

**UNIVERSITÀ DEGLI STUDI DI NAPOLI
FEDERICO II**



CORSO DI INGEGNERIA DEL SOFTWARE

DIPARTIMENTO DI INGEGNERIA ELETTRICA E TECNOLOGIE
DELL'INFORMAZIONE

NATOUR21

<UNA MODERNA PIATTAFORMA SOCIAL PER APPASSIONATI DI ESCURSIONI>

Candidati:

Luca Bianco - N86003200
Antonio Garofalo - N86003129

Ordered by:
SoftEngUniNA

ANNO ACCADEMICO 2021/2022

Indice

1 Introduzione	4
1.1 Che cos'è NaTour21?	4
1.2 Cosa offre?	4
1.3 Cosa contiene questa documentazione?	4
1.4 Tecnologia adoperata	5
1.5 Versioni e informazioni	5
2 Modello Funzionale	6
2.1 Requisiti dell'applicazione	6
2.1.1 Funzionali	6
2.1.2 Non funzionali	7
2.1.3 Dominio	7
2.2 Modellazione dei casi d'uso	8
2.2.1 Use Case Generale	8
2.2.2 Use Case Autenticazione	10
2.2.3 Use Case Gestione Compilation	11
2.2.4 Use Case Inserisci Itinerario	12
2.2.5 Use case Visualizza profilo	13
2.3 Tabelle di Cockburn	14
2.4 Mockup	19
2.4.1 Carica Foto	20
2.4.2 Pubblicazione nuovo itinerario	25
2.5 Presentazione dell'idea progettuale	32
2.5.1 Che cos'è il trekking?	32
2.5.2 Analisi delle funzionalità	32
2.5.3 Stato di sviluppo delle funzionalità	34
2.6 Individuazione del target di utenti	37
2.6.1 Personas	38
2.7 Valutazione dell'usabilità a priori	42
2.7.1 Tabelle di valutazione	44
2.7.2 Tecnica utilizzata	45
2.8 Prototipazione funzionale via statechart dell'interfaccia grafica	47
2.9 Glossario	49
2.9.1 Termini	49
2.9.2 Acronimi	51
3 Modelli di Dominio	53
3.1 Classi, oggetti e relazioni di analisi	54
3.1.1 Classi ed entità	54
3.1.2 Class diagram di analisi	55
3.1.3 Class diagram - Collezione	55
3.1.4 Class diagram - Login	56

3.1.5	Class diagram - Registrazione	57
3.1.6	Class diagram - PhotoAdd	58
3.1.7	Class diagram - Profilo	59
3.1.8	Class diagram - Pubblica Nuovo Itinerario	60
3.1.9	Class diagram - Support	61
3.1.10	Class diagram - Visualizza e ricerca itinerario	62
3.2	Sequence Diagram	63
3.2.1	Sequence Diagram - Inserisci Itinerario	63
3.2.2	Sequence Diagram - Carica Foto	64
3.3	Activity Diagram	65
3.3.1	Activity Diagram - Compilation	65
3.3.2	Activity Diagram - Inserimento nuovo itinerario (Manuale)	66
3.3.3	Activity Diagram - Inserimento nuovo itinerario (Registrato)	67
3.3.4	Activity Diagram - Inserisci Foto	68
3.3.5	Activity Diagram - Autenticazione	69
3.3.6	Activity Diagram - Ricerca Itinerari	70
3.3.7	Activity Diagram - Visualizza Profilo	71
4	Design di Sistema	72
4.1	Analisi Architetturale	72
4.1.1	Descrizione architettura cloud	73
4.1.2	Il server	76
4.1.3	REST API	78
4.1.4	Il client	79
4.1.5	Supporti	81
4.2	Gantt Diagram	82
4.3	Class Diagram di Design	83
4.3.1	Class Diagram Design - Login	83
4.3.2	Class Diagram Design - Registrazione	84
4.3.3	Class Diagram Design - Gestione Collezione	85
4.3.4	Class Diagram Design - Nuova Collezione	86
4.3.5	Class Diagram Design - Pubblicazione Itinerario	87
4.3.6	Class Diagram Design - Registrazione Itinerario	88
4.3.7	Class Diagram Design - Visualizzazione Itinerari	89
4.4	Sequence Diagram di Design	90
4.4.1	Sequence Diagram Design - Inserimento Itinerario	91
4.4.2	Sequence Diagram Design - Carica Foto	92
4.5	Gerarchie Funzionali	93
4.5.1	Login	93
4.5.2	HomePage pt.1	93
4.5.3	HomePage pt.2	94

5 Codice sorgente sviluppato	95
5.1 Versionamento e licenza	95
5.1.1 Permessi	95
5.1.2 Limitazioni	95
5.1.3 Condizioni	95
6 Codice xUnit	96
6.1 Filter Query	96
6.2 Request Generator	101
6.2.1 BlackBox	101
6.2.2 WhiteBox	105
6.3 Verifica policy	107
7 Valutazione dell’usabilità sul campo	110
7.1 Metodo euristico	110
7.2 Metodo con analisi di logging e monitoraggio	114

1 Introduzione

La società SoftEngUniNA commissiona al gruppo progetto INGSW2122_N_05 la realizzazione di un sistema informatico multi-piattaforma atta a commercializzare un applicativo software chiamato NaTour21.

1.1 Che cos'è NaTour21?

NaTour è un servizio definito come un "Moderno social network multi-piattaforma per appassionati di escursioni".

1.2 Cosa offre?

L'applicativo nella sua prima versione di lancio offrirà le seguenti funzionalità richieste dal mittente del progetto:

- Login/Registrazione di utenti sulla piattaforma mediante email, Google Login e Facebook Login.
- Possibilità di inserire itinerari personalizzati, aggiungendo caratteristiche specifiche dell'itinerario (i.e Difficoltà, Tracciato, etc...) con la possibilità di farlo in-place o tramite file GPX.
- Possibilità di fare un tracciato personalizzato tramite GPS, dall'applicazione stessa NaTour21.
- Effettuare ricerche di itinerari di altri utenti registrati con vari filtri integrati nella ricerca.
- Possibilità di creare "Compilation" personalizzate di sentieri propri o altrui con descrizione e titolo personalizzati.
- Possibilità di caricare fotografie (filtrate da apposito controllo) dei tracciati percorsi ed inserirne le coordinate di scatto della fotografia.

1.3 Cosa contiene questa documentazione?

La documentazione contiene:

- Documento dei Requisiti Software.
- Documento di Design del sistema.
- Codice sorgente (Link Drive).
- Definizione di un piano di testing e valutazione sul campo dell'usabilità.

1.4 Tecnologia adoperata

L'applicativo è sviluppato interamente in Android in linguaggio Object-Oriented (in questo caso Java), tale tecnologia è affiancata da un sistema di logging per testing efficace (i.e sistema MBaaS Firebase) e un Backend attraverso tecnologie allo stato dell'arte quali servizi di public Cloud Computing come **Azure** o **AWS**, al fine di massimizzare la scalabilità del sistema in vista di un possibile repentino aumento del numero degli utenti nelle fasi iniziali di rilascio al pubblico.

1.5 Versioni e informazioni

L'applicativo è rilasciato ad una versione stabile delle funzionalità richieste come versione pilota da rilasciare sul mercato, sono altresì prese in considerazione altre richieste dal committente del progetto che fanno sperare in un continuo futuro sviluppo dell'applicativo.

Per qualsiasi informazione, fare riferimento ai seguenti contatti:

- **CEO** - Luca Bianco (N86003200)
 - Email: luca.bianco4@studenti.unina.it
- **CEO** - Antonio Garofalo (N86003129)
 - Email: antonio.garofalo12@studenti.unina.it

Testi utilizzati per arricchire questa documentazione e relativo progetto

- Facile da Usare[[1](#)];
- Clean Code[[2](#)];
- Clean Architecture[[3](#)];

2 Modello Funzionale

In questa sezione andremo a descrivere il modello funzionale del software partendo da un analisi dei casi d'uso assegnati per l'applicativo, ciò che verrà descritto è il risultato della riunione di consulto con gli "StackHolders" per definire i requisiti funzionali del software.

2.1 Requisiti dell'applicazione

2.1.1 Funzionali

Visualizzazione di itinerari Visualizzare itinerari propri e pubblicati da altri utenti.

Pubblicazione di itinerari Pubblicare itinerari in piattaforma e renderli pubblici alla community NaTour21, manualmente o registrandoli tramite sequenza di posizioni GPS oppure caricando file GPX.

Ricerca degli itinerari Ricerca tramite nome o per filtri selezionati degli itinerari in piattaforma.

Autenticazione locale Autenticazione tramite la creazione di un account NaTour21 mediante utilizzo di un email valida.

Autenticazione social Autenticazione tramite social network, in particolare tramite il provider OAuth 2.0 di Meta e Google.

Gestione di compilation L'utente ha la possibilità di creare, modificare ed eliminare compilation di itinerari.

Inserimento di itinerario in compilation L'utente ha la possibilità di inserire uno o più itinerari all'interno di una compilation propria.

Gestione e visualizzazione del profilo L'utente in quanto iscritto ad un social network ha la possibilità di gestire un proprio profilo personale dove è possibile anche visualizzare itinerari pubblicati.

2.1.2 Non funzionali

Usabilità L'applicazione è stata sottoposta a una ricerca nel dettaglio di "User Personas" e monitoraggio tramite servizi official Google.inc.

Scalabilità Il sistema deve avere un back-end in cloud scalabile per adattarsi a frequenze di accesso elevate.

Password policy security Le password dell'utente verrano salvate in cloud con un controllo regex avanzato (vedesi testing) e sicurezza su crittografia SHA-256.

Performance di ricerca tramite filtri Il sistema implementa una gestione delle query di ricerca tramite filtri con pochi metodi e alte prestazioni, il sistema è in grado di rispondere in tempi pari a singola ricerca (circa 3 secondi).

Astrazione al datastore Il back-end è capace di adattarsi (mediante modifica di parametro source) a qualunque sottosistema di storage (i.e SQL, NOSQL).

Utilizzo di single-activity, multi-fragment L'applicazione utilizza per fluidità e facile gestione il pattern di single-activity e multi-fragment in modo da dare precisi scopi alle activity che gestiscono fragment comuni.

2.1.3 Dominio

ISO/IEC Il sistema deve essere conforme alle direttive ISO/IEC del trattamento dei dati privati su servizi di hosting in cloiud¹

GDPR Il sistema segue le direttive euorpee sulla GDPR consultabili qui², mentre quelle delle policy della privacy qui³.

¹<https://www.iso.org/organization/70.html>.

²<https://www.freeprivacypolicy.com/live/35a19d31-61fc-46b4-9c5f-036e836a49ec>

³<https://www.freeprivacypolicy.com/live/39fe83f2-4d7a-4c8d-bc69-09b5680533ee>

2.2 Modellazione dei casi d'uso

Per la documentazione dei casi d'uso è stato utilizzato un software CASE Tool, chiamato Visual Paradigm⁴.

2.2.1 Use Case Generale

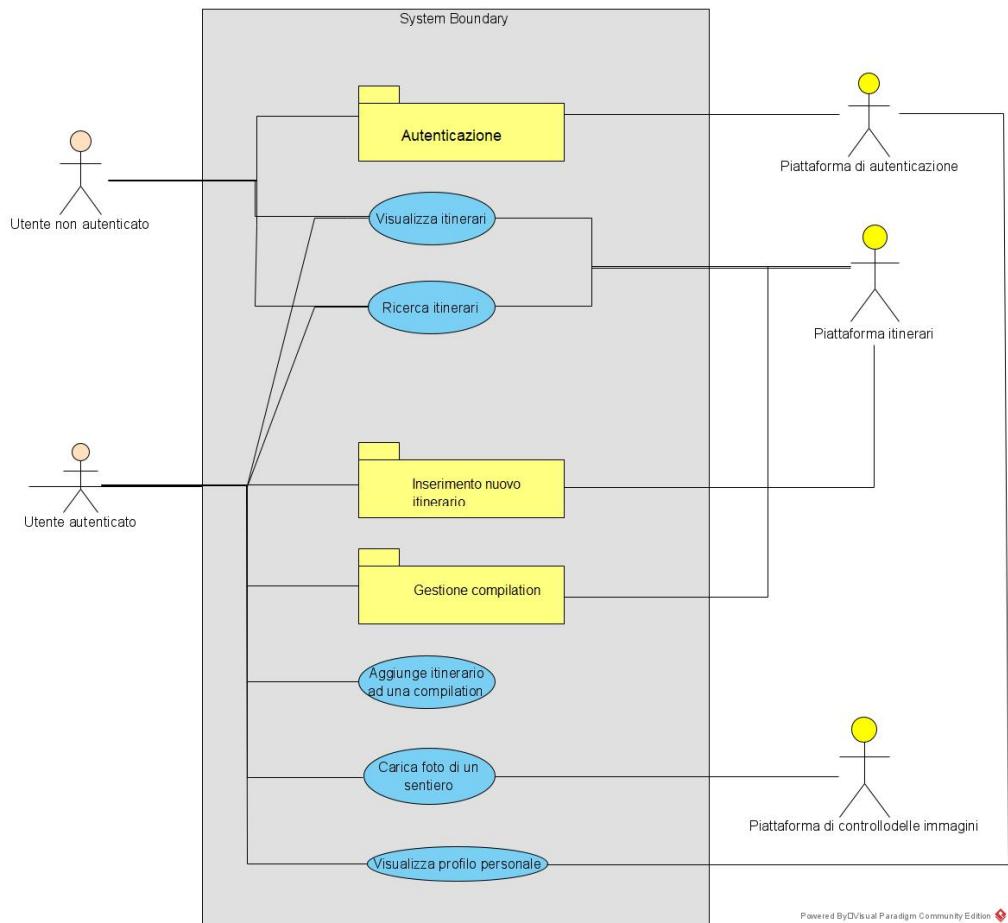


Figura 1: Use Case Generale

Come sopra citato questo use case in 1 è nato dopo l'intervista fatta con gli StakeHolders di SoftEngUniNA e rappresenta l'insieme delle funzionalità del sistema assegnate alla software house.

Le funzionalità più complesse sono rappresentate mediante "Package" e verranno specificate in seguito.

⁴<https://www.visual-paradigm.com/editions/community/>

In oltre abbiamo, per completezza, aggiunto la funzionalità "Visualizza Profilo" che non ci è stata assegnata ma che abbiamo concluso essere una funzionalità utile anche per sviluppi futuri.

2.2.2 Use Case Autenticazione

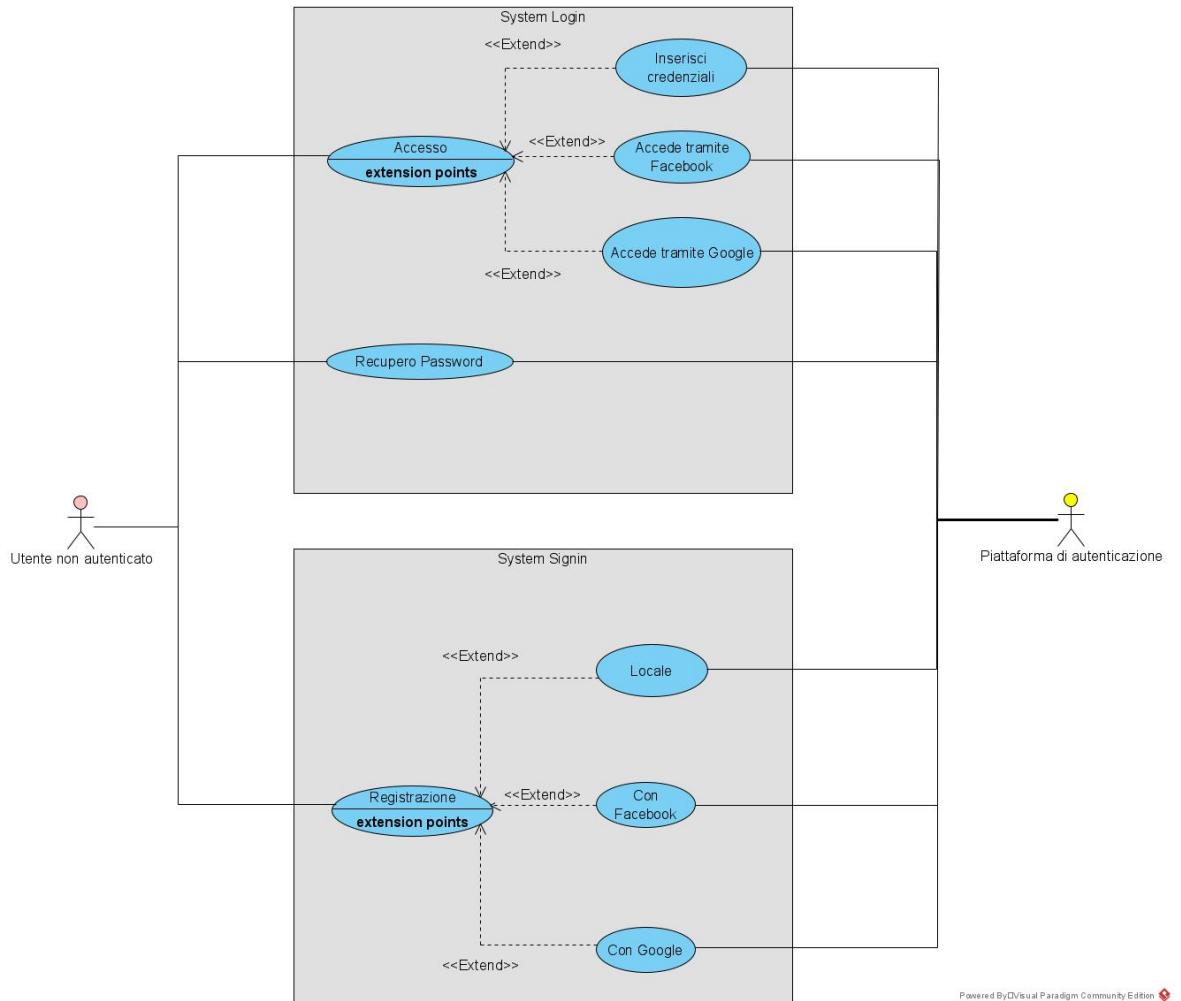


Figura 2: Autenticazione

In questo Use Case abbiamo descritto la fase di autenticazione (Login e registrazione) sotto il punto di vista dell'utente non ancora autenticato. D'altra parte abbiamo come attore secondario la piattaforma di autenticazione che gestisce sia la registrazione che il login con piattaforme come Facebook e Google oppure manualmente direttamente dall'applicazione.

2.2.3 Use Case Gestione Compilation

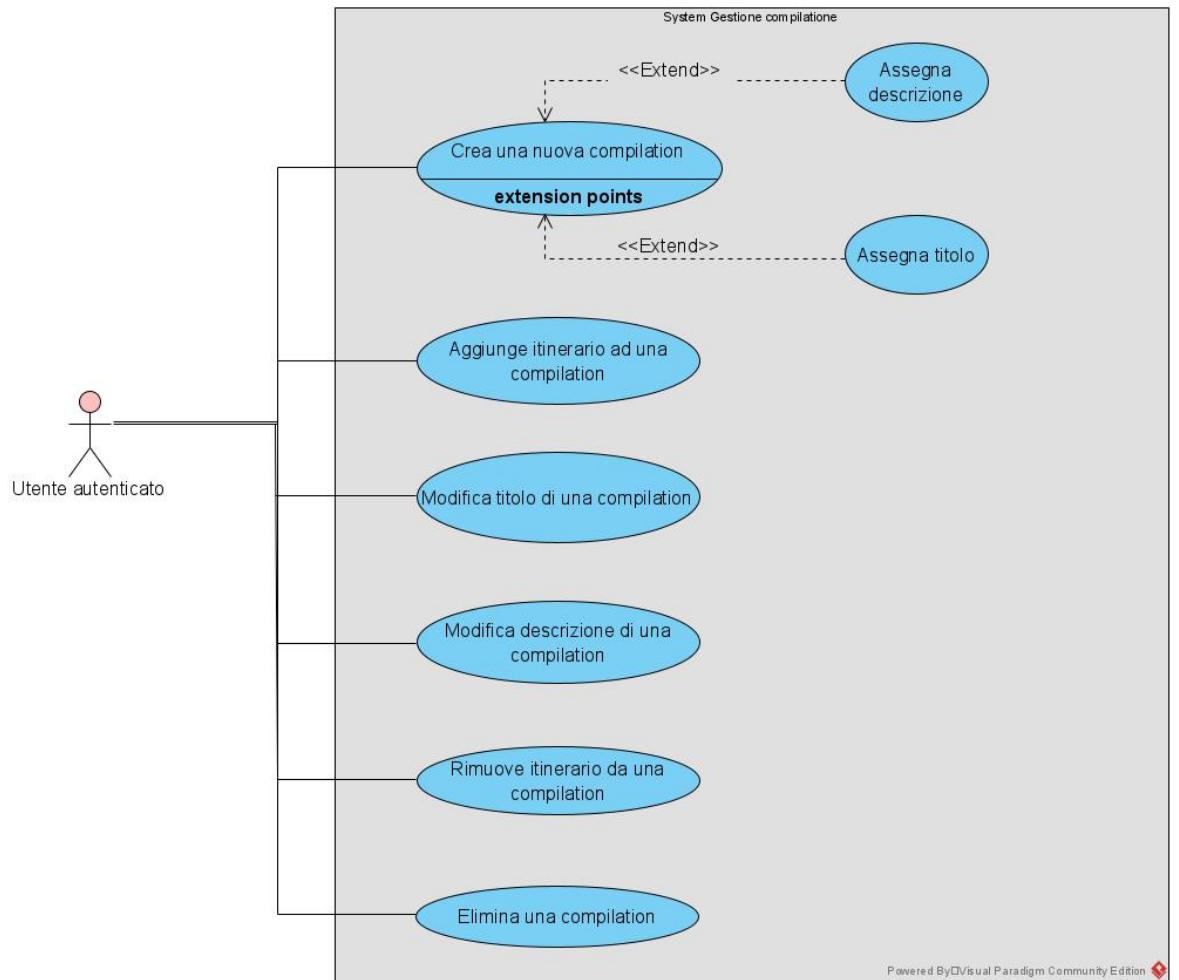


Figura 3: Gestione Compilation

Nella figura 3 è descritto l'insieme delle azioni riguardanti le compilation (i.e Playlist di Spotify) che un utente AUTENTICATO può fare.
Distinguiamo le seguenti azioni:

- Creare una compilation;
- Aggiungere un itinerario alla compilation;
- Modifica della compilation;
- Rimuovere itinerario da compilation;
- Elimina compilation;

2.2.4 Use Case Inserisci Itinerario

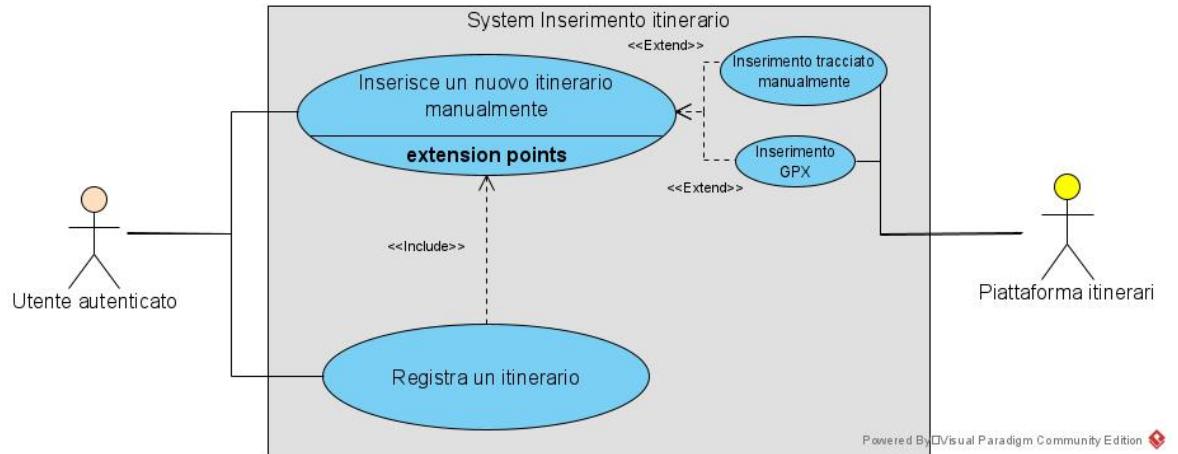


Figura 4: Inserisci Itinerario

In questo use case in 4 abbiamo rappresentato le due modalità di inserimento di un itinerario. L’utente AUTENTICATO può scegliere se pubblicare l’itinerario manualmente inserendo tutti i dati necessari oppure caricare un file GPX dove estrarre il tracciato.

La seconda modalità è quella di registrare l’itinerario mentre l’utente cammina.

2.2.5 Use case Visualizza profilo

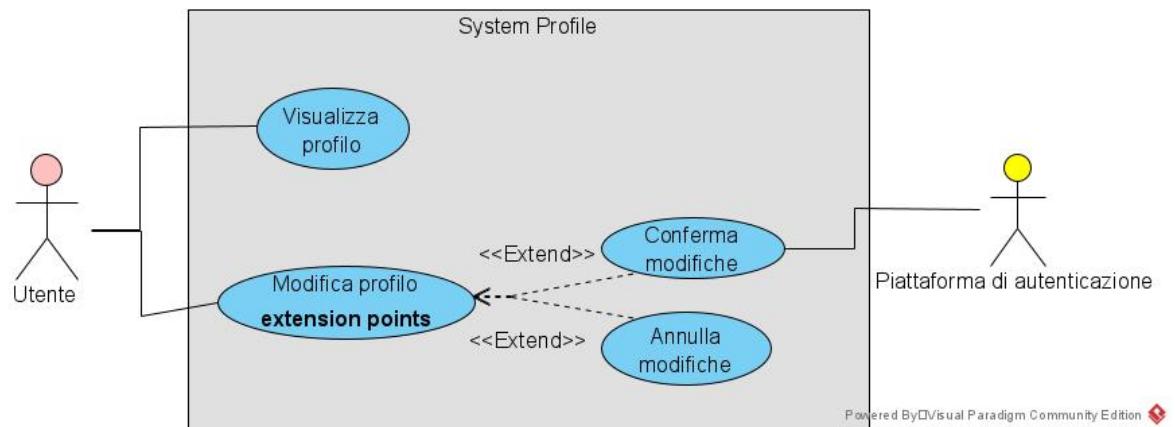


Figura 5: Visualizza profilo

Visualizza profilo è un caso che è stato aggiunto dalla software house per completezza dato che NaTour21 si registra come "Social Network".

2.3 Tabelle di Cockburn

Il committente ci ha richiesto di presentare tra i casi d'uso assegnati due casi specifici a scelta descrivendoli tramite tabelle di Cockburn.

Formalismo Le tabelle di Cockburn (create da Alistair Cockburn) sono un formalismo di rappresentazione dei casi d'uso, la politica adotta la rappresentazione di un Main scenario nella quale uno o più attori interagiscono tra loro attraverso l'invocazione di trigger e descrivendo (in formato tabellare) gli eventi.

Per situazioni estranee al main scenario abbiamo pensato di rappresentare anche le Extension e le Subvariation per rappresentare scenari di errore o anche strade alternative per raggiungere al fine dello scenario rappresentato dallo use case.

Abbiamo deciso di rappresentare i seguenti use case:

- Inserimento di un nuovo itinerario.
- Caricamento di una nuova foto nel sistema.

USE CASE #01	Inserimento nuovo itinerario in piattaforma		
Goal in Context	L'utente autenticato vuole pubblicare un nuovo itinerario in piattaforma.		
Preconditions	L'utente deve essere autenticato.		
Success End Condition	L'utente è riuscito ad inserire in piattaforma un nuovo itinerario.		
Failed End Condition	L'utente non è riuscito ad inserire in piattaforma un nuovo itinerario.		
Primary Actor	Utente Autenticato.		
Trigger	Entra nell'Home page.		
DESCRIPTION	Step n°	Utente Autenticato	Sistema
	1	Clicca sul bottone "Aggiungi percorso" di MockupHomePage.	
	2		Mostra Mockup InserisciNuovoltinerario.
	3	Inserisce il titolo dell'itinerario.	
	4	Inserisce la descrizione dell'itinerario.	
	5	Seleziona altre opzioni.	
	6	Inserisce il tracciato.	
	7		Calcola la durata e la lunghezza del tracciato inserito e lo mostra.
	8		Calcola punto di inizio e punto di fine li mostra.
	9	(Opzionale) Aggiunge POI all'itinerario.	
	10	Preme "Pubblica".	
	11		Mostra popup di inserimento andato a buon fine.
	12		Reindirizza l'utente alla home page.

EXTENSIONS #1		Utente Autenticato	Sistema
Il sistema non riesce a collegarsi al server cloud.			
	<i>13.a</i>		Non riesce a collegarsi al server e mostra un toast con descrizione del problema.
	<i>14.a</i>		Reindirizza Mockup HomePage.
EXTENSIONS #2		Utente Autenticato	Sistema
Itinerario già presente in piattaforma.			
	<i>12.b</i>		Mostra popup di errore inserimento di itinerario già presente.
	<i>13.b</i>		Torna al punto 2 del Main scenario.
SUBVARIATION #1		Utente Autenticato	Sistema
<u>Itinerario caricato tramite file GPX</u>	2	Clicca sull'icona "Carica tramite GPX".	
	3		Richiede i permessi di accesso alla memoria.
	4	Seleziona un file .gpx.	
	5		Controlla i file .gpx selezionato.
	6		Estrae i metadati dal file .gpx.
	7		Mostra itinerario pre-calcolato sulla schermata.
	8	Inserisce campi obbligatori e opzionali.	
	9		Torna al punto 9 del Main scenario.

