



UNIVERSITÀ DEGLI STUDI DI NAPOLI
FEDERICO II

DIPARTIMENTO DI INGEGNERIA ELETTRICA
E DELLE TECNOLOGIE DELL'INFORMAZIONE

Specifiche, progettazione,
implementazione e validazione del
Sistema Informativo
NaTour21

Studenti

Bianca Giada CHEHADE N86003209
Mario LIGUORI N86003258
Mattia ROSSI N86003211

Docenti

Sergio DI MARTINO
Francesco CUTUGNO



Indice

1 Descrizione del progetto	3
1.1 Presentazione dell'idea progettuale	3
1.2 Individuazione del target degli utenti	4
1.3 Valutazione dell'usabilità a priori	5
1.3.1 Definizione delle Personas	5
1.3.2 Progettazione dell'esperienza utente	8
1.3.3 Definizione di uno stile di design	9
2 Documento dei Requisiti Software	10
2.1 Requisiti funzionali	10
2.1.1 Autenticazione	10
2.1.2 Interazione con un itinerario	11
2.1.3 Interazione con un post	12
2.1.4 Gestione dati personali e conversazioni private	13
2.1.5 Interazione con una compilation	13
2.1.6 Requisiti amministratore	14
2.2 Requisiti non funzionali	15
2.3 Requisiti di dominio	17
2.4 Modellazione dei Casi d'Uso	18
2.4.1 Autenticazione	19
2.4.2 Interazione con un itinerario	20
2.4.3 Interazione con un post	21
2.4.4 Gestione profilo	22
2.4.5 Gestione messaggi	23
2.4.6 Funzionalità amministratore	24
2.5 Tabelle di Cockburn dei casi d'uso	25
2.5.1 Inserisce un itinerario	25
2.5.2 Segnala un itinerario	28
2.6 Mock-Up dell'applicazione	29
2.7 Prototipazione funzionale via statechart dell'interfaccia grafica	30
2.7.1 Segnalazione itinerario	30
2.7.2 Inserimento itinerario	31
2.7.3 Inserimento dettagli itinerario	32
2.7.4 Inserimento mappa itinerario	33
2.7.5 Inserimento foto itinerario	34
2.8 Classi, oggetti e relazioni di analisi	35
2.9 Diagrammi di sequenza di analisi	36
2.9.1 Ricerca itinerario	37
2.10 Diagrammi di attività	38
2.10.1 Autenticazione	38
2.10.2 Interazione con un itinerario	42
2.10.3 Interazione con un post	48
2.10.4 Interazione con una compilation	51
2.10.5 Gestione profilo e interazione con utenti	54
2.10.6 Funzionalità riservate agli amministratori	58

3 Documento di Design del sistema	60
3.1 Analisi dell'architettura e criteri di design	60
3.1.1 Diagramma di design del sistema	60
3.1.2 Software deployment	61
3.1.3 Google Maps Platform	62
3.1.4 Architettura del REST Service	62
3.2 Architettura dell'applicativo	65
3.3 Le scelte implementative	68
3.3.1 Applicazioni mobile native e ibride	68
3.3.2 Perchè Kotlin	68
3.3.3 Diagramma di design dell'applicativo	69
3.3.4 Pattern architettonicale utilizzato	70
3.4 Diagramma delle classi di design	71
3.4.1 Autenticazione	71
3.4.2 Interazione con una compilation	72
3.4.3 Aggiunta di un itinerario	73
3.5 Diagrammi di sequenza di design	74
3.6 Definizione delle gerarchie funzionali	75
4 Definizione di un piano di testing e valutazione sul campo dell'usabilità	76
4.1 Valutazione dell'usabilità	76

1 Descrizione del progetto

NaTour21 è un sistema complesso e distribuito finalizzato ad offrire un moderno social network multipiattaforma per appassionati di escursioni.

Il sistema consiste in:

- un Back-End sicuro, performante e scalabile;
- un client mobile attraverso cui gli utenti possono fruire delle funzionalità del sistema in modo intuitivo, rapido e piacevole;
- un client mobile attraverso cui gli amministratori possono gestire i contenuti inseriti in piattaforma.

NaTour21 ha lo scopo di creare una community sicura e affidabile dove condividere la propria passione per l'escursionismo.

In questo scenario, l'utente si configura come protagonista: oltre alla possibilità di inserire itinerari (dettagliati da informazioni), compilation e post personali sulla piattaforma, è lasciato ampio spazio all'individualità personale.

Tutto ciò si concretizza con la possibilità di interagire con gli altri utenti, in modo da poter avere un contatto più diretto con la realtà dell'escursionismo, e di lasciare valutazioni personali su qualunque itinerario si desideri.

Il sistema valuta essenziale la sicurezza degli utenti: questi potranno segnalare informazioni inesatte o contenuti inappropriati al fine di rendere la permanenza nella community piacevole per tutti.

1.1 Presentazione dell'idea progettuale

L'applicazione "NaTour21" nasce in seguito all'esigenza di uno spazio virtuale dove poter condividere la passione per l'escursionismo.

Creare una community di escursionisti, esperti o meno, ha lo scopo di rendere le esperienze individuali degli utenti più sicure e informate, oltre a promuovere la condivisione di contenuti personali.

NaTour21 mette a disposizione degli utenti registrati diverse funzionalità.

Il lato prettamente informativo consente di ricercare - nonchè inserire in piattaforma - diversi itinerari, e tutte le informazioni relative ad essi. Ciò comprende la possibilità di visualizzarli su mappa, e, per garantire un'accuratezza maggiore delle informazioni, di lasciare un feedback personale per ciascuno di essi.

Ci sono, però, ulteriori funzionalità che coinvolgono direttamente l'utente: tra queste la condivisione di post, la creazione di compilation personalizzate e la visualizzazione di una homepage con post riguardanti diversi itinerari.

Particolare attenzione è riservata alla sicurezza degli utenti, i quali possono segnalare contenuti inappropriati o informazioni inesatte.

1.2 Individuazione del target degli utenti

NaTour21 si configura come community in cui chiunque può intraprendere la passione per l'escursionismo e condividerla.

Ciononostante, è stato possibile individuare un target di utenti ben definito grazie ai dati statistici¹ raccolti nell'anno 2020 dall'ISTAT, Istituto Nazionale di Statistica.

In seguito all'analisi delle informazioni statistiche sull'escursionismo italiano è stato individuato come target utenti la popolazione, caratterizzata da un numero di circa pari entità di uomini e di donne, nella fascia di età 35-64. La maggior parte della platea considerata risulta essere appartenente alla condizione lavorativa di *dirigente, quadro o impiegato*.



L'Istat ha inoltre conteggiato nell'anno 2020 circa 41194 escursioni giornaliere, di cui il 42.7% situate temporalmente nel terