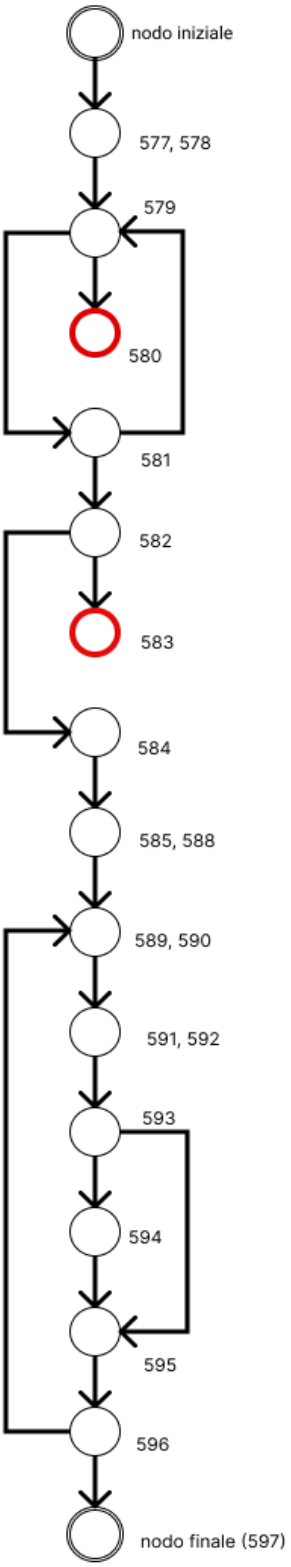
5.2.3 Testing del metodo “isImageValid”

Il testing del metodo in oggetto è stato eseguito con la strategia white box con tecnica di branch coverage. In questo caso, quindi, si cerca di far in modo che ogni possibile arco del GFC sia esaminato, così da capire come si comporta il metodo per ogni possibile “movimento” all’interno del codice (ad esempio se torna indietro per un loop oppure se compie un certo codice invece di un altro con costrutti if else)

Il metodo in oggetto è il seguente (il metodo isLatLonValid serve come supporto per verificare che la latitudine e longitudine siano valori validi)

```
public boolean isImageValid(Immagine immagine, List<GeoPoint> geoPoints)
{
    String exceptionMsg = "Latitudine o longitudine errati!";
    for (GeoPoint geoPoint: geoPoints){
        if(!isLatLonValid(geoPoint.getLatitude(), geoPoint.getLongitude()))
            throw new IllegalPositionException(exceptionMsg);
    }
    if(!isLatLonValid(immagine.getLatitude(), immagine.getLongitude()))
        throw new IllegalPositionException(exceptionMsg);
    float minDistance;
    float[] dist = new float[1];
    Location.distanceBetween(immagine.getLatitude(),immagine.getLongitude(),
            geoPoints.get(0).getLatitude(),geoPoints.get(0).getLongitude(),
dist);
    minDistance = dist[0];
    for (GeoPoint geoPoint: geoPoints)
    {
        Location.distanceBetween(immagine.getLatitude(),immagine.getLongitude(),
                geoPoint.getLatitude(),geoPoint.getLongitude(), dist);
        if(dist[0] < minDistance){
            minDistance = dist[0];
        }
    }
    return minDistance < 300f;
}

private boolean isLatLonValid(double lat, double lng){
    return (lat >= -90f && lat <= 90f) && (lng >= -180f && lng <= 180f);
}
```

GFC del metodo in questione:



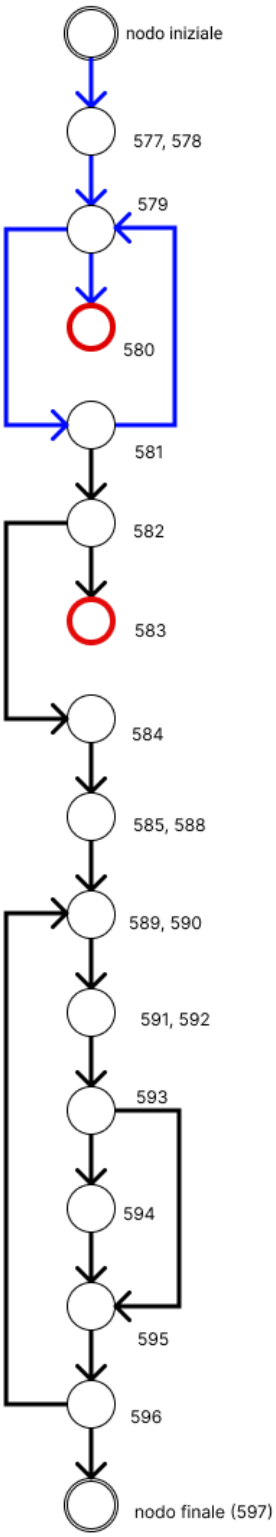
Di seguito il codice per il testing:

```
package com.example.natour.controller;

import static org.junit.Assert.assertTrue;
import static org.junit.Assert.fail;

import com.example.natour.exception.IllegalPositionException;
import com.example.natour.model.Immagine;

import org.junit.Test;
import org.osmdroid.util.GeoPoint;

import java.util.ArrayList;
import java.util.List;

public class ControllerItinerarioTest
{

    /*
     * Questo metodo determina se la distanza tra
     * l'immagine inserita e i punti dell'itinerario
     * sono effettivamente vicini (100 metri di differenza) tra loro.
     * */

    /*
     * Immagine ha dei campi latitudine e longitudine
     *
     * I waypoints sono una lista di geopoints
     * con latitudine e longitudine
     * */

    /*il metodo preso in esame verrà testato
    con la metodologia whitebox con
     *
     * strategia di branch coverage */

    /**
     * Branch esaminato: lancia l'eccezione dopo
     * aver confrontato almeno un geopoint
     * correttamente con la posizione dell'immagine
     * */
    @Test
    public void isImageValidGeopointsException(){
        Immagine immagine = new Immagine();
        immagine.setLatitude(15);
        immagine.setLongitude(34);
        GeoPoint geoPoint = new GeoPoint(10f,35f);
        GeoPoint geoPoint1 = new GeoPoint(-300f, -300f);
        ControllerItinerario controllerItinerario =
                new ControllerItinerario(null,
                        null,null);
        List<GeoPoint> waypoints = new ArrayList<>();
        waypoints.add(geoPoint);
        waypoints.add(geoPoint1);
        try{
            controllerItinerario.isImageValid(immagine, waypoints);
```

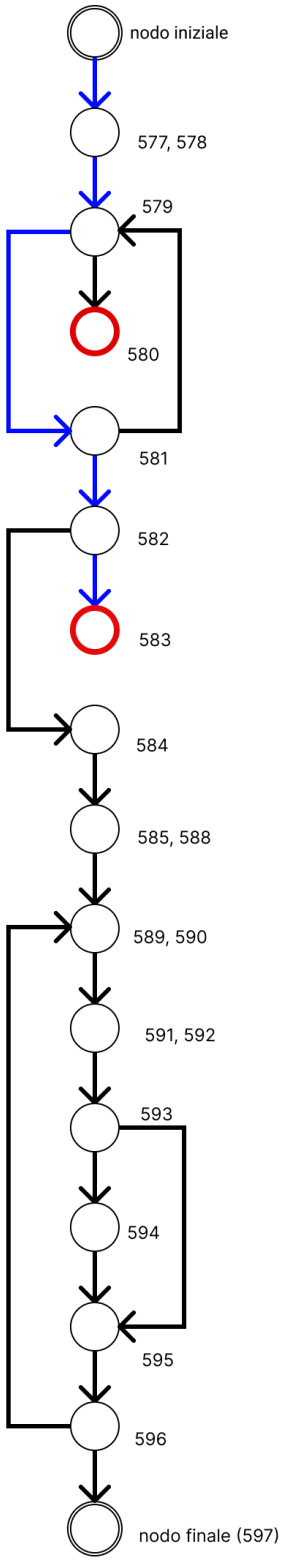
```
            fail();
        }catch (IllegalPositionException e){
            e.printStackTrace();
            assertTrue(true);
        }catch (Exception e){
            fail();
        }
    }

    /**
     * branch esaminato: dopo aver esaminato i
     * geopoints nella lista e averli trovati corretti
     * lancia l'eccezione per la latitudine e
     * longitudine invalidi dell'immagine
     * */
    @Test
    public void isImageValidImageException()
    {
        Immagine immagine = new Immagine();
        immagine.setLatitude(-300);
        immagine.setLongitude(300);
        GeoPoint geoPoint = new GeoPoint(15f,26f);
        ControllerItinerario controllerItinerario =
                new ControllerItinerario(null,
                        null,null);
        List<GeoPoint> waypoints = new ArrayList<>();
        waypoints.add(geoPoint);
        try{
            controllerItinerario.isImageValid(immagine, waypoints);
            fail();
        }catch (IllegalPositionException e){
            e.printStackTrace();
            assertTrue(true);
        }catch (Exception e){
            fail();
        }
    }

/**
 * in questo branch: dopo aver controllato
 * che gli indirizzi sono inseriti correttamente
 * verifica che il secondo punto è più vicino
 * e lo selezione come distanza minima.
 * In fine restituisce true in quanto la distanza
 * tra il punto minore e il punto dell'immagine è
 * inferiore a 300 metri
 * */

    @Test
    public void isImageValidCorretto()
    {

        Immagine immagine = new Immagine();
        immagine.setLatitude(40.899628001261995);
        immagine.setLongitude(14.092006225335721);
        GeoPoint geoPoint = new GeoPoint(40.89839534981892,
                14.092006225335721);
```

```
        GeoPoint geoPoint1 = new GeoPoint(40.89969534981892,
                14.092006225335731);
        ControllerItinerario controllerItinerario =
                new ControllerItinerario(null,
                        null,null);
        List<GeoPoint> waypoints = new ArrayList<>();
        waypoints.add(geoPoint);
        waypoints.add(geoPoint1);
        assertTrue(controllerItinerario
                .isImageValid(immagine, waypoints));
    }
}
```

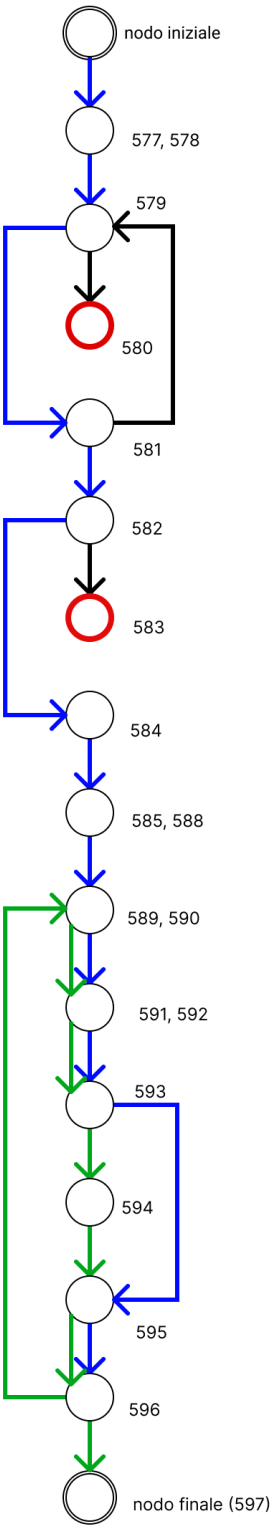
GFC per il test “isImageValidGeopointsException”



GFC per il test “isImageValidImageException”



GFC per il test “isImageValidCorretto”



Questo è il logo dell'applicazione sul dispositivo.

Grazie per la vostra attenzione.