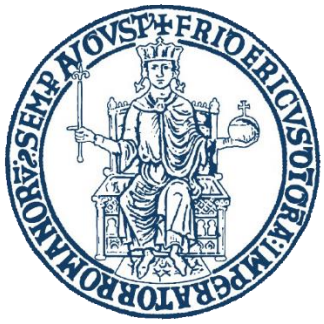


UNIVERSITÀ DEGLI STUDI
DI NAPOLI FEDERICO II

CORSO DI LAUREA IN INFORMATICA
INSEGNAMENTO DI INGEGNERIA DEL SOFTWARE

ANNO ACCADEMICO 2021/2022

DOCUMENTAZIONE PIATTAFORMA

NATOUR21

Di Maio Aurora – N86002829
Fonseta Mike - N86003410

Indice

Capitolo I. Documento dei Requisiti Software

Introduzione

NaTour21 è un sistema basato su un client mobile volto ad offrire un nuovo social network per gli appassionati di escursionismo.

Il sistema offre diverse funzionalità, tra le quali quella di accesso/registrazione tramite piattaforme Google e Facebook.

Darà la possibilità all'utente di inserire in piattaforma i sentieri che ha percorso condividendoli così, attraverso la homepage, a tanti altri appassionati di escursionismo. Questi avranno la possibilità di visionare il sentiero e di poter lasciare una propria recensione.

Il sistema dispone di una chat, favorendo così lo scambio di informazioni circa un itinerario tra gli utenti.

Gli utenti potranno inoltre segnalare un sentiero qualora le informazioni inserite dovessero essere scorrette.
Il proprietario del post del sentiero segnalato riceverà una notifica e potrà poi rispondere all'utente che ha effettuato la segnalazione.

1 Modello funzionale

I requisiti fondamentali che il sistema deve offrire sono:

Requisiti Funzionali

Registrazione alla piattaforma:

Il sistema deve offrire la possibilità ad un utente che lo desidera di potersi registrare alla piattaforma utilizzando il servizio NaTour21 o servizi esterni (Google, Facebook).

Accesso alla piattaforma:

Il sistema deve offrire la possibilità ad un utente registrato di poter accedere alla piattaforma utilizzando le credenziali inserite in fase di registrazione.
L'utente potrà inoltre decidere di rimanere connesso in piattaforma utilizzando la funzione "Ricordami".

Recupero Password:

Il sistema deve offrire la possibilità di recupero password indicando l'e-mail utilizzata in fase di registrazione.

Inserimento Post:

Il sistema deve offrire la possibilità di poter inserire in piattaforma degli itinerari, sotto forma di post.
I quali conterranno:
-Titolo del sentiero
-Descrizione opzionale del sentiero
-Durata del sentiero
-Difficoltà del sentiero
-Punto di inizio del sentiero
-Tracciato geografico opzionale inseribile manualmente tramite interazione con la mappa o tramite inserimento di un file .GPX.

Modifica durata/difficoltà:

Il sistema deve offrire la possibilità di poter modificare la durata e/o difficoltà di un sentiero, già inserito in piattaforma, se un utente lo ritiene necessario.
La durata/difficoltà di tale sentiero verranno poi ricalcolati come la media di tutte le durate e difficoltà inserite dagli utenti.

Inserimento Recensione:

Il sistema deve offrire la possibilità di poter inserire una recensione inerente ad un post (sentiero).
La recensione deve contenere un punteggio da 1 a 5 ed una descrizione testuale.

Inserimento Segnalazione:

Il sistema deve offrire la possibilità di poter inserire una segnalazione inerente ad un post (sentiero).
La segnalazione deve contenere una descrizione testuale ed un titolo.

Riposta Segnalazione:

Il sistema deve offrire la possibilità, all'utente proprietario di un post, di poter rispondere alle segnalazioni ricevute.

Visualizzazione dettagli Post:

Il sistema deve offrire la possibilità di poter visualizzare i dettagli inerenti un post (sentiero).
I dettagli devono contenere:
-Descrizione completa
-Durata del sentiero
-Difficoltà del sentiero
-Recensioni se presenti.
-Alert di segnalazioni attive se presenti

Invio e risposte messaggi chat:

Il sistema deve offrire la possibilità di poter inviare e rispondere a messaggi utilizzando le funzionalità della Chat.

Non funzionali

Limitazione recensioni:

Il sistema permette di recensire un post una singola volta per utenti non proprietari del post. Se l'utente è proprietario del post il sistema non permette di lasciare recensioni.

Limitazione segnalazioni:

Il sistema permette di segnalare un post una singola volta per utenti non proprietari del post. Se l'utente è proprietario del post il sistema non permette di lasciare segnalazioni.

Regole Password (Criteri):

Il sistema impone all'utente di inserire una password di almeno 6 caratteri.

Regole Nome Utente (Criteri):

Il sistema impone all'utente di inserire un nome utente, privo di caratteri speciali, di almeno 4 caratteri minuscoli, fino ad un massimo di 32 caratteri.

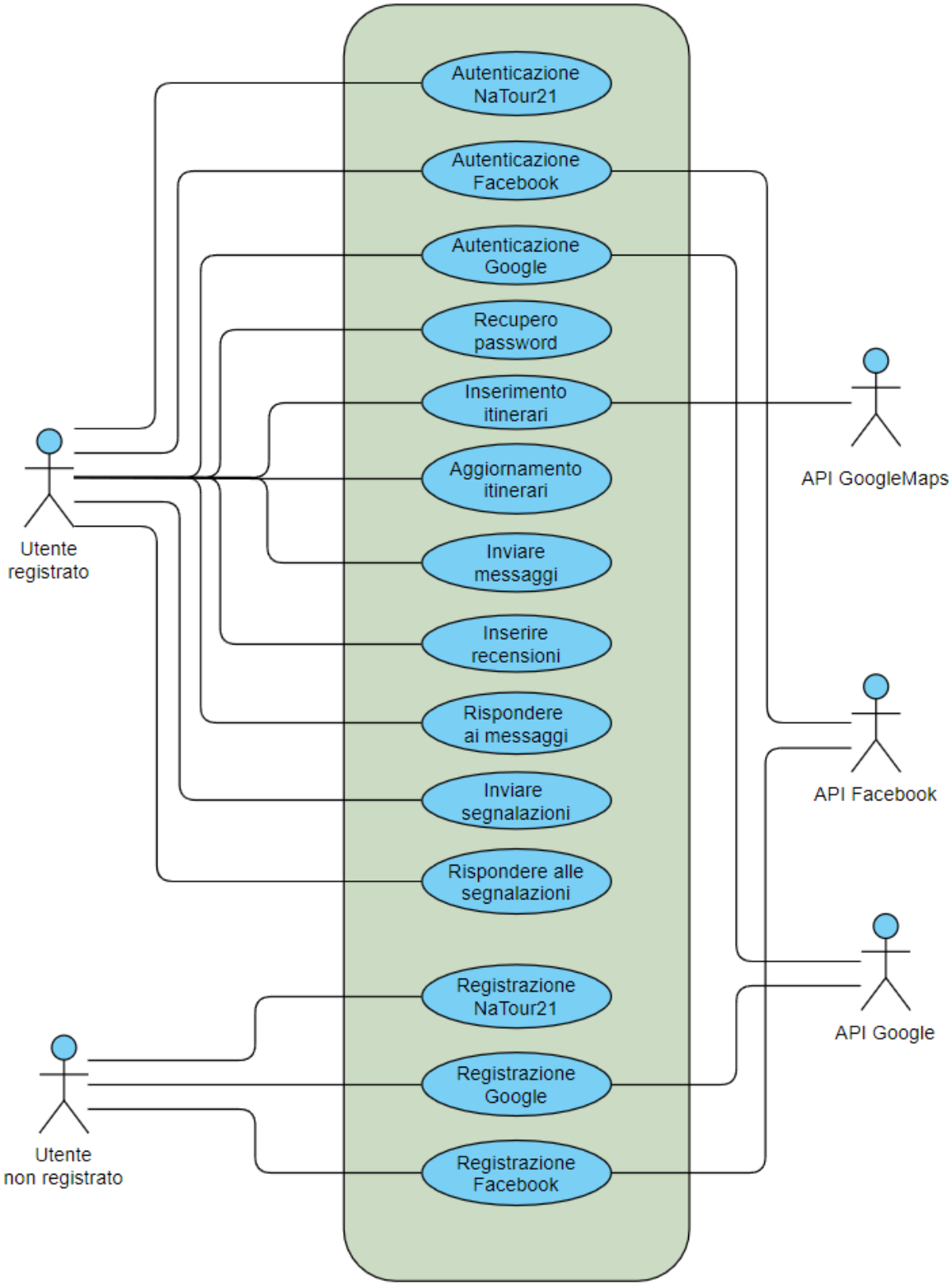
Regole E-Mail (Criteri):

Il sistema controlla la correttezza dell'e-mail e non permette l'inserimento nel caso la correttezza non venga rispettata.

Regole Messaggi Chat:

Il sistema non permette l'invio di messaggi tramite l'utilizzo della chat a sé stessi o ad utenti non registrati.

1.1 Modellazione dei casi d’uso



1.2 Tabelle di Cockburn per due casi d'uso significativi

Riportiamo le tabelle di Cockburn per i casi d'uso di **Inserimento Itinerario** e **Inserimento Recensione**

1.2.1 Inserimento Itinerario

USE CASE	Inserimento itinerario		
Goal in Context	L'utente vuole inserire un itinerario in piattaforma		
Preconditions	L'utente deve aver effettuato l'accesso		
Success End Condition	L'utente pubblica un itinerario		
Failed End Condition	L'utente non pubblica un itinerario. Chiude la schermata di inserimento o c'è un errore durante l'inserimento.		
Primary Actor	Utente		
Trigger	L'utente preme il pulsante "Inserisci un nuovo itinerario…" nella schermata HomePage		
DESCRIPTION	**Step n°**	**Utente**	**Sistema**
	1	Preme il pulsante "Inserisci un nuovo itinerario…" nella schermata HomePage	
	2		Mostra la schermata di inserimento itinerario
	3	Inserisce un nome, durata, difficoltà, punto di inizio, descrizione e tracciato geografico	
	4	Preme il pulsante "PUBBLICA"	
	5		Salva l'itinerario
	6		Torna alla schermata HomePage
	7		Mostra la schermata Avviso
	8	Preme il pulsante "OK" della schermata Avviso	

EXTENSIONS #1 L'utente preme il pulsante "Back" annullando l'operazione di inserimento itinerario.	**Step n°**	**Utente**	**Sistema**
	4a	Preme il pulsante "Back"	
	5a		Torna alla schermata HomePage
EXTENSIONS #2 L'utente preme il pulsante "PUBBLICA" ma l'inserimento non va a buon fine.	5a		Non salva l'itinerario
	6a		Mostra la schermata Avviso
	7a	Preme il pulsante "OK" della schermata Avviso	
SUBVARIATIONS # 1 L'utente non compila i/il campi/o opzionali/e "descrizione" e/o "tracciato geografico"	*Step*	**Utente**	**Sistema**
	3a	L'utente lascia il campo "descrizione" e/o "tracciato geografico vuoto	

1.2.2 Inserimento Recensione

USE CASE	Inserimento recensione		
Goal in Context	L'utente vuole inserire una recensione per un itinerario esistente		
Preconditions	L'utente deve aver effettuato l'accesso		
Success End Condition	L'utente pubblica una recensione per un itinerario		
Failed End Condition	L'utente non pubblica una recensione. Chiude la schermata di inserimento o c'è un errore durante l'inserimento.		
Primary Actor	Utente		
Trigger	L'utente preme il pulsante "AGGIUNGI RECENSIONE" nella schermata DettagliSentiero		
DESCRIPTION	**Step n°**	**Utente**	**Sistema**
	1	Preme il pulsante "AGGIUNGI RECENSIONE" nella schermata DettagliSentiero	
	2		Mostra la schermata di inserimento recensione
	3	Inserisce commento e valutazione	
	4	Preme il pulsante "PUBBLICA"	
	5		Salva la recensione
	6		Mostra la schermata Avviso con messaggio "Recensione pubblicata"
	7	Preme il pulsante "OK" della schermata Avviso	
	8		Torna alla schermata DettagliSentiero

	Step n°	Utente	Sistema
EXTENSIONS #1 L'utente preme il pulsante "Back" annullando l'operazione di inserimento recensione.	4a	Preme il pulsante "Back"	
	5a		Torna alla schermata DettagliSentiero
EXTENSIONS #2 L'utente preme il pulsante "PUBBLICA" ma l'inserimento non va a buon fine.	5a		Non salva la recensione
	6a		Mostra la schermata Avviso con messaggio "Non è stato possibile inserire la recensione"
	7a	Preme il pulsante "OK" della schermata Avviso	

1.3 Prototipazione visuale via Mock-Up per due casi d'uso significativi

Verranno di seguito mostrati i mock-up per i due casi d'uso significativi esplicitati al punto 1.2

1.3.1 Schermata Inserimento Itinerario

Componente 1	
Tipo	Pulsante
Funzionalità	Pubblica l'itinerario e mostra la schermata di Avviso
Componente 2	
Tipo	Pulsante
Funzionalità	Ritorna alla schermata precedente di Homepage
Componente 3	
Tipo	Campo testuale
Funzionalità	Inserimento nome itinerario
Componente 4	
Tipo	Campo testuale
Funzionalità	Inserimento durata itineriario
Componente 5	
Tipo	Drop Down List
Funzionalità	Inserimento difficoltà itinerario
Componente 6	
Tipo	Campo testuale
Funzionalità	Inserimento punto di inizio itinerario
Componente 7	
Tipo	Campo testuale
Funzionalità	Inserimento descrizione itinerario
Componente 8	
Tipo	Campo ricerca
Funzionalità	Ricerca sulla mappa di un punto
Componente 9	
Tipo	Pulsante
Funzionalità	Elimina i punti inseriti dalla mappa
Componente 10	
Tipo	Pulsante
Funzionalità	Fornisce informazioni sui punti da inserire sulla mappa
Componente 11	
Tipo	Schermata
Funzionalità	Mostra avviso di successo/errore
Componente 12	
Tipo	Pulsante
Funzionalità	Ritorna alla schermata precedente di Homepage

1.3.2 Schermata Inserimento Recensione

Componente 1	
Tipo	Pulsante
Funzionalità	Pubblica la recensione e mostra la schermata Avviso
Componente 2	
Tipo	Pulsante
Funzionalità	Ritorna alla schermata precedente DettagliSentiero
Componente 3	
Tipo	Campo testuale
Funzionalità	Mostrare all'utente il nome dell'itinerario che si sta recensendo
Componente 4	
Tipo	Campo testuale
Funzionalità	Inserimento commento recensione
Componente 5	
Tipo	Campo testuale
Funzionalità	Inserimento valutazione itinerario
Componente 6	
Tipo	Schermata
Funzionalità	Mostra avviso di successo/errore
Componente 7	
Tipo	Pulsante
Funzionalità	Ritorna alla schermata precedente DettagliSentiero

1.4 Idea Progettuale

L’idea progettuale alla base di NaTour 21 è quella di fornire un applicativo semplice ed intuitivo da usare.
Ciò è stato possibile tramite l’utilizzo di un’interfaccia grafica dallo stile minimal, lineare e straight-forward.

È stato deciso di adottare questo stile dopo aver considerato l’età media degli utenti dell’applicativo (vedi 1.5).

Questo è stato realizzato attraverso la flessibilità e linearità che comportano i dispositivi mobile.

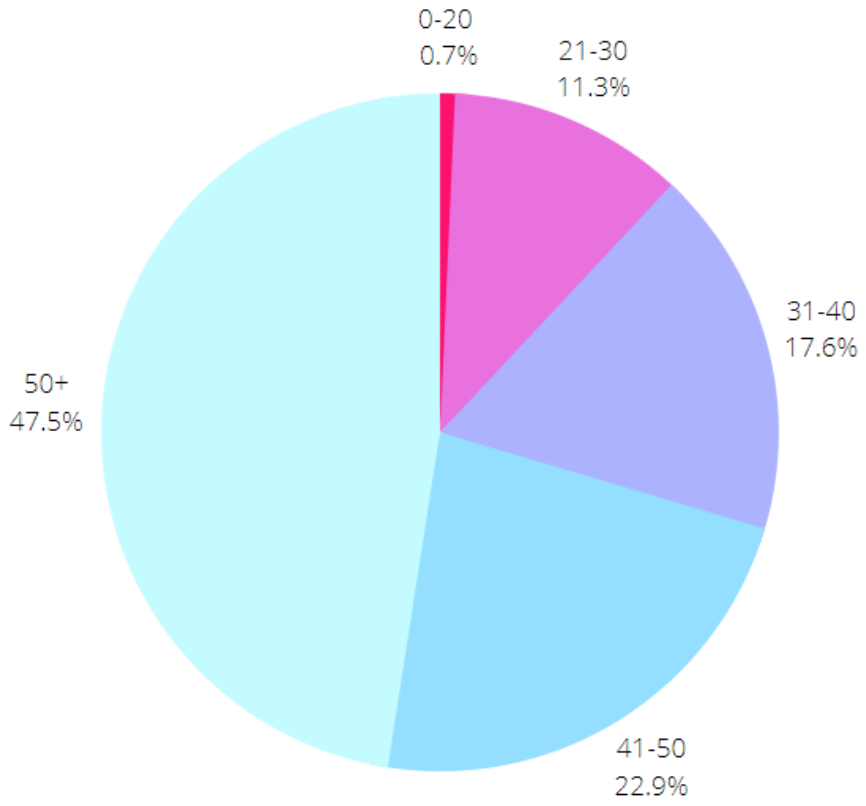
L’utente riuscirà quindi in pochi minuti ad effettuare l’accesso sull’applicativo e pubblicare il suo primo post!

L’utente potrà poi navigare facilmente tra le varie schermate del sistema e poter così accedere alle diverse funzionalità fornite, tra cui: l’inserimento di una recensione, di una segnalazione o la possibilità di avviare una chat con un altro appassionato di escursionismo.

Ricordiamo inoltre la possibilità che l’utente ha di poter recuperare la sua password quando quest’ultima è stata smarrita.
Il processo, ancora una volta, si presenta molto semplice. L’utente, infatti non dovrà fare altro che immettere l’e-mail che ha utilizzato in fase di registrazione.
Gli verrà poi inviata una mail, dal nostro sistema, che conterrà un semplice codice da inserire sull’applicazione per impostare una nuova password.

1.5 Individuazione del target degli utenti

L’individuazione del target degli utenti è stata basata sull’analisi di statistiche reperite online.
Il grafico seguente mostra le percentuali dell’età di chi fa escursionismo.