USE CASE #02	Carica Foto		
Goal in Context	L'utente vuole caricare una foto nelle informazioni di un itinerario.		
Preconditions	L'utente deve essere correttamente registrato alla piattaforma (o possedere un account esterno Facebook/Google).		
Success End Condition	L'utente riesce a pubblicare una foto sulla pagina di un itinerario.		
Failed End Condition	L'utente non riesce a pubblicare la foto.		
Primary Actor	Utente Autenticato.		
Trigger	Clicca "Icona foto" nella pagina informazioni del percorso.		
DESCRIPTION	Step n°	Utente Autenticato	Sistema
	1	Clicca in "Più informazioni" in "Homepage".	
	2		Carica Mockup "Dettagli itinerario".
	3	Clicca su "Icona Foto" in "Dettagli Tracciato".	
	4		Apre galleria.
	5	Seleziona una foto.	
	6		Estrae metadati.
	7		Inserisce marker con metadati estratti.
	8		Controlla correttezza foto.
	9		Inserisce foto nell'album dell'itinerario.

EXTENSIONS #1		Utente Autenticato	Sistema
Non sono presenti metadati.			
	<i>6.a</i>		Tentativo di estrazione dei metadati.
	<i>7.a</i>		Invio feedback negativo all'utente.
	<i>8.a</i>		Torna al punto 3 del Main scenario.
EXTENSIONS #2		Utente Autenticato	Sistema
La foto è inappropriata/offensiva.			
	<i>7.b</i>		La foto viene analizzata dal sistema.
	<i>4.b</i>		Invia feedback negativo.
	<i>5.b</i>		Torna al punto 3 nel Main scenario.

2.4 Mockup

Un **mockup**, o **mock-up**⁵, è una realizzazione a scopo illustrativo o meramente espositivo di un oggetto o un sistema, senza le complete funzioni dell'originale; un mockup può rappresentare la totalità o solo una parte dell'originale di riferimento (già esistente o in fase di progetto), essere in scala reale oppure variata.

I mockup sono stati fatti in 2 fasi della progettazione utilizzando lo stesso tool-case **Axure RP**⁶ attraverso l'utilizzo di implementazioni di varie librerie di design.

Mostriamo di seguito una prima versione del mockup (disponibile nella repository del progetto):

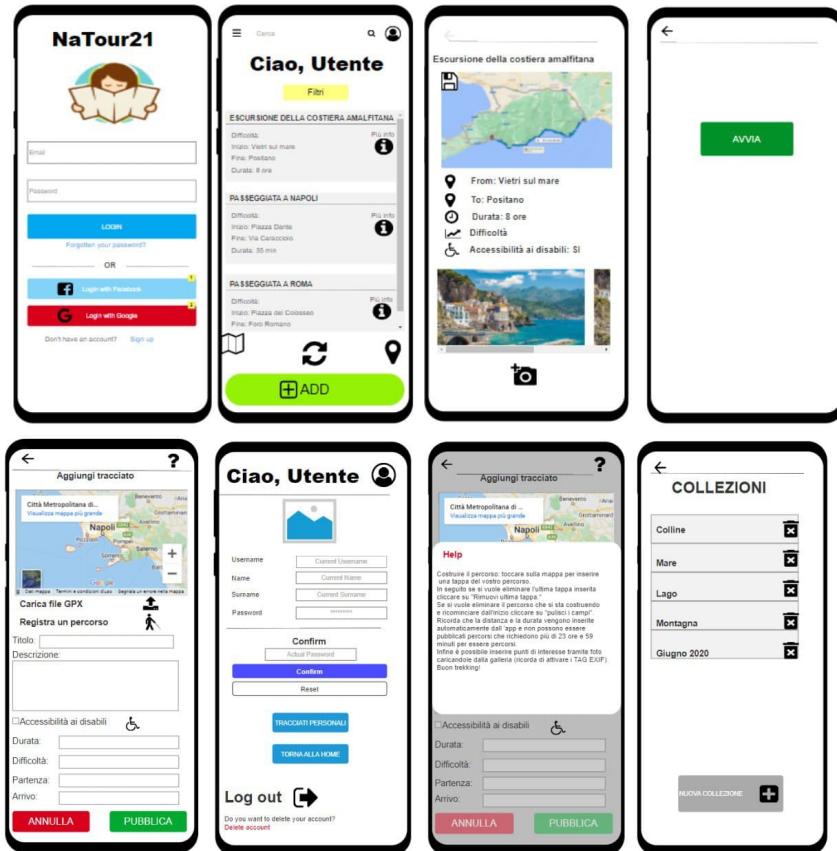


Figura 10: Vecchi mockup

⁵<https://it.wikipedia.org/wiki/Mockup>

⁶<https://www.axure.com/>

Adesso mostriamo i mock-up dell’interfaccia grafica dei due metodi scelti, ovvero: **Pubblicazione di un nuovo itinerario** nella piattaforma (in cui sono stati rappresentati anche due scenari di errore) e il **caricamento di una foto** per un dato itinerario.

Nel footnote di questa pagina è possibile accedere ai mockup vecchi e nuovi dell’applicativo⁷.

2.4.1 Carica Foto

⁷Vecchi mockup qui, nuovi mockup di pubblica itinerario qui e di carica foto qui.

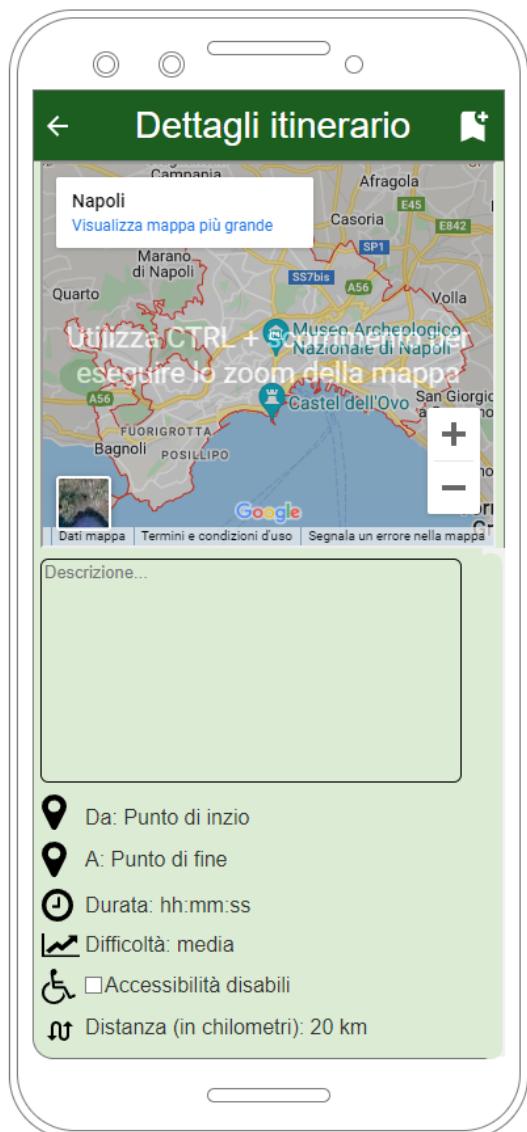


Figura 11: Dettagli itinerario - 1

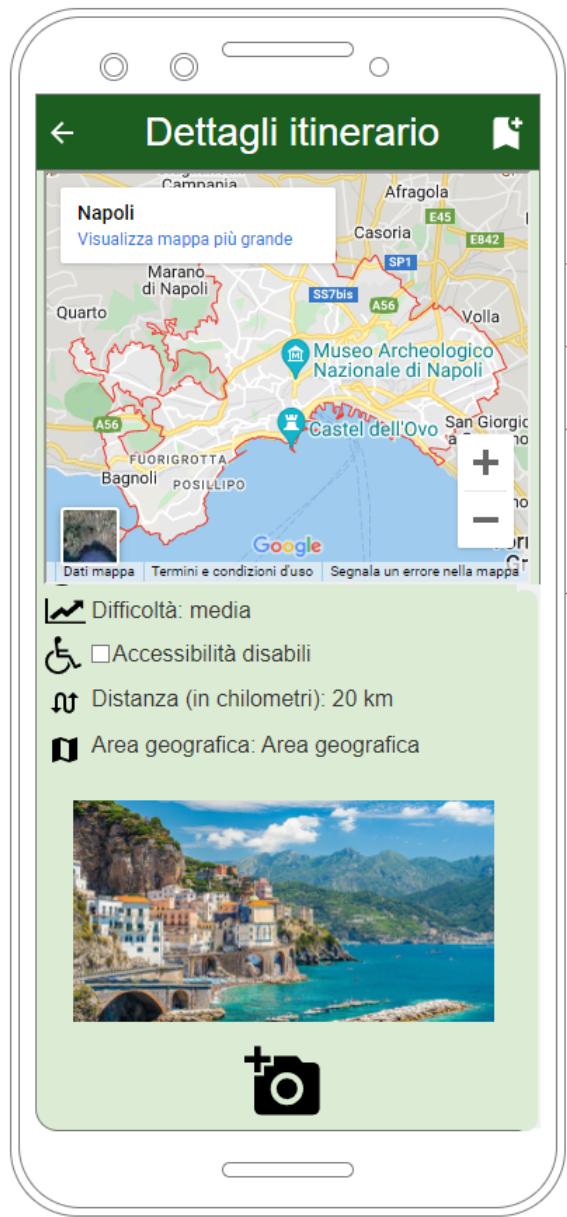


Figura 12: Dettagli itinerario - 2



Figura 13: Pubblica Foto

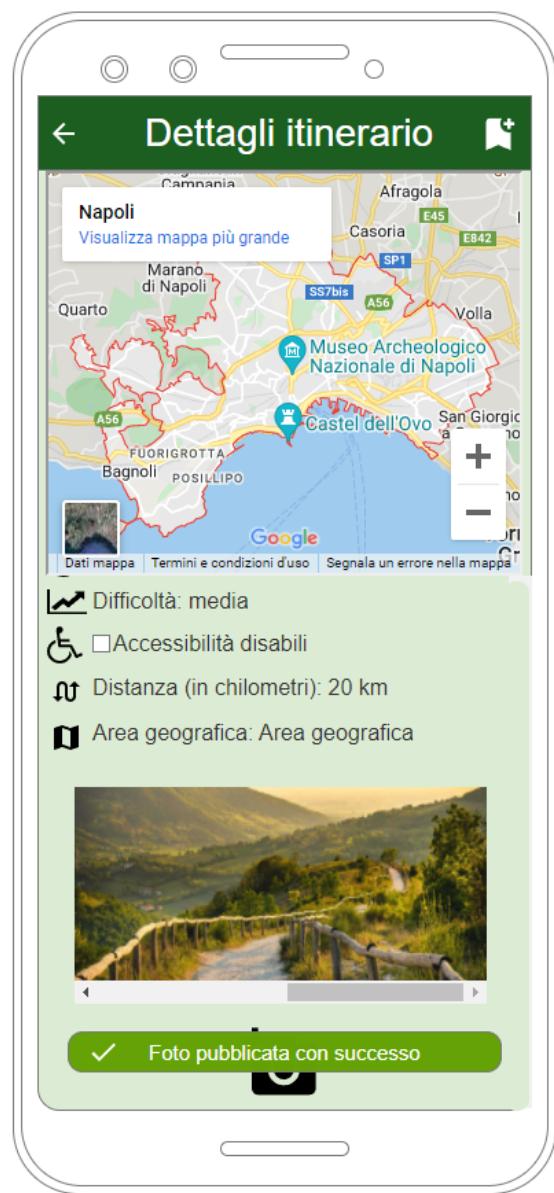


Figura 14: Foto Pubblicata

2.4.2 Pubblicazione nuovo itinerario



Figura 15: Pubblica Itinerario - 1



Figura 16: Pubblica Itinerario - 2

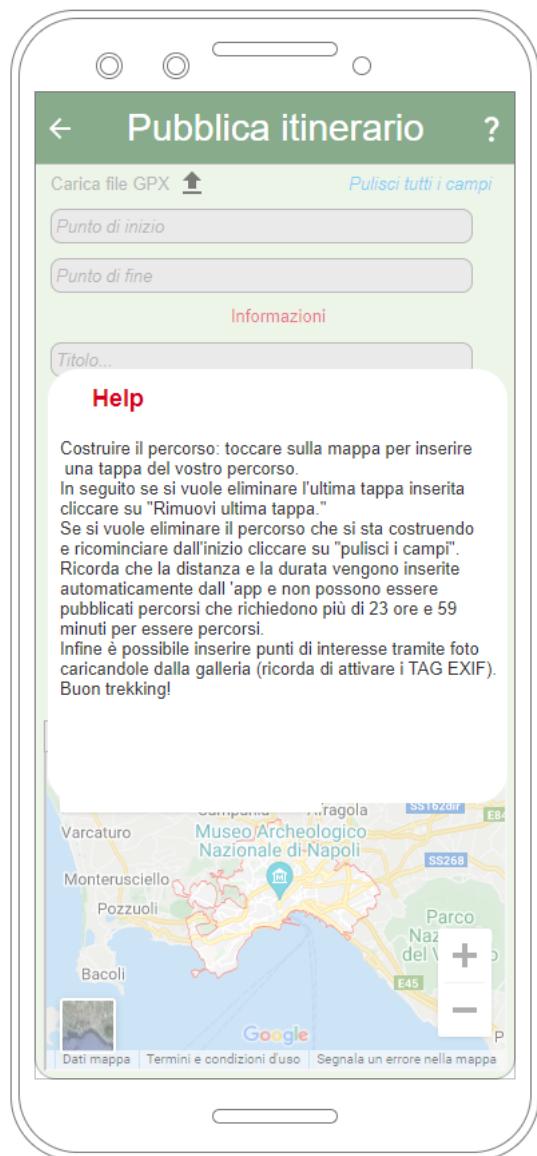


Figura 17: Help



Figura 18: Interesting Point



Figura 19: Success - Itinerario Pubblicato



Figura 20: Error 1 - Itinerario già presente



Figura 21: Errore 2 - Titolo o tappe

2.5 Presentazione dell'idea progettuale

In riferimento alla sezione 1 e ai paragrafi 1.1 e 1.2 di questo documento possiamo presentare più nello specifico NaTour21.

NaTour21 nasce dall'idea di creare un social network che in qualche modo, sotto specifici aspetti del contesto di riferimento (il trekking), metta in contatto amanti della materia, sia esperti che neofiti.

2.5.1 Che cos'è il trekking?

Citando parte del paragrafo di Wikipedia⁸, la traduzione di "Trekking" è parallelamente in italiano "Escursionismo":

"L'escursionismo è un'attività motoria e sportiva basata sul camminare nel territorio a scopo di studio o svago, lungo percorsi poco agevoli che tipicamente non possono essere percorsi con i mezzi di trasporto convenzionali (sentieri, sentieri a lunga percorrenza, alte vie, mulattiere, ippovie ecc.); spesso, derivando il termine dalla lingua inglese, viene indicato anche come trekking o hiking con il primo termine che deriva dal verbo inglese to trek, che significa camminare lentamente o anche fare un lungo viaggio, mentre il secondo deriva dal verbo inglese to hike, che significa camminare."

2.5.2 Analisi delle funzionalità

Un utente può registrarsi e autenticarsi Come richiesto l'utente può registrarsi e autenticarsi fornendo pochi dati essenziali (in futuro espandibili) quali username univoco, email comune, nome e cognome. Abbiamo deciso di integrare allo stesso social anche il linking a social più famosi così da poter in futuro permettere la condivisione su di essi quali Facebook (Meta) e Google.

Un utente autenticato può inserire nuovi itinerari in piattaforma Itinerari o sentieri qualsivoglia nominarli rappresentano l'oggetto fulcro della piattaforma, la possibilità di condividere i propri tracciati (per esempio se si è esperti) oppure percorrerne qualcuno già pubblicato (per esempio se si è neofiti).

Un utente può inserire un tracciato tramite l'app registrandolo Per "Registrare" un percorso si intende la capacità di poter, mentre si fa trekking, registrarlo tramite l'app in modo tale che a fine percorso l'utente deve solo inserire eventuali (e facoltativi) attributi per poi pubblicarlo.

⁸<https://it.wikipedia.org/wiki/Escursionismo>

Effettuare ricerche di itinerari presenti in piattaforma Questo punto è un altro tassello fondamentale per l'app NaTour21 e, come vedremo nel prossimo paragrafo, essere un requisito fondamentale soprattutto per chi è definito "Utente spettatore" ovvero per coloro che hanno esigenze aperticolari e quindi offriamo la possibilità di ricercare tramite appositi filtri gli itinerari che più vanno bene per loro.

Un utente può creare compilation di sentieri Questa funzionalità è adatta a chi nello specifico vuole crearsi una serie di itinerari, magari vicini tra loro, che può percorrere e quindi (su future implementazioni) dare la possibilità di creare dei MACRO-Itinerari, ovvero insiemi di itinerari magari simili, vicini o che compongono un itinerario più grande (i.e Cammino di Santiago).

Un utente può caricare delle fotografie L'utente avrà la possibilità di poter pubblicare le foto di un itinerario che verranno aggiunte ad un album personale dellos tesso, ovviamente tali foto andranno analizzate e con la possibilità di mostrare i punti GPS sull'itinerario stesso meglio definiti come "Interesting Point".

2.5.3 Stato di sviluppo delle funzionalità

Nelle seguenti tabelle abbiamo rappresentato lo stato di sviluppo delle funzionalità ad oggi:

	Login	Sigin	Forgot Password	Google Login	Meta Login
Idea Progettuale					
Mockup e StateChart					
Prima implementazione					
Diagramma delle classi					
Testing					
Beta-Testing					
Prodotto finto	X	X	X	X	X

	Estrazione metadati (POI)	Estrazione metadati (Common)	Rekognition
Idea Progettuale			X
Mockup e StateChart			
Prima implementazione		X	
Diagramma delle classi			
Testing			
Beta-Testing			
Prodotto finto	X		

	Pubblica Itinerario	Registra Itinerario	Import da GPX	Preferiti	Chat
Idea Progettuale					
Mockup e StateChart				X	X
Prima implementazione					
Diagramma delle classi					
Testing					
Beta-Testing					
Prodotto finto	X	X	X		

	Visualizza Profilo	Modifica Profilo	Cancella Profilo	Assistenza	Aggiunta POI
Idea Progettuale					
Mockup e StateChart					
Prima implementazione					
Diagramma delle classi					
Testing					
Beta-Testing			X		
Prodotto finto	X	X		X	X

	Creazione Compilation	Modifica Compilation	Cancella Compilation	Aggiunta Itinerario in Compilation	Aggiunta foto itinerario tramite marker
Idea Progettuale					
Mockup e StateChart					
Prima implementazione		X			X
Diagramma delle classi					
Testing					
Beta-Testing					
Prodotto finto	X		X	X	

2.6 Individuazione del target di utenti

Il target di utenti che si può definire da una prima (e relativa) analisi dei casi d'uso sono ovviamente:

- Utenti che siano appassionati di Trekking.
- Utenti che utilizzino dei social network.
- Utenti che viaggiano per il mondo.

Questa prima analisi è molto buona ma non completa.

Secondo i dati ISTAT in Italia, abbiamo notato che anche nel periodo di pandemia, la crescita del Agriturismo (spesso centri comuni per amanti del escursionismo) non si è arrestata, negli ultimi anni, infatti (riferimento a dati dal 2010) l'escursionismo (soprattutto alpino) ha avuto una crescita di circa il 33% crescita da non sottovalutare dato che in alcune regioni (i.e Toscana, Abruzzo, Piemonte) si arriva a una quasi saturazione del mercato agritouristico⁹.

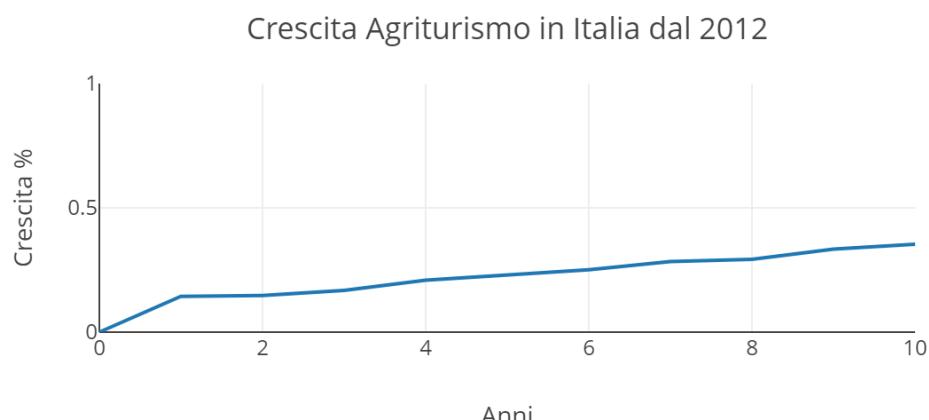


Figura 22: Grafico di crescita agritourismo

⁹Ufficiali Trekking Italia - Soci adulti: 7.129 partecipanti all'attività, trek classici: 22.790, giorni sui sentieri in Italia: 2.155, regioni attraversate: 20, giorni sui sentieri nel Mondo: 876, nazioni attraversate: 35, partecipanti a tutte le attività: 30.458, giorni sui sentieri: 3.263.

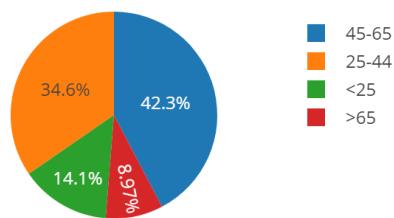
2.6.1 Personas

Il focus principale per noi è stato quello di analizzare 3 aspetti fondamentali per individuare le user personas:

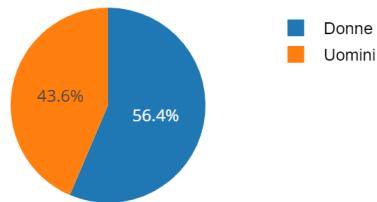
- Età - sommariamente si parla di un'età compresa tra 25-65 anni.
- Sesso - sommariamente le persone che praticano il trekking sono per lo più donne.
- Impiego - secondo una nostra ipotesi è possibile che un target di utenti siano quelli che viaggiano o che consigliano viaggi, per esempio proprietari di agenzia di viaggi.

Possiamo quindi dire che un nostro focus futuro potrebbe essere quello di fidelizzare i nostri clienti magari dando anche consigli di viaggi o di agenzie di viaggio mirate proprio all'escursionismo.

Età media



Sesso



Personas 1 La prima user personas abbiamo raccolto il target d'utenti giovane, amanti della tecnologia, coloro che sono studenti o giovani lavoratori in cerca di avventure:



Figura 23: User Personas 1

Personas 2 La seconda user personas individuata a parer nostro riguarda anche quella che potrebbe a livello di marketing essere oggetto di approfondito studio. In questa classe di utenti prendiamo coloro che sono appassionati della materia ma che magari l'hanno trasformata nel proprio business, include coloro che hanno un età mediana a quella individuata nel paragrafo precedente.



Figura 24: User Personas 2

Personas 3 L'ultima classe comprende quella di età avanzata nonché estremo destro del range individuato. Questa classe prevede persone che non sono confidente con le tecnologie odierne e che potrebbero avere problemi ma che invece hanno grande passione.

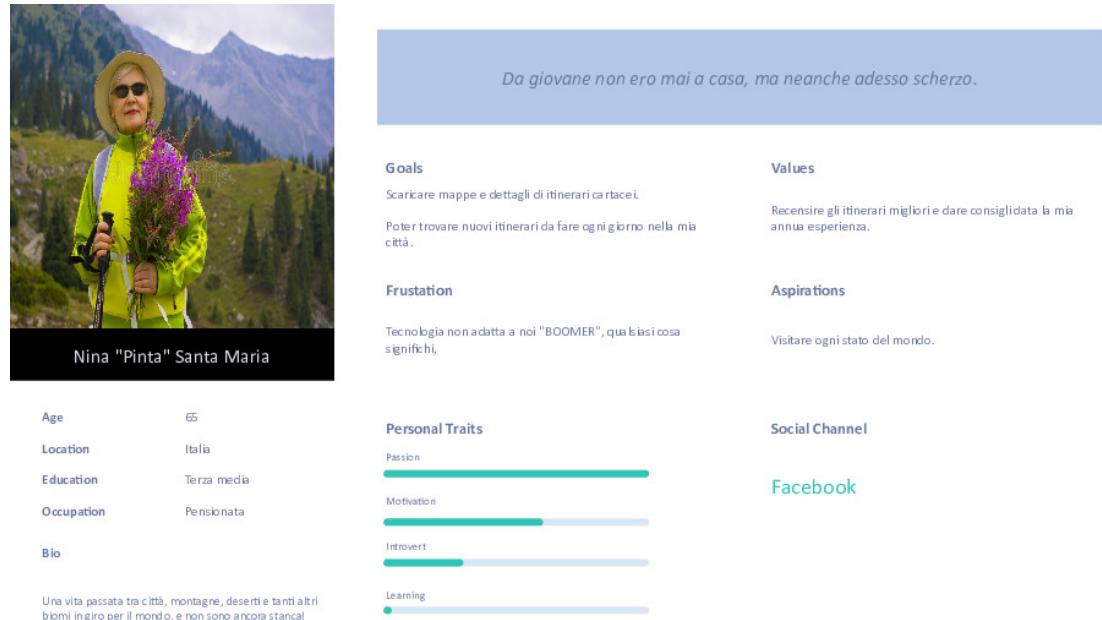


Figura 25: User Personas 3

2.7 Valutazione dell’usabilità a priori

Per quanto riguarda la valutazione dell’usabilità a priori sono state adoperate varie tecniche e strumenti che avevamo a disposizione. In particolare, per ottenere una valutazione dell’usabilità a priori più corretta possibile, abbiamo cercato di riprodurre dei prototipi avendo le idee ben chiare sull’aspetto e delle funzionalità che doveva avere la nostra app, quindi abbiamo deciso di rimanere più fedeli possibile ai prototipi così da avere una valutazione a priori quanto più vicina a quella a posteriori in beta testing.

Per realizzare i mockup è stato utilizzato Axure, quest’ultimo ci permette di avere dei mockup dinamici simulando una sorta di computazione e di transazione tra le activity/fragment.

I mockup, e soprattutto l’app, sono stati realizzati seguendo le 8 regole d’oro di **Shneiderman**¹⁰ con particolare attenzione a 3 regole, ovvero: **usabilità universale, riscontri informativi e riduzione del carico di memoria a breve termine.**

Per quanto riguarda l’usabilità universale è stato deciso di aggiungere un tasto “Help” nella activity di pubblicazione di un itinerario che spiega passo passo come effettuare la pubblicazione di un nuovo itinerario in piattaforma. Abbiamo scelto di metterla in questa activity perché l’abbiamo ritenuta molto importante ma soprattutto delicata in quanto devono essere aggiunte molte informazioni riguardo l’itinerario che si sta pubblicando ed alcune di esse vengono anche calcate automaticamente dall’app e non inserite manualmente dall’utente. Un altro tipo di aiuto che è stato dato all’utente è quello dell’aggiunta della funzionalità di assistenza che può essere consultata per qualsiasi motivo inviando una mail dall’app, e in qualsiasi momento avendo accesso a questa funzionalità direttamente dalla toolbar.

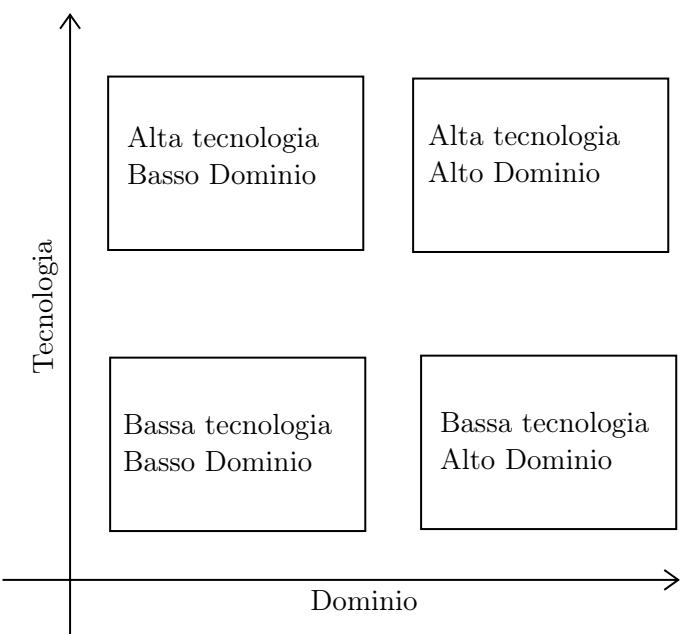
Per i riscontri informativi sono stati aggiunti diversi popup che permettono una sorta di dialogo costante con l’utente così da non farlo sentire abbandonato durante lo svolgimento delle funzionalità. In particolare ad ogni azione dell’utente che richiede un certo tempo di attesa (es. Visualizzazione di un itinerario) viene mostrato un popup di caricamento. Un altro tipo di feedback che viene dato, è quando l’utente effettua azioni delicate (es. eliminazione di una compilation, uscita dal proprio account) mostrando un altro popup che richiede la conferma o l’annullamento dell’azione richiesta.

La riduzione del carico di memoria a breve termine è stato garantito non richiedendo informazioni troppo complesse all’utente ma semplici dati da inserire, inoltre abbiamo deciso di calcolare la distanza dell’itinerario e la sua

¹⁰<https://principles.design/examples/shneidermans-eight-golden-rules-of-interface-design>

durata invece di farli inserire all’utente, questo sia per avere delle informazioni sicuramente più consistenti ma anche per semplificare il lavoro dell’utente e non richiedergli troppe informazioni. Un altro modo per garantire meno carico di memoria a breve termine è quello della ridondanza, ovvero la stessa funzionalità viene proposta più volte in differenti posizioni.