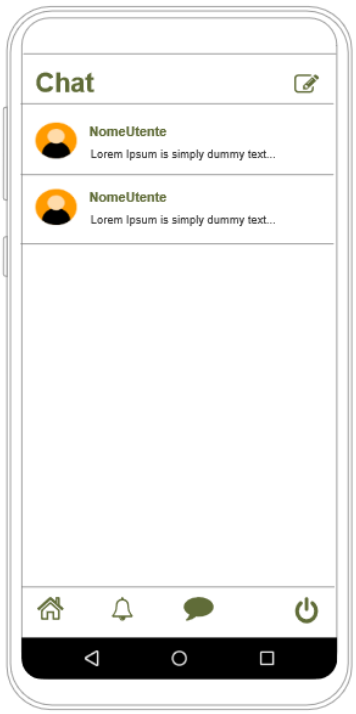


Si può dunque notare come l’età prevalente sia dai 41 anni a salire.
Per questo motivo NaTour21 si presenterà con un’interfaccia grafica semplice, di facile utilizzo alla portata di tutti non tenendo conto solo della fascia di età 18-30 che ad oggi è quella che fa un utilizzo maggiore di social network.

1.6 Valutazione dell'usabilità a priori

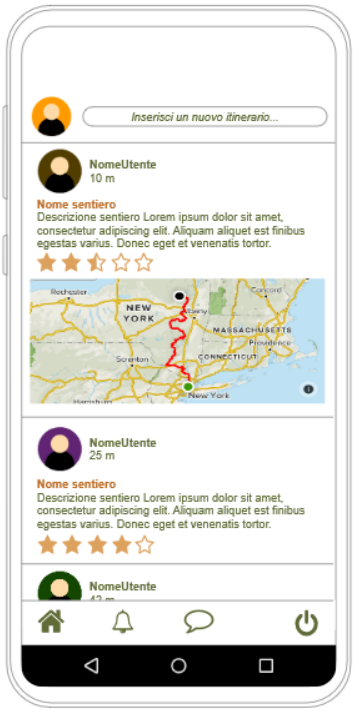
Ai fini della valutazione dell'usabilità sono stati progettati dei Mock-up (non definitivi) con Axure.
I prototipi sono poi stati mostrati a un gruppo di tester, i quali hanno dovuto valutare e comprendere il potenziale funzionamento e la logica di tali schermate.
Di seguito riportiamo alcuni dei prototipi mostrati ai tester e le loro valutazioni.

MK-01



MK-02

MK-03



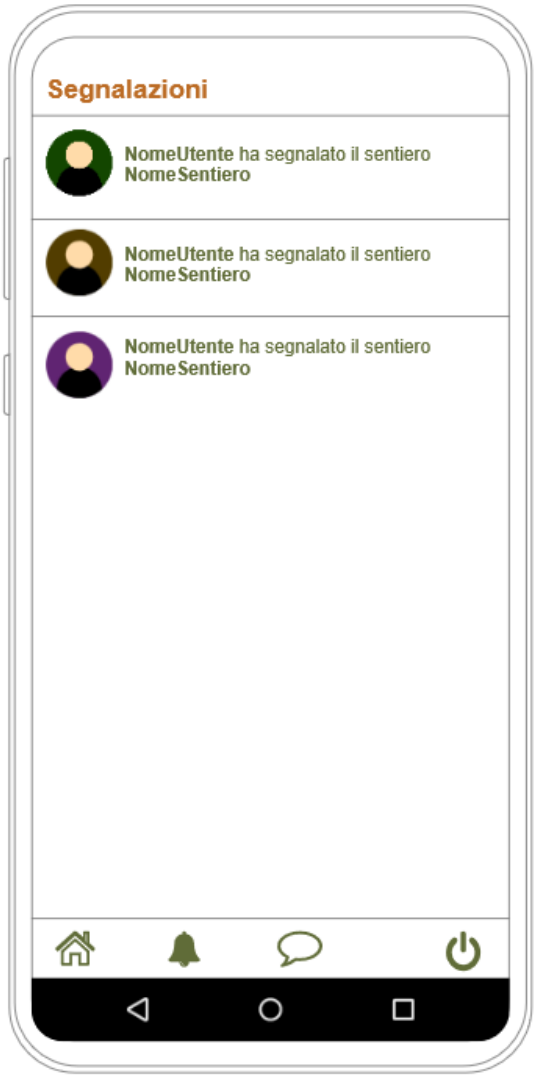
- **MK-01**: nessuna perplessità o dubbio riguardo l'utilizzo di tale schermata ma è stato suggerita la comodità dell'avere l'icona  (homepage che permette di andare alla schermata **MK-03**) nella zona centrale della barra di navigazione inferiore.

- **MK-02**: in questa schermata si sono presentate differenti domande. Come inserisco l'itinerario? Come cerco un luogo sulla mappa? Mi sposto manualmente? Oppure il pulsante INSERISCI TRACCIATO mi permette di aprire la mappa e selezionare un indirizzo?

  L'inserimento di un tracciato, dunque non si è mostrato così semplice e chiaro come credevamo e queste valutazioni ci hanno permesso di perfezionare al meglio tale schermata al fine di renderla più chiara possibile e soprattutto di facile utilizzo.

- **MK-03**: nessuna perplessità ne dubbio riguardo l'utilizzo di tale schermata ma è stato suggerito la comodità dell'avere l'icona  (homepage che permette di andare alla schermata **MK-03**) nella zona centrale della barra di navigazione inferiore.

MK-04



Un'altra valutazione importante è quella dell'icona (segnalazioni che permette di andare alla schermata **MK-04**) nelle schermate **MK-01, MK-02, MK-03**.

MK-04 è un prototipo di una schermata in cui l'utente potrà vedere le segnalazioni inviate o ricevute e nient'altro.

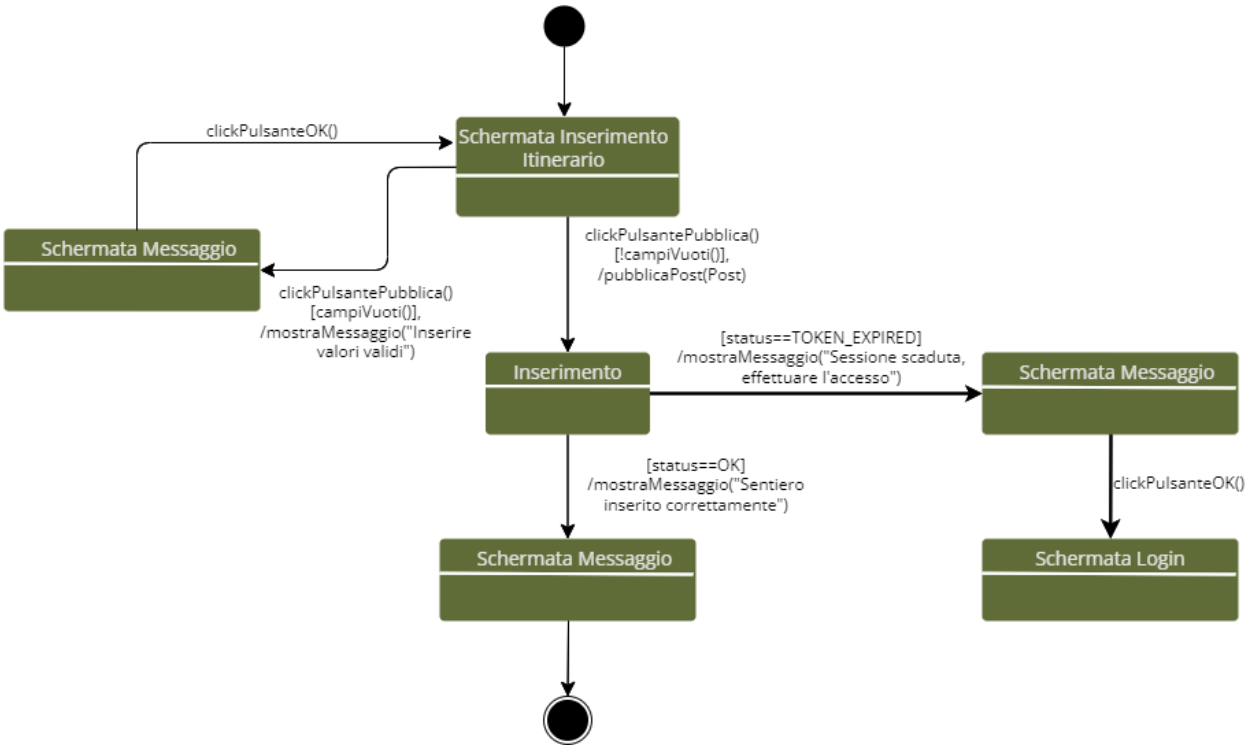
I tester hanno ritenuto che tale icona ( )non rispecchiasse il contenuto e l'utilizzo di tale schermata ma lasciasse intendere tale schermata come area di notifiche generali.

Al seguito di queste valutazioni abbiamo dunque pensato ad un'icona che rispecchiasse al meglio la schermata **MK-04**

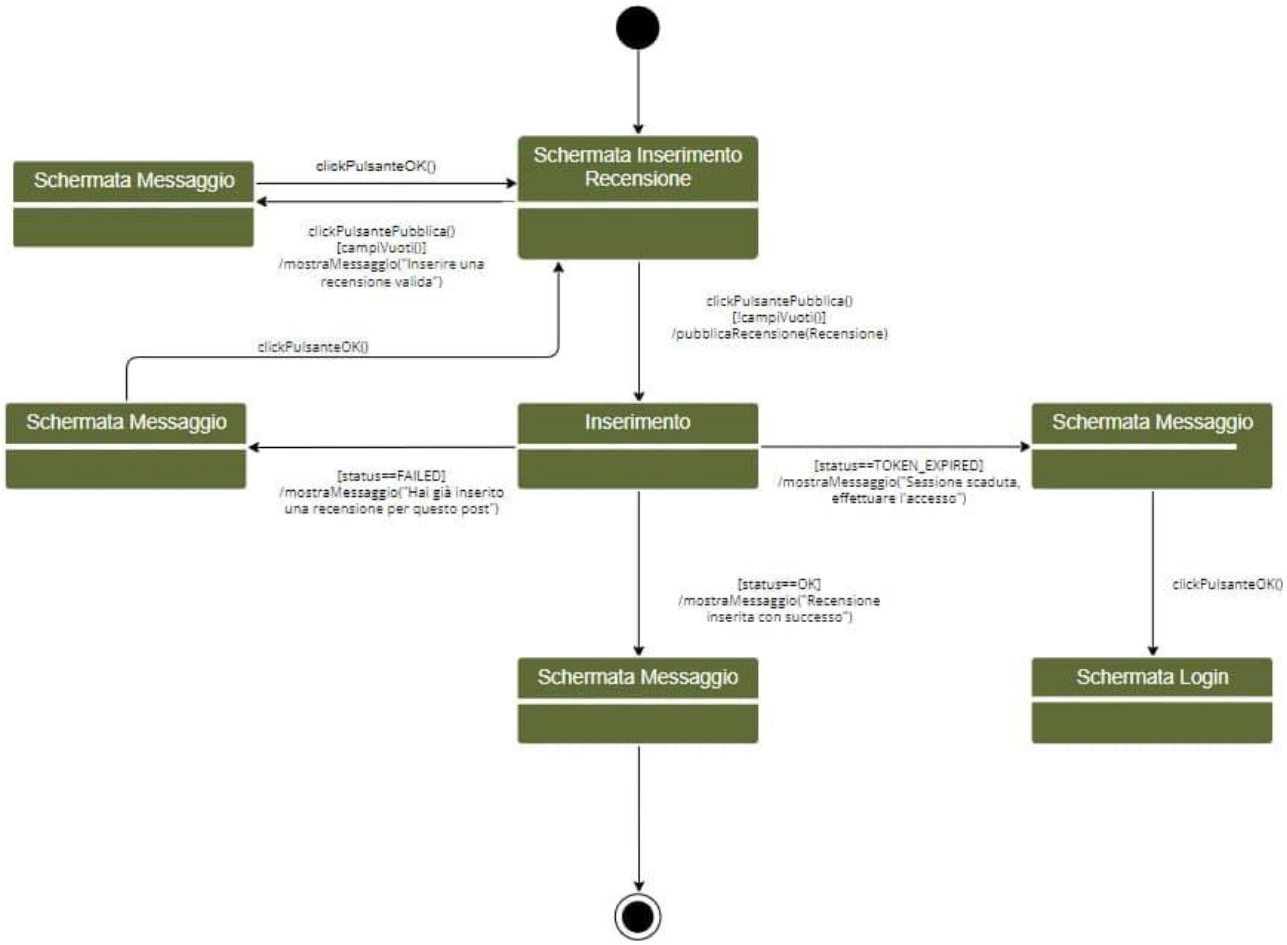
1.7 Statechart di interfaccia grafica

Riportiamo gli statechart per le interfacce grafiche di **Inserimento Itinerario** e **Inserimento Recensione** viste al punto 1.3

1.7.1 Inserimento itinerario



1.7.2 Inserimento recensione



1.8 Glossario

	Termine	Descrizione
1	*Utente*	Indentifica colui che utilizza l'applicazione ed è registrato in piattaforma
2	*Sentiero*	Percorso calcolato dopo l'inserimento in piattaforma da parte dell'utente utilizzando la mappa integrata
3	GPS	Global Position System, sistema per la determinazione di coordinate geografiche relative ad un punto
4	GPX	GPS exchange Format un tipo di formato caratterizzato da uno schema XML realizzato per il trasferimento di dati tra dispositivi GPS e applicazioni software
5	Token	Autorizza un utente che ha effettuato l'accesso a comunicare col server. Valido per due settimane. Dopo due settimane, sarà necessario effettuare di nuovo l'accesso per ottenere un nuovo token

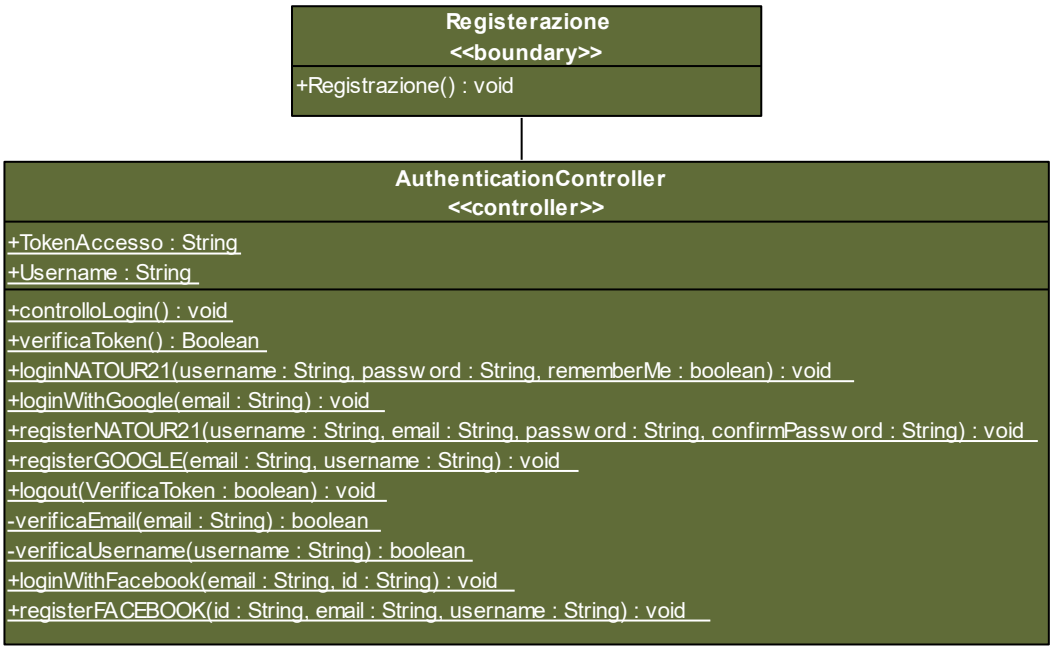
2 Modelli di Dominio

2.1 Classi, oggetti e relazioni di analisi

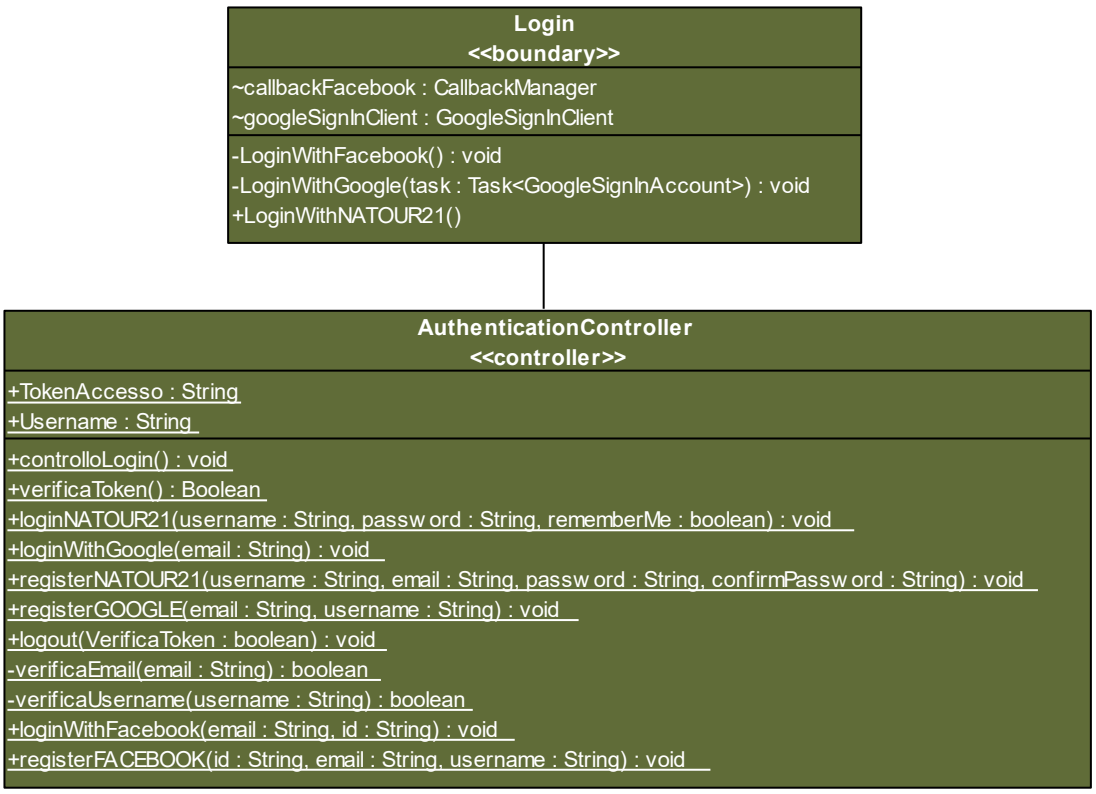
Elencati di seguito vi sono i Class Diagram relativi alle specifiche che l'applicativo deve possedere.
I seguenti Class Diagram sono stati distinti in 3 categorie:

- **Entity** – Rappresentano l'informazione di un tale dato
- **Boundary** – Rappresentano le interazioni tra utente e applicativo
- **Controller** – Rappresentano la logica che lega le precedenti categorie

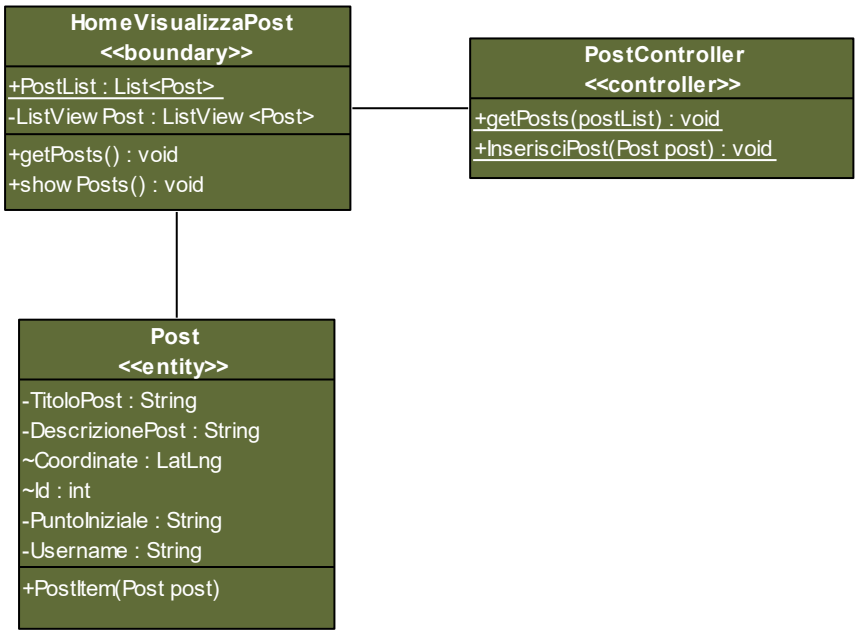
2.1.1 Registrazione



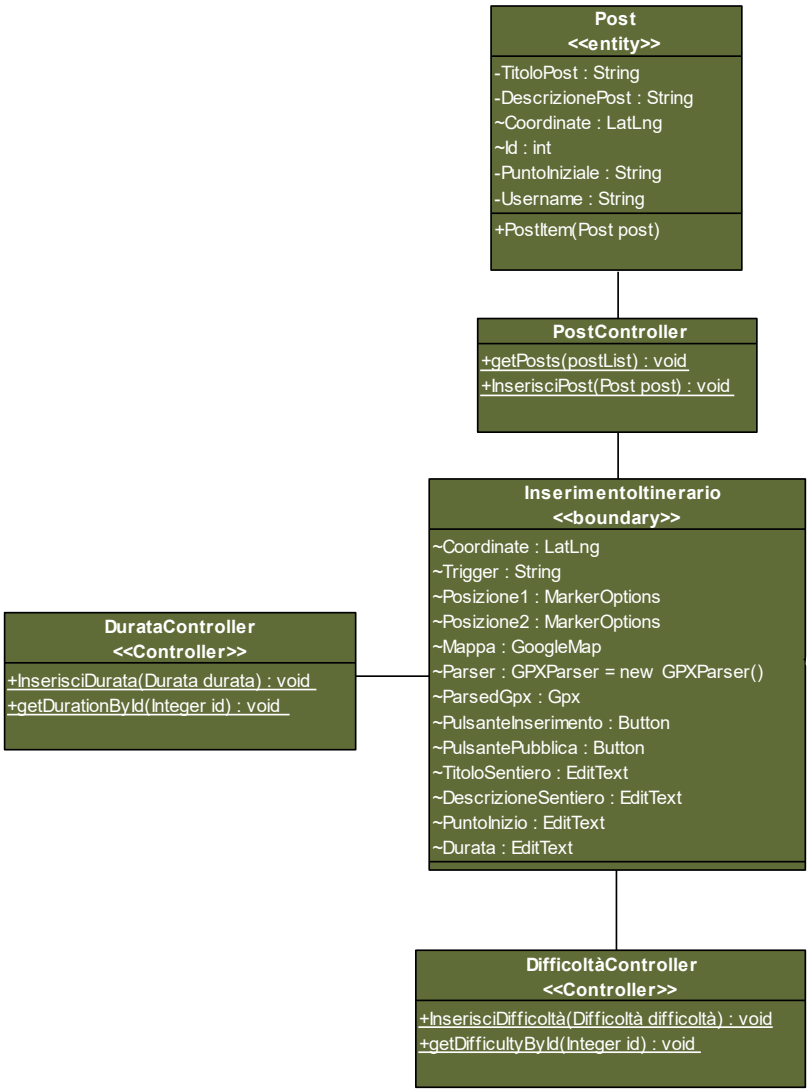
2.1.2 Login



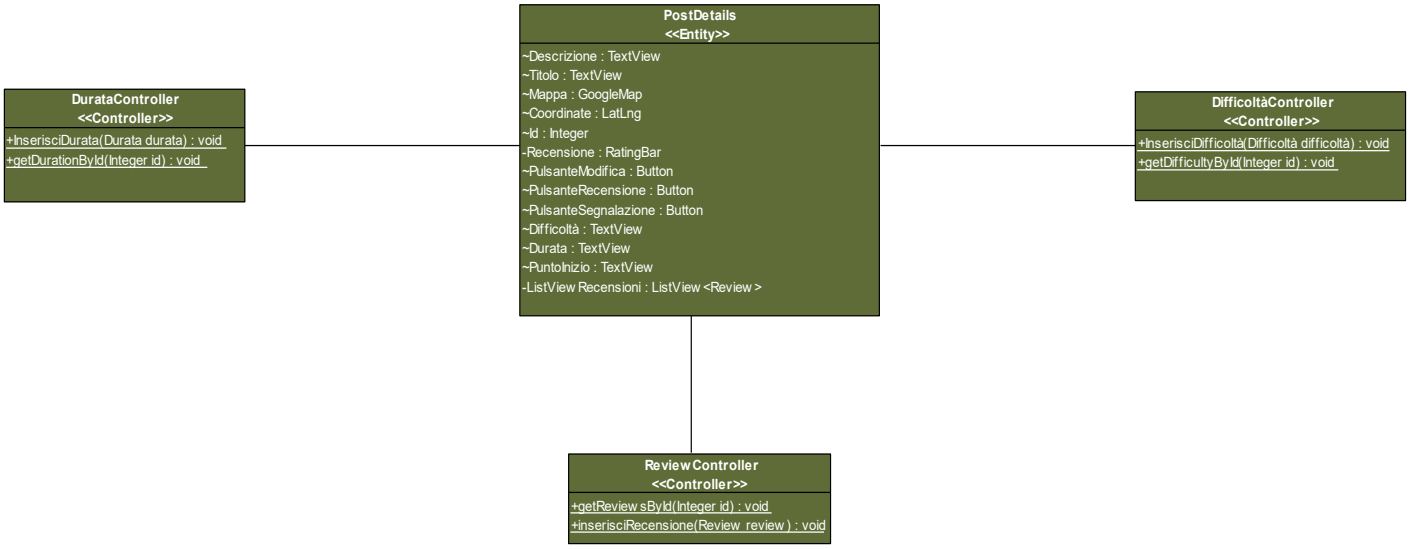
2.1.3 Home



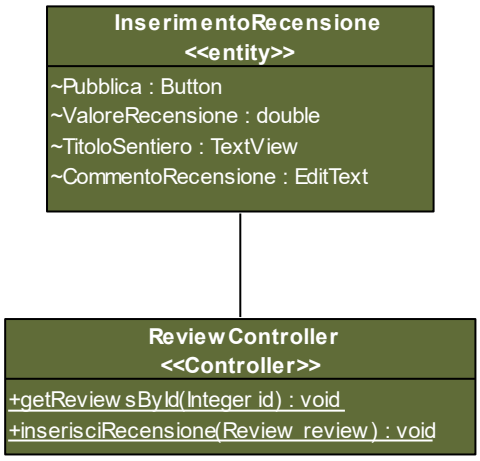
2.1.4 Inserimento Itinerario



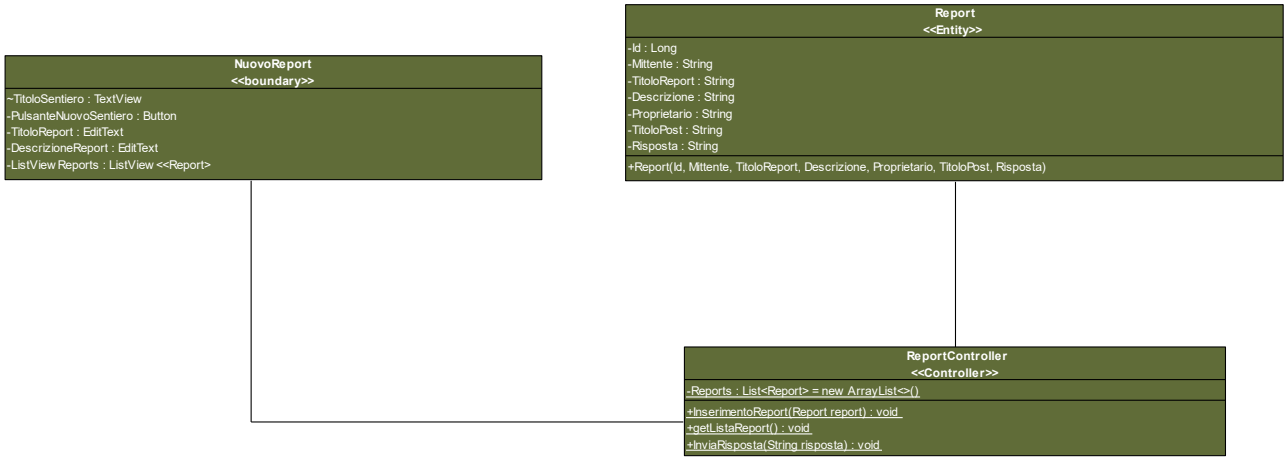
2.1.5 Dettagli Post e modifica durata/difficoltà



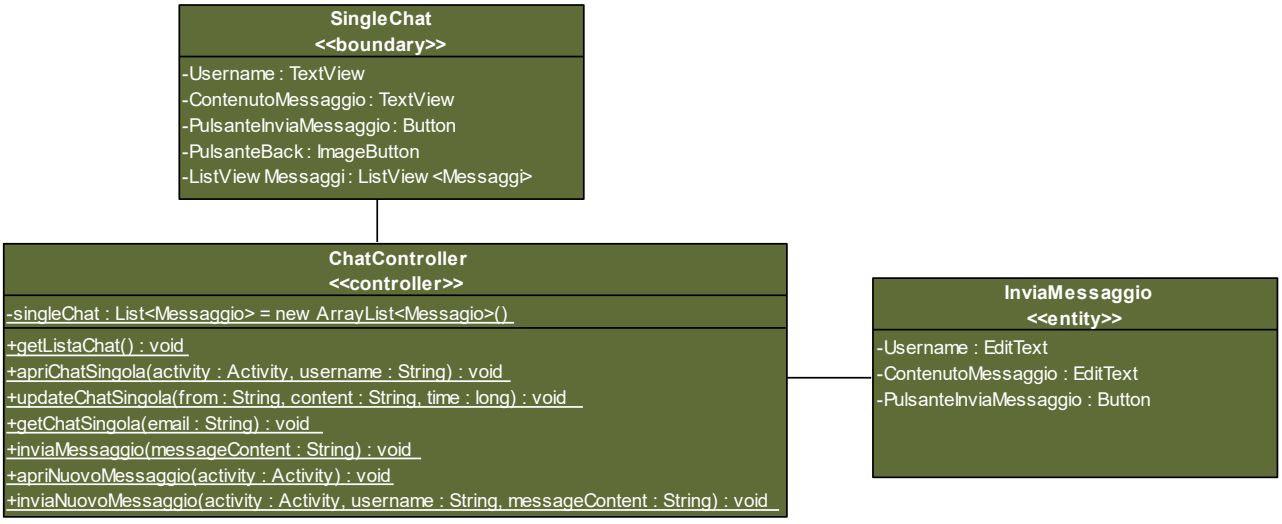
2.1.6 Inserimento Recensione



2.1.7 Inserimento Report e visualizza report



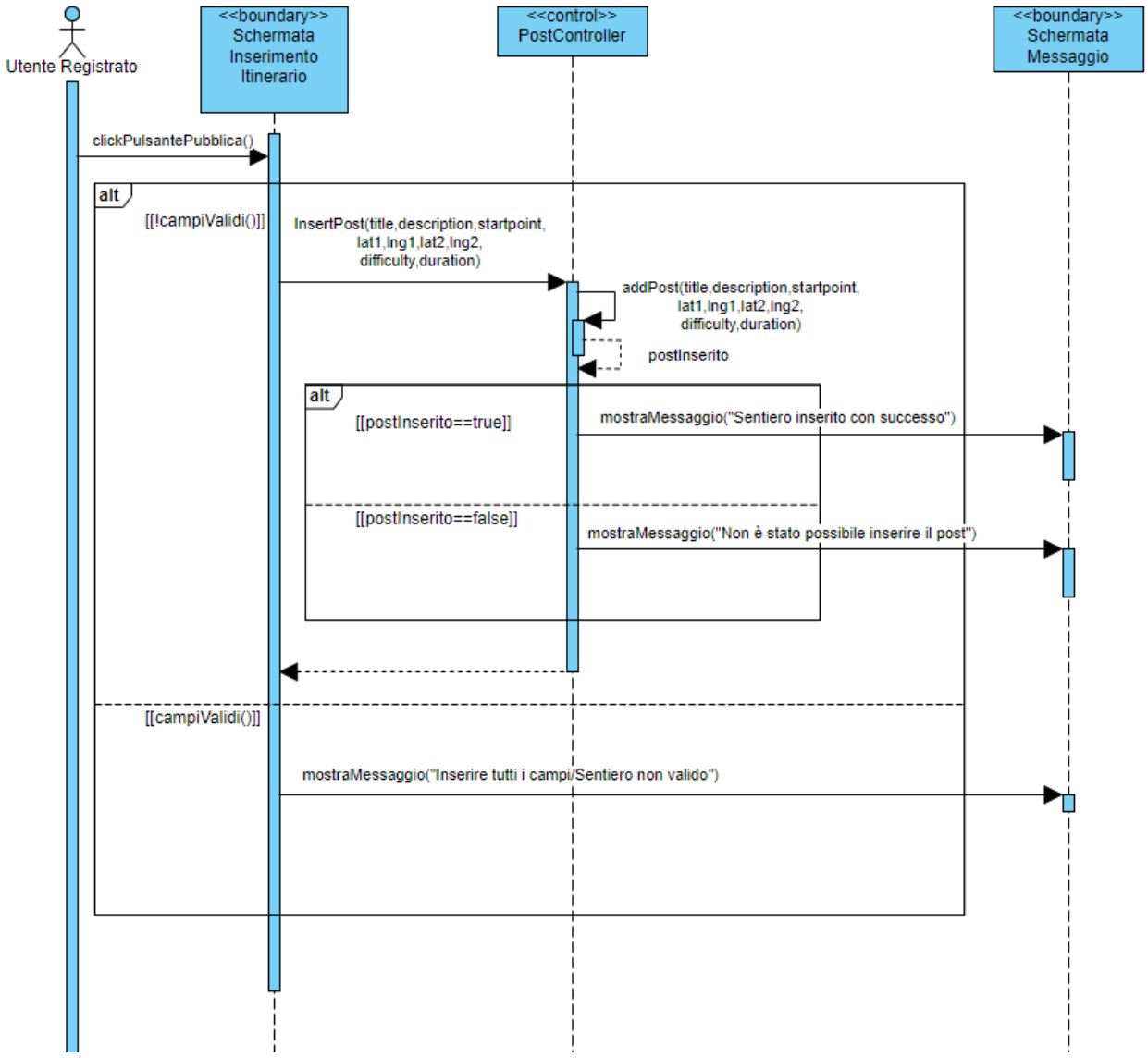
2.1.8 Chat



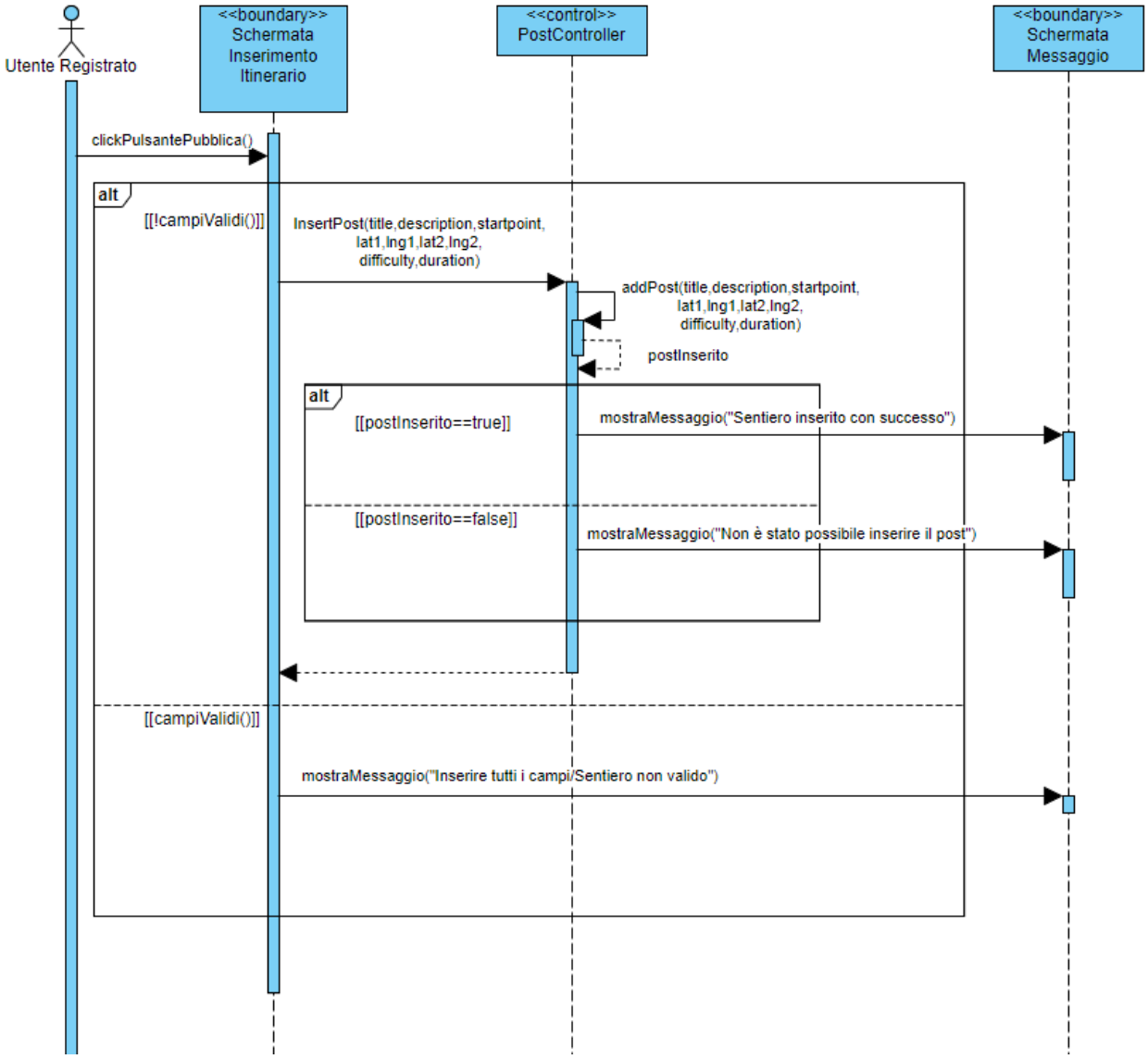
2.2 Diagrammi di sequenza di analisi per due casi d'uso significativi

Riportiamo i diagrammi di sequenza di analisi per i casi d'uso di **Inserimento Itinerario** e **Inserimento Recensione** visti nel dettaglio nel punto 1.2

2.2.1 Inserimento Recensione



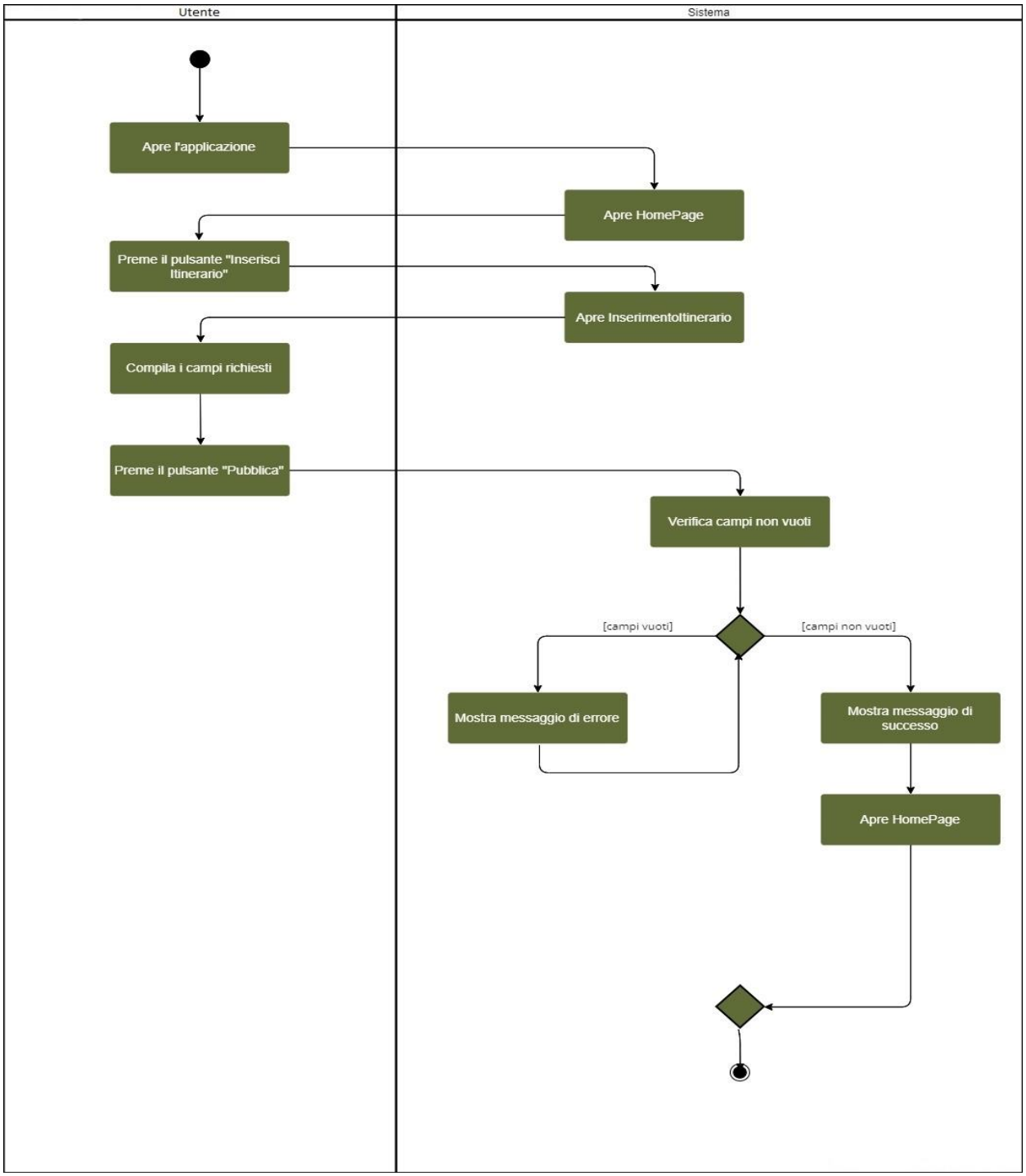
2.2.2 Inserimento Itinerario



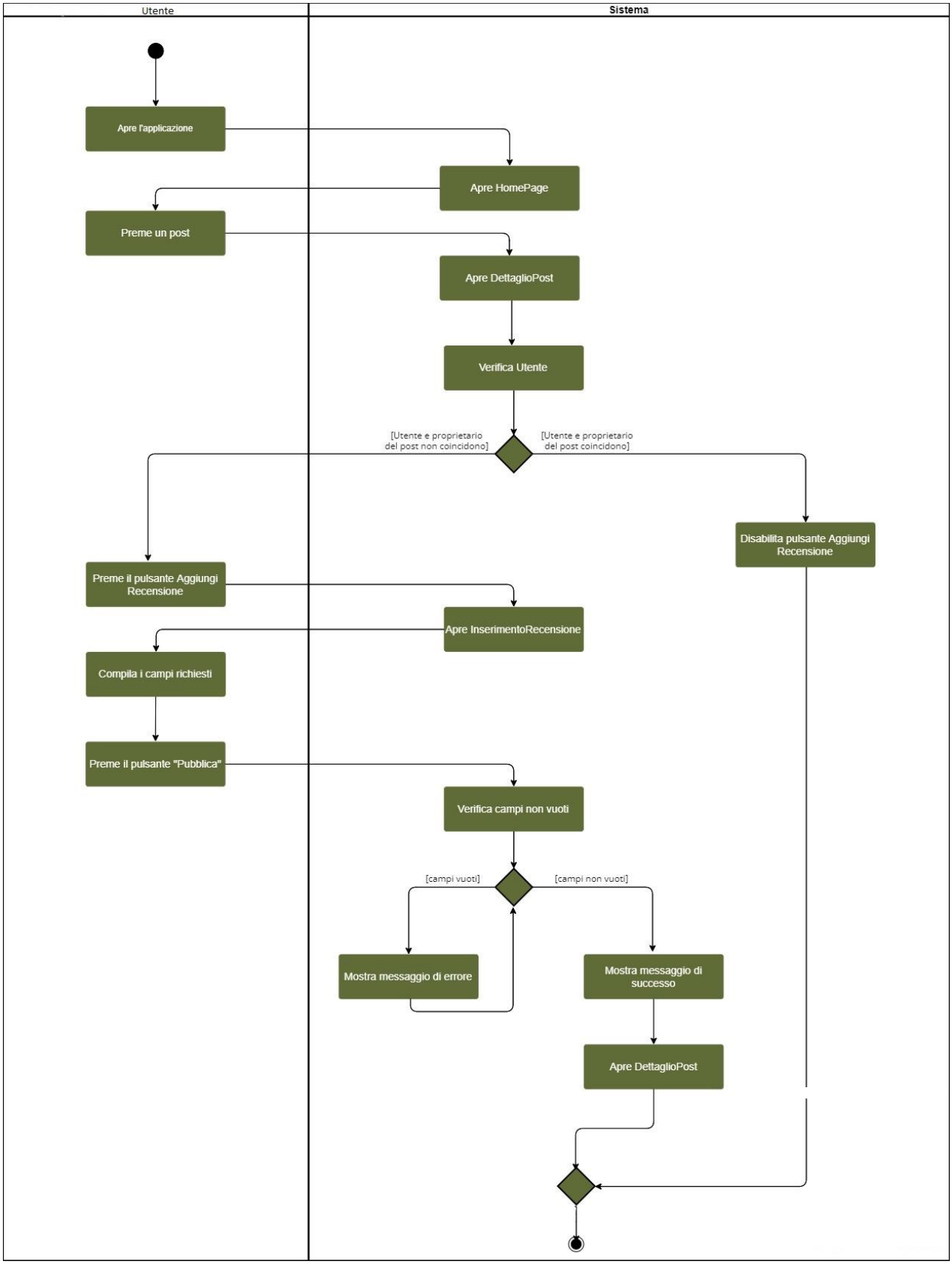
2.3 Diagrammi di attività

Vengono qui di seguito elencati i diagrammi di attività non banali dell'applicativo.
Si noti che si è presupposto che il parametro "**token**", che gestisce la validità dell'autenticazione, sia **NOT expired**

2.3.1 Inserimento itinerario



2.3.2 Inserimento Recensione



Capitolo II. Diagrammi di Design del sistema

3.1 Analisi dell'architettura con definizione dei criteri di design

Nella progettazione dell'architettura di NaTour21 abbiamo scelto di suddividere tale applicazione in due moduli:

• **Frontend**: Applicazione mobile Android

• **Backend**: Server Spring Boot e Server Database

La comunicazione tra applicazione android e server database è stata possibile grazie al server springboot tramite richieste http su endpoint specifici (vedremo in dettaglio il funzionamento nella sezione **Backend**). Per realizzare tale comunicazione lato client e tale infrastruttura lato server sono stati utilizzati diversi servizi che adesso vedremo nel dettaglio.

3.1.1 Frontend

Per la realizzazione di questo modulo è stato scelto Android utilizzando come linguaggio di programmazione Java. Nell'implementazione sono state utilizzate diverse librerie esterne:

- **Volley:** questa libreria ci ha permesso la comunicazione tra applicazione android e il server springboot. Per spiegare come essa funziona parliamo prima di tutto, di due concetti fondamentali: Request e RequestQueue. Come già anticipato abbiamo utilizzato il protocollo http. Esso è basato sull'interazione tra client e server in termini di richiesta-risposta: con un derivato della classe Request formuleremo la richiesta e l'accoderemo nella RequestQueue. Derivato della classe Request poiché esistono svariate sottoclassi di Request come, per esempio StringRequest che andremo ad utilizzare nel nostro applicativo.

  Non andremo mai ad eseguire la richiesta immediatamente, ma ci limiteremo ad inserirla in una coda (la RequestQueue). Sarà poi Volley ad eseguirla appena possibile, secondo le sue politiche e le condizioni del sistema.

  In ogni oggetto Request, saranno inclusi i riferimenti a due listener: uno derivante da Response.Listener invocato nel caso in cui la richiesta venga svolta con successo; l'altro, ErrorListener, in caso di errori. In quest'ultimo listener utilizzeremo un oggetto VolleyError che non è altro che un'eccezione che maschera un errore HTTP.

Al fine di poter integrare volley nel nostro codice abbiamo deciso di creare un'interfaccia (VolleyCallBack) con i seguenti metodi:

- onSuccess (String response)
- onError (String response)

Tale interfaccia sarà utile per passare la risposta dalle classi API (UserAPI, MessageAPI, etc) che effettuano effettivamente le richieste http, alle classi controller che gestiscono la risposta.

- **Facebook Android Sdk**: utilizzata per permettere agli utenti l'autenticazione tramite Facebook. L'utente sarà comunque tenuto a scegliere un nome utente per identificarsi nel socialnetwork.

- **Google Play services**: utilizzata per permettere agli utenti l'autenticazione tramite Google. L'utente sarà comunque tenuto a scegliere un nome utente per identificarsi nel socialnetwork.

- **Pusher Java Client**: utilizzata per permettere agli utenti di ricevere messaggi e/o segnalazioni in tempo reale. Esso è strutturato in canali ed eventi. Ad ogni utente che ha effettuato il login sarà associato un canale col proprio nome utente e ogni canale avrà tre eventi in "ascolto" sul client, **newMessage** per catturare nuovi messaggi e notificare l'utente, **report** per catturare nuove segnalazioni ricevute e notificare l'utente e infine **report_response** nel caso in cui ci siano risposte a segnalazioni precedentemente inviate.

- **Google Play Services Maps:** utilizzata per permettere agli utenti di visualizzare le mappe, fornite da Google, sul nostro applicativo.
  Le mappe sono state implementate con una MapView all'interno delle seguenti classi:
  - HomeFragment: Qui il servizio viene utilizzato per mostrare, all'interno di un CardLayout, la MapView versione Lite, contenente il percorso scelto dall'utente.

-insertPostFragment: Qui il servizio viene utilizzato per permettere all'utente di visualizzare graficamente una ricerca sulla mappa, effettuata con Google Places API, mostrare il percorso del file .GPX appena inserito e per selezionare punti geografici interattivamente interagendo con la mappa.
-postDetailsFragment: Qui il servizio viene utilizzato per mostrare nel dettaglio il percorso che l'utente ha inserito. L'utente potrà zoomare e muoversi sulla mappa a suo piacimento.

- **Google Play Services Location:** Questo servizio permette di facilitare il processo di rilevamento automatico di posizione, il rilevamento del wrong-side-of-the-street, il geofencing ed il riconoscimento delle attività.
  Questo è da inserire obbligatoriamente per poter avere accesso al servizio di Google Play Services Maps.

- **Google Directions Android:** utilizzata per poter far sì che la nostra MapView potesse graficamente contenere tracciati costituiti da molteplici polilinee tenendo conto delle reali strade presenti sulla mappa.
  Questo servizio è stato utilizzato nelle seguenti classi:
  -HomeFragment: per poter visualizzare il tracciato contenuto da un post.
  -inserPostFragment: per poter visualizzare l'effettivo tracciato inserito dall'utente tramite interazione con la mappa oppure tramite inserimento del file .GPX
  -postDetailsFragment: per poter visualizzare il tracciato contenuto nel post in dettaglio con possibilità di zoom e spostamenti sulla mappa.

- **Google Places API:** utilizzata per poter permettere la ricerca di un determinato luogo, indirizzo o coordinata geografica su di una mappa.
  Questo servizio è stato utilizzato nelle seguenti classi:
  -insertPostFragment: per poter facilitare la ricerca di un luogo geografico per il conseguente inserimento di punti sulla mappa.

- **Android GPX Parser:** Questa libreria ha facilitato il processo di Parsing di file .GPX. Dopo aver con successo verificato la correttezza del file .GPX inserito, ed aver aggiunto tutti i punti di tipo LatLng all'interno di un ArrayList, è stato possibile, grazie all'aiuto di questa libreria, poter scomporre in segmenti tutti i punti ricevuti in input dal file e salvarli come valori di Latitudine e Longitudine.
  A questo punto dopo aver scomposto in segmenti tutti i punti, è stato possibile inserire il tracciato all'interno della mappa.
  Questo servizio è stato utilizzato all'interno delle seguenti classi:
  -insertPostFragment: utilizzato durante l'inserimento di un file .GPX .

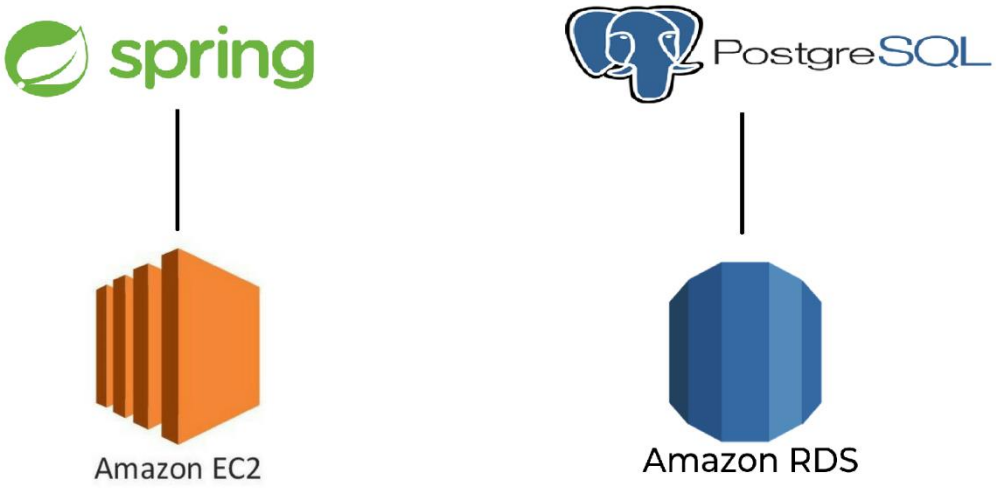
- **Handle Path Oz:** utilizzata per poter ricevere il path reale di un file .GPX di un dispositivo.
  Data l'impossibilità di Android di fornire un path veritiero, ci siamo avvalsi di questa libreria che dopo aver selezionato il file, a partire da un FilePicker, ricava il path reale prendendo in input l'Uri del file.
  Questo servizio è stato utilizzato dall'interno delle seguenti classi:
  -insertPostFragment: utilizzato dopo aver selezionato con successo un file .GPX dal FilePicker.

- **SimpleToolTip:** utilizzata per poter creare un tooltip all'interno del processo di Inserimento di Itinerario per spiegare come inserire un tracciato in maniera interattiva utilizzando la mappa.
  Questo servizio è stato utilizzato all'interno delle seguenti classi:
  -insertPostFragment: indicato con un'icona di informazioni in prossimità della MapView.

- **RoutingListener:** utilizzata per poter calcolare il percorso tra due o più punti e mostrare graficamente il percorso sulla MapView utilizzando l'API Google Directions Android. Con questo servizio è possibile specificare i Waypoints per cui si vuole calcolare un percorso aggiungendo il travelMode, nel nostro caso è stato utilizzato "WALKING". Questo servizio ci permette di poter trattare diversi scenari.
  In particolare abbiamo implementato lo scenario in cui il processo ha successo: onRoutingSuccess() in cui il servizio calcola correttamente il percorso e lo scenario in cui il processo fallisce, onRoutingFailure() in cui il servizio lancia una RouteException.

3.1.2 Backend

Questo modulo si divide in due sotto-moduli. Il primo, il *server springboot* sarà hostato su un'istanza Amazon EC2 (Linux/UNIX) e il secondo, il *server database* su un'istanza Amazon RDS (PostgreSQL) in cui saranno immagazzinati tutti i dati.



Springboot

Il server spring boot (framework Java) scritto in Java realizza tutta la logica di comunicazione tra l'applicazione mobile e il *server database*

La scelta di Spring Boot è dovuta alla sua robustezza e sicurezza. Infatti, tale framework tramite l'implementazione di un token permette di rendere sicure le richieste (protocollo http) inviate al server e dunque non accessibili a tutti ma solo a chi dispone di un token valido. Il token risulterà invalido dopo due settimane dal login. Effettuando nuovamente l'accesso il countdown ripartirà nuovamente. Senza un token valido non sarà possibile effettuare richieste al server

Sul server springboot troveremo per ogni entità presente sul database:

- **Controller**: definisce gli endpoint (URL della richiesta), gestisce e cattura le richieste http effettuate dal client
- **Entity**: definisce la struttura dell'entità sul database.
- **Repository**: definisce eventualmente funzioni personalizzate (sfruttando la classe JpaRepository) che non siano già presenti nel Service
- **Service**: definisce la logica di gestione dei dati (sul database) per il controller

Così come sull'applicativo android sul server springboot troveremo il servizio *Pusher.* Nello specifico la classe **PusherManager** gestirà l'invio di notifiche di nuovi messaggi/segnalazioni ai relativi canali degli utenti.

Il server springboot gestirà anche la logica per il recupero password; infatti, è stata utilizzata JavaMailSender (appartenente al framework Springboot) per rendere possibile l'invio di un codice OTP per il recupero password. Di seguito lasciamo le impostazioni utilizzate nel file applicaiton.properties del server springboot:

```
spring.mail.host=smtp.gmail.com
spring.mail.port=465
spring.mail.username=natour21.otp@gmail.com
spring.mail.password=jdndxyvzifpcatbg
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.connectiontimeout=5000
spring.mail.properties.mail.smtp.timeout=5000
spring.mail.properties.mail.smtp.writetimeout=5000

spring.mail.properties.mail.smtp.socketFactory.port = 465
spring.mail.properties.mail.smtp.socketFactory.class = javax.net.ssl.SSLSocketFactor
```

Database

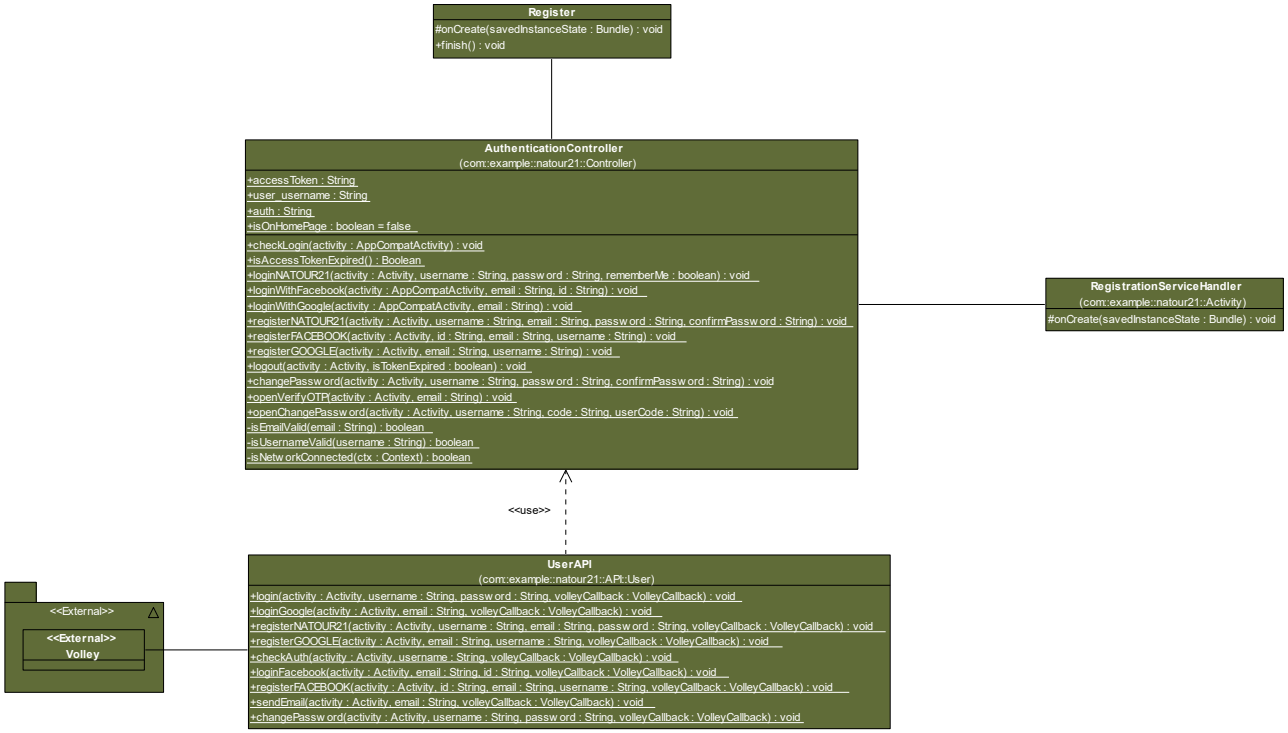
Questo modulo è un DBMS (Database Management System) realizzato con PostgreSQL ed avrà il compito di immagazzinare i dati. Grazie all'utilizzo di un'istanza Amazon RDS sarà possibile in un futuro trasferire il database in un altro tipo di istanza, o effettuare backup periodici.

3.2 Diagramma delle classi di design

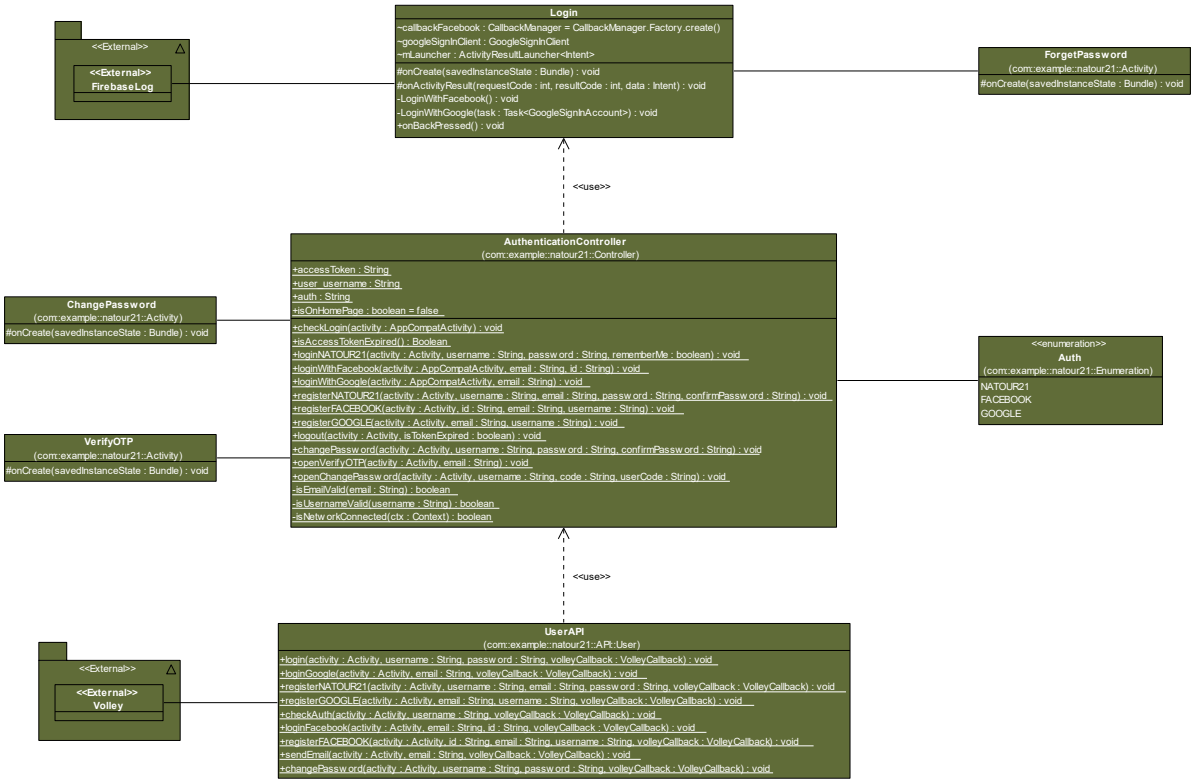
Verrà di seguito riportato il Class Diagram di Design completo di tutte le classi e relazioni dell'appilcativo. Verranno poi di seguito riportati, per una maggiore leggibilità, i Class Diagram singoli per ogni funzionalità.

Tutti i diagram riportati di seguito hanno formato .SVG per cui è possibile zoommare per visualizzare ogni dettaglio evitando l'effetto "pixel".

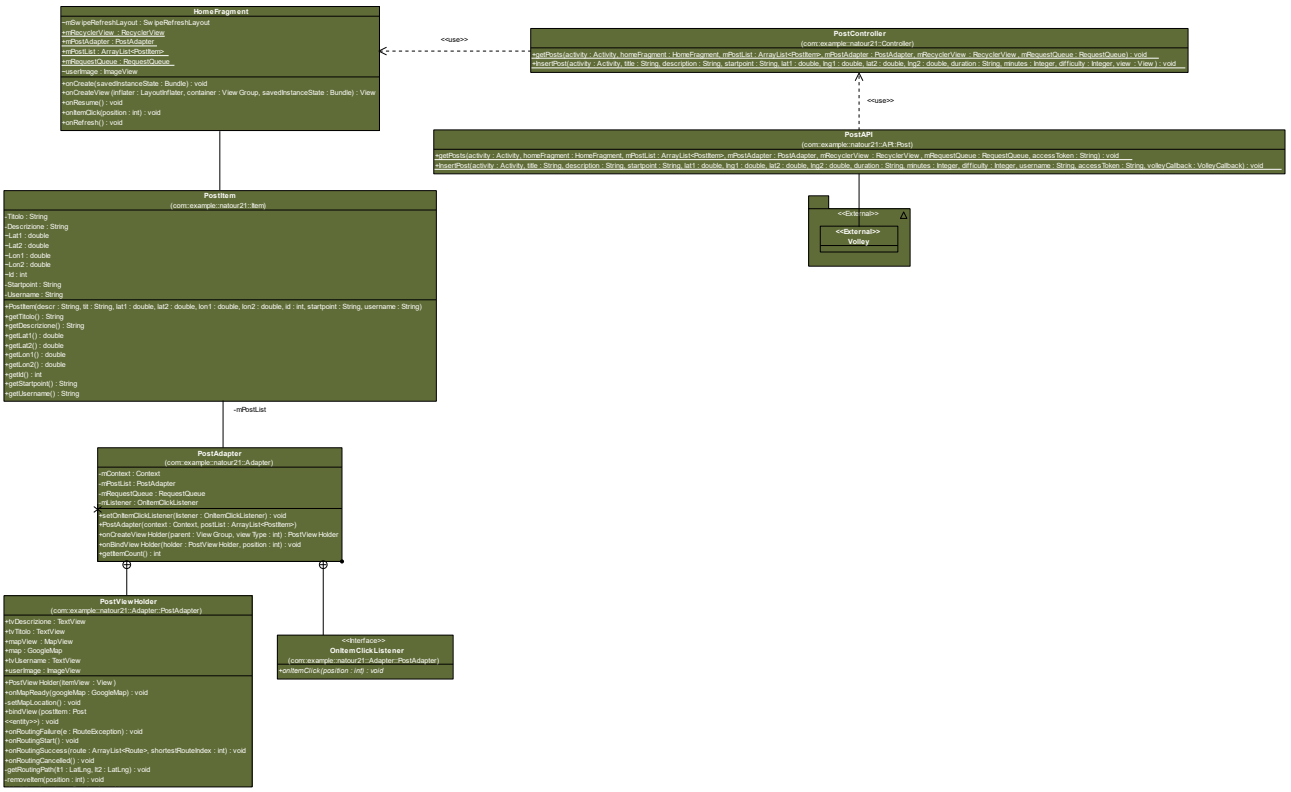
3.2.1 Registrazione



3.2.2 Login

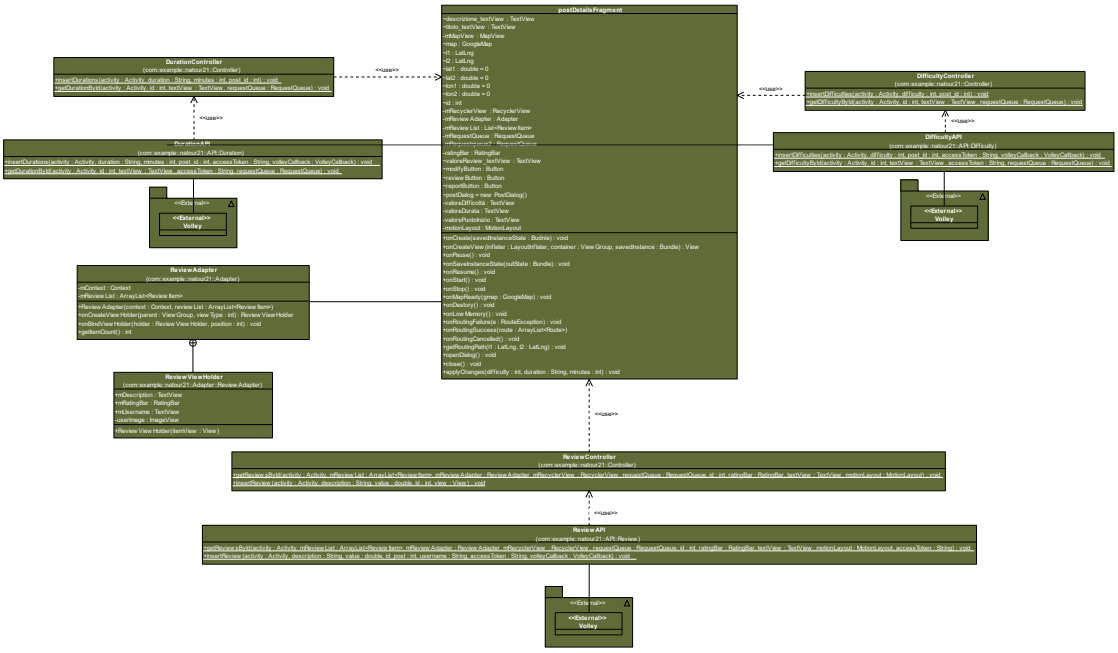

Login
~callbackFacebook : CallbackManager = CallbackManager.Factory.create()
~googleSignInClient : GoogleSignInClient
~mLauncher : ActivityResultLauncher<Intent>
#onCreate(savedInstanceState : Bundle) : void
#onActivityResult(requestCode : int, resultCode : int, data : Intent) : void
-LoginWithFacebook() : void
-LoginWithGoogle(task : Task<GoogleSignInAccount>) : void
+onBackPressed() : void
<<External>>
<<External>>
FirebaseLog
ForgetPassword
(com::example::natour21::Activity)
#onCreate(savedInstanceState : Bundle) : void
<<use>>
AuthenticationController
(com::example::natour21::Controller)
+accessToken : String
+user_username : String
+auth : String
+isOnHomePage : boolean = false
+checkLogin(activity : AppCompatActivity) : void
+isAccessTokenExpired() : Boolean
+loginNATOUR21(activity : Activity, username : String, password : String, rememberMe : boolean) : void
+loginWithFacebook(activity : AppCompatActivity, email : String, id : String) : void
+loginWithGoogle(activity : AppCompatActivity, email : String) : void
+registerNATOUR21(activity : Activity, username : String, email : String, password : String, confirmPassword : String) : void
+registerFACEBOOK(activity : Activity, id : String, email : String, username : String) : void
+registerGOOGLE(activity : Activity, email : String, username : String) : void
+logout(activity : Activity, isTokenExpired : boolean) : void
+changePassword(activity : Activity, username : String, password : String, confirmPassword : String) : void
+openVerifyOTP(activity : Activity, email : String) : void
+openChangePassword(activity : Activity, username : String, code : String, userCode : String) : void
-isEmailValid(email : String) : boolean
-isUsernameValid(username : String) : boolean
-isNetworkConnected(ctx : Context) : boolean
ChangePassword
(com::example::natour21::Activity)
#onCreate(savedInstanceState : Bundle) : void
VerifyOTP
(com::example::natour21::Activity)
#onCreate(savedInstanceState : Bundle) : void
<<enumeration>>
Auth
(com::example::natour21::Enumeration)
NATOUR21
FACEBOOK
GOOGLE
<<use>>
UserAPI
(com::example::natour21::API::User)
+login(activity : Activity, username : String, password : String, volleyCallback : VolleyCallback) : void
+loginGoogle(activity : Activity, email : String, volleyCallback : VolleyCallback) : void
+registerNATOUR21(activity : Activity, username : String, email : String, password : String, volleyCallback : VolleyCallback) : void
+registerGOOGLE(activity : Activity, email : String, username : String, volleyCallback : VolleyCallback) : void
+checkAuth(activity : Activity, username : String, volleyCallback : VolleyCallback) : void
+loginFacebook(activity : Activity, email : String, id : String, volleyCallback : VolleyCallback) : void
+registerFACEBOOK(activity : Activity, id : String, email : String, username : String, volleyCallback : VolleyCallback) : void
+sendEmail(activity : Activity, email : String, volleyCallback : VolleyCallback) : void
+changePassword(activity : Activity, username : String, password : String, volleyCallback : VolleyCallback) : void
<<External>>
<<External>>
Volley