Le tecniche utilizzate per la valutazione dell’usabilità a priori sono varie. Inizialmente, dopo la realizzazione dei prototipi, abbiamo deciso di effettuare un test di usabilità. La pianificazione del test di usabilità inizia con una attenta scelta dei valutatori che devono utilizzare i prototipi e poi valutarli. Come nel grafico qui rappresentato, abbiamo 4 tipi di valutatori.



Abbiamo scelto 5 persone, in quanto con 5 persone abbiamo oltre l’85% di problemi di usabilità trovati (come ci ricorda la regola di Nielsen¹¹) ovvero:

- **Valutatori** (2) con bassa conoscenza tecnologica e bassa conoscenza del dominio (Alessandro G., Leonardo P.).
- **Valutatore** (1) con bassa conoscenza tecnologica e alta conoscenza del dominio (Ginevra T.).
- **Valutatore** (1) con alta conoscenza della tecnologia e bassa conoscenza del dominio (Giovanni V.).
- **Valutatore** (1) con alta conoscenza della tecnologia e alta conoscenza del dominio (Ciro P.).

¹¹<https://www.neurowebdesign.it/it/alla-scoperta-delle-10-euristiche-di-nielsen/>

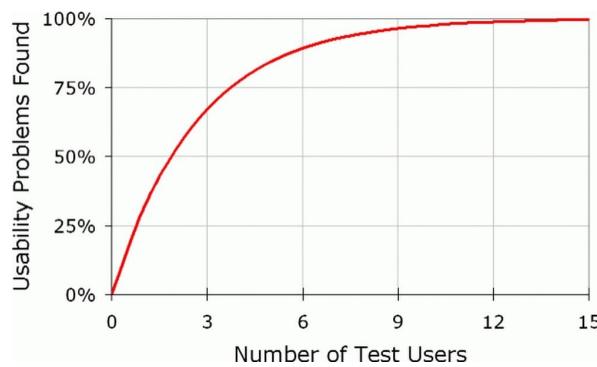
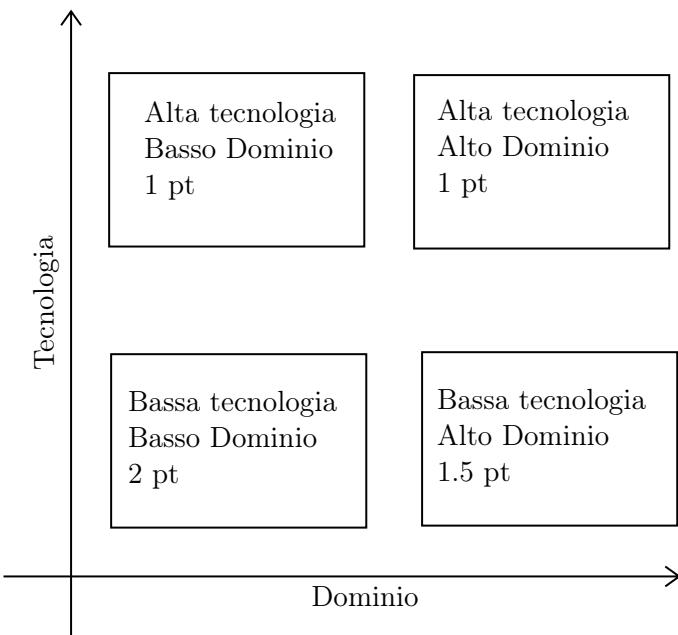


Figura 26: Regola di Nielsen

2.7.1 Tabelle di valutazione

Per una maggiore affidabilità del test di usabilità abbiamo deciso di dare dei pesi diversi ai diversi tipi di utenti per ogni funzionalità assegnata:



Inoltre abbiamo assegnato anche un peso al tempo che impiegano i nostri valutatori ad effettuare un dato compito:

0-2 minuti	2-4 minuti	più di 4 minuti
2 pt	1 pt	0.5 pt

Infine abbiamo deciso di dare anche un peso alla difficoltà del compito proposto agli utenti:

COMPITO 1	COMPITO 2	COMPITO 3	COMPITO 4
1 pt	2 pt	2 pt	1.5 pt

Avevamo intenzione di dare anche un peso all'età degli utenti ma i nostri utilizzatori fanno parte tutti della stessa fascia di età che va dai 20 ai 25 anni e quindi sarebbe risultato inutile dare a tutti lo stesso peso aggiuntivo.

2.7.2 Tecnica utilizzata

Per effettuare la valutazione dell'usabilità abbiamo cercato di simulare più fedelmente possibile la tecnica del Mago di Oz.

Ai nostri utilizzatori sono stati proposti i seguenti compiti:

- **Compito 1:** Registrazione alla piattaforma.
- **Compito 2:** pubblicazione di un itinerario in piattaforma.
- **Compito 3:** Creazione di una collezione.
- **Compito 4:** Visualizzazione di un itinerario qualsiasi.

I risultati del test di usabilità è visibile nella seguente tabella:

	Compito 1	Compito 2	Compito 3	Compito 4
Alessandro G.	P	S	P	S
Leonardo P.	S	P	F	S
Ginevra P.	S	S	F	S
Giovanni V.	S	S	S	S
Ciro P.	S	S	S	S

- **P:** successo parziale (0.5).
- **S:** successo (1).
- **F:** failure (0).

Ricordando i pesi dati moltiplichiamo i valori ottenuti per i loro pesi e dividiamo questo valore per la somma dei pesi, avremo quindi: $47/50 = 94\%$;

Possiamo ritenerci abbastanza soddisfatti del risultato e oltre ad aver dato una misura all'usabilità abbiamo anche raccolto delle porposte di modifica dei prototipi, in particolare:

1. Il tasto “Torna alla home” presente nel profilo andava tolta perché non era abbastanza utile vista la presenza del back button.
2. Nel profilo eliminare il pulsante di logout in quanto molto vicino al pulsante di eliminazione dell’account, ciò potrebbe portare alla cancellazione involontaria dell’account.
3. Nella Home Page i pulsanti in basso erano troppi e venivano cliccati involontariamente.

Abbiamo accolto queste proposte ed infatti questi cambiamenti sono stati adoperanti anche all’interno dell’applicazione, ad esempio il pulsante di logout è presente soltanto nel menu laterale e non più nel profilo.

2.8 Prototipazione funzionale via statechart dell’interfaccia grafica

Abbiamo rappresentato mediante statechart il processo di pubblicazione di un nuovo itinerario in piattaforma e del caricamento di una foto nel sistema (foto che un utente associa ad un itinerario).

Inserisci Itinerario è mostrato quanto segue:

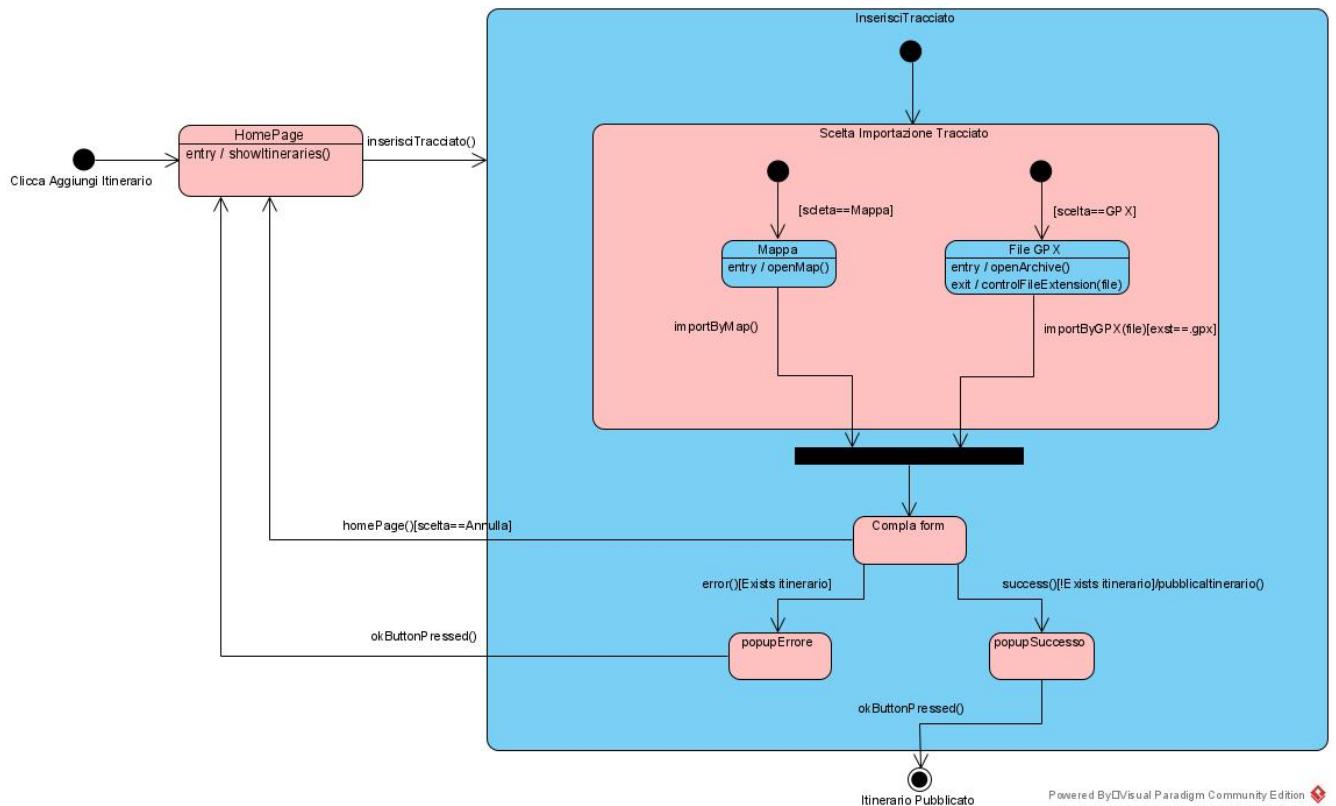


Figura 27: Statechart: Inserisci Itinerario

Per quanto riguarda il caricamento di una foto nel sistema abbiamo simulato il processo di stato dell’interfaccia grafica e del processo parallelo di verifica della foto.

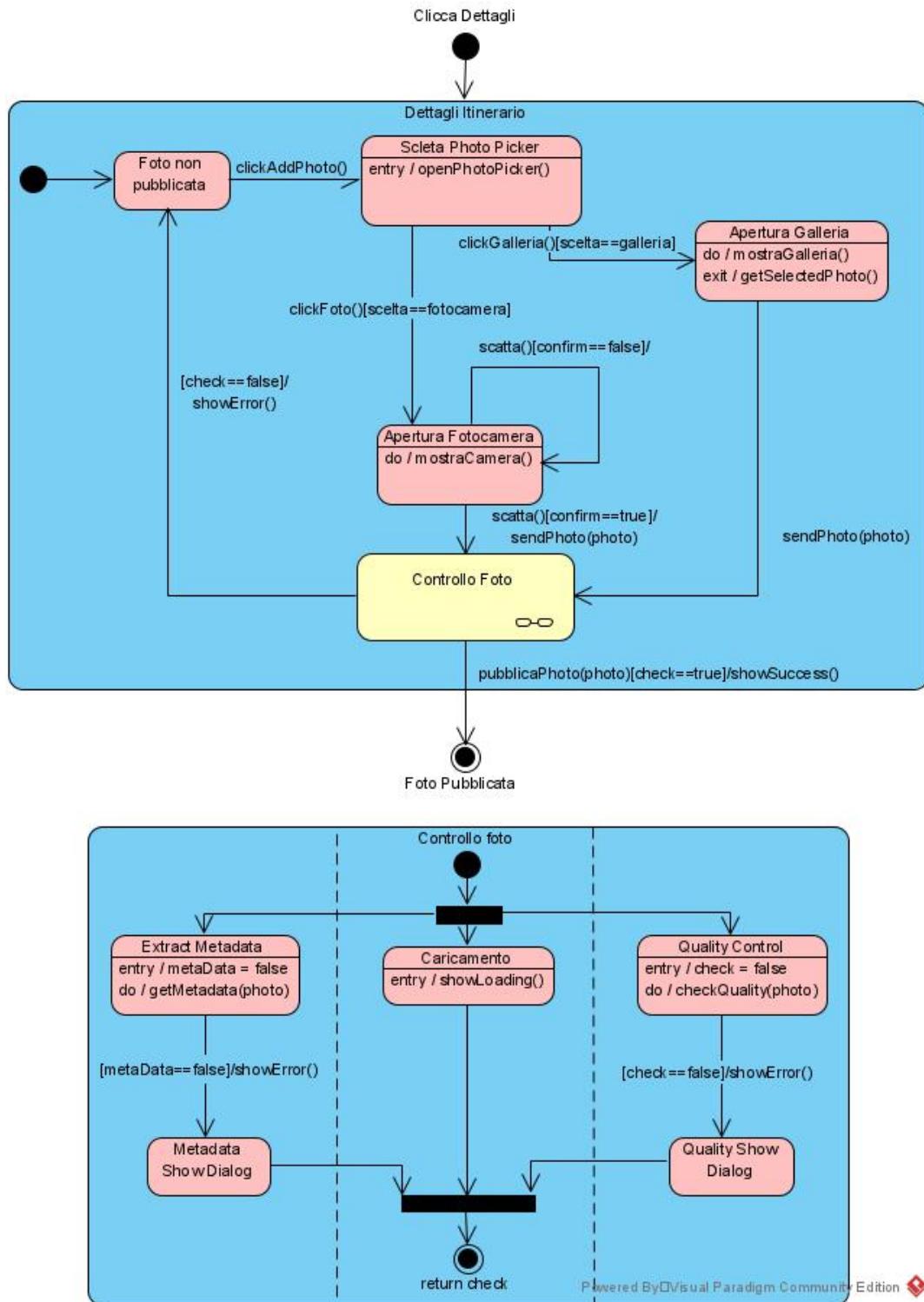


Figura 28: Statechart: Carica Foto

2.9 Glossario

Qui si possono trovare le definizioni agli acronimi e ai termini poco conosciuti nel linguaggio comune.

2.9.1 Termini

Elasticità - si riferisce alla capacità di un servizio cloud di offrire servizi su richiesta aumentando o diminuendo le risorse quando la domanda sale o scende.

Scalabilità - capacità di un sistema di aumentare o diminuire di scala in funzione delle necessità.

Usabilità - l'usabilità è il "grado in cui un prodotto può essere usato da particolari utenti per raggiungere certi obiettivi con efficacia, efficienza e soddisfazione in uno specifico contesto d'uso.

Design Pattern - è una soluzione progettuale generale ad un problema ricorrente.

Mock-up - progetti astratti che realizzano una o più funzionalità del prodotto e possono esserci diversi gradi di approfondimento dello sviluppo.

Class Diagram - diagrammi che possono comparire in un modello UML che consentono di descrivere tipi di entità, con le loro caratteristiche e le eventuali relazioni.

Sequence Diagram - diagramma previsto dall'UML utilizzato per descrivere una determinata sequenza di azioni.

StateChart - descrive il comportamento di sistemi reattivi. Le transizioni descrivono come l'entità modellata reagisce a eventi (generati dall'esterno o da se stesso).

Activity Diagram - diagramma che permette di descrivere un processo attraverso dei grafi in cui i nodi rappresentano le attività e gli archi l'ordine con cui vengono eseguite.

Personas - sono veri e propri identikit di clienti ideali, una sorta di profilo fittizio dell'utente, che rappresentano i bisogni, i comportamenti, gli interessi e le aspirazioni degli utenti reali. Sono una rappresentazione dei tratti caratterizzanti di ciascun utente e di quelli che li accomunano.

White Box - Modalità di testing in cui un metodo viene testato in base al codice che contiene, alla struttura dell'applicazion.

Black Box - Modalità di testing in cui un'unità viene testata in base ai requisiti di quest'ultima.

Framework - architettura logica di supporto sul quale un software può essere progettato e realizzato, spesso facilitandone lo sviluppo da parte del programmatore.

Back-end - parte di sviluppo che non opera sul client e che fornisce un servizio.

Android - sistema operativo mobile.

Server - macchina che ha il compito di offrire servizi ai client che li richiedono.

Tabelle di CockBurn - Tabelle di rappresentazione di un caso d'uso create da Alistair Cockburn.

Use Case - rappresenta una funzione o servizio offerto dal sistema a uno o più attori.

2.9.2 Acronimi

EC2 - Elastic Compute Cloud è un servizio offerto da AWS che consente l'utilizzo di macchine virtuali in cloud.

RDS - Relational Database Service, è un servizio di hosting di server database di Amazon Web Service.

S3 - Simple Storage Service, è un servizio di storage di Amazon Web Service.

CRUD - Create Read U Delete, operazioni standard di un database.

REST API - Application Programming Interface, è un'interfaccia di programmazione delle applicazioni conforme ai vincoli dello stile architettonico REST, che consente l'interazione con servizi web RESTful. Il termine REST, coniato dall'informatico Roy Fielding, è l'acronimo di REpresentational State Transfer.

HTTP - HyperText Transfer Protocol, è un protocollo a livello applicativo usato come principale sistema per la trasmissione d'informazioni sul web ovvero in un'architettura tipica client-server.

ISO/IEC - International Organization for Standardization/International Electrotechnical Commission, L'ISO coopera strettamente con la Commissione eletrotecnica internazionale (IEC), responsabile per la standardizzazione dei dispositivi elettrici ed elettronici, e con l'Unione internazionale delle telecomunicazioni (ITU) per quanto riguarda le norme tecniche nell'ambito delle telecomunicazioni.

GDPR - General Data Protection Regulation, è un regolamento europeo che disciplina il modo in cui le aziende e le altre organizzazioni trattano i dati personali.

MVC - Model View Controller, è uno dei più importanti design pattern dell'ingegneria del software orientato soprattutto alla programmazione ad oggetti.

MVP - Model View Presester, è un design pattern molto simile ad MVC usato per la programmazione diretta all'utente (tramite GUI).

GUI - Graphic User Interface, interfaccia grafica.

DAO - Data Access Object, è un design pattern che viene utilizzato per separare l'API o le operazioni di accesso ai dati di basso livello dai servizi aziendali di alto livello. Di seguito sono riportati i partecipanti a Data Access Object Pattern.

DTO - Data Transfer Object, è un design pattern usato per trasferire dati tra sottosistemi di un'applicazione software. I DTO sono spesso usati in congiunzione con gli oggetti di accesso ai dati (DAO) per recuperare i suddetti da una base di dati.

SECT - Modalità di sviluppo di test in cui le classi di equivalenza di parametri diversi vengono testate effettuando tutte le combinazioni.

WECT - Modalità di sviluppo di test in cui ci sono il minimo numero di test case che ricoprono tutte le classi di equivalenza valide e uno use case per ogni classe di equivalenza non valida.

3 Modelli di Dominio

In questa sezione si analizzeranno quelli che sono i modelli di dominio dell'applicazione ancora in fase di analisi dei requisiti.

In particolar modo analizzeremo 3 fondamentali diagrammi utili sia in questa fase di analisi che come vedremo più avanti anche in fase di design.

I diagrammi sono i seguenti:

- Class Diagram - diagramma di classi.
- Sequence diagram - analisi di sequenza dell'interazione tra oggetti di un singolo use case.
- Diagramma di attività - evoluzione di un flowchart.

Per astrarre l'implementazione e per far sì che ci sia una facile comprensione, per individuare le classi e le entità in gioco abbiamo usato l'euristica **Three-Object-Type** o anche conosciuta con 3 acronimi:

- **Entity** - entità o modelli che sono in gioco.
- **Boundary** - detti anche "Confini" rappresentano le interazioni tra utente e sistema (i.e GUI).
- **Control** - rappresentano i controller o meglio definita la logica del sistema che si occupa di definire gli Use case sotto sviluppo.

3.1 Classi, oggetti e relazioni di analisi

3.1.1 Classi ed entità

Le entità che abbiamo individuato sono le seguenti e saranno esse stesse gli oggetti in gioco per i vari use case assegnati:

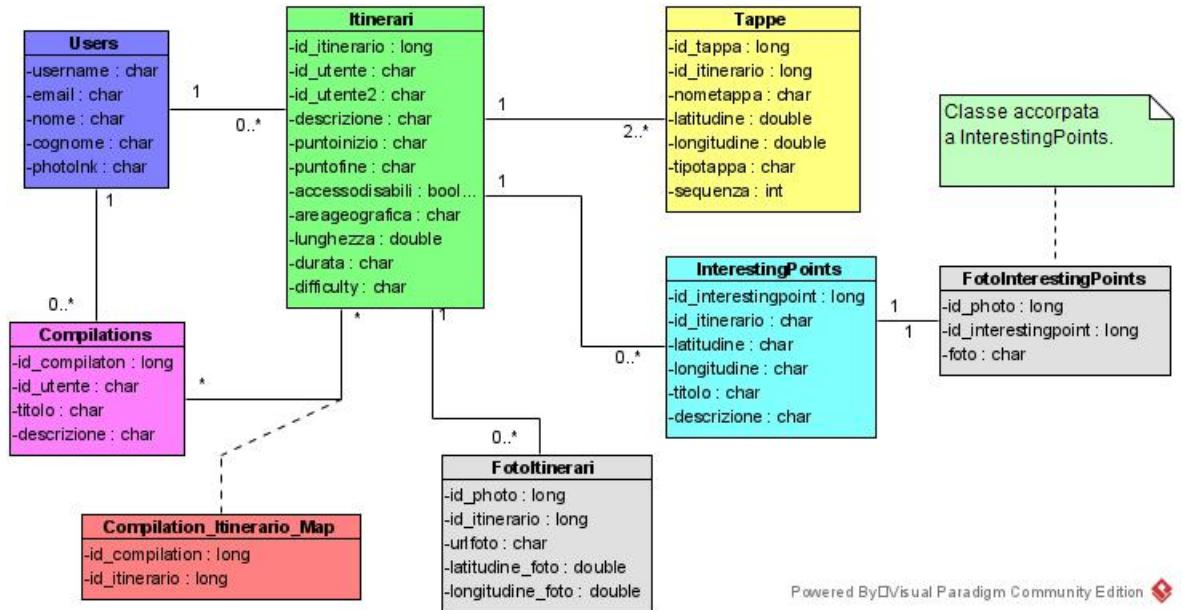


Figura 29: Class Diagram Generale

3.1.2 Class diagram di analisi

In questa sezione elecheremo tutti i classdiagram suddivisi (questo dato chè tutto l'insieme delle funzionalità può essere molto grande e difficile da capire.