3.2.3 Home

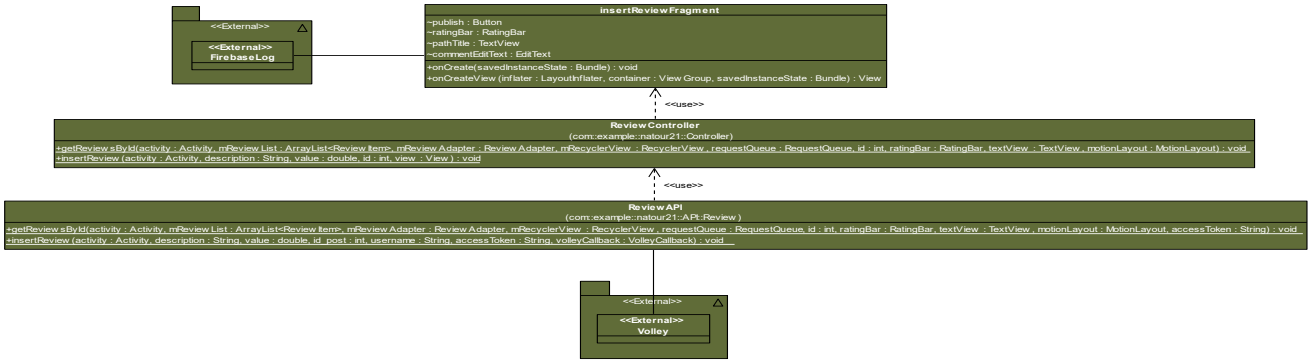

HomeFragment
-mSwipeRefreshLayout : SwipeRefreshLayout
-mRecyclerView : RecyclerView
-mPostAdapter : PostAdapter
-mPostList : ArrayList<PostItem>
-mRequestQueue : RequestQueue
-userImage : ImageView
+onCreate(savedInstanceState : Bundle) : void
+onCreateView (inflater : LayoutInflater, container : View Group, savedInstanceState : Bundle) : View
+onResume() : void
+onItemClick(position : int) : void
+onRefresh() : void
<<use>>
PostController
(com::example::natour21::Controller)
+getPosts(activity : Activity, homeFragment : HomeFragment, mPostList : ArrayList<PostItem>, mPostAdapter : PostAdapter, mRecyclerView : RecyclerView, mRequestQueue : RequestQueue) : void
+insertPost(activity : Activity, title : String, description : String, startpoint : String, lat1 : double, lon1 : double, lat2 : double, lon2 : double, duration : String, minutes : Integer, difficulty : Integer, view : View) : void
<<use>>
PostAPI
(com::example::natour21::API::Post)
+getPosts(activity : Activity, homeFragment : HomeFragment, mPostList : ArrayList<PostItem>, mPostAdapter : PostAdapter, mRecyclerView : RecyclerView, mRequestQueue : RequestQueue, accessToken : String) : void
+insertPost(activity : Activity, title : String, description : String, startpoint : String, lat1 : double, lon1 : double, lat2 : double, lon2 : double, duration : String, minutes : Integer, difficulty : Integer, username : String, accessToken : String, volleyCallback : VolleyCallback) : void
<<External>>
Volley
PostItem
(com::example::natour21::Item)
-Titolo : String
-Descrizione : String
-Lat1 : double
-Lat2 : double
-Lon1 : double
-Lon2 : double
-Id : int
-Startpoint : String
-Username : String
+PostItem(descr : String, tit : String, lat1 : double, lat2 : double, lon1 : double, lon2 : double, id : int, startpoint : String, username : String)
+getTitolo() : String
+getDescrizione() : String
+getLat1() : double
+getLat2() : double
+getLon1() : double
+getLon2() : double
+getId() : int
+getStartpoint() : String
+getUsername() : String
-mPostList
PostAdapter
(com::example::natour21::Adapter)
-mContext : Context
-mPostList : PostAdapter
-mRequestQueue : RequestQueue
-mListener : OnItemClickListener
+setOnItemClickListener(listener : OnItemClickListener) : void
+PostAdapter(context : Context, postList : ArrayList<PostItem>)
+onCreateView Holder(parent : View Group, view Type : int) : PostView Holder
+onBindView Holder(holder : PostView Holder, position : int) : void
+getItemCount() : int
PostViewHolder
(com::example::natour21::Adapter::PostAdapter)
+tvDescrizione : TextView
+tvTitolo : TextView
+mapView : MapView
+map : GoogleMap
+tvUsername : TextView
+userImage : ImageView
+PostView Holder(itemView : View)
+onMapReady(googleMap : GoogleMap) : void
-setMapLocation() : void
+bindView (postItem : Post
<<synthesized>> : void
+onRoutingFailure(e : RouteException) : void
+onRoutingStart() : void
+onRoutingSuccess(route : ArrayList<Route>, shortestRouteIndex : int) : void
+onRoutingCancelled() : void
-getRoutingPath(lt1 : LatLng, lt2 : LatLng) : void
-removeItem(position : int) : void
<<Interface>>
OnItemClickListener
(com::example::natour21::Adapter::PostAdapter)
+onItemClick(position : int) : void

3.2.4 Inserimento Itinerario

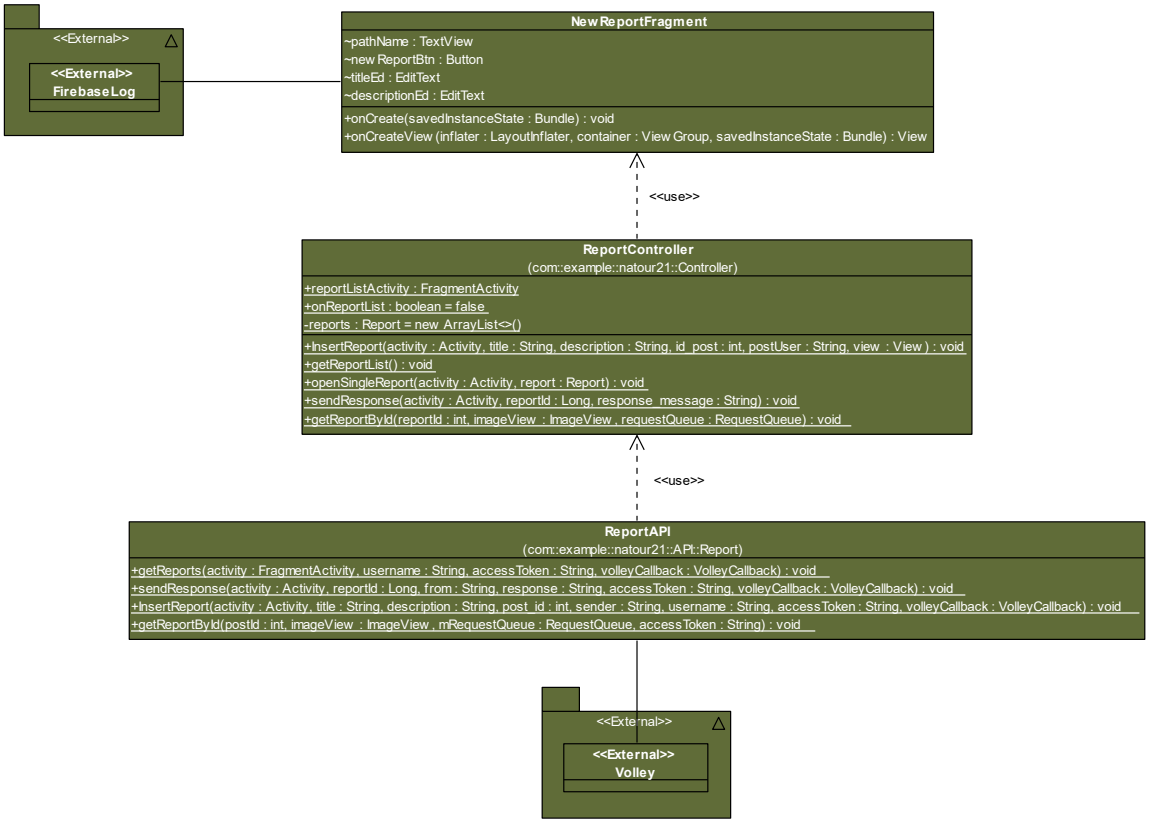
3.2.5 Dettagli Post e modifica durata/difficoltà



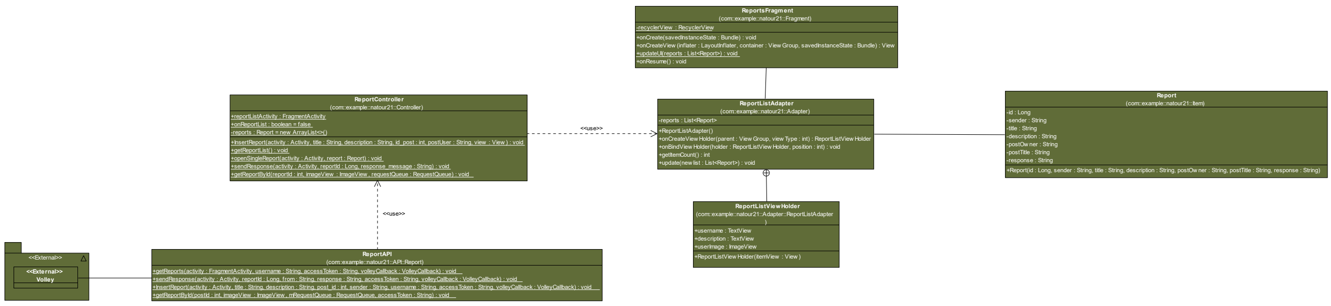
3.2.6 Inserimento Recensione



3.2.7 Inserimento Report

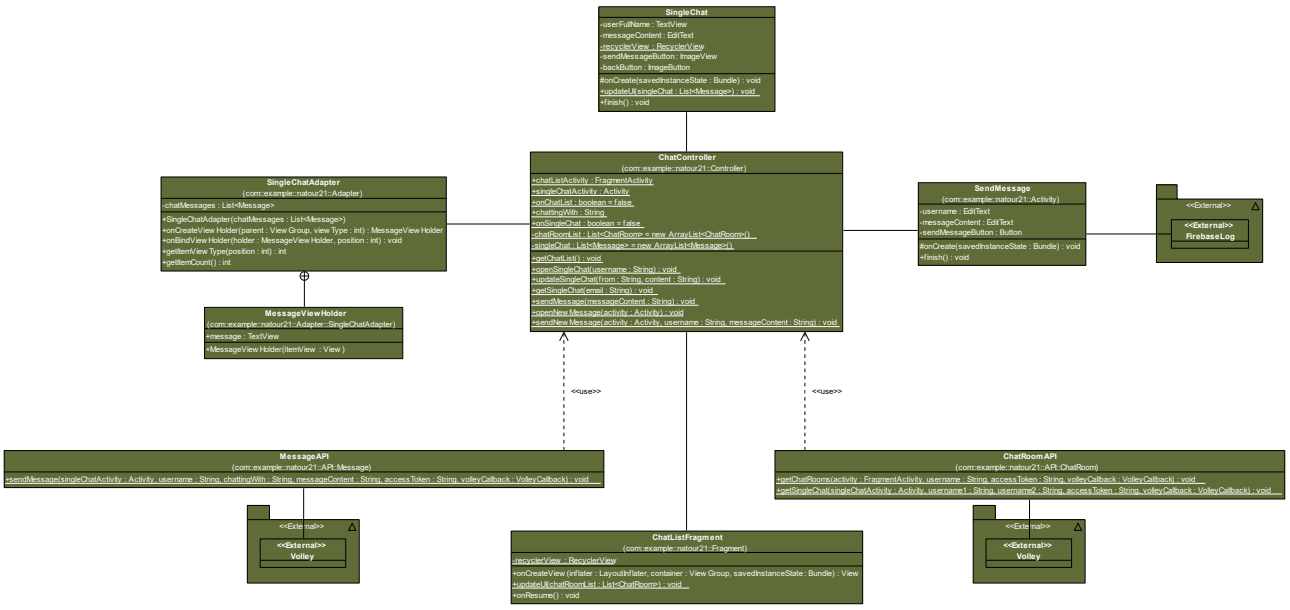


3.2.8 Reports



ReportsFragment
(com::example::natour21::Fragment)
-recyclerView : RecyclerView
+onCreate(savedInstanceState : Bundle) : void
+onCreateView (inflater : LayoutInflater, container : View Group, savedInstanceState : Bundle) : View
+updateUI(reports : List<Report>) : void
+onResume() : void
ReportController
(com::example::natour21::Controller)
+reportListActivity : FragmentActivity
+onReportList : boolean = false
-reports : Report = new ArrayList<>()
+InsertReport(activity : Activity, title : String, description : String, id_post : int, postUser : String, view : View) : void
+getReportList() : void
+openSingleReport(activity : Activity, report : Report) : void
+sendResponse(activity : Activity, reportId : Long, response_message : String) : void
+getReportById(reportId : int, imageView : ImageView , requestQueue : RequestQueue) : void
<<use>>
ReportListAdapter
(com::example::natour21::Adapter)
-reports : List<Report>
+ReportListAdapter()
+onCreateView Holder(parent : View Group, view Type : int) : ReportListView Holder
+onBindView Holder(holder : ReportListView Holder, position : int) : void
+getItemCount() : int
+update(new list : List<Report>) : void
Report
(com::example::natour21::Item)
-id : Long
-sender : String
-title : String
-description : String
-postOw ner : String
-postTitle : String
-response : String
+Report(id : Long, sender : String, title : String, description : String, postOw ner : String, postTitle : String, response : String)
<<use>>
ReportListViewHolder
(com::example::natour21::Adapter::ReportListAdapter
)
+username : TextView
+description : TextView
+userImage : ImageView
+ReportListView Holder(itemView : View)
<<External>>
<<External>>
Volley
ReportAPI
(com::example::natour21::API::Report)
+getReports(activity : FragmentActivity, username : String, accessToken : String, volleyCallback : VolleyCallback) : void
+sendResponse(activity : Activity, reportId : Long, from : String, response : String, accessToken : String, volleyCallback : VolleyCallback) : void
+InsertReport(activity : Activity, title : String, description : String, post_id : int, sender : String, username : String, accessToken : String, volleyCallback : VolleyCallback) : void
+getReportById(postId : int, imageView : ImageView , mRequestQueue : RequestQueue, accessToken : String) : void

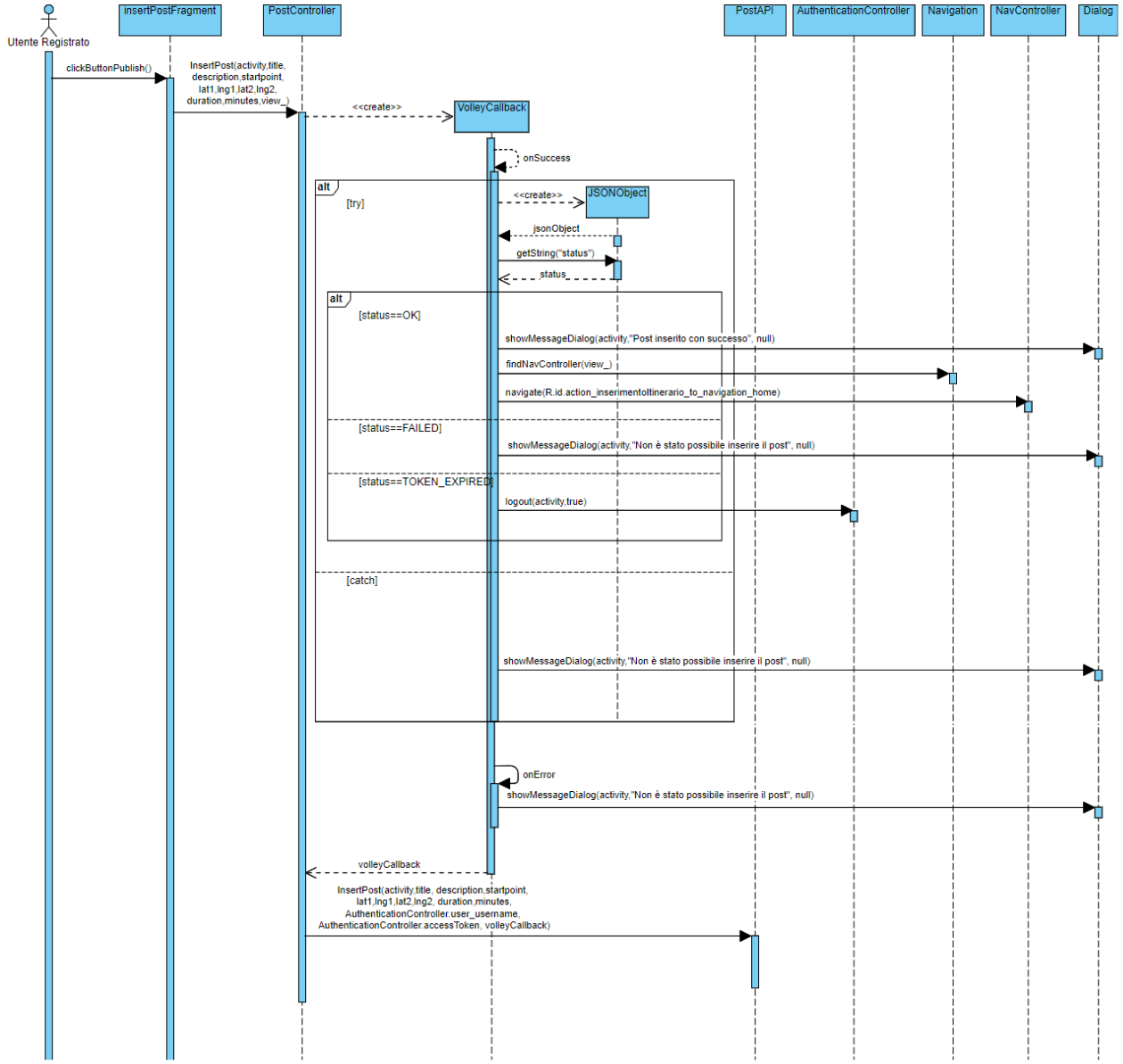
3.2.9 Chat


SingleChat
-userFullName : TextView
-messageContent : EditText
-recyclerView : RecyclerView
-sendMessageButton : ImageView
-backButton : ImageButton
#onCreate(savedInstanceState : Bundle) : void
+updateUI(singleChat : List<Message>) : void
+finish() : void
ChatController
(com::example::natour21::Controller)
+chatListActivity : FragmentActivity
+singleChatActivity : Activity
+onChatList : boolean = false
+chattingWith : String
+onSingleChat : boolean = false
-chatRoomList : List<ChatRoom> = new ArrayList<ChatRoom>()
-singleChat : List<Message> = new ArrayList<Message>()
+getChatList() : void
+openSingleChat(username : String) : void
+updateSingleChat(from : String, content : String) : void
+getSingleChat(email : String) : void
+sendMessage(messageContent : String) : void
+openNewMessage(activity : Activity) : void
+sendNewMessage(activity : Activity, username : String, messageContent : String) : void
SingleChatAdapter
(com::example::natour21::Adapter)
-chatMessages : List<Message>
+SingleChatAdapter(chatMessages : List<Message>)
+onCreateView Holder(parent : View Group, view Type : int) : MessageView Holder
+onBindView Holder(holder : MessageView Holder, position : int) : void
+getItemView Type(position : int) : int
+getItemCount() : int
MessageViewHolder
(com::example::natour21::Adapter::SingleChatAdapter)
+message : TextView
+MessageView Holder(itemView : View)
SendMessage
(com::example::natour21::Activity)
-username : EditText
-messageContent : EditText
-sendMessageButton : Button
#onCreate(savedInstanceState : Bundle) : void
+finish() : void
<<External>>
<<External>>
FirebaseLog
<<use>>
<<use>>
MessageAPI
(com::example::natour21::API::Message)
+sendMessage(singleChatActivity : Activity,username : String, chattingWith : String, messageContent : String,accessToken : String, volleyCallback : VolleyCallback) : void
<<External>>
<<External>>
Volley
ChatListFragment
(com::example::natour21::Fragment)
-recyclerView : RecyclerView
+onCreateView (inflater : LayoutInflater, container : View Group, savedInstanceState : Bundle) : View
+updateUI(chatRoomList : List<ChatRoom>) : void
+onResume() : void
ChatRoomAPI
(com::example::natour21::API::ChatRoom)
+getChatRooms(activity : FragmentActivity, username : String, accessToken : String, volleyCallback : VolleyCallback) :void
+getSingleChat(singleChatActivity : Activity, username1 : String, username2 : String, accessToken : String, volleyCallback : VolleyCallback) : void
<<External>>
<<External>>
Volley

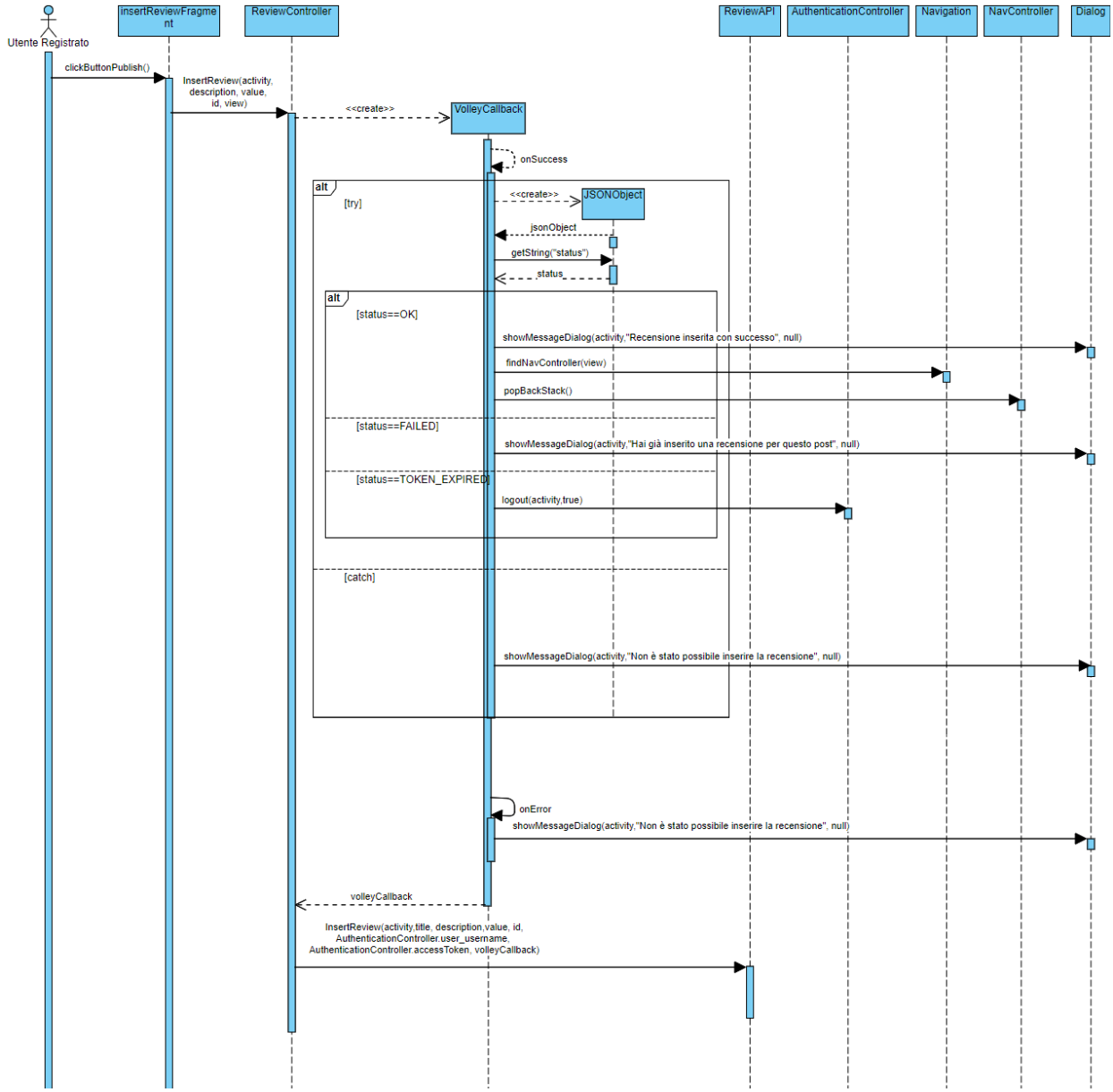
3.3 Diagrammi di sequenza di design per due casi d'uso significativi

Riportiamo i diagrammi di sequenza di design per i casi d'uso di **Inserimento Itinerario** e **Inserimento Recensione** visti nel dettaglio nel punto 1.2

3.3.1 Inserimento itinerario



3.3.2 Inserimento recensione



3.4 Definizione delle gerarchie funzionali.

Di seguito riportate vi sono le specifiche riguardanti il documento dell'intero progetto.

La tabella conterrà le seguenti voci:

- **Data**: Specifica la data dell'aggiunta/modifica di un elemento all'interno del documento.
- **Autore**: Specifica l'autore che ha compiuto l'aggiunta/modifica all'interno del documento.
- **Descrizione**: Breve descrizione riguardante le aggiunte/modifiche.

Data	Autore	Descrizione
09/11/2021	Aurora Di Maio	Introduzione
12/11/2021	Aurora Di Maio Mike Fonseta	Modellazione Casi D'uso
13/11/2021	Aurora Di Maio	Tabella di Cockburn – Inserimento Itinerario
12/11/2021	Mike Fonseta	Tabella di Cockburn – Inserimento Recensione
14/11/2021	Aurora Di Maio	Prototipazione visuale via Mock-Up – Inserimento Itinerario
14/11/2021	Mike Fonseta	Prototipazione visuale via Mock-Up – Inserimento Recensione
02/01/2022	Aurora Di Maio	Modello Funzionale
03/01/2022	Mike Fonseta	Aggiornamento Casi D'uso
03/01/2022	Mike Fonseta	Aggiornamento Tabelle di Cockburn
04/01/2022	Aurora Di Maio	Aggiornamento prototipazione visuale via Mock-Up
07/01/2022	Aurora Di Maio	Stesura Idea Progettuale
09/01/2022	Mike Fonseta	Individuazione del target degli utenti
11/01/2022	Mike Fonseta	Valutazione dell'usabilità a priori
13/01/2022	Aurora Di Maio	Statechart di interfaccia grafica – Inserimento Itinerario
13/01/2022	Mike Fonseta	Statechart di interfaccia grafica – Inserimento Recensione
15/01/2022	Aurora Di Maio	Class Diagram di Dominio
16/01/2022	Mike Fonseta	Sequence Diagram di Dominio
18/01/2022	Aurora Di Maio	Activity Diagrams
20/01/2022	Mike Fonseta	Analisi dell'architettura con definizione dei criteri di Design
21/01/2022	Aurora Di Maio	Class Diagram di Design
22/01/2022	Mike Fonseta	Sequence Diagram di Design
26/01/2022	Mike Fonseta	Valutazione dell'usabilità sul campo

29/01/2022	Aurora Di Maio	Glossario
01/02/2022	Aurora Di Maio	Gerarchie funzionali
10/02/2022	Mike Fonseta	Aggiornamento Tabelle di Cockburn e prototipazione visuale via Mock-Up – Inserimento Recensione
11/02/2022	Mike Fonseta	Normalizzazione documento e creazione indice
03/03/2022	Mike Fonseta	Inserimento testing
04/03/2022	Mike Fonseta	Aggiornamento usabilità

Capitolo III. Valutazione dell'usabilità sul campo

Al fine di valutare l'usabilità dell'applicazione abbiamo scelto di sottoporre un gruppo di tester a determinate azioni da svolgere e in più abbiamo optato per l'integrazione di Firebase nel nostro applicativo al fine di poter monitorare le performance della nostra applicazione.

Testing applicazione

La prima valutazione è stata possibile dando ai tester una versione dell'applicazione da installare sul proprio smartphone.

Di seguito le azioni che hanno svolto i tester:

1. **Inserire un itinerario tramite mappa interattiva**
2. **Recensire un itinerario**
3. **Segnalare un itinerario**
4. **Mandare un messaggio ad un altro utente**

Ai fini di analisi statistiche è stato chiesto loro di compilare *un form Google* per valutare la facilità di tali azioni da "Molto facile" a "Molto difficile".

Si precisa che tali test sono stati fatti con tester che non hanno esperienze pregresse con l'escursionismo né con l'utilizzo di social network o applicazioni per escursionisti. Abbiamo scelto questa particolarità per poter valutare soprattutto in che modo una persona priva di esperienze nell'ambito, possa approcciarsi al social network e soprattutto con che facilità.

	Azione 1	Azione 2	Azione 3	Azione 4
Tester 1	F	S	S	S
Tester 2	S	S	S	S
Tester 3	S	S	S	S
Tester 4	S	S	S	S
Tester 5	S	S	S	S

Legenda: S = successo F = fallimento P = successo parziale

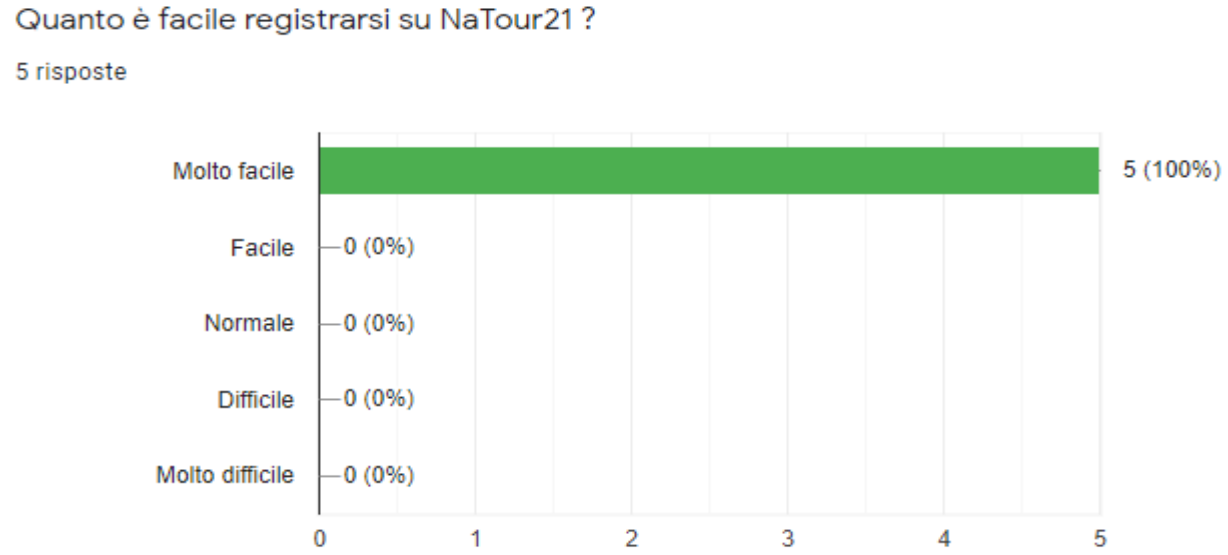
Da tali risultati il tasso di successo è del 95%

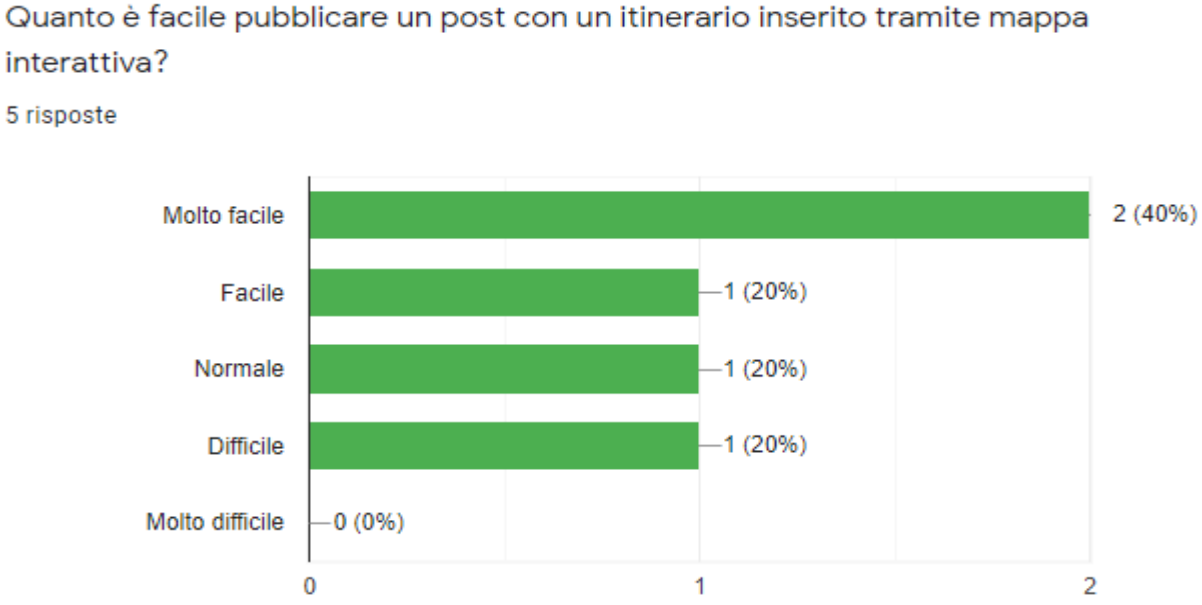
L’unica azione che ha influenzate il risultato finale è “**Inserire un itinerario tramite mappa interattiva”** che ha portato alcune difficoltà o incomprensioni. In linea generale considerano la particolarità dei tester il risultato finale mostra un ottimo livello di successo e in generale l’app si presenta accessibile anche chi ha poca esperienza nell’utilizzo di social network o in generali di applicazioni legate all’escursionismo.

Per quanto riguarda il form Google, come già accennato gli utenti potevano valutare la facilita da “Molto facile” a “Molto difficile” e nello specifico avevano 5 opzioni:

- **Molto facile**
- **Facile**
- **Normale**
- **Difficile**
- **Molto difficile**

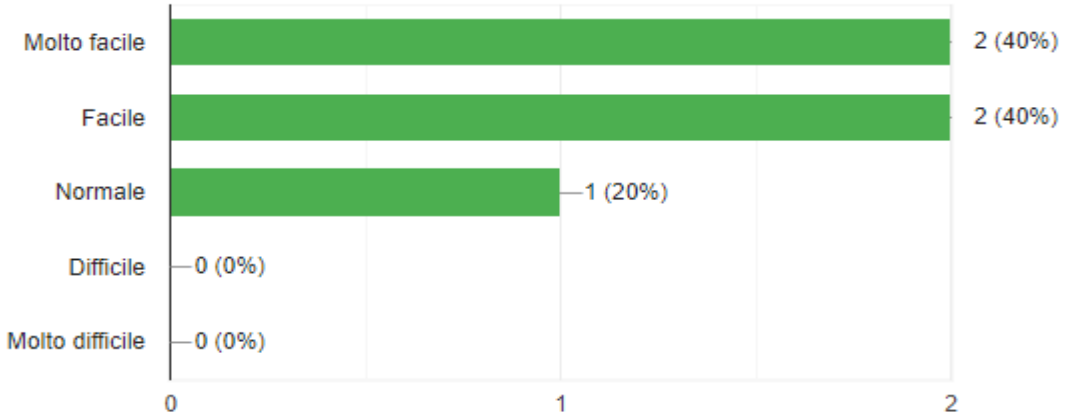
Di seguito vengono riportati i risultati del form:





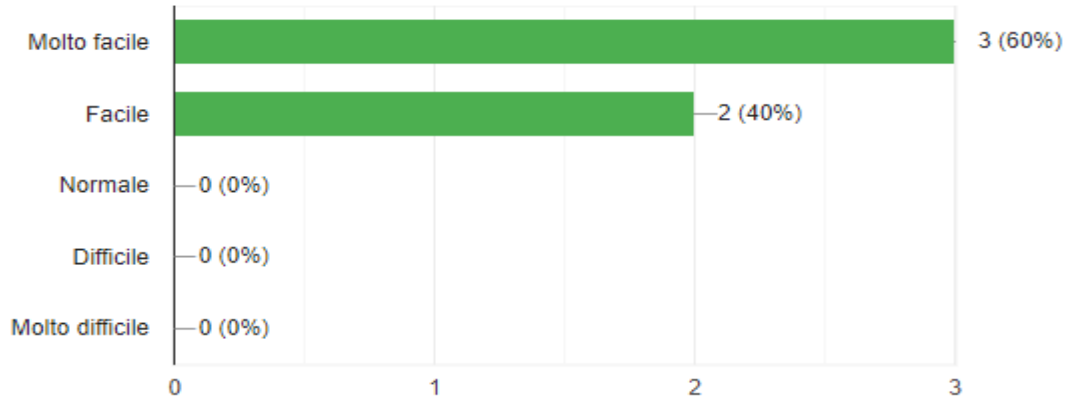
Quanto è facile inviare un messaggio ad un altro utente ?