3.1.3 Class diagram - Collezione

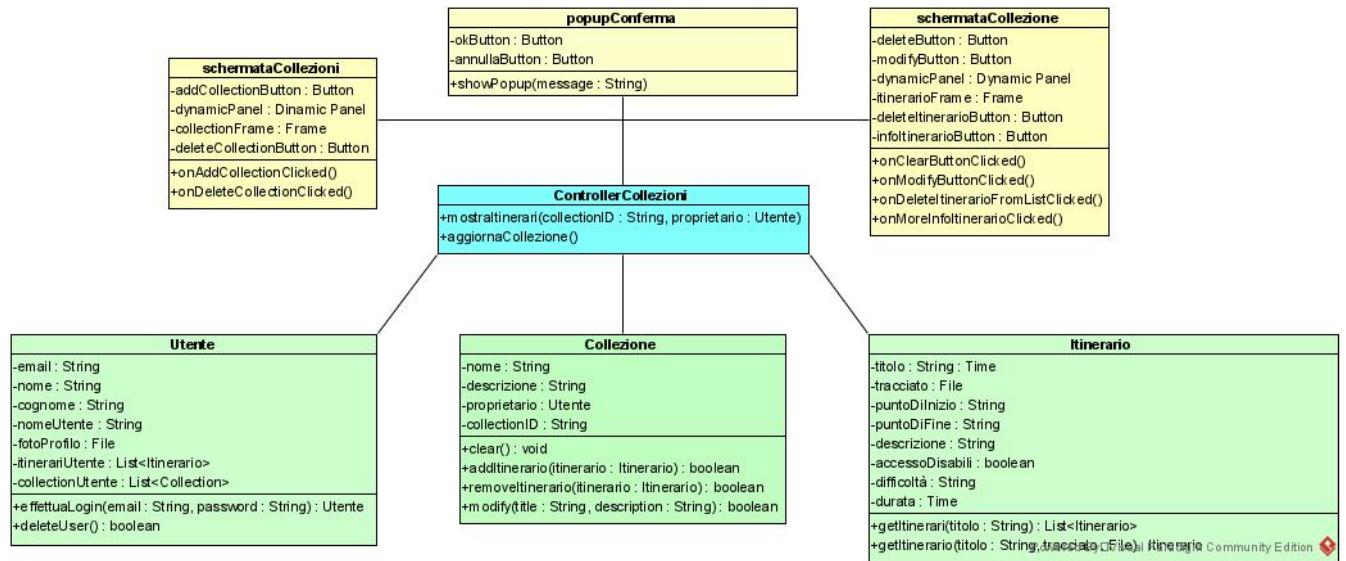


Figura 30: Collezione

3.1.4 Class diagram - Login

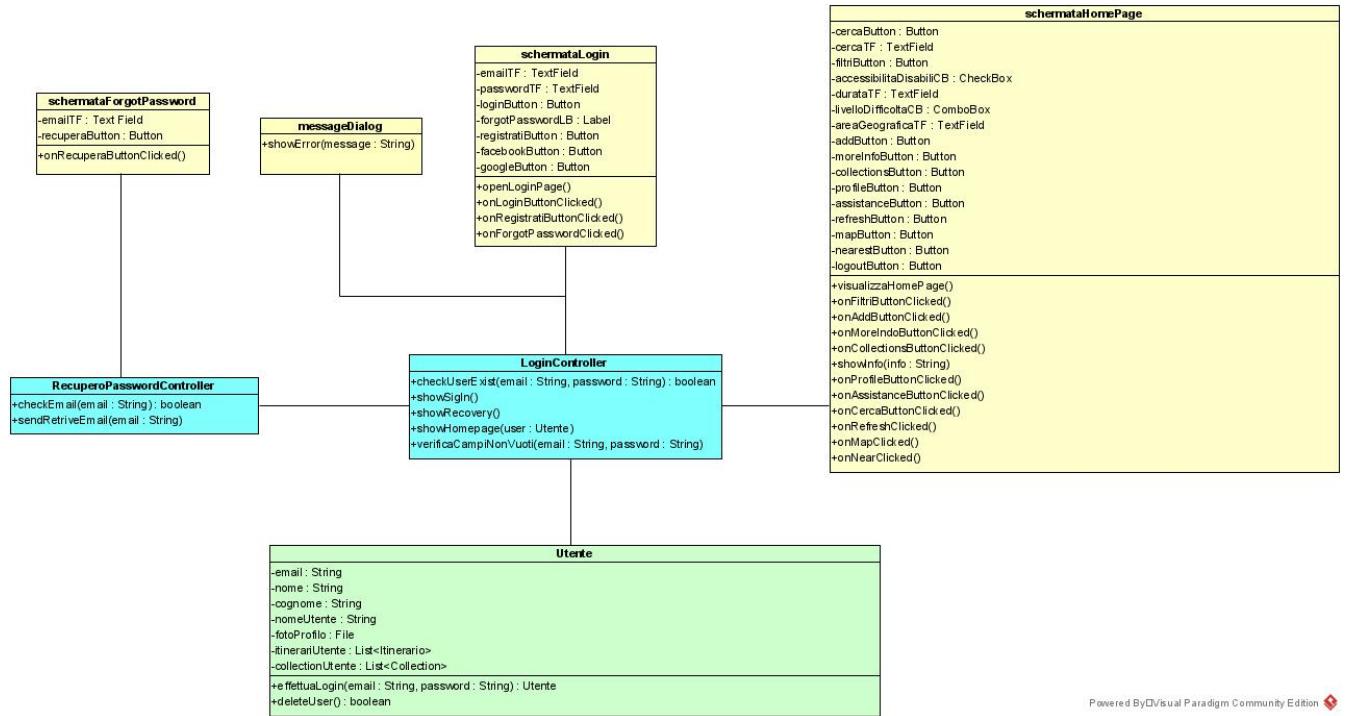


Figura 31: Login

Powered By Visual Paradigm Community Edition

3.1.5 Class diagram - Registrazione

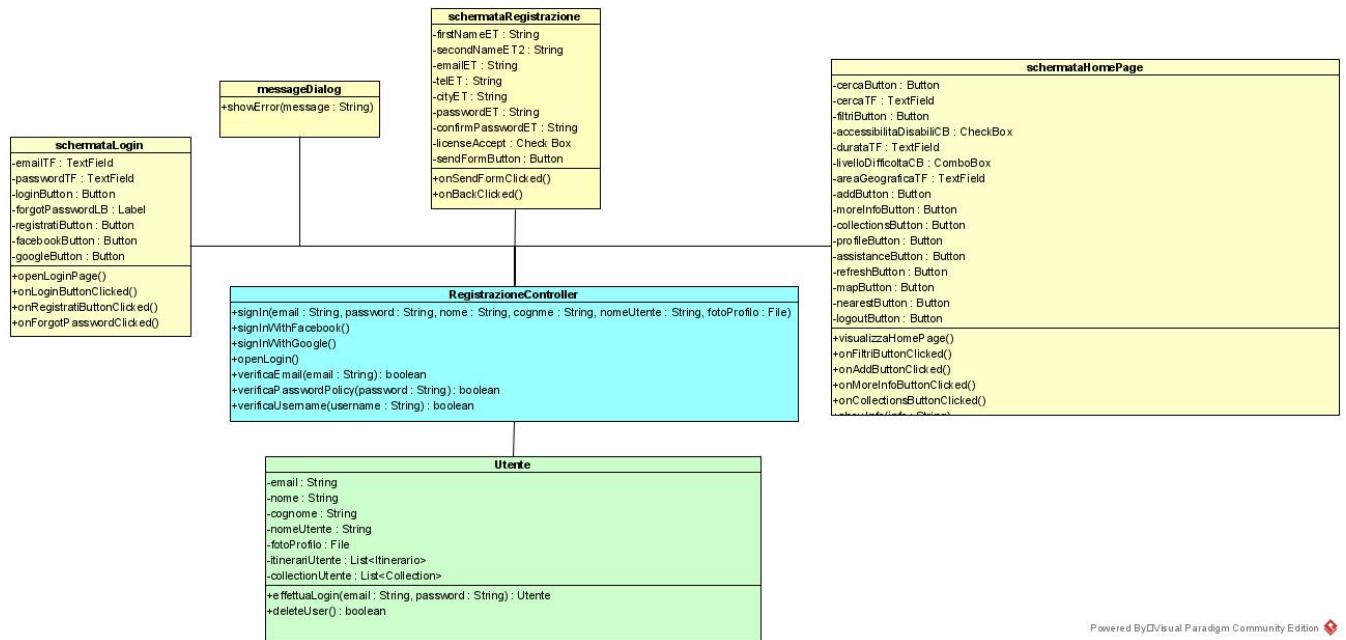


Figura 32: Registrazione

Powered By Visual Paradigm Community Edition

3.1.6 Class diagram - PhotoAdd

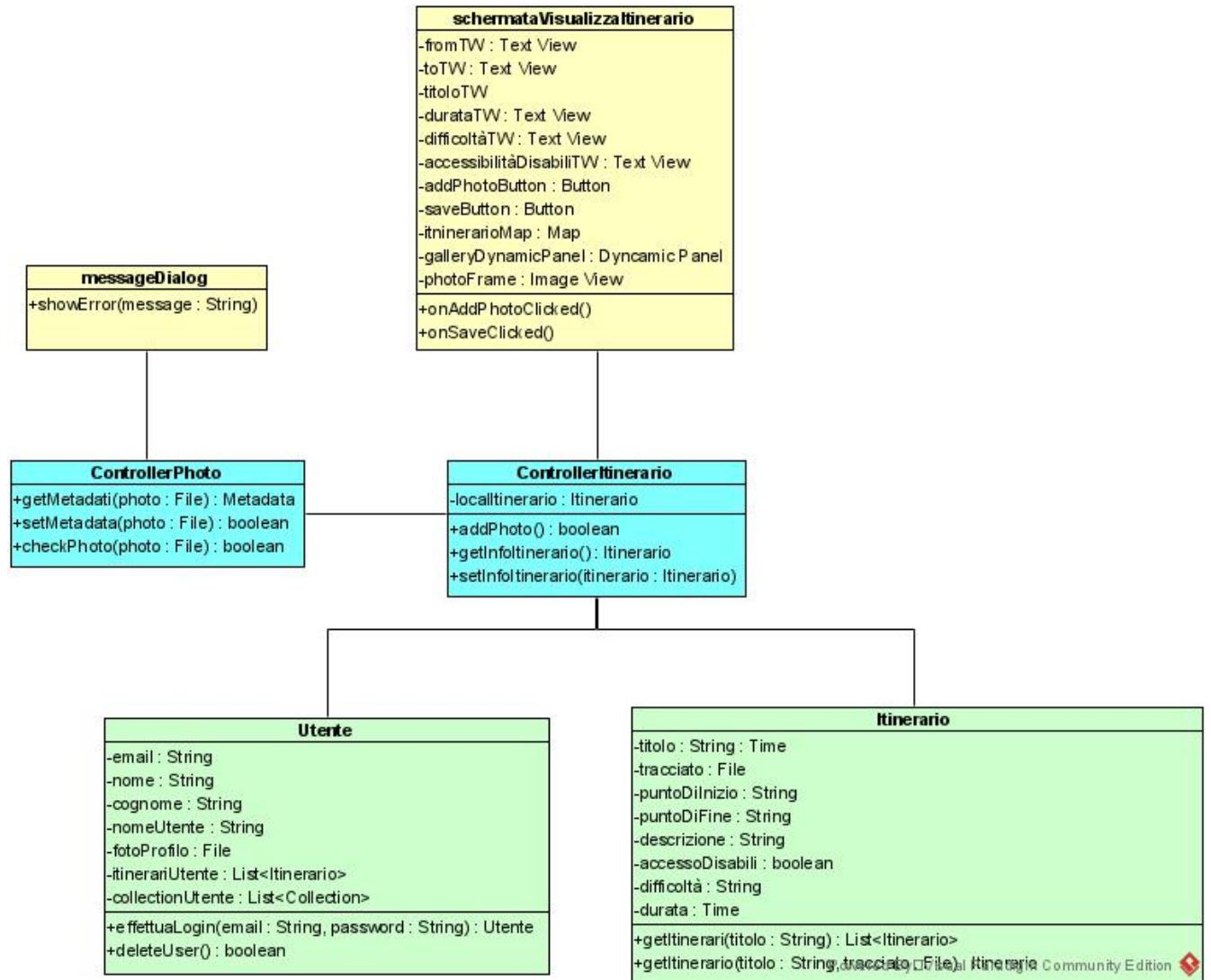


Figura 33: PhotoAdd

3.1.7 Class diagram - Profilo

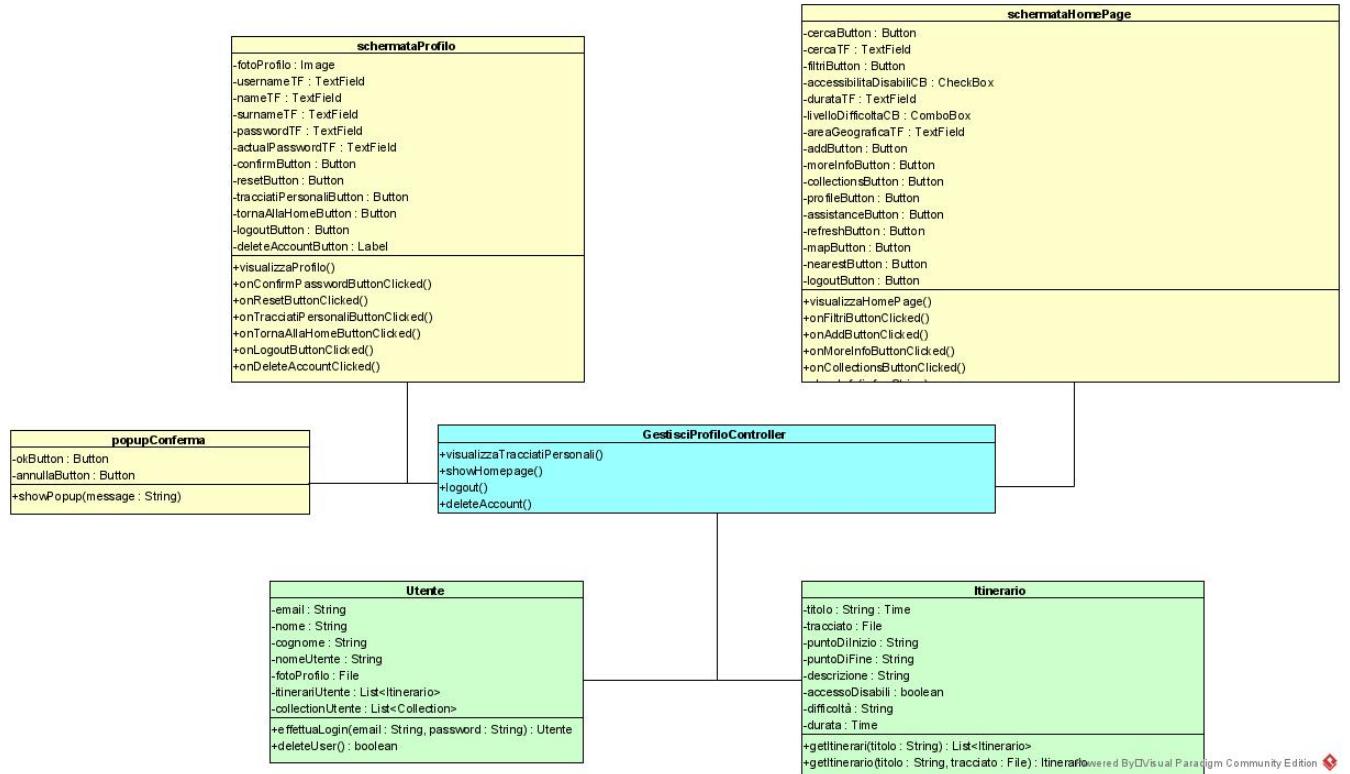


Figura 34: Profilo

3.1.8 Class diagram - Pubblica Nuovo Itinerario

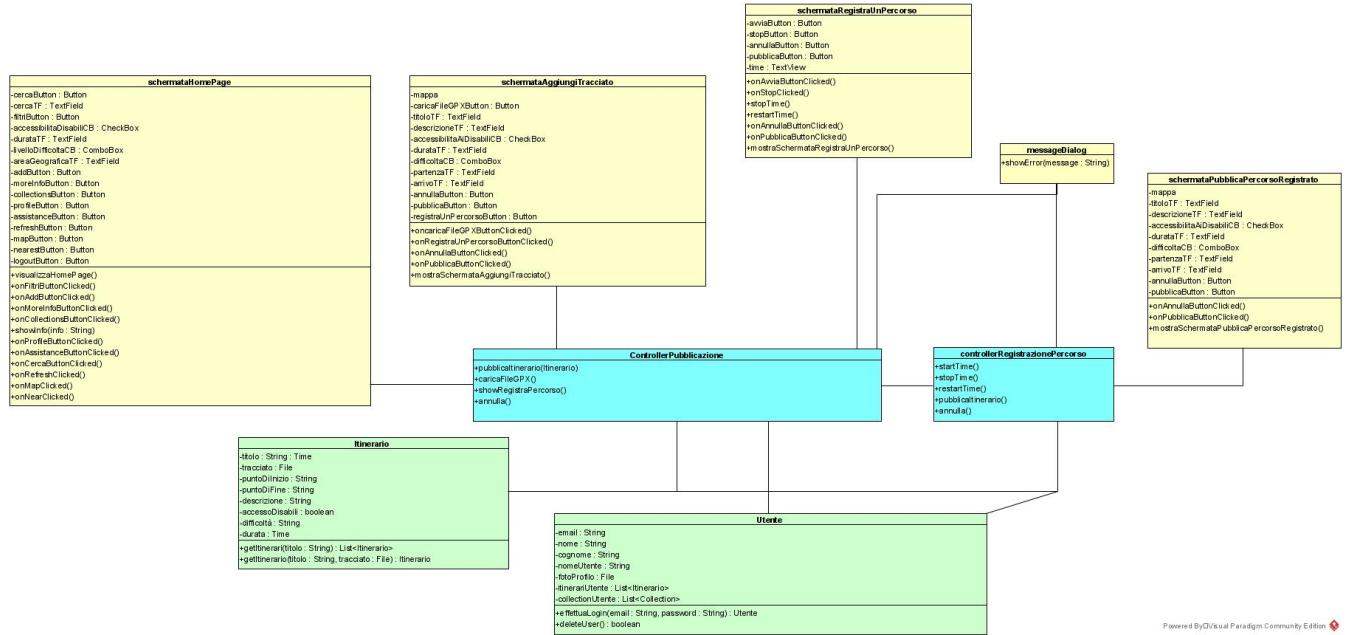


Figura 35: Pubblica nuovo itinerario

Powered By Visual Paradigm Community Edition

3.1.9 Class diagram - Support

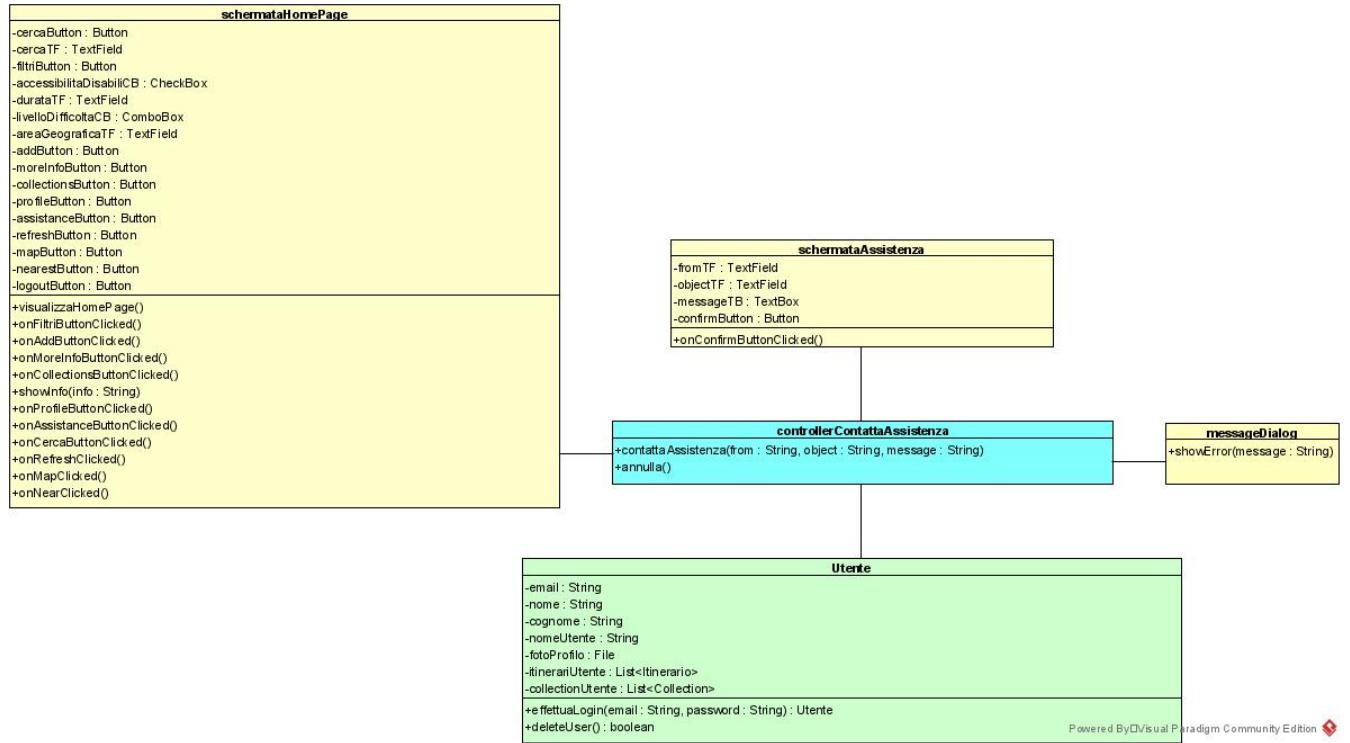


Figura 36: Support

3.1.10 Class diagram - Visualizza e ricerca itinerario

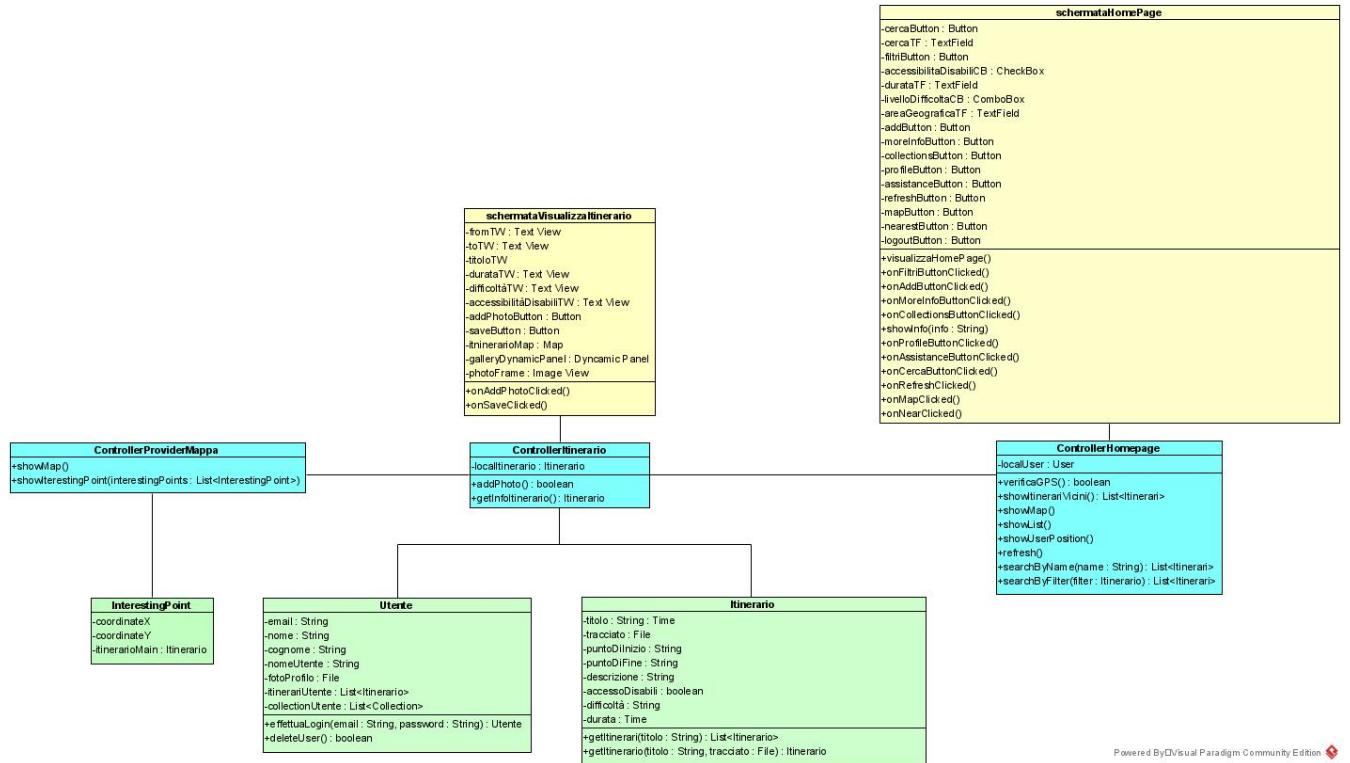


Figura 37: Visualizza e ricerca itinerario

3.2 Sequence Diagram

I sequence diagram, citando la definizione di visual paradigm descrivono:

"High-level interactions between user of the system and the system, between the system and other systems, or between subsystems (sometimes known as system sequence diagrams)."

3.2.1 Sequence Diagram - Inserisci Itinerario

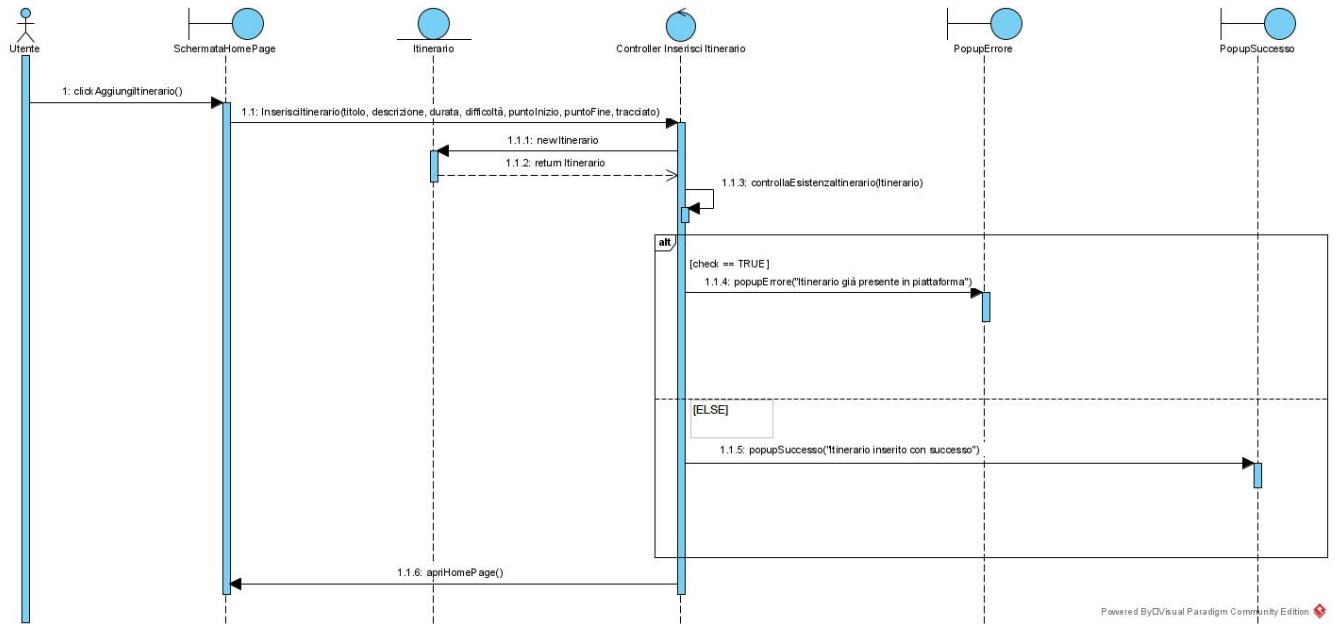


Figura 38: Sequence Diagram - Inserisci Itinerario

3.2.2 Sequence Diagram - Carica Foto

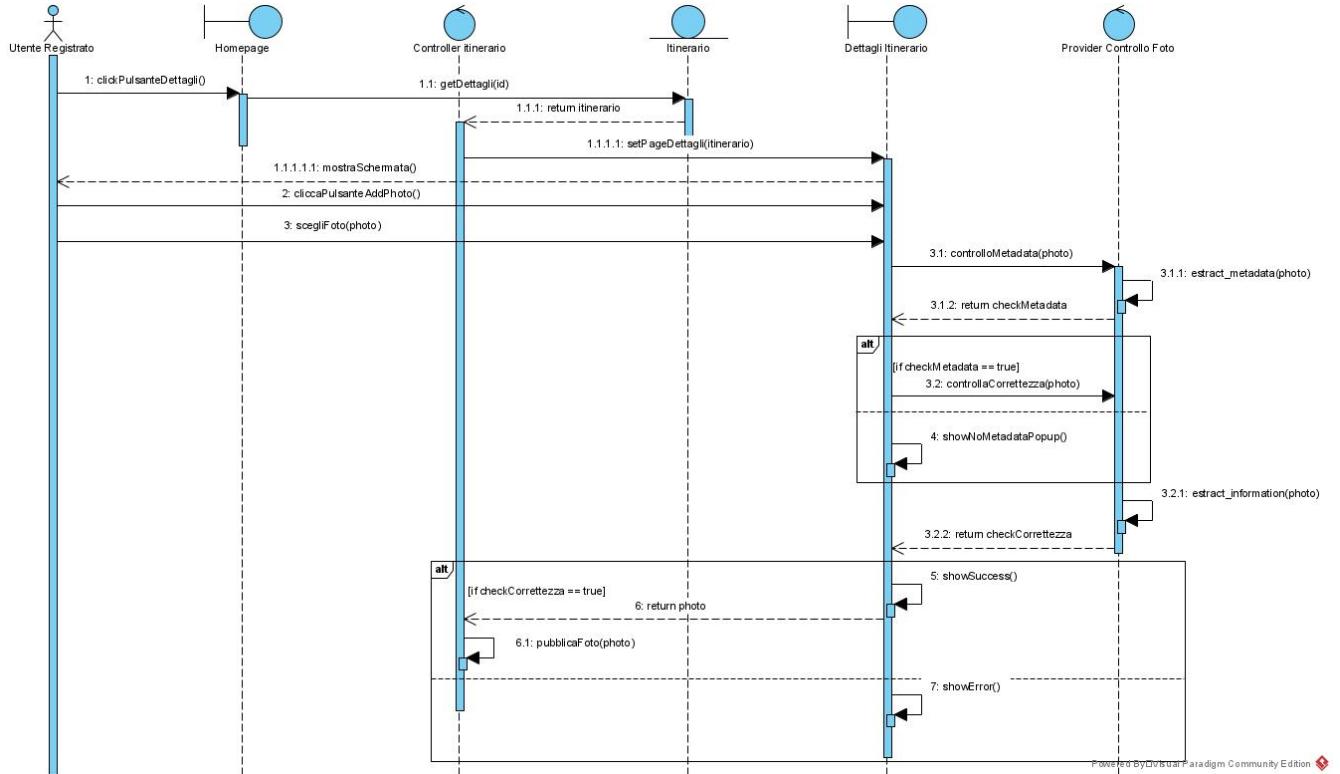


Figura 39: Sequence Diagram - Carica Foto

3.3 Activity Diagram

In questa sezione e ultima dei modelli di dominio (appartenente alla categoria dei requisiti di software) mostriremo gli activity diagram creati per le funzionalità

3.3.1 Activity Diagram - Compilation

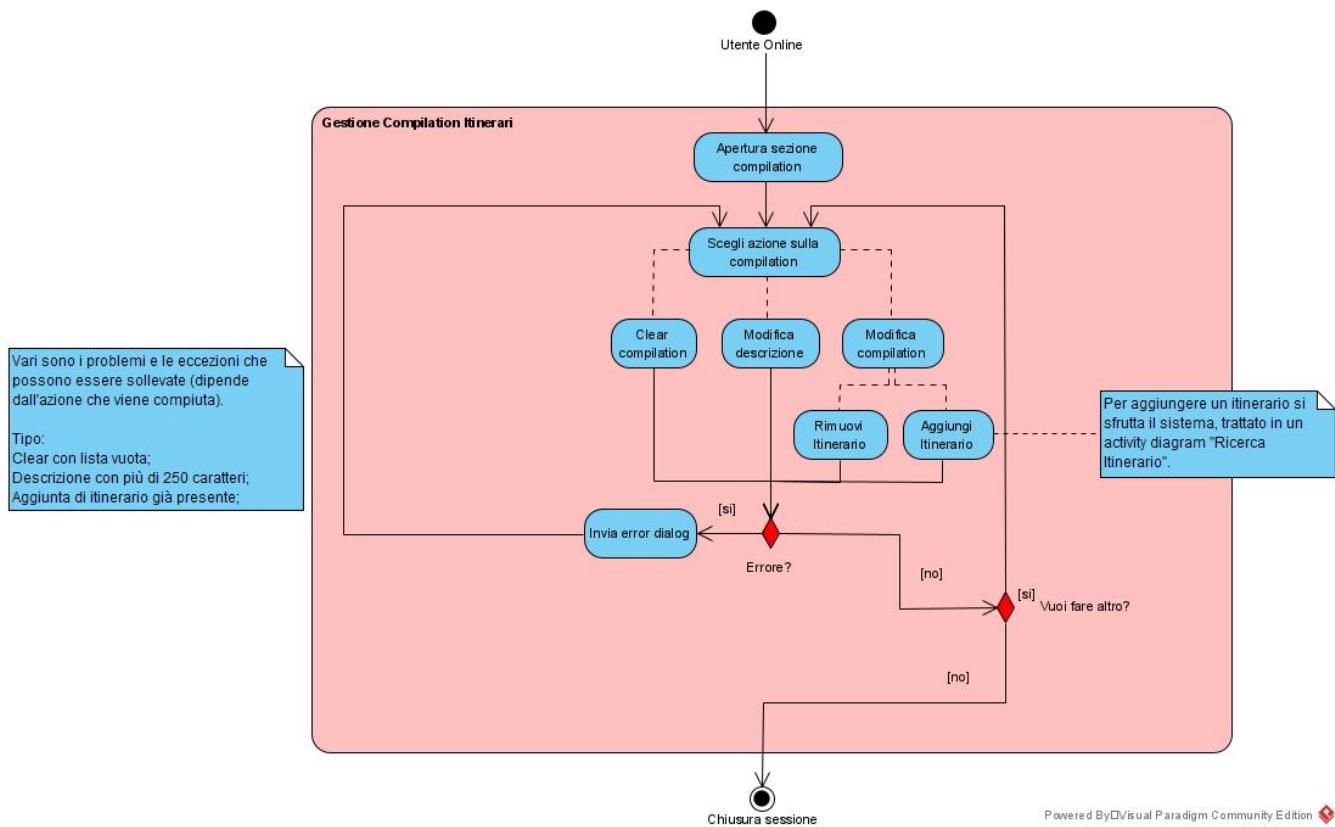


Figura 40: Activity Diagram - Compilation

3.3.2 Activity Diagram - Inserimento nuovo itinerario (Manuale)

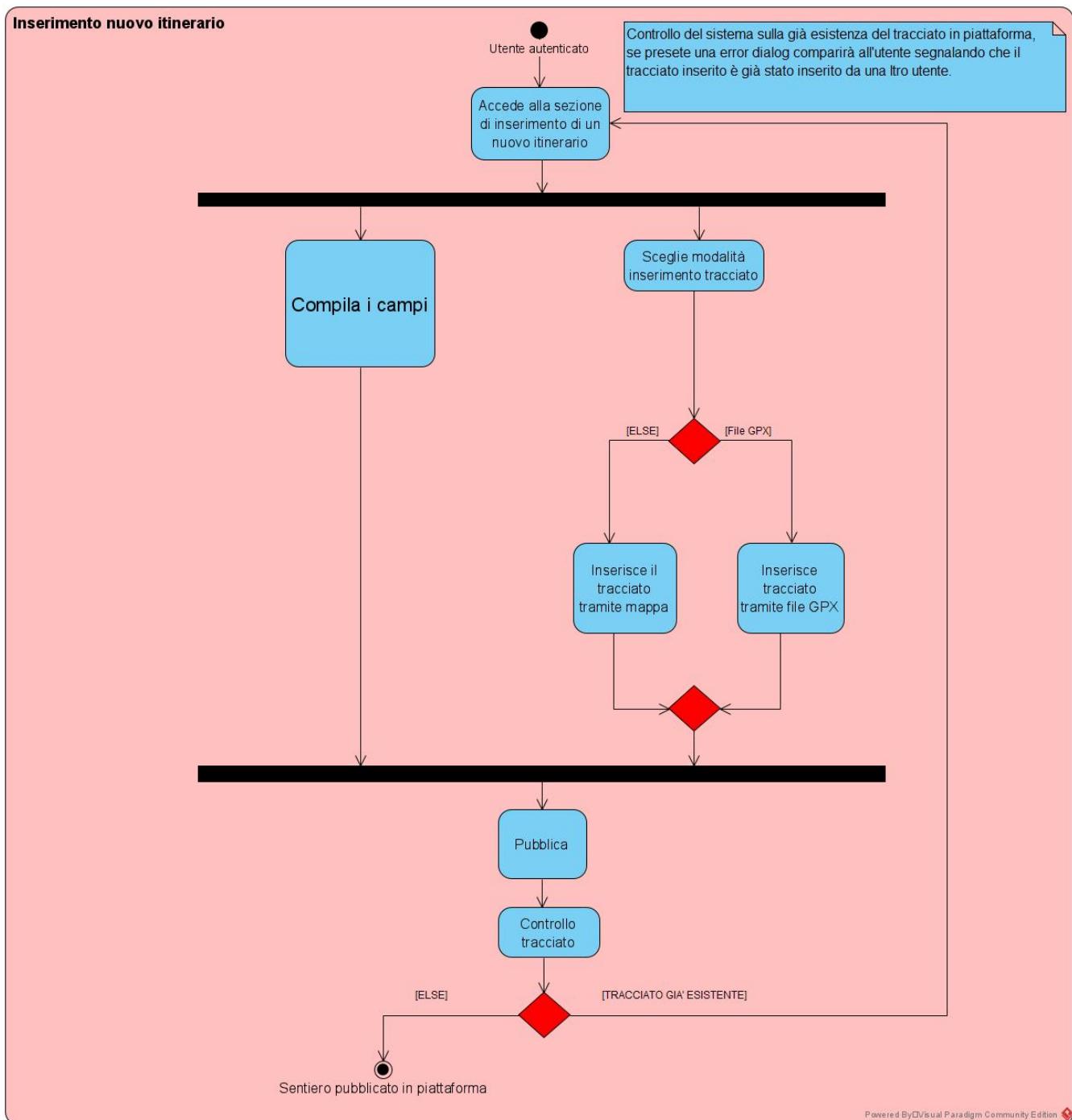


Figura 41: Activity Diagram - Insert itinerary (Manuale)

3.3.3 Activity Diagram - Inserimento nuovo itinerario (Registrato)

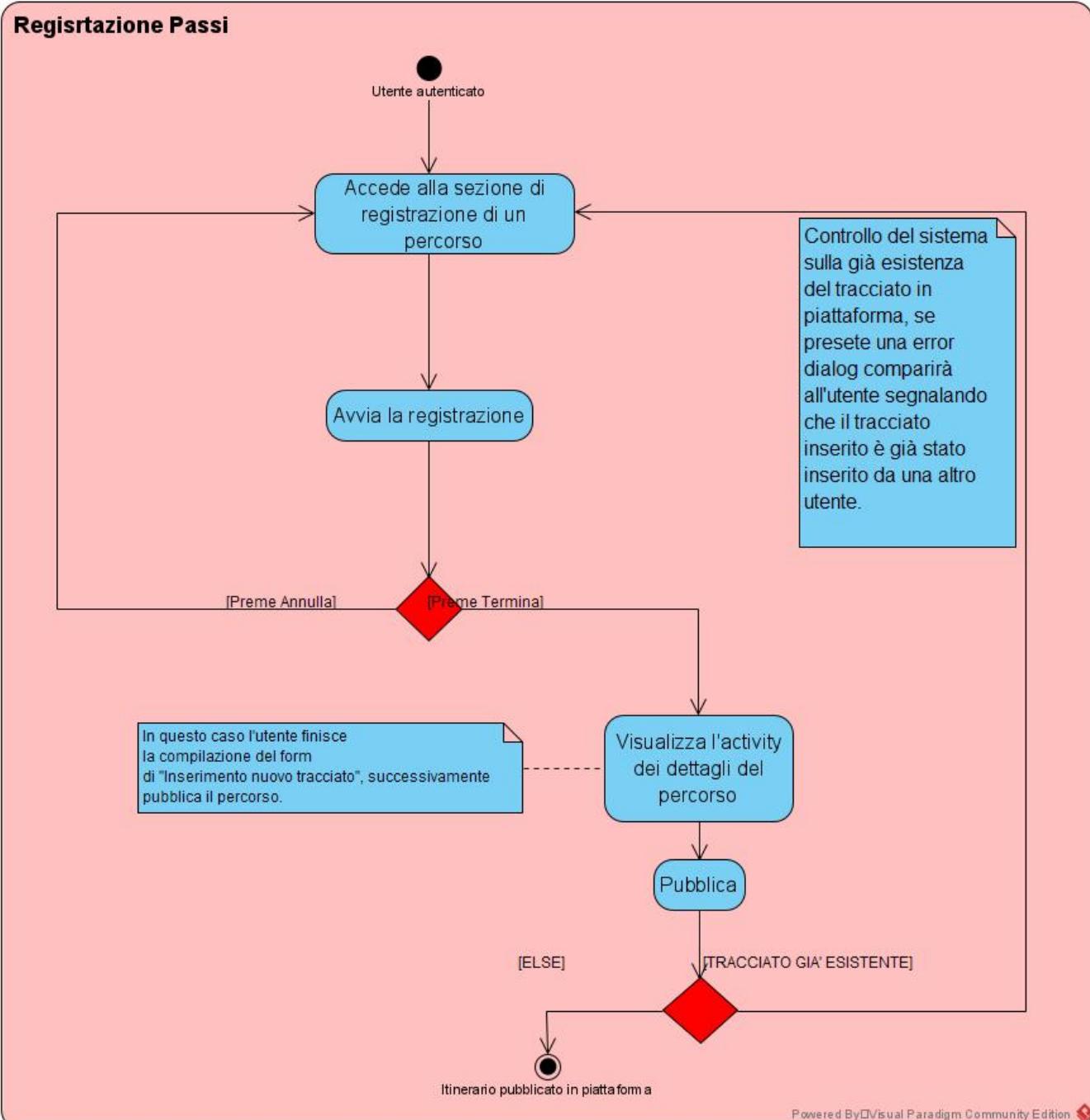


Figura 42: Activity Diagram - Insert itinerario (Registrato)

3.3.4 Activity Diagram - Inserisci Foto

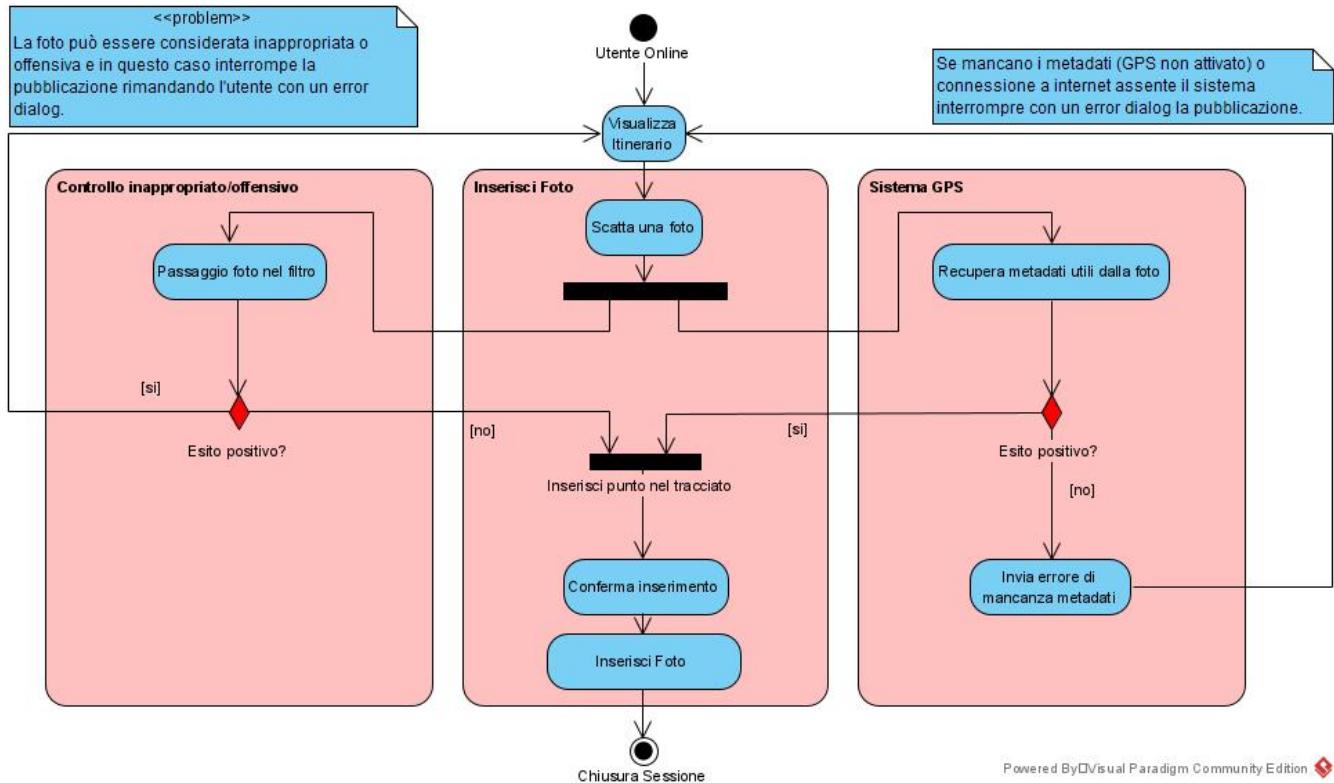


Figura 43: Activity Diagram - Inserisci Foto

3.3.5 Activity Diagram - Autenticazione

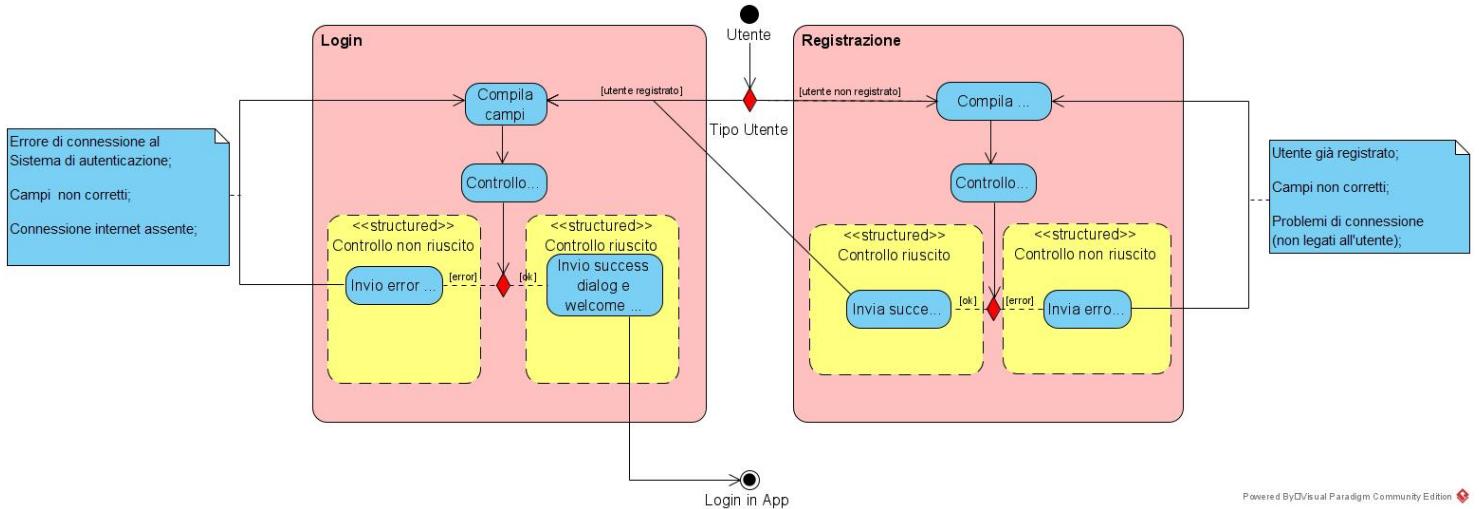


Figura 44: Activity Diagram - Autenticazione

3.3.6 Activity Diagram - Ricerca Itinerari

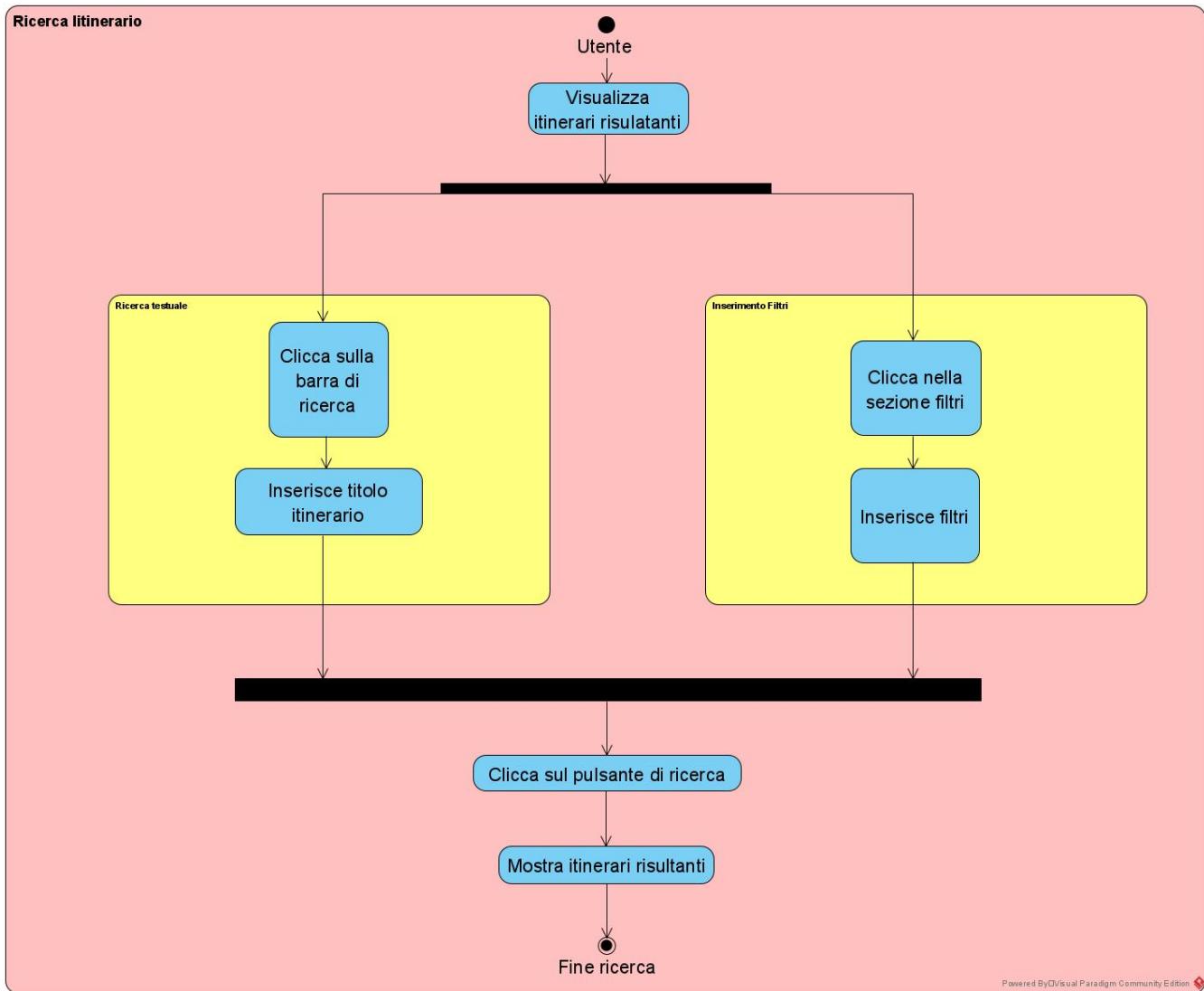


Figura 45: Activity Diagram - Ricerca Itinerari

3.3.7 Activity Diagram - Visualizza Profilo

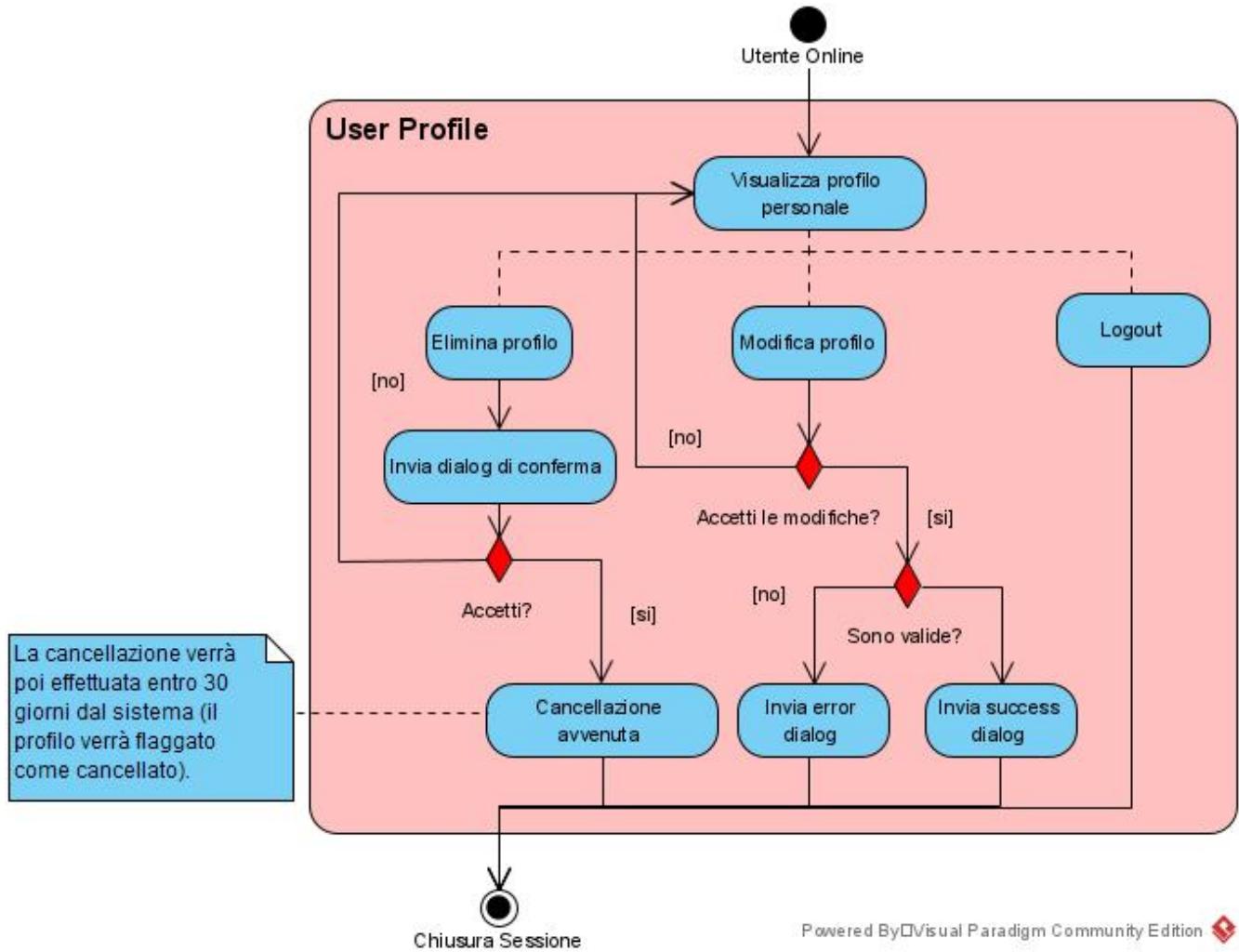


Figura 46: Activity Diagram - Visualizza Profilo

4 Design di Sistema

4.1 Analisi Architetturale

L'architettura usata per questo progetto comprende un insieme di soluzioni e strategie di famosi **design pattern**.

In particolare in questo documento analizzeremo l'architettura client server utilizzata, dove i client corrispondono nel nostro caso ai dispositivi android mobile, e il/i server al backend utilizzato per gestire i servizi che NaTour21 offre ai suoi clienti.

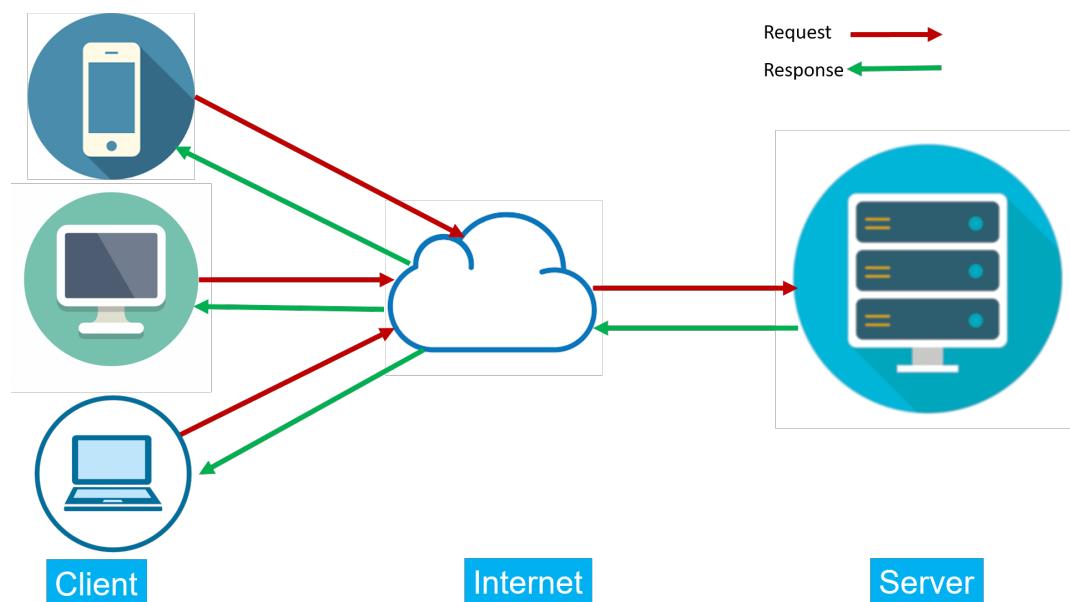


Figura 47: Architettura client server

In questa semplice illustrazione possiamo notare le frecce di **Request** e **Response** esse indicate che c'è un comunicazione asincrona tra client e server tramite un protocollo di comunicazione.

Il protocollo di comunicazione implementato è quello HTTP, analizzeremo in seguito le specifiche.

4.1.1 Descrizione architettura cloud

NaTour21 vuole offrire un back-end che abbia le seguenti caratteristiche:

- Elasticità;
- Scalabilità;
- Economicità;
- Astrazione;
- Sicurezza;
- Virtualizzazione;

In particolare noi della software house ci siamo affidati a i servizi on demand in Cloud di **AWS**¹² applicando la seguente architettura:

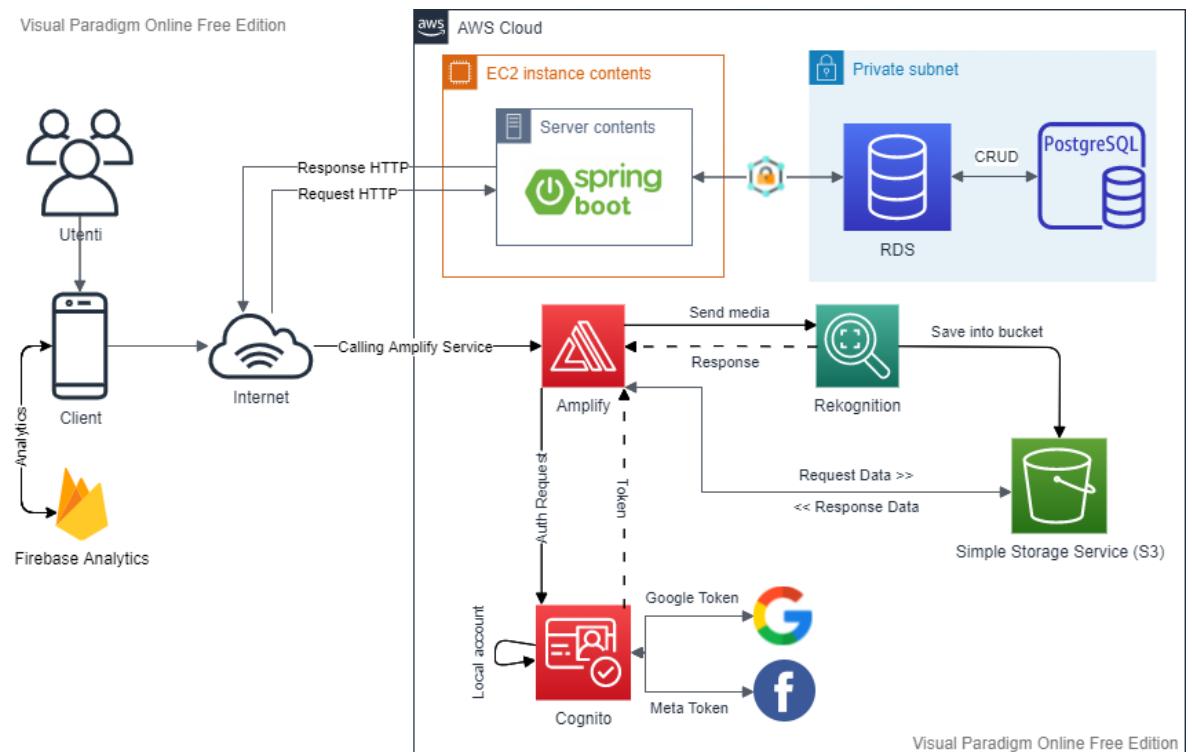


Figura 48: Caption

¹²<https://aws.amazon.com/it/>

Le tecnologie che abbiamo adoperato sono le seguenti:

Firebase Analytics Firebase è un sistema cloud di proprietà Google.inc, essendo proprietario di Android è stato molto conveniente importare alcuni dei servizi che firebase offre, in particolare il nostro focus è stato quello di attivare:

- **Crashlytics** - sistema di monitoraggio post release con informazioni riguardanti crash, failure, bug e warning.
- **Performance** - sistema di monitoraggio delle performance proveniente da tutti i dispositivi in grado di far capire quali dispositivi sono più o meno compatibili con l'app.
- **Analytics** - sistema di logging esterno all'applicazione in grado di fornire molte informazioni di usabilità post-release

SpringBoot SpringBoot¹³ è un tassello importantissimo per il nostro backend, questo famosissimo framework di Java ci ha permesso di fare un proprio insieme personalizzato di route di REST API personalizzate, nonché SpringBoot è il diretto connesso al nostro server Database per operazioni CRUD.

Amazon AWS - RDS RDS è un sistema che semplifica l'impostazione, il funzionamento e il dimensionamento di database relazionali nel cloud. Questo servizio fornisce una capacità ridimensionabile efficiente nei costi, automatizzando al tempo stesso le attività di amministrazione più dispendiose in termini di tempo, quali il provisioning di hardware, l'impostazione di database, l'applicazione di patch e i backup.

Amazon AWS - S3 Amazon Simple Storage Service (Amazon S3) è un servizio di archiviazione di oggetti che offre scalabilità, disponibilità dei dati, sicurezza e prestazioni all'avanguardia nel settore. I clienti di tutte le entità e settori possono archiviare e proteggere qualsiasi quantità di dati per qualsiasi caso d'uso, come data lake, applicazioni native per il cloud e app mobili. Con classi di archiviazione convenienti e caratteristiche di gestione di facile utilizzo, è possibile ottimizzare i costi, organizzare i dati e configurare controlli di accesso ottimizzati per soddisfare specifici requisiti aziendali, organizzativi e di conformità.

¹³<https://spring.io/projects/spring-boot>

Amazon AWS - Cognito Amazon Cognito permette di aggiungere strumenti di registrazione degli utenti, accesso e controllo degli accessi alle app Web e per dispositivi mobili, in modo rapido e semplice. Amazon Cognito permette di ridimensionare le risorse per milioni di utenti e supporta l'accesso con provider di identità social quali Apple, Facebook, Google e Amazon e provider di identità aziendali tramite SAML 2.0 e OpenID Connect.

Amazon AWS - Rekognition NON IMPLEMENTATO - Amazon Rekognition offre capacità di visione artificiale (CV) pre-addestrate e personalizzabili per estrarre informazioni e informazioni dettagliate dalle tue immagini e video.

Amazon AWS - Amplify Ultimo ma non per importanza (importante mediatore per un buon fit delle tecnologie per mobile), AWS Amplify consiste in un set di strumenti e caratteristiche appositamente progettati per consentire agli sviluppatori front-end di applicazioni Web e per dispositivi mobili di costruire rapidamente e facilmente applicazioni full-stack in AWS, con la flessibilità di sfruttare i vari servizi AWS man mano che i casi d'uso si evolvono. Con Amplify, è possibile configurare un back-end per app Web o per dispositivi mobili, connettere la tua app in pochi minuti, costruire visualmente un'interfaccia utente front-end Web e gestire facilmente i contenuti dell'app al di fuori della console AWS. Distribuisci più velocemente e dimensiona facilmente, senza dover disporre di competenze cloud.

Lasciamo i dettagli di Firebase¹⁴ e AWS¹⁵ nelle footnote di questa pagina.

¹⁴<https://firebase.google.com/>

¹⁵https://aws.amazon.com/it/?nc2=h_1g

4.1.2 Il server

Come preannunciato il server è scritto interamente in SpringBoot, famoso framework di java per il back-end, facile da implementare e da gestire. Per una buona gestione e per facilitare tante qualità quali il riutilizzo di codice, dependency injection e altro abbiamo optato per un famoso **Design Pattern**, ovvero **MVC**.

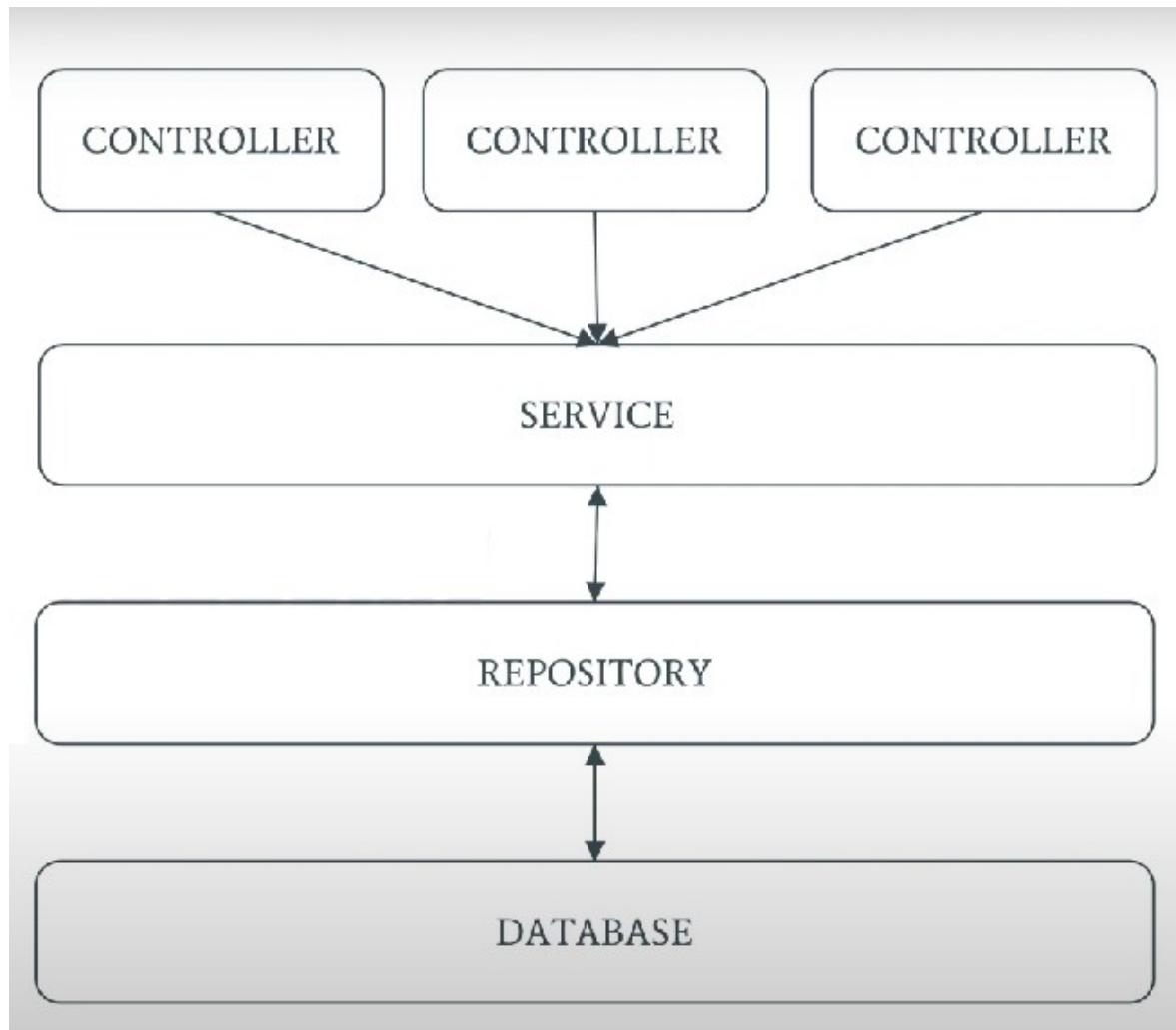


Figura 49: MVC - SpringBoot

Il design pattern segue delle linee guida per SpringBoot, infatti individuiamo i seguenti componenti:

- Components - componenti del backend quali controller, service, repository (tutte sottoclassi).
- Model - oggetti che rappresentano una figura concreta che mappata viene per essere entità di un database.
- Model DTO - Data Transfer Object sono il mezzo di comunicazione entrante/uscente del server.
- Controller - gestiscono i metodi da HTTP (GET, POST, PUT, DELETE) e relative rotte e sotto-rotte.
- Service - layer soggetto a dependency injection molto importante e intermediario tra repository e controller (qui avviene la magia).
- Repository - diretta comunicazione con la logica di database sottostante, attraverso JPA e altri moduli si interfacciano con interfacce per fare operazioni CRUD.

Springboot ha il vantaggio di **astrarre** completamente il codice da qualsiasi implementazione esterna, un esempio concreto è l'utilizzo della dependency **JPA** famoso modulo che permette la gestione tramite REST API e con l'aiuto di SOLE annotation, la persistenza dei dati in un database. Il database appunto non è altro che una riga con un link source, in questo caso, cambiare questa riga nelle proprietà del server, adatterà il server ad un database diverso senza preoccuparsi effettivamente di che tipo è, infatti JPA gestirà le query SQL adattandosi al database e al JDBC a cui ci associamo.

Nel prossimo paragrafo elencheremo le REST API create appositamente per offrire i servizi tramite springboot per i nostri clienti NaTour21.

4.1.3 REST API

In questa sezione nello specifico elencheremo le **REST API** implementate e relative route:

CAMPO	ROUTE	DESCRIZIONE
Compilation	http://server-ip:port/api/compilation	Main Route per operazioni sulle compilation.
FotoItinerario	http://server-ip:port/api/fotoItinerario	Main Route per operazioni sulle foto di un itinerario.
InterestingPoint	http://server-ip:port/api/interestingpoint	Main Route per operazioni su gli interestingpoint di un itinerario.
Tappa	http://server-ip:port/api/tappa	Main Route per operazioni sulle tappe di un itinerario.
Itinerario	http://server-ip:port/api/itinerario	Main Route per operazioni su un itinerario.
Utente	http://server-ip:port/api/user	Main Route per operazioni su gli utenti.

Tabella 1: Tabella REST API

Ogni main route ha delle sotto-route che compiono azioni più specifiche, queste sono visibili della javadoc del server SpringBoot.

Il protocollo di comunicazione, come accennato in precedenza è quello HTTP.

4.1.4 Il client

Il client di NaTour21 è stato sviluppato su piattaforma **Android**. I vantaggi di lavorare su android sono molteplici uno tra quali l'**Open Source**, infatti grazie ad android abbiamo avuto la possibilità di avere accesso a numerose librerie proprie di google oppure sviluppate dalle comunità.

Per il progetto abbiamo usato il design pattern **MVP**:

- **Model** - rappresenta il modello dei dati di interesse per l'applicazione. Tale livello si occupa di incapsulare lo stato dell'applicazione, gestisce l'accesso alla sorgente dei dati, fornisce funzionalità per l'aggiornamento dello stato e l'accesso ai dati. Nel client android sviluppato, è compito di questo livello gestire la comunicazione diretta con le API Lambda e quindi si occupa della formattazione delle richieste e della decodifica delle risposte. Infine, il MODEL notifica al PRESENTER il cambiamento di stato.
- **View** - questo livello rappresenta l'interfaccia utente, permettendo l'interazione con esso e fornendo una rappresentazione grafica ed interattiva del model. La responsabilità di questo livello è la presentazione dei dati e dello stato dell'applicazione. Inoltre, esso riceve notifiche dal PRESENTER e aggiorna la visualizzazione.
- **Presenter** - tale livello definisce la logica di controllo e le funzionalità applicative. Quindi, è compito di questo livello gestire gli eventi ed i comandi generati dall'utente: in base a questi ultimi, esso opera sul model. Infine, tale livello si occupa di selezionare/aggiornare il livello VIEW in base ai dati recuperati.

MVP Pattern

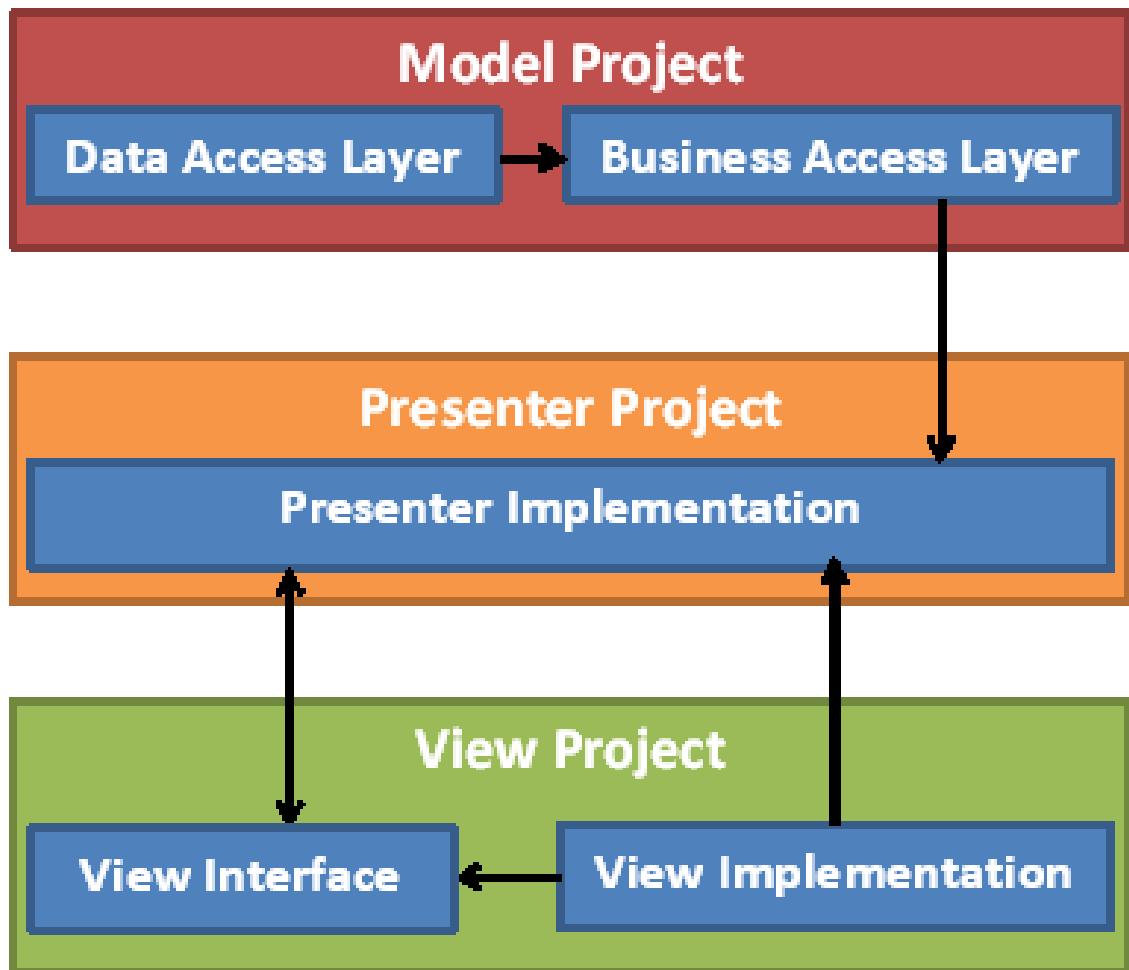


Figura 50: MVP

4.1.5 Supporti

Open Street Map è la libreria API che abbiamo utilizzato per quanto riguarda tutte le azioni che hanno bisogno di una MAPPA. OSM è stata una scelta ponderata con Google Maps come alternativa, la scelta è stata finalizzata dopo aver visto i seguenti dettagli:

- Licenza: Open Source (Google Maps è di Google).
- Costo: Gratis (Google Maps limitante sul numero di richieste).
- Supporto: librerie degli utenti e della community (Google Maps ha solo la propria documentazione).
- Auto Routeing: attraverso l'utilizzo di flag è possibile calcolare percorsi in modo semplice e veloce.

Unico svantaggio di OSM è la latenza server, spesso soggetti a crash temporanei.

Retrofit Retrofit¹⁶ è un modulo di comunicazione tramite protocollo HTTP per chiamate a REST API

RxJava RxJava¹⁷ è un modulo per integrare Reactive-x java e inserire codice asincrono utilizzando il noto design pattern **Observer**.

¹⁶<https://square.github.io/retrofit/>

¹⁷<https://github.com/ReactiveX/RxJava>

4.2 Gantt Diagram

Per completezza lasciamo l'organizzazione del lavoro del team di sviluppo:

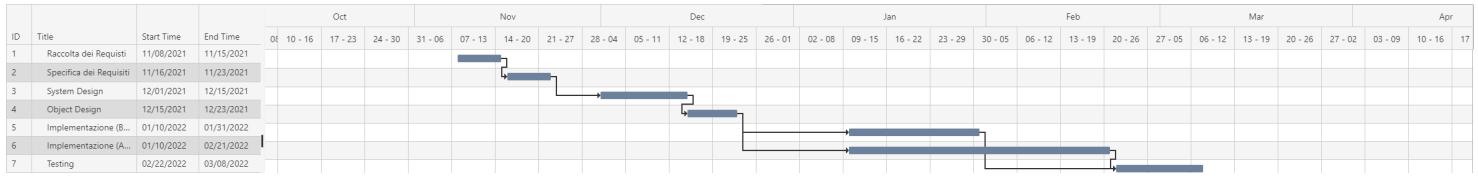


Figura 51: Gantt Chart

Come si può notare per finire il progetto entro la **dead line** del 31/03/2022 il team di sviluppo ha lavorato senza sosta per far sì che ogni periodo utile sia sfruttato al meglio.

Particolare attenzione allo sviluppo di **back-end** (ID=4) e **front-end** (ID=5) che sono stati sviluppati in modo parallelo.

4.3 Class Diagram di Design

I class dagram che possiamo individuare qui sono quelli che effettivamente rappresentano il sistema sviluppato nell'applicativo andorid.

Nel class diagram essendo molto grande è stato sia suddiviso in sotto-diagrammi che omesso metodi semplici come getter e setter o semplici notify da una riga di codice caduno e sono stati omessi quelli che non fanno parte dell'input principale (punti commissionati).

4.3.1 Class Diagram Design - Login

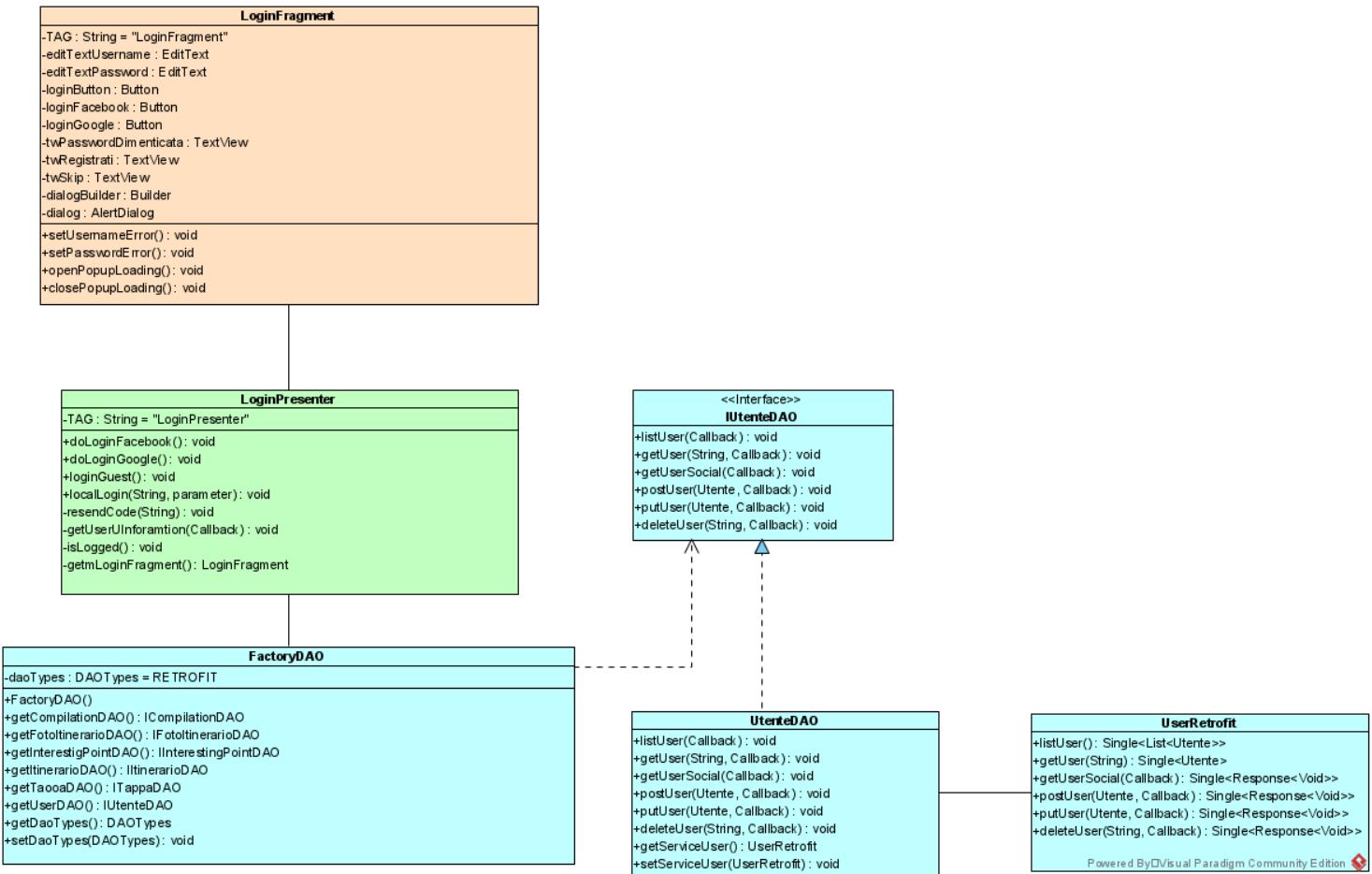


Figura 52: Class Diagram Design - Login

4.3.2 Class Diagram Design - Registrazione

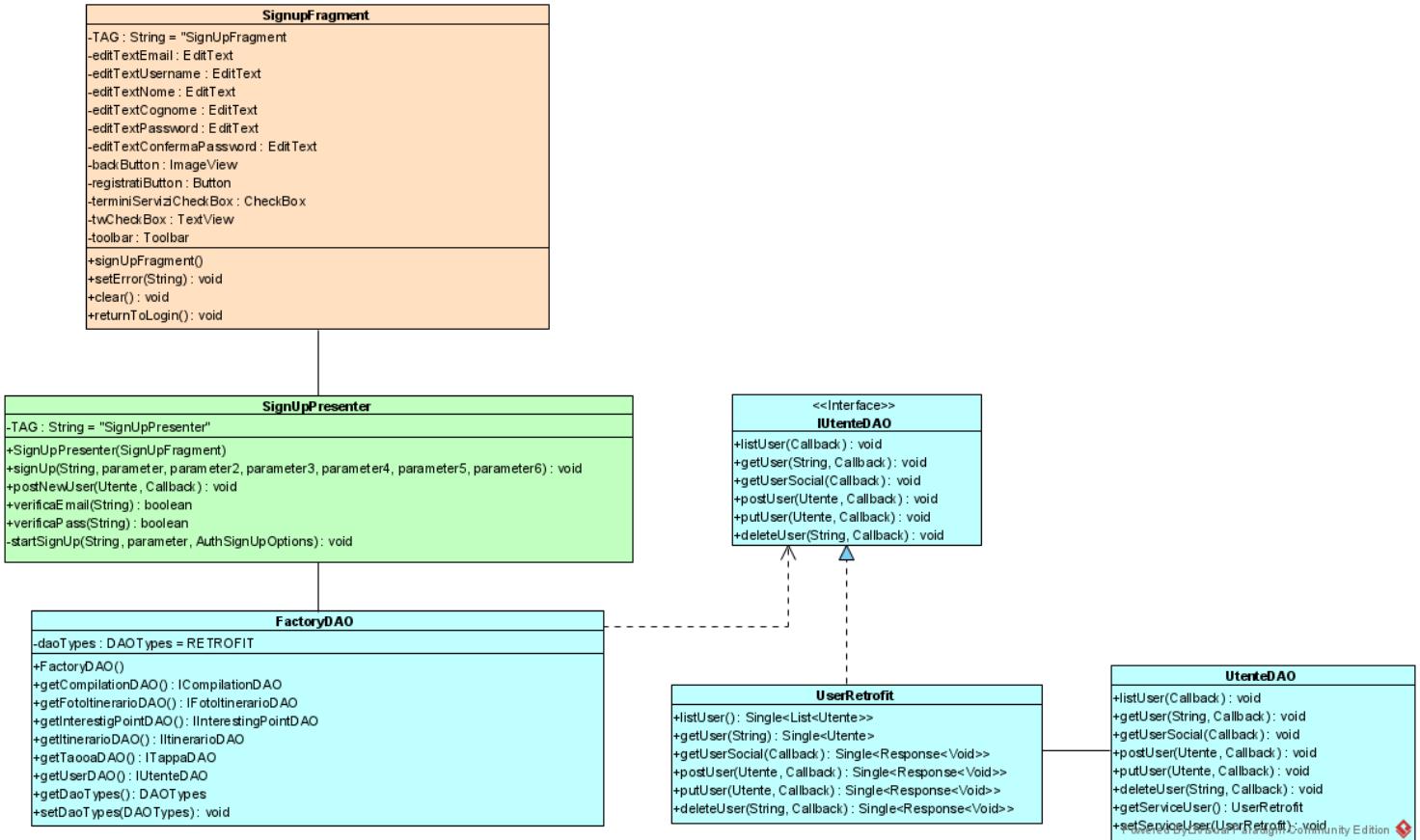


Figura 53: Class Diagram Design - Registrazione

4.3.3 Class Diagram Design - Gestione Collezione

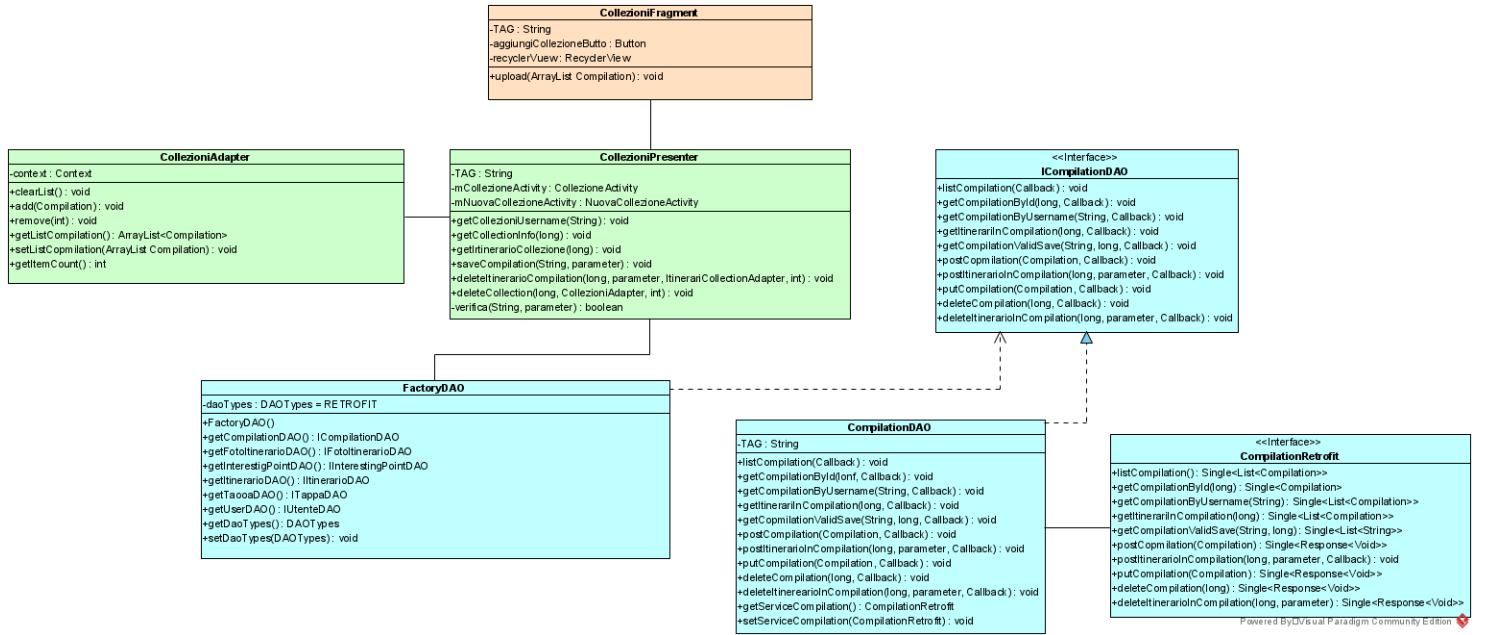


Figura 54: Class Diagram Design - Gestione Collezione

4.3.4 Class Diagram Design - Nuova Collezione

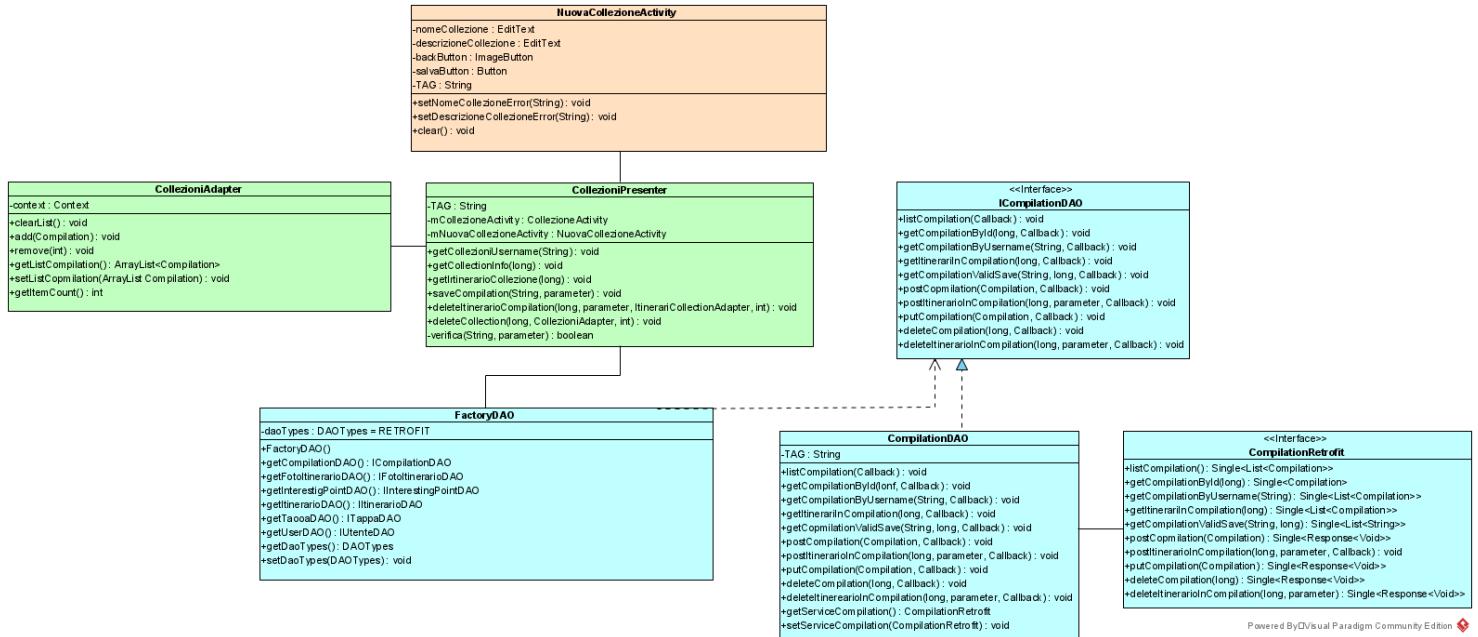


Figura 55: Class Diagram Design - Nuova Collezione

4.3.5 Class Diagram Design - Pubblicazione Itinerario

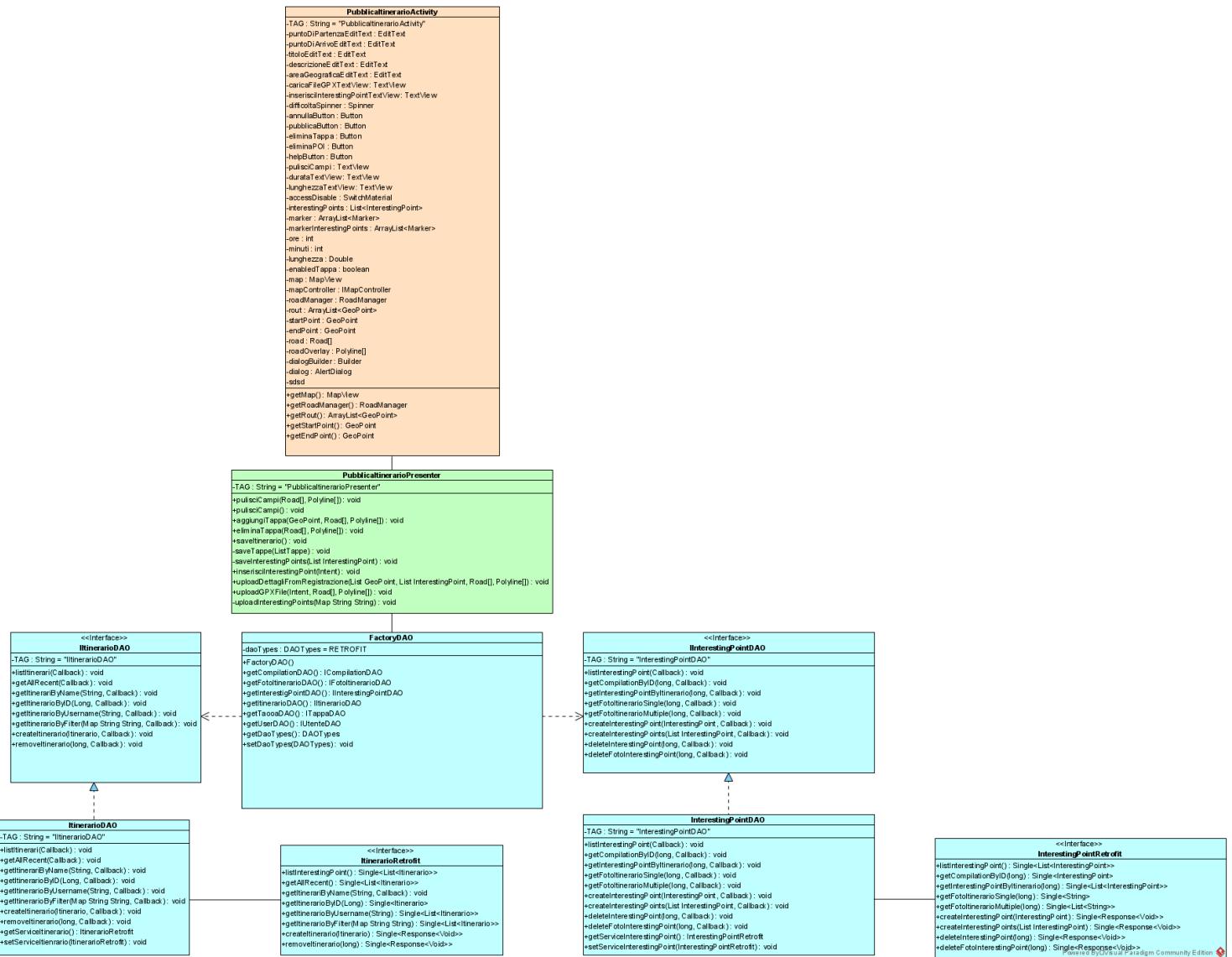


Figura 56: Class Diagram Design - Pubblicazione Itinerario

4.3.6 Class Diagram Design - Registrazione Itinerario

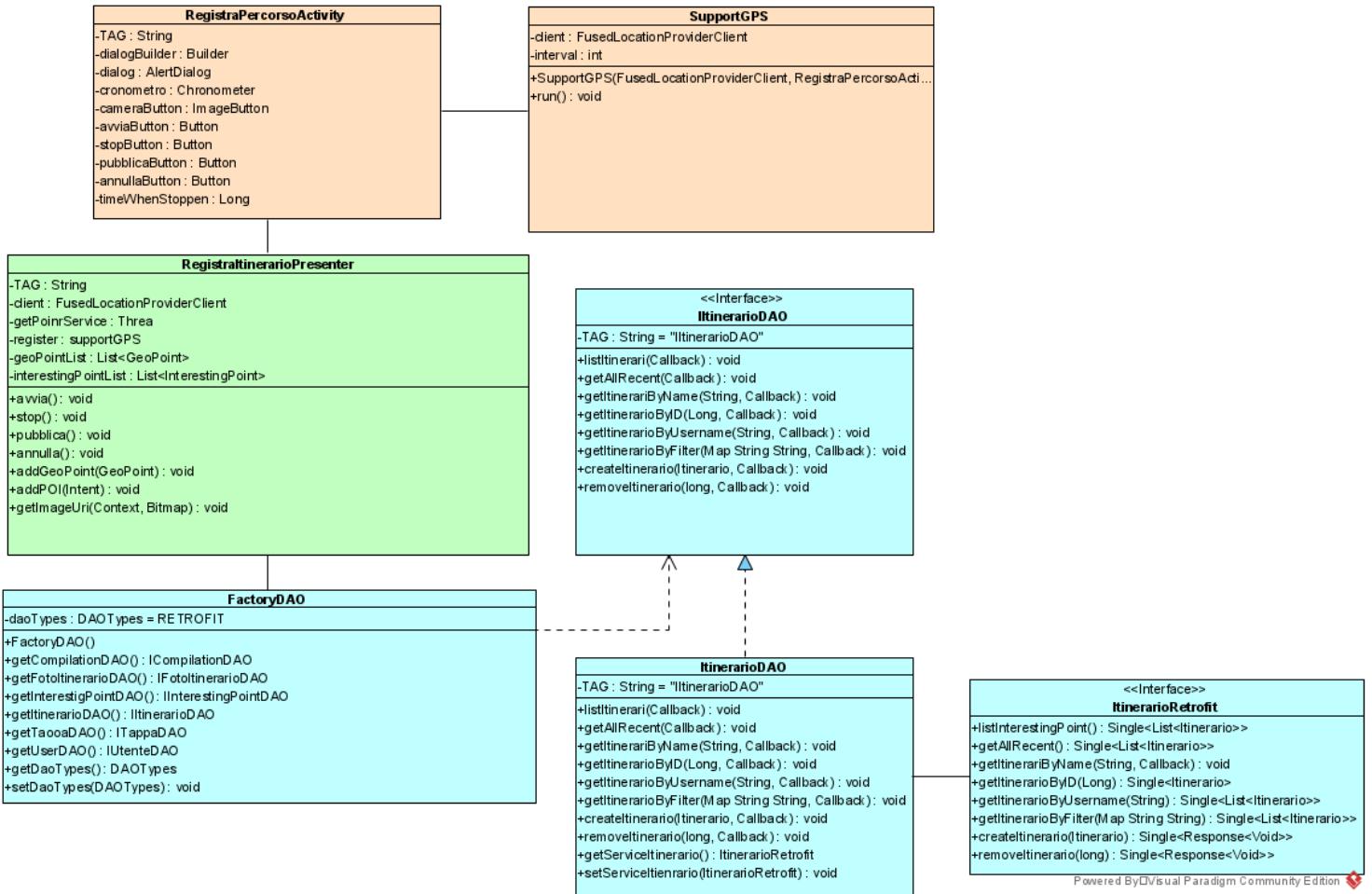


Figura 57: Class Diagram Design - Registrazione Itinerario

4.3.7 Class Diagram Design - Visualizzazione Itinerari

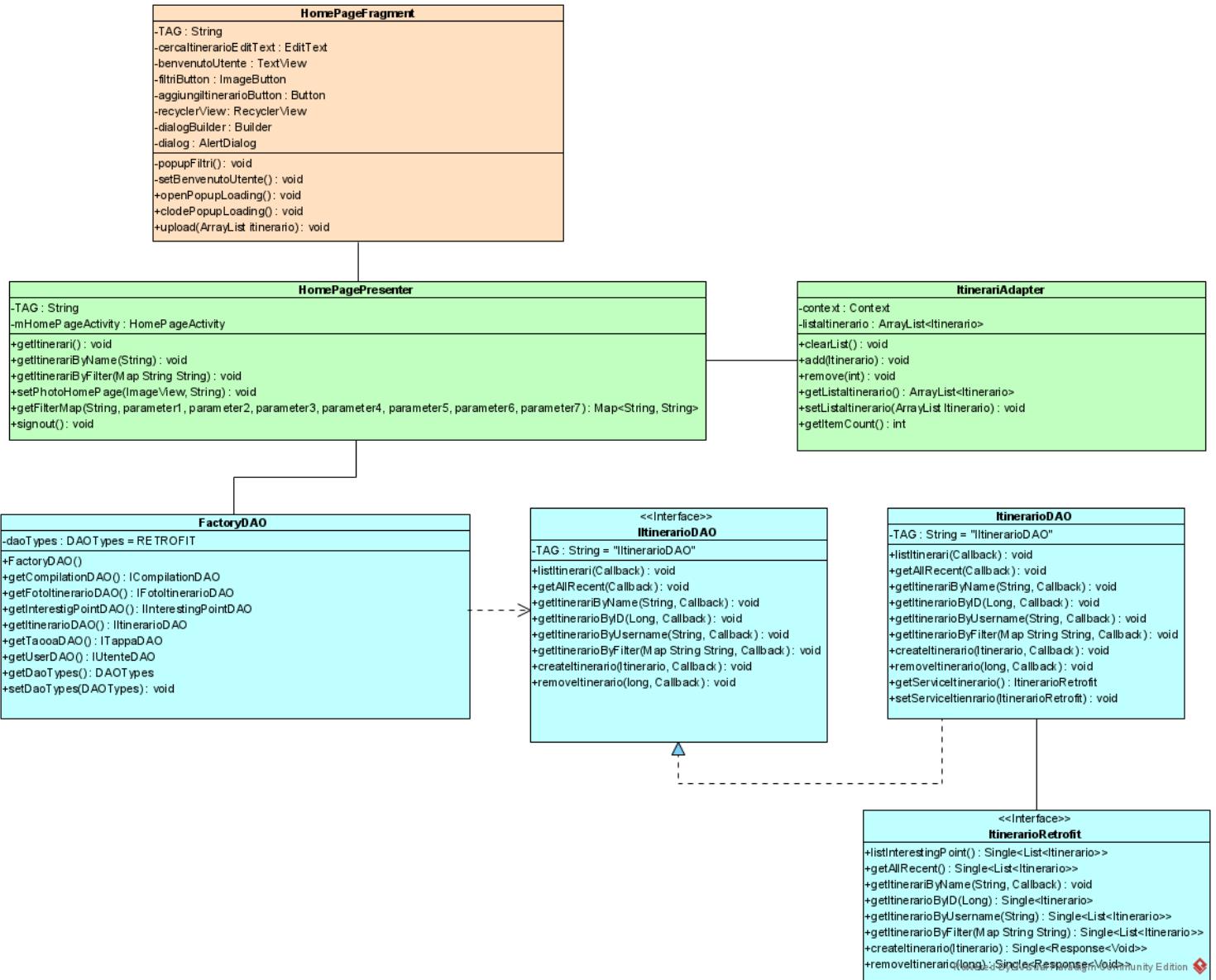


Figura 58: Class Diagram Design - Visualizzazione Itinerari

4.4 Sequence Diagram di Design

Questa sezione contiene i sequence diagram di design, qui si specificano concretamente come gli oggetti nei due metodi da noi selezionati.

Nota bene abbiamo per semplicità ampliato nello specifico le callback e chiamate async proprie del nostro server ma non quelle proprie di AWS Amplify, inoltre le callback differenziate in:

- Callback - onSuccess(Object);
- Callback - onFailure(Throwable e);

sono state specificate in un ramo (if/else) per convenzione UML).

4.4.1 Sequence Diagram Design - Inserimento Itinerario

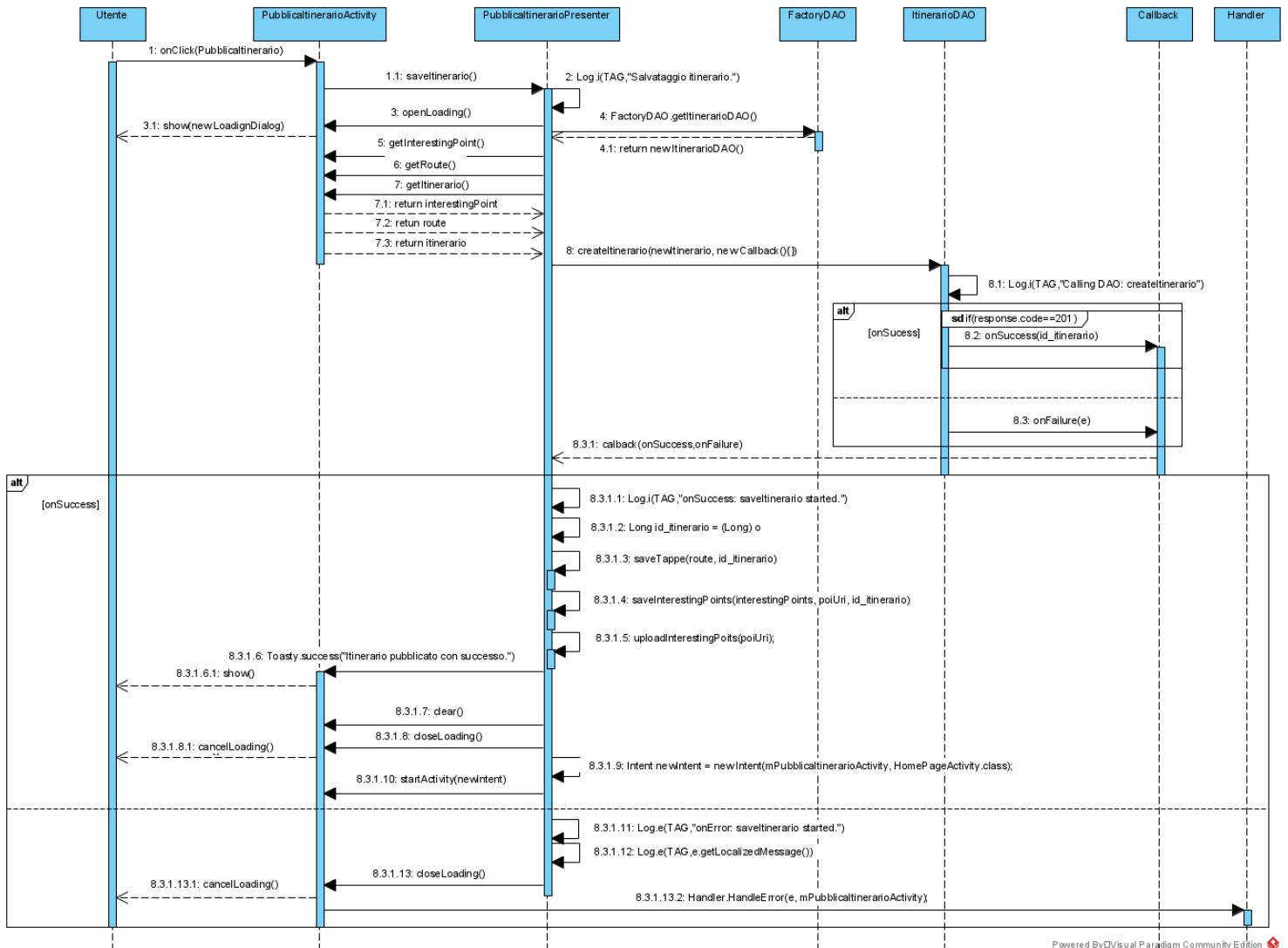


Figura 59: Sequence Diagram - Design: Inserisci Itinerario

4.4.2 Sequence Diagram Design - Carica Foto

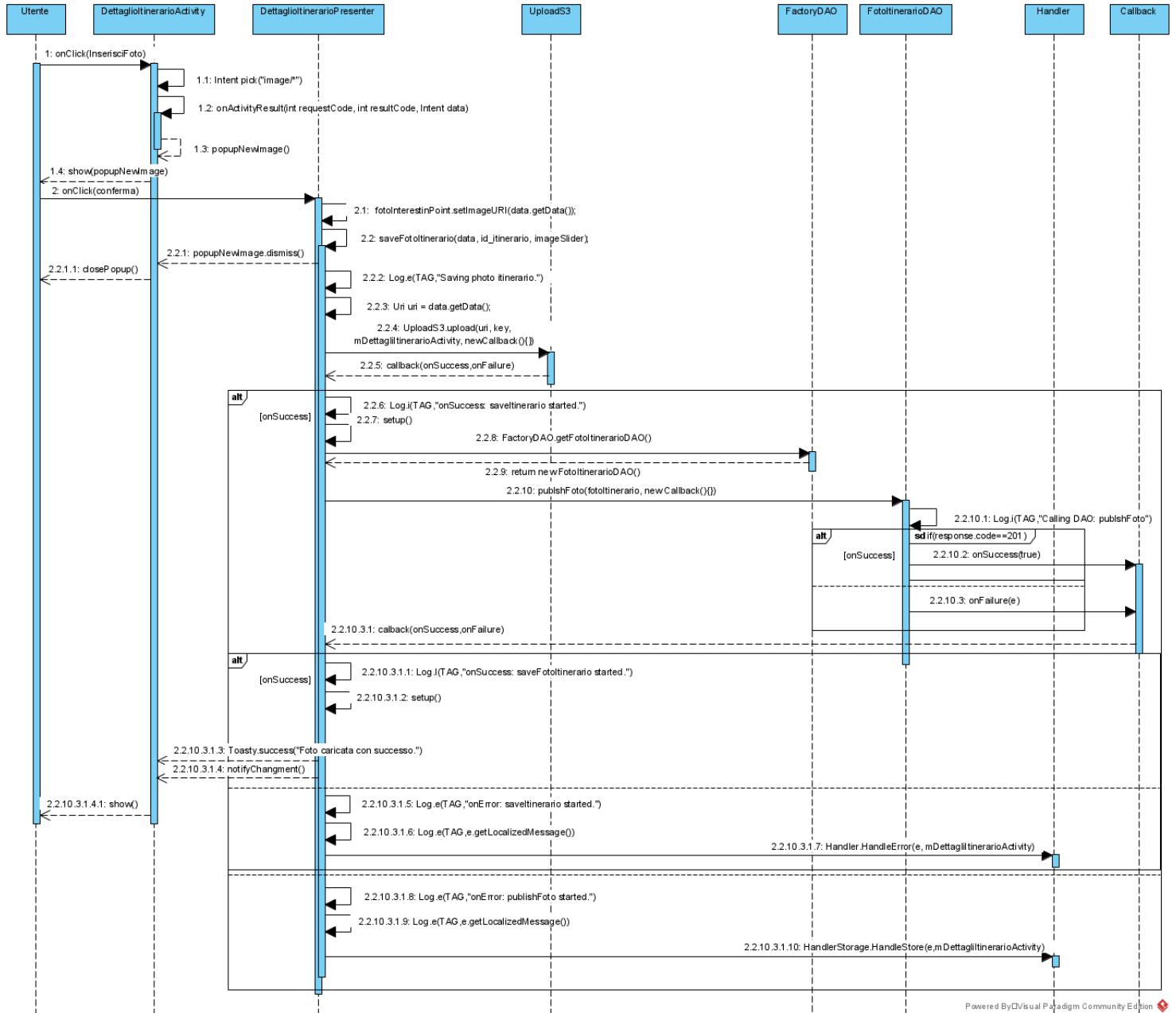
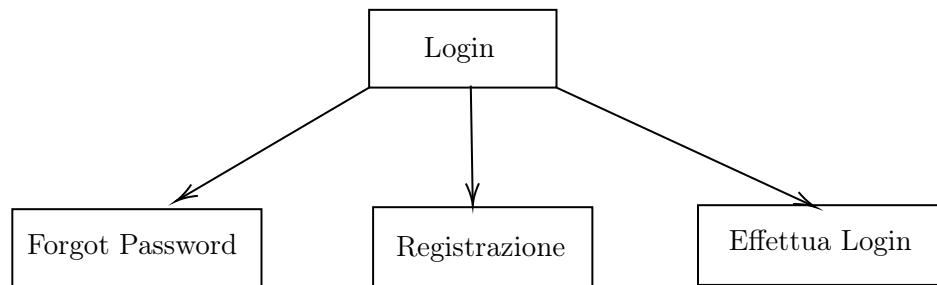


Figura 60: Sequence Diagram - Design: Carica Foto

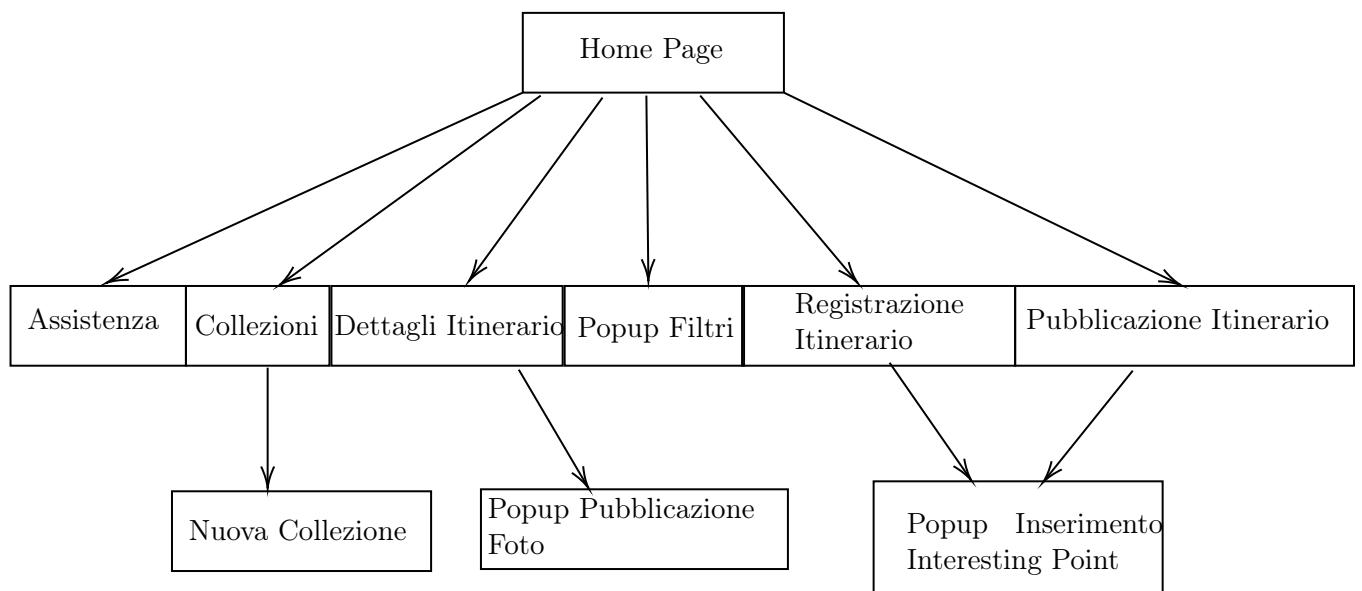
4.5 Gerarchie Funzionali

Rappresentiamo in questo paragrafo le gerarchie funzionali dell'applicativo, per semplicità abbiamo suddiviso le funzionalità in 2 macro categorie che abbiamo gestito come insiemi a parte ma comunicanti in qualche modo tra di loro.

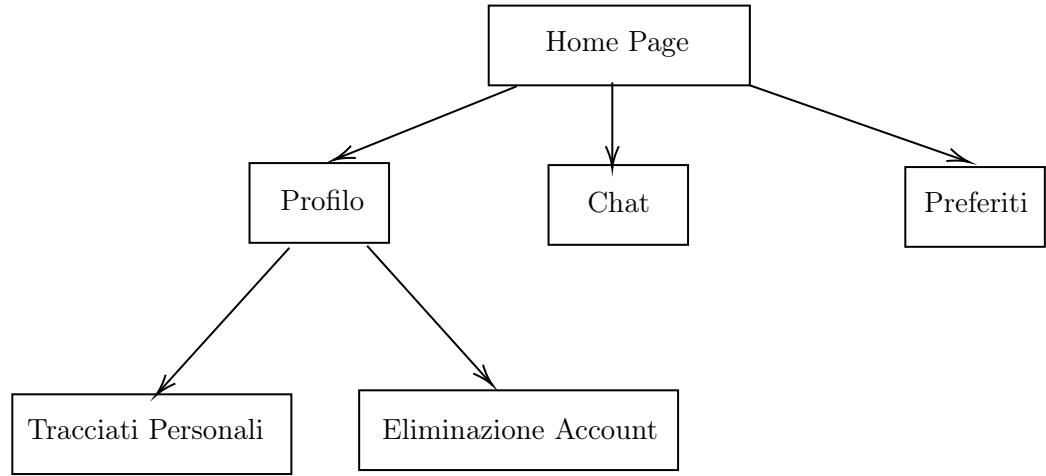
4.5.1 Login



4.5.2 HomePage pt.1



4.5.3 HomePage pt.2



5 Codice sorgente sviluppato

Il codice sorgente sviluppato dell'applicativo NaTour21 è possibile reperirlo cliccando qui per la parte di android e qui per quella di progettazione (con server e documentazione).

5.1 Versionamento e licenza

La licenza dell'applicativo e dell'intero software è la GNU - General Public License v3.0:

"Permissions of this strong copyleft license are conditioned on making available complete source code of licensed works and modifications, which include larger works using a licensed work, under the same license. Copyright and license notices must be preserved. Contributors provide an express grant of patent rights."

5.1.1 Permessi

- Commercial use.
- Modification.
- Distribution.
- Patent use.
- Private use.

5.1.2 Limitazioni

- Liability.
- Warranty.

5.1.3 Condizioni

- License and copyright notice.
- State changes.
- Disclose source.
- Same license.

6 Codice xUnit

Come richiesto dalla SoftEngUniNA abbiamo pensato di creare una suite di test per 3 metodi non banali con JUnit Testing Framework.

I metodi scelti sono i seguenti:

- Filter Query (query per ricerca tramite filtri).
- RequestGenerator (generatore di path REST Api).
- Verifica delle policy di email e password.

6.1 Filter Query

```
/*
Class test for filter map (a map for query to filter search):

We Mocked the method for a easy reuse (i.e @Before annotation).

All parameter are converted in string for a easy manipulation.

A specific query is valuated with "LIKE" keyword in SQL, so the
empty string mean "*" in Regex.

Length is a double >= 0, else is converted automatically
in 100.0 (illegal argument if is not a double);

Time have a standard format hh:mm:ss, if is empty is automatically
converted in 23:59:00
(illegal argument if don't match with
"^(\\d\\d:[0-5]\\d:[0-5]\\d)");

Disable Access is a boolean
(case_insensitive, true==True/false==False, illegal argument if is
not a boolean);

Difficulty have only 5 values that is the grade scale
of Trekking, if value is "Qualsiasi" string is
automatically converted in empty string;

Other parameters are string and they can have any string value
(not only numbers), empty string and null
(server have a controller for this).
```

```

Query (insert in the map for a easy retry of query url in case
of reuse):
?titolo=&puntoinizio=&puntofine=&durata=&lenght=&difficulty=&accessodisabili=

Strategy used for testing:
Black Box - WECT
*/



public class FilterQueryTesting {

    Map<String, String> queryMapper = new HashMap<>();
    List<String> parameters = new ArrayList<>();
    SearchFilterMock searchFilterMock = new SearchFilterMock();
    String goodQuery =
        "?titolo=Viaggio a Napoli" +
        "&puntoinizio=Via Roma, 80133, NA" +
        "&puntofine=Mergellina, 80122, NA" +
        "&durata=03:30:00" +
        "&lenght=44.0" +
        "&difficulty=T-Turistico" +
        "&accessodisabili=true" +
        "&areageografica=Napoli";

    @Before
    public void setQuery() {
        queryMapper = new HashMap<>();
        parameters = new ArrayList<>();
        //Itinerario titolo (id=0)
        parameters.add("Viaggio a Napoli");
        //Punto di Inizio (id=1)
        parameters.add("Via Roma, 80133, NA");
        //Punto di Fine (id=2)
        parameters.add("Mergellina, 80122, NA");
        //Durata (standard hh:mm:ss) (id=3)
        parameters.add("03:30:00");
        //Distanza (in km) (id=4)
        parameters.add("44.0");
        //Difficulty (id=5)
        parameters.add("T-Turistico");
        //Accesso disabili (id=6)
        parameters.add("true");
        //Area geografica (id=7)
        parameters.add("Napoli");
    }
}

```

```

@Test
public void allGoodParameter() {
    queryMapper = searchFilterMock.getMap(parameters);
    assertEquals(goodQuery,queryMapper.get("url"));
}

@Test
public void anyParamNull() {
    parameters.set(0,null);
    queryMapper = searchFilterMock.getMap(parameters);
    assertEquals(null,queryMapper.get(0));
}

@Test(expected = IllegalArgumentException.class)
public void titoloNumeric() {
    parameters.set(0,"123");
    queryMapper = searchFilterMock.getMap(parameters);
}

@Test(expected = IllegalArgumentException.class)
public void puntoInizioNumeric() {
    parameters.set(1,"123");
    queryMapper = searchFilterMock.getMap(parameters);
}

@Test(expected = IllegalArgumentException.class)
public void puntoFineNumeric() {
    parameters.set(2,"123");
    queryMapper = searchFilterMock.getMap(parameters);
}

@Test(expected = IllegalArgumentException.class)
public void durataNoMatch() {
    parameters.set(3,"321:88:");
    queryMapper = searchFilterMock.getMap(parameters);
}

@Test(expected = IllegalArgumentException.class)
public void durataNoMatchHH() {
    parameters.set(3,"321:30:00");
    queryMapper = searchFilterMock.getMap(parameters);
}

@Test(expected = IllegalArgumentException.class)
public void durataNoMatchMM() {
    parameters.set(3,"03:88:00");
    queryMapper = searchFilterMock.getMap(parameters);
}

```

```

@Test(expected = IllegalArgumentException.class)
public void durataNoMatchSS() {
    parameters.set(3, "03:30:77");
    queryMapper = searchFilterMock.getMap(parameters);
}

@Test
public void durataEmpty() {
    parameters.set(3, "");
    queryMapper = searchFilterMock.getMap(parameters);
    assertEquals("23:59:00", queryMapper.get("durata"));
}

@Test(expected = IllegalArgumentException.class)
public void lengthNotNumber() {
    parameters.set(4, "CENTO");
    queryMapper = searchFilterMock.getMap(parameters);
}

@Test(expected = IllegalArgumentException.class)
public void lengthLowerThenZero() {
    parameters.set(4, "-1");
    queryMapper = searchFilterMock.getMap(parameters);
}

@Test
public void lengthEmpty() {
    parameters.set(4, "");
    queryMapper = searchFilterMock.getMap(parameters);
    assertEquals("100.0", queryMapper.get("length"));
}

@Test
public void difficultyQualsiasi() {
    parameters.set(5, "Qualsiasi");
    queryMapper = searchFilterMock.getMap(parameters);
    assertEquals("", queryMapper.get("difficulty"));
}

@Test(expected = IllegalArgumentException.class)
public void accessoDiabiliNotBoolean() {
    parameters.set(6, "vero");
    queryMapper = searchFilterMock.getMap(parameters);
}

```

```
@Test(expected = IllegalArgumentException.class)
public void areaGeograficaNumeric() {
    parameters.set(7, "123");
    queryMapper = searchFilterMock.getMap(parameters);
}
}
```

6.2 Request Generator

6.2.1 BlackBox

```
/*
Class test for Retrofit instance generator:

Request generator is a class of Utils package
created to instantiate retrofit that returns a
connection using base url and a suffix.

To mock retrofit we will build a new mock class.

Parameters:
Enumeration API with some values:
--FOTO_ITINERARIO;
--INTERESTINGPOINT_API;
--TAPPA_API;
--ITINERARIO_API;
--COMPILEMENTATION_API;
--SERVER_API;
--USER_API;

The base url can be changed into code (is a Static Public String).

Strategy used for testing:
Black Box - SECT
*/
public class RequestGeneratorTestingBlackBox {

    RequestGeneratorMock requestGeneratorMock;

    @Before
    public void setup() {
        requestGeneratorMock = new RequestGeneratorMock();
    }

    @Test
    public void routeFotoItinerario() {
        String expected = requestGeneratorMock.getBaseUrl() +
            "/api/fotoItinerario/";
        requestGeneratorMock.RetrofitInstance(API.FOTO_ITINERARIO);
        assertEquals(expected, requestGeneratorMock.getCurrentUrl());
    }
}
```

```

@Test
public void routeInterestingPoint(){
    String expected = requestGeneratorMock.getBaseUrl() +
        "/api/interestingpoint";
    requestGeneratorMock.retrofitInstance(API.INTERESTINGPOINT_API);
    assertEquals(expected,requestGeneratorMock.getCurrentUrl());
}

@Test
public void routeTappa(){
    String expected = requestGeneratorMock.getBaseUrl() +
        "/api/tappa/";
    requestGeneratorMock.retrofitInstance(API.TAPPA_API);
    assertEquals(expected,requestGeneratorMock.getCurrentUrl());
}

@Test
public void routeItinerario(){
    String expected = requestGeneratorMock.getBaseUrl() +
        "/api/itinerario/";
    requestGeneratorMock.retrofitInstance(API.ITINERARIO_API);
    assertEquals(expected,requestGeneratorMock.getCurrentUrl());
}

@Test
public void routeCompilation(){
    String expected = requestGeneratorMock.getBaseUrl() +
        "/api/compilation/";
    requestGeneratorMock.retrofitInstance(API.COMPILEATION_API);
    assertEquals(expected,requestGeneratorMock.getCurrentUrl());
}

@Test
public void routeServer(){
    String expected = requestGeneratorMock.getBaseUrl() + "/";
    requestGeneratorMock.retrofitInstance(API.SERVER_API);
    assertEquals(expected,requestGeneratorMock.getCurrentUrl());
}

@Test
public void routeUser(){
    String expected = requestGeneratorMock.getBaseUrl() +
        "/api/user/";
    requestGeneratorMock.retrofitInstance(API.USER_API);
    assertEquals(expected,requestGeneratorMock.getCurrentUrl());
}

```

```

@Test
public void routeFotoItinerarioCurrentNotNull() {
    requestGeneratorMock.setCURRENT_URL(requestGeneratorMock.getBaseUrl()
        + "/api/fotoItinerario/");
    String expected = requestGeneratorMock.getBaseUrl() +
        "/api/fotoItinerario/";
    requestGeneratorMock.RetrofitInstance(API.FOTO_ITINERARIO);
    assertEquals(expected,requestGeneratorMock.getCURRENT_URL());
}

@Test
public void routeInterestingPointCurrentNotNull(){
    requestGeneratorMock.setCURRENT_URL(requestGeneratorMock.getBaseUrl()
        + "/api/fotoItinerario/");
    String expected = requestGeneratorMock.getBaseUrl() +
        "/api/interestingpoint/";
    requestGeneratorMock.RetrofitInstance(API.INTERESTINGPOINT_API);
    assertEquals(expected,requestGeneratorMock.getCURRENT_URL());
}

@Test
public void routeTappaCurrentNotNull(){
    requestGeneratorMock.setCURRENT_URL(requestGeneratorMock.getBaseUrl()
        + "/api/fotoItinerario/");
    String expected = requestGeneratorMock.getBaseUrl() +
        "/api/tappa/";
    requestGeneratorMock.RetrofitInstance(API.TAPPA_API);
    assertEquals(expected,requestGeneratorMock.getCURRENT_URL());
}

@Test
public void routeItinerarioCurrentNotNull(){
    requestGeneratorMock.setCURRENT_URL(requestGeneratorMock.getBaseUrl()
        + "/api/fotoItinerario/");
    String expected = requestGeneratorMock.getBaseUrl() +
        "/api/itinerario/";
    requestGeneratorMock.RetrofitInstance(API.ITINERARIO_API);
    assertEquals(expected,requestGeneratorMock.getCURRENT_URL());
}

```

```

    @Test
    public void routeCompilationCurrentNotNull(){
        requestGeneratorMock.setCURRENT_URL(requestGeneratorMock.getBaseUrl()
            + "/api/fotoItinerario/");
        String expected = requestGeneratorMock.getBaseUrl() +
            "/api/compilation/";
        requestGeneratorMock.RetrofitInstance(API.COMPILEATION_API);
        assertEquals(expected,requestGeneratorMock.getCURRENT_URL());
    }

    @Test
    public void routeServerCurrentNotNull(){
        requestGeneratorMock.setCURRENT_URL(requestGeneratorMock.getBaseUrl()
            + "/api/fotoItinerario/");
        String expected = requestGeneratorMock.getBaseUrl() + "/";
        requestGeneratorMock.RetrofitInstance(API.SERVER_API);
        assertEquals(expected,requestGeneratorMock.getCURRENT_URL());
    }

    @Test
    public void routeUserCurrentNotNull(){
        requestGeneratorMock.setCURRENT_URL(requestGeneratorMock.getBaseUrl()
            + "/api/fotoItinerario/");
        String expected = requestGeneratorMock.getBaseUrl() +
            "/api/user/";
        requestGeneratorMock.RetrofitInstance(API.USER_API);
        assertEquals(expected,requestGeneratorMock.getCURRENT_URL());
    }

    @Test
    public void routeUndefined(){
        String expected = requestGeneratorMock.getBaseUrl() +
            "/api/undefined/";
        requestGeneratorMock.RetrofitInstance(API.CHAT_API);
        assertEquals(expected,
            requestGeneratorMock.getCURRENT_URL());
    }

    @Test (expected = IllegalArgumentException.class)
    public void routeIllegalArgumentException(){
        requestGeneratorMock.RetrofitInstance(null);
    }
}

```

6.2.2 WhiteBox

```
/*
Class test for Retrofit instance generator:

Request generator is a class of Utils package
created to instantiate retrofit that returns a
connection using base url and a suffix.

To mock retrofit we will build a new mock class.

Parameters:
Enumeration API with some values:
--FOTO_ITINERARIO;
--INTERESTINGPOINT_API;
--TAPPA_API;
--ITINERARIO_API;
--COMPILATION_API;
--SERVER_API;
--USER_API;

The base url can be changed into code (is a Static Public String).

Strategy used for testing:
White Box - All condition coverage.
*/
public class RequestGeneratorTestingWhiteBox {

    RequestGeneratorMock requestGeneratorMock;

    @Before
    public void setup() {
        requestGeneratorMock = new RequestGeneratorMock();
    }

    @Test
    public void FirstTrueSecondTrue(){
        String expected = requestGeneratorMock.getBaseUrl() +
            "/api/user/";
        requestGeneratorMock.RetrofitInstance(API.USER_API);
        assertEquals(expected,requestGeneratorMock.getCurrent_URL());
    }

    @Test
    public void FirstTrueSecondFalse(){
        String expected = requestGeneratorMock.getBaseUrl() +
            "/api/compilation/";
        requestGeneratorMock.RetrofitInstance(API.COMPILATION_API);
        assertEquals(expected,requestGeneratorMock.getCurrent_URL());
    }
}
```

```
}

@Test
public void FirstFalseSecondTrue(){
    requestGeneratorMock.setCURRENT_URL(requestGeneratorMock.getBaseUrl()
        + API.FOTO_ITINERARIO);
    String expected = requestGeneratorMock.getBaseUrl() +
        "/api/tappa/";
    requestGeneratorMock.RetrofitInstance(API.TAPPA_API);
    assertEquals(expected,
        requestGeneratorMock.getCurrent_URL());
}

@Test
public void FirstFlaseSecondFalse(){
    requestGeneratorMock.setCURRENT_URL(requestGeneratorMock.getBaseUrl()
        + API.TAPPA_API);
    String expected = requestGeneratorMock.getBaseUrl() +
        "/api/tappa/";
    requestGeneratorMock.RetrofitInstance(API.TAPPA_API);
    assertEquals(expected,
        requestGeneratorMock.getCurrent_URL());
}

@Test(expected = IllegalArgumentException.class)
public void routeIllegalArgumentException(){
    requestGeneratorMock.RetrofitInstance(null);
}
}
```

6.3 Verifica policy

```
/*
Class test for email and password policy, according with AWS a
password to be matched:
^(?=.*?[A-Z])(?=.*?[a-z])(?=.*?[0-9])(?=.*?[^#$%^&*-]).{8,16}$

Length between: [8-16]
One char: a-z;
One char: A-Z;
One number: 0-9;
One special char: #?!@$%^&*-;

An email is a valid email in standard regex:
[a-zA-Z0-9\+\.\_\%\-\+]{1,256}\@[a-zA-Z0-9][a-zA-Z0-9\-\-]{0,64}
(\.[a-zA-Z0-9 ][a-zA-Z0-9\-\-]{0,25}) 

You can try it here:
https://regexr.com/

Strategy used for testing:
Black Box - SECT
*/
public class PolicyTesting {

    LoginPolicyMock loginPolicyMock = new LoginPolicyMock();

    @Test
    public void emailValidPasswordValid() {
        String email = "support.natour21@libero.it";
        String password = "#Password123";

        assertTrue(loginPolicyMock.isValid(email,password));
    }

    @Test
    public void emailWithoutAtPasswordValid() {
        String email = "support.natour21";
        String password = "#Password123";

        assertFalse(loginPolicyMock.isValid(email,password));
    }

    @Test
    public void emailWithSpecialCharPasswordValid() {
```

```

        String email = "support.natour#21@livero.it";
        String password = "#Password123";

        assertFalse(loginPolicyMock.isValid(email,password));
    }

    @Test
    public void emailWithoutPrefixPasswordValid() {
        String email = "@libero.it";
        String password = "#Password123";

        assertFalse(loginPolicyMock.isValid(email,password));
    }

    @Test
    public void emailWithoutDomainPasswordValid() {
        String email = "support.natour21@";
        String password = "#Password123";

        assertFalse(loginPolicyMock.isValid(email,password));
    }

    @Test
    public void emailValidPasswordTooLong() {
        String email = "support.natour21@libero.it";
        String password = "#ThisPasswordIsToooooLong123";

        assertFalse(loginPolicyMock.isValid(email,password));
    }

    @Test
    public void emailValidPasswordTooShort() {
        String email = "support.natour21@libero.it";
        String password = "#Pa123";

        assertFalse(loginPolicyMock.isValid(email,password));
    }

    @Test
    public void emailValidPasswordWithoutAZ() {
        String email = "support.natour21@libero.it";
        String password = "#password123";

        assertFalse(loginPolicyMock.isValid(email,password));
    }

    @Test
    public void emailValidPasswordWithoutaz() {

```

```
String email = "support.natour21@libero.it";
String password = "#PASSWORD123";

assertFalse(loginPolicyMock.isValid(email,password));
}

@Test
public void emailValidPasswordWithoutNumber() {
    String email = "support.natour21@libero.it";
    String password = "#Password";

    assertFalse(loginPolicyMock.isValid(email,password));
}

@Test
public void emailValidPasswordWithoutSpecial() {
    String email = "support.natour21@libero.it";
    String password = "Password123";

    assertFalse(loginPolicyMock.isValid(email,password));
}

@Test (expected = IllegalArgumentException.class)
public void argumentNull() {
    loginPolicyMock.isValid(null, null);
}

@Test
public void allInvalid() {
    String email = "libero.it";
    String password = "notagoodpassword";

    assertFalse(loginPolicyMock.isValid(email,password));
}
}
```

7 Valutazione dell'usabilità sul campo

In questa ultima sezione analizzeremo l'usabilità sul campo con un prodotto semi-finito, qui analizzzeremo tecniche simili a quelle viste del paragrafo precedente riguardante l'usabilità a priori del sistema.

7.1 Metodo euristico

L'usabilità sul campo è stata valutata grazie a due operazioni adottate, la prima è quella delle valutazioni euristiche, la seconda è l'utilizzo dei file di log che verranno trattati in seguito.

In particolare per quanto riguarda la valutazione euristica siamo riusciti ad individuare 3 esperti a cui sottoporre sia qualche domanda, con particolare attenzione alle 8 regole d'oro di Shneiderman, sia il nostro prodotto, così da trovare dei difetti.

Le domande sottoposte ai valutatori sono le seguenti:

1. C'è un alto grado di usabilità universale?
2. Ci sono abbastanza riscontri informativi?
3. Gli errori sono facilmente attivabili? E se sì, c'è modo di annullarli?
4. Il carico di memoria a breve termine è basso?
5. L'applicazione è in grado di rispondere in tempi utili?
6. È all'altezza di altri competitors?

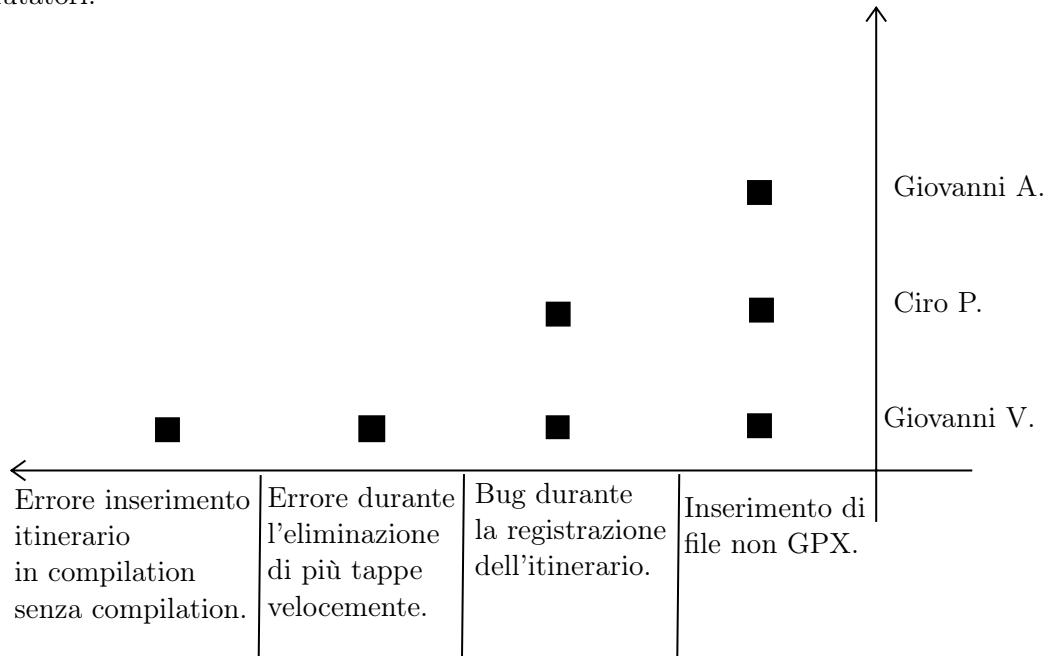
Mentre per quanto riguarda i compiti assegnati, durante l'utilizzo dell'app, abbiamo dato libertà ai nostri utilizzatori di esplorare l'app, abbiamo però dato 2 compiti generali da eseguire, ovvero:

- Compito 1: Pubblicare un itinerario in piattaforma tramite file GPX.
- Compito 2: Aggiungere un itinerario ad una compilation già esistente.

Successivamente alla valutazione euristica gli stessi utenti sono stati valutati per scoprire chi è stato l'utilizzatore più attento e chi meno.

I compiti assegnati sono stati eseguiti tutti e in tempi brevi dai nostri valutatori, quindi abbiamo una misura dell’usabilità abbastanza elevata che va comunque a centrare il nostro obiettivo iniziale, ovvero quello di avere un grado di usabilità sul campo molto vicino a quello dell’usabilità sui prototipi, e quindi evitando cambiamenti durante lo sviluppo dell’app che avrebbero portato ad una maggiore quantità di tempo e risorse impiegate e probabilmente abbassato il grado di usabilità del prodotto rispetto al prototipo falsando e invalidando quest’ultimo.

Di seguito viene mostrata la tabella che raffigura i difetti trovati dai nostri valutatori:



Come possiamo osservare, i difetti trovati sono principalmente 4, ed andremo a spiegarli nello specifico:

1. Inserimento di file non GPX: Questo errore si presentava quando, all'inserimento di un nuovo itinerario in piattaforma, l'utente poteva scegliere anche file che non erano di tipo GPX. Questo errore è stato risolto e adesso l'utente può scegliere soltanto file di tipo GPX.
 2. Bug durante la registrazione dell'itinerario: Se non veniva avviata nessuna registrazione, quando l'utente tornava alla Home Page l'app andava in crash.
 3. Errore durante l'eliminazione di più tappe velocemente: Se, durante la pubblicazione di un nuovo itinerario, l'utente inseriva manualmente delle tappe e poi le cancellava velocemente l'app andava in crash, questo errore è stato risolto disabilitando il tasto di eliminazione della

tappa fino a quando la tappa non veniva effettivamente eliminata dal sistema.

4. Errore inserimento itinerario in compilation senza compilation: Quando l'utente, all'interno della schermata di dettaglio di un itinerario, cliccava sul pulsante per inserire l'itinerario in una delle sue compilation, ma l'utente non aveva nessuna compilation, allora l'app andava in crash.

È possibile osservare che l'errore trovato più frequentemente è quello dell'inserimento di un itinerario tramite file GPX, questo grazie anche al compito che abbiamo dato ai nostri utilizzatori, infine l'utilizzatore più attento è stato Giovanni A.

Dopo aver risolto questi errori abbiamo riproposto l'app ai nostri utilizzatori che non hanno riscontrato altri errori.

Adesso vediamo come hanno risposto i nostri utilizzatori alle domande che gli abbiamo sottoposto.

Giovanni A.

1. Sì, l'app è abbastanza chiara ed esplicativa, un aiuto molto importante è stata la guida durante la pubblicazione di un nuovo itinerario.
2. Sì, le azioni vengono spesso accompagnate da schermate di loading.
3. Non ci sono tanti errori facilmente attivabili grazie ad una buona quantità di controlli, ma purtroppo quando viene effettuato un errore non è possibile risolverlo.
4. Sì, molto basso.
5. Sì, anche se non ha tempi molto veloci.
6. Con l'inserimento di nuove funzionalità potrebbe essere all'altezza dei competitors.

Ciro P.

1. Sì, molto esplicativa.
2. Sembra di sì.
3. Sì, facilmente attivabili.
4. Molto basso, non bisogna ricordare quasi nulla, anche durante la registrazione alla piattaforma non viene inviato un codice OTP ma un link di verifica che consente di non dover ricordare il codice.
5. Sì, ma non è molto veloce.
6. Probabilmente sì.

Giovanna A.

1. Si.
2. Abbastanza, soprattutto durante la pubblicazione di un nuovo itinerario.
3. Non sembra.
4. Molto basso.
5. Abbastanza.
6. Sì, ma sarebbe molto più competitiva se venissero introdotte le funzionalità di chat e preferiti.

Grazie a queste interviste possiamo ritenerci abbastanza soddisfatti del risultato in quanto: gli errori trovati sono stati risolti, non sono stati trovati altri errori ed infine le domande poste all'utente sono state abbastanza positive. Purtroppo l'applicazione non è molto veloce ma questo viene compensato con il grado di affidabilità che diamo all'utente, grazie ai controlli che vengono effettuati e alle conferme richieste durante le azioni più importanti fatte dall'utente.

7.2 Metodo con analisi di logging e monitoraggio

In questo ultimo paragrafo della documentazione mostriamo la raccolta di un breve periodo di testing applicando tecnologie di logging.

Le tecnologie utilizzate sono le seguenti (come citato in paragrafi precedenti):

- **Firebase Analytics;**
- **Crashlytics;**
- **Performance;**
- **Debug library di Android;**

Debug library di Android - per utilizzare un metodo facile e intuitivo di debuggare il codice android durante lo sviluppo abbiamo usufruito della libreria Log¹⁸, i log possono essere visualizzati all'interno del **Logcat** dell'applicativo, in seguito un esempio:

```
Log.i(TAG, msg: "Log informativo.");
Log.d(TAG, msg: "Log di debug.");
Log.w(TAG, msg: "Log di warning");
Log.e(TAG, msg: "Log di errore.");
Log.wtf(TAG, msg: "Log di Failure");
Log.v(TAG, msg: "Log di verbose.");
```

```
2022-03-10 11:09:18.847 12572-12572/com.example.natour21 I/LoginFragment: Log informativo.
2022-03-10 11:09:18.847 12572-12572/com.example.natour21 D/LoginFragment: Log di debug.
2022-03-10 11:09:18.847 12572-12572/com.example.natour21 W/LoginFragment: Log di warning
2022-03-10 11:09:18.847 12572-12572/com.example.natour21 E/LoginFragment: Log di errore.
2022-03-10 11:09:18.848 12572-12572/com.example.natour21 E/LoginFragment: Log di failure.
2022-03-10 11:09:18.852 12572-12572/com.example.natour21 V/LoginFragment: Log di verbose.
```

Figura 61: Debug Log Library

¹⁸<https://developer.android.com/reference/android/util/Log>

Crashlytics - è uno strumento di logging appartenente alla suite cloud di **Firebase**, crashlytics ci permette di tracciare e risolvere eventuali bug e diversi tipi di arresti anomali.

Attraverso crashlytics abbiamo potuto osservare che il 91.67% degli utenti che hanno provato l'app NON hanno riscontrato problemi, mentre 1 solo utente (che si è rivelato essere il più esperto in materia) ha riscontrato un arresto anomalo nella splashscreen che abbiamo prontamente risolto, in seguito i dettagli:



Figura 62: Crashlytics

Performance - questo è un altro modulo della suite cloud di **Firebase**, con performance è possibile inserire le metriche desiderate che si vogliono loggare nel sistema.

Insieme a Crashlytics, Performance è uno strumento utilissimo per il monitoraggio il beta-testing e release. Performance ci ha segnalato che con la risoluzione del bug in splashscreen l'applicazione adesso risulta più veloce del 7% ad avviarsi:

Tempo di avvio dell'app

Tempo di avvio dell'app (924 ms) è 7% più veloce rispetto a 1 giorno prima



Figura 63: Performance

Firebase Analytics - è uno dei moduli più importanti della suite cloud di Firbease (Google.inc), attraverso l'ausilio degli strumenti di base di Analytics possiamo analizzare varie informazioni utili ad una completa valutazione dell'usabilità nel campo.

Firebase analytics lavora in background monitorando eventi casuali o indotti dall'utente (tutto gestibile dalla console online di fire base).

Come prima analisi, vediamo il coinvolgimento per sessione (ed eventuali sessioni attive dell'utente) sul testing che abbiamo effettuato sul campo. In particolare abbiamo selezionato gli stessi utilizzatori scelti nella valutazione euristica e abbiamo chiesto loro di fare piccole sessioni di utilizzo dell'app liberamente (chiedendo loro pochi task).

Le sessioni sono durate circa una settimana, ogni utente ha utilizzato l'app per circa 1 ora al giorno e abbiamo monitorato il tutto traendo le seguenti conclusioni:



Figura 64: Analytics 1

Come si può notare dal grafico sovrastante, su 35 utenti (creati da gli stessi valutatori e anche utenti in beta-testing), le sessioni sono durate in media circa 22 minuti e 38 secondi con un numero di sessioni per utente di circa 4 (per la precisione 3.9).

Inoltre abbiamo visualizzato le schermate che gli utenti hanno utilizzato di più nelle sessioni, maggiormente e liberamente (le sessioni sono state, per avere dei dati quanto più veritieri, fatte in modo ASINCRONO, ovvero senza far sì che tutti gli utenti utilizzassero l'applicazione nello stesso momento).

Titolo pagina e classe schermata ▾		+	Visualizzazioni	Utenti	Nuovi utenti	Visualizzazioni per utente	Durata media de coinvolgimento
Totali		4.573	35	27	130,66	22 m 38 s	Uguale alla media
		100% del totale	100% del totale	100% del totale	Uguale alla media	Uguale alla media	
1	HomePageActivity	1.334	30	0	44,47	9 m 42 s	
2	MainActivity	1.232	35	0	35,20	5 m 45 s	
3	SplashScreen	959	29	0	33,07	1 m 18 s	
4	PubblicaitinerarioActivity	437	18	0	24,28	9 m 30 s	
5	DettagliitinerarioActivity	240	18	0	13,33	2 m 21 s	
6	RegistraPercorsoActivity	138	15	0	9,20	1 m 02 s	
7	CustomTabsManagerActivity	73	11	0	6,64	0 m 00 s	
8	CollezioneActivity	52	5	0	10,40	2 m 14 s	

Figura 65: Analytics 2

Dal grafico in figura 65 possiamo trarre delle conclusioni:

- Gli utenti hanno usufruito poco delle collezioni, magari ciò può essere direttamente sostituito da un modulo più semplice di preferiti oppure Like/Dislike.
- Tra i due metodi di pubblicazione di un itinerario abbiamo notato che gli utenti preferiscono di gran lunga pubblicare manualmente un itinerario invece di registralo.
- Come si può notare le activity "MainActivity" e "HomePageActivity" sono quelle più utilizzate (con numeri nettamente superiori a le altre), questo perchè esse sono radici delle gerarchie funzionali (vedesi paragrafo apposito) e quindi invocate per ogni singola azione.
- Le pubblicazioni di un itinerario sono il doppio delle visualizzazioni (DettaglioItinerarioActivity).

Alleghiamo anche il grafico di riferimento:

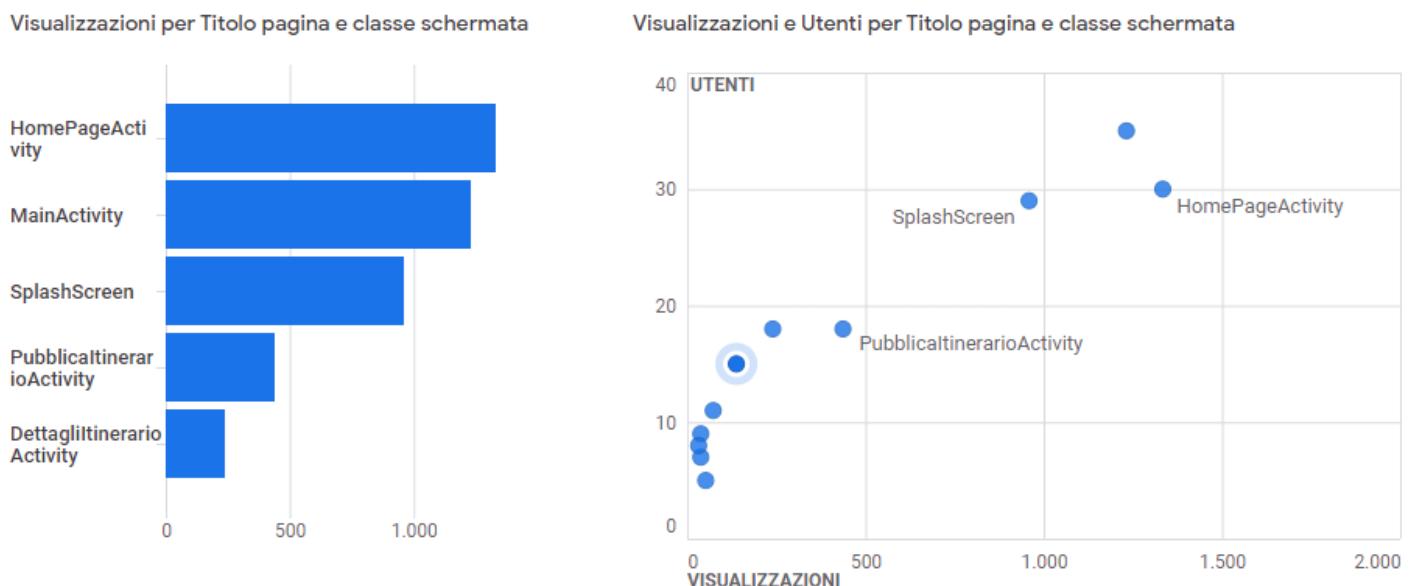


Figura 66: Analytics 3

Firebase ci ha poi offerto oltre a svariati tool di monitoraggio (quali per esempio quelli di "Country Control", per sapere in quali paesi l'app è arrivata) la visualizzazione di un grafico di **User stickiness** che ci permette di avere 3 dati fondamentali:

- DAU/MAU - (Daily Active User/Monthly Active User);
- DAU/WAU - (Daily Active User/Weekly Active User);
- WAU/MAU - (Weekly Active User/Monthly Active User);

User stickiness



Figura 67: Analytics 4

Riferimenti bibliografici

- [1] R. Polillo, *Facile da usare.* Idee e strumenti, Apogeo Education, 2010.
- [2] R. Martin, *Clean Code: Guida per diventare bravi artigiani nello sviluppo agile di software.* Maestri di programmazione, Feltrinelli Editore, 2018.
- [3] R. Martin, *Clean Architecture: Guida per diventare abili progettisti di architetture software.* Maestri di programmazione, Feltrinelli Editore, 2018.