5 risposte



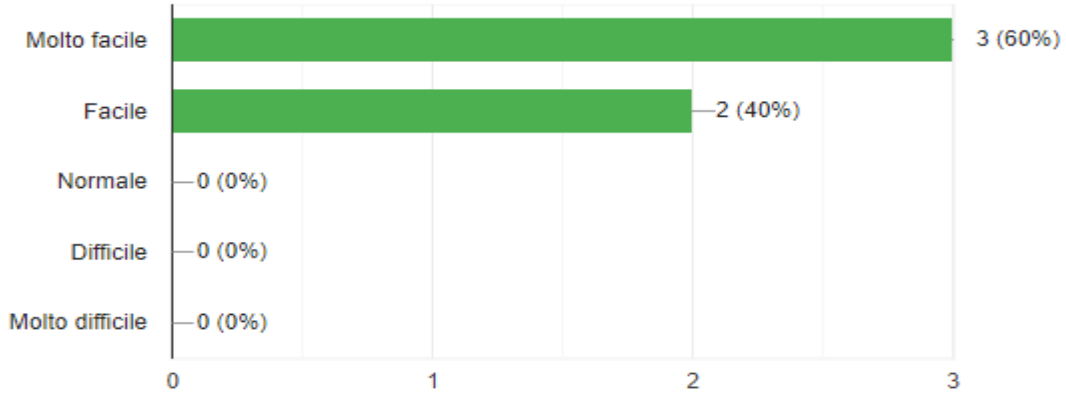
Quanto è facile recensire un itinerario?

5 risposte



Quanto è facile segnalare un itinerario?

5 risposte



Firebase e monitoraggio performance

Non meno importante è stata l'integrazione di Firebase nel nostro applicativo al fine di monitorare diversi aspetti. Firebase è un servizio che ci ha permesso di analizzare in quanto tempo gli utenti svolgevano determinate azioni e inoltre ci permette la possibilità di monitorare il tempo di avvio o eventuali crash dell'applicazione.

Per rendere facile l'utilizzo di Firebase nel nostro Frontend abbiamo creato una classe (*singleton*) che ha principalmente due metodi:

- startTrace (String traceName);
- stopTrace (String traceName);

Il primo metodo ci permette da quel punto in poi di avviare un timer per il *traceName* che ci interessa. Cosa sono i Trace? Nel gergo di firebase esse sono Tracce che monitorano il tempo in cui determinate azioni vengono svolte. Di base firebase ne offre già svariate ma al fine della valutazione delle azioni che ci interessavano, abbiamo creato delle tracce personalizzate.

Infine, il secondo metodo ferma il timer e ci informa del tempo trascorso a partire dall'avvio del *traceName* in considerazione.

Abbiamo monitorato le seguenti tracce:

- login_natour21 (accesso tramite natour21)
- insertPost (inserimento di un post tramite mappa interattiva)
- insertReview (inserimento di una recensione)
- insertReport (inserimento di una segnalazione)
- send_new_message (invio di un messaggio ad un altro utente)

Questo tipo di test, utilizzando firebase è stato eseguito scegliendo un altro gruppo di tester.

Per ognuno di loro sarà possibile osservare il relativo file di log dell'utilizzo dell'applicazione, pertanto saranno consegnati i seguenti file:

- logFile_tester1.txt
- logFile_tester2.txt
- logFile_tester3.txt
- logFile_tester4.txt
- logFile_tester5.txt

Dall’analisi di questi log si evincono i seguenti risultati:

	login_natour21	insertPost	insertReview	insertReport	send_new_message
1	12205,048ms	32356,981ms	14260,753ms	11304,944ms	25090,799ms
2	16332,151ms	41100,078ms	9819,272ms	14548,561ms	9838,503
3	18531,327ms	55766,915ms	21395,135ms	16065,185ms	6925,859
4	18037,585ms	58044,412ms	12040,832ms	15157,941ms	11799,484ms
5	13844,986ms	42256,008ms	12007,613ms	10205,463ms	14599,21ms

La media per ogni trace risulta essere:

- login_natour21 15,79 secondi
- insertPost 45,90 secondi
- insertReview 13,90 secondi
- insertReport 13,45 secondi
- send_new_message 13,65 secondi

Inoltre, dai log è possibile valutare il tempo di risposta da parte del backend (server springboot) che risulta essere di circa 200ms !

Testing

Prima di parlare del codice sviluppato per testare alcuni metodi bisogna fare delle premesse. Al fine di avere test utili e con corretta valutazione del codice è necessario che sul database siano stati precedentemente inseriti due account, entrambi con autenticazione NaTour21. Per facilitare tale opzione abbiamo lasciato il seguente codice nel runner del server springboot nella classe NaTour21Application:

```
@Bean
CommandLineRunner run(UserService userService) {
    return args -> {
        userService.saveUser(new User( email: "seta.mik@gmail.com", username: "mike.fonseta", password: "admin12",
                Auth.NATOUR21.toString(), role: "ROLE_USER", facebookId: null));
        userService.saveUser(new User( email: "mike.fonseta@gmail.com", username: "mike13", password: "admin12",
                Auth.NATOUR21.toString(), role: "ROLE_USER", facebookId: null));
    };
}
```

Questo permetterà all'avvio del server di inserire due account sul database con le relative informazioni.

Inoltre, per il corretto avvio del server springboot è necessario che nel file application.properties del server spring boot ci sia tale impostazione

```
spring.jpa.hibernate.ddl-auto=create
```

Questa impostazione inizializzerà il database in modo tale da non aver conflitti nell'inserimento dei due account qui sopra riportati.

Nel caso in cui si voglia testare il codice quando l'applicativo risulta già online per far si che i dati degli utenti non vengano persi dal database sarà necessario cambiare l'impostazione vista poco fa in:

```
spring.jpa.hibernate.ddl-auto=update
```

Infine, come vedremo in seguito nel codice dei test è stato necessario l'utilizzo di alcune Annotation per le classi di test sul server springboot:

```
@SpringBootTest
@RunWith(SpringRunner.class)
```

```
@Autowired
private UserController userController;
```
(getUserAuhtTest)

```
@Autowired
private MessageController messageController;
```
(sendMessageTest)

Queste hanno permesso alle classi dei test di avviare correttamente il server al fine di poterlo utilizzare correttamente.

I metodi che abbiamo scelto per il testing sono:

- **getUserAuth** : questo metodo è presente nella classe UserController del server springboot e grazie ad esso è possibile reperire il tipo di autenticazione dell'utente (NATOUR21, FACEBOOK, GOOGLE).
  Come unico attributo troviamo il nome utente.
- **sendMessage:** questo metodo è presente nella classe MessageController del server springboot e grazie ad esso è possibile inviare un messaggio ad un altro utente.
  Come attributo troviamo un oggetto di tipo MessageRequest che è composto dai seguenti attributi:
  - **from**: persona che vuole inviare un messaggio
  - **to**: persona al quale inviare il messaggio
  - **content**: messaggio da inviare
- **getImage:** questo metodo si trova nella classe ImagePicker dell'applicativo android e grazie ad esso è possibile assegnare immagini agli utenti in base alla lunghezza del loro nome.
  Come unico attributo troviamo il nome utente.

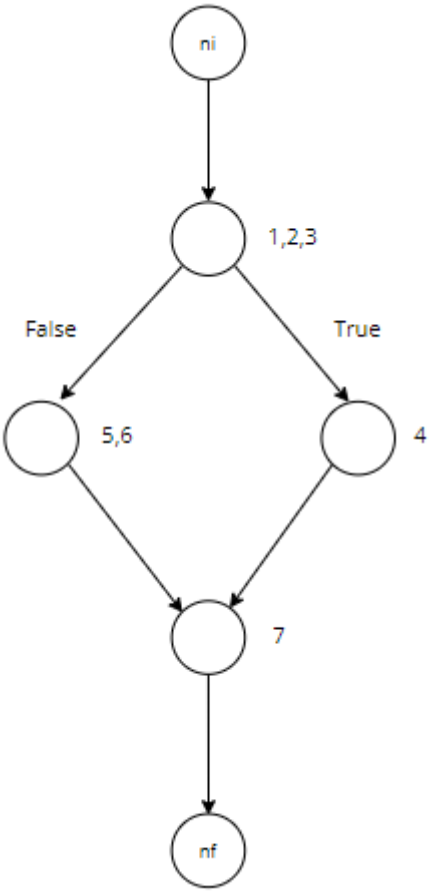
Adotteremo per i primi due metodi la strategia *White-Box* e per il terzo metodo la strategia *Black-Box.*

getUserAuth

Il codice del metodo *getUserAuth* è il seguente:

```
public ResponseEntity<BasicResponse>getUserAuth(@Param("username") String username) {
1   User user = userService.getUser(username.toLowerCase(Locale.ROOT));
2   BasicResponse response;
3   if(user != null) {
4       response = new BasicResponse(user.getAuth(),  status: "OK");
    }
5   else
    {
6       response = new BasicResponse( result: "Nome utente non registrato",  status: "FAILED");
    }
7   return ResponseEntity.ok().body(response);
}
```

e il suo CFG (Control Flow Graph) è:



Dal CFG ricaviamo i seguenti metodi:

- test_1_2_3_4_7
- test_1_2_3_5_6_7

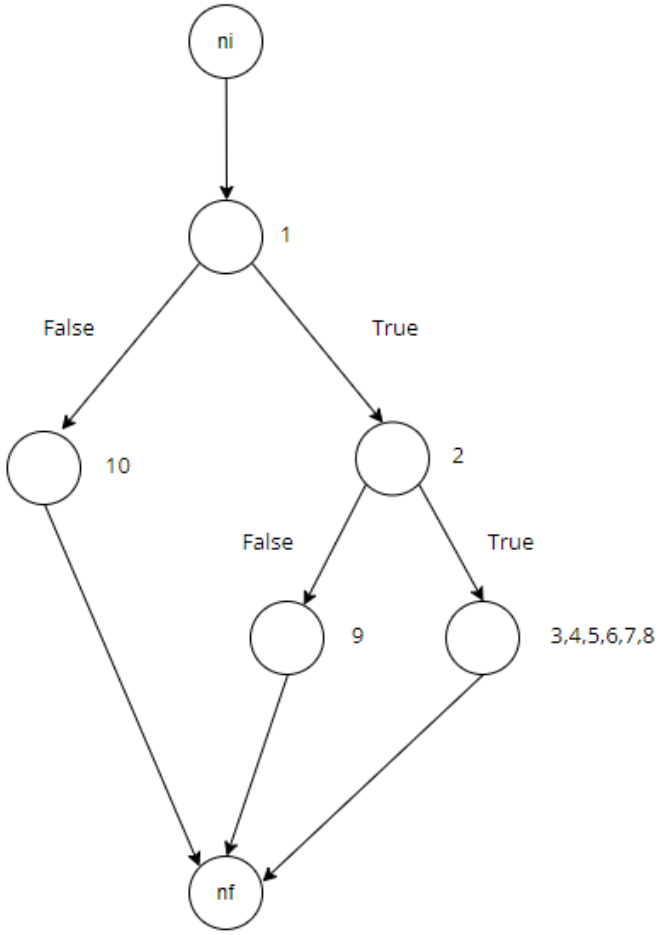
```java
package com.example.NaTour21;

import com.example.NaTour21.User.Controller.UserController;
import com.example.NaTour21.Utils.ResponseTemplate.BasicResponse;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.http.ResponseEntity;
import org.springframework.test.context.junit4.SpringRunner;

import static org.junit.Assert.assertEquals;

@SpringBootTest
@RunWith(SpringRunner.class)
public class getUserAuthTest {

    @Autowired
    private UserController userController;

    String t1 = "mike.fonseta";
    String t2 = "invalidUsername";

    @Test
    public void test_1_2_3_4_7()
    {
        assertEquals(userController.getUserAuth(t1),
                ResponseEntity
                .ok()
                .body(new BasicResponse("NATOUR21", "OK")));
    }

    @Test
    public void test_1_2_3_5_6_7()
    {
        assertEquals(userController.getUserAuth(t2),
                ResponseEntity
                .ok()
                .body(new BasicResponse("Nome utente non registrato","FAILED")));
    }

}
```

sendMessage

Il codice del metodo *sendMessage* è il seguente:

```
@PostMapping("/message/send")
public ResponseEntity<BasicResponse> sendMessage(@RequestBody MessageRequest messageRequest) {
 1  if(userService.getUser(messageRequest.getTo()) != null)
    {
 2      if(userService.getUser(messageRequest.getFrom()) != null
              && messageRequest.getContent() != null
              && !messageRequest.getFrom().equals(messageRequest.getTo())) {
 3          Long time = new Date().getTime();
 4          ChatRoom chatRoom = chatRoomService.saveChatRoom(messageRequest.getTo().toLowerCase(Locale.ROOT),
                    messageRequest.getFrom().toLowerCase(Locale.ROOT), messageRequest.getContent(), time);
 5          messageService.newMessage(new Message( id: null, chatRoom.getId(),
                    messageRequest.getFrom().toLowerCase(Locale.ROOT), messageRequest.getContent(), time));
 6          MessageResponse messageResponse = new MessageResponse(messageRequest.getFrom().toLowerCase(Locale.ROOT), messageRequest.getContent());
 7          PusherManager.SendMessage(messageRequest.getTo().toLowerCase(Locale.ROOT), messageResponse);
 8          return ResponseEntity.ok(new BasicResponse(messageResponse,  status: "OK"));
      }
 9      return ResponseEntity.ok(new BasicResponse( result: "Impossibile inviare il messagio",  status: "FAILED"));
    }
10  return ResponseEntity.ok(new BasicResponse( result: "Impossibile trovare " + messageRequest.getTo(),  status: "FAILED"));
}
```

E il suo CFG (Control Flow Graph) è:



Dal CFG ricaviamo i seguenti metodi:

- test_1_2_3_4_5_6_7_8
- test_1_2_9
- test_1_10

```java
package com.example.NaTour21;

import com.example.NaTour21.Message.Controller.MessageController;
import com.example.NaTour21.Utils.RequestTemplate.MessageRequest;
import com.example.NaTour21.Utils.ResponseTemplate.BasicResponse;
import com.example.NaTour21.Utils.ResponseTemplate.MessageResponse;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.http.ResponseEntity;
import org.springframework.test.context.junit4.SpringRunner;

import static org.junit.Assert.assertEquals;

@SpringBootTest
@RunWith(SpringRunner.class)
public class sendMessageTest {

    @Autowired
    private MessageController messageController;

    String validUsername1 = "mike.fonseta";
    String validUsername2 = "mike13";
    String invalidUsername = "invalidUsername";
    String message = "Ciao!";

    @Test
    public void test_1_10()
    {
        assertEquals(messageController.sendMessage(
                new MessageRequest(validUsername1,invalidUsername,message)),
                ResponseEntity.ok(new BasicResponse("Impossibile trovare
invalidUsername", "FAILED")));
    }

    @Test
    public void test_1_2_9()
    {
        assertEquals(messageController.sendMessage(
                new MessageRequest(invalidUsername,validUsername1,message)),
                ResponseEntity.ok(new BasicResponse("Impossibile inviare il
messagio", "FAILED")));
    }

    @Test
    public void test_1_2_3_4_5_6_7_8()
    {
        assertEquals(messageController.sendMessage(
                new MessageRequest(validUsername1,validUsername2,message)),
                ResponseEntity.ok(new BasicResponse(
               new MessageResponse("mike.fonseta", message), "OK")));
    }

}
```

getImage

Il codice del metodo *getImage* è il seguente:

```
public static int getImage(String username) {
    if (username != null) {

        if (username.length() == 4 || username.length() == 5) {
            return R.drawable.user1;
        } else if (username.length() == 6 || username.length() == 7) {
            return R.drawable.user2;
        } else if (username.length() == 8 || username.length() == 9) {
            return R.drawable.user3;
        } else if (username.length() == 10 || username.length() == 11) {
            return R.drawable.user4;
        } else if (username.length() == 12 || username.length() == 13) {
            return R.drawable.user5;
        } else if (username.length() == 14 || username.length() == 15) {
            return R.drawable.user6;
        } else if (username.length() == 16 || username.length() == 17) {
            return R.drawable.user7;
        } else if (username.length() == 18 || username.length() == 19) {
            return R.drawable.user8;
        } else if (username.length() == 20 || username.length() == 21) {
            return R.drawable.user9;
        } else if (username.length() == 22 || username.length() == 23) {
            return R.drawable.user10;
        } else if (username.length() > 23 && username.length() <= 27) {
            return R.drawable.user11;
        } else if (username.length() > 27 && username.length() <= 32) {
            return R.drawable.user12;
        }
        return R.drawable.user1;
    } else {
        return R.drawable.user1;
    }
}
```

Il suo unico attributo è **username** ed esso potrà essere una stringa dai 4 ai 32 caratteri stando ai limiti imposti dalla nostra registrazione. Applicando la strategia Black-Box andremo a testare i seguenti valori:

- username di lunghezza minore di 4 caratteri (test1)
- username di lunghezza uguale a 4 caratteri (test2)
- username di lunghezza uguale a 18 caratteri (test3)
- username di lunghezza uguale a 32 caratteri (test4)
- username di lunghezza maggiore di 32 caratteri (test5)

ci aspetteremo rispettivamente tali risultati:

 user1 (<4 caratteri)

 user1 (=4 caratteri)

 user8 (=18 caratteri)

 user12 (=32 caratteri)

 user1 (>32 caratteri)

In questo caso nel codice non saranno presenti annotazioni se non quelle di junit `(@Test)`

```java
package com.example.natour21;

import static org.junit.jupiter.api.Assertions.assertEquals;
import com.example.natour21.Utils.ImagePicker;
import org.junit.Test;

public class getImageTest {

    String t1 = "mik";
    String t2 = "mike";
    String t3 = "mike.fonseta171217";
    String t4 = "mike.fonseta17121719235461746231";
    String t5 = "mike.fonseta171217192354617462318712625";

    @Test
    public void test1(){
        assertEquals(ImagePicker.getImage(t1), R.drawable.user1);
    }

    @Test
    public void test2(){
        assertEquals(ImagePicker.getImage(t2), R.drawable.user1);
    }

    @Test
    public void test3(){
        assertEquals(ImagePicker.getImage(t3), R.drawable.user8);
    }

    @Test
    public void test4(){
        assertEquals(ImagePicker.getImage(t4), R.drawable.user12);
    }

    @Test
    public void test5(){
        assertEquals(ImagePicker.getImage(t5), R.drawable.user1);
    }
}
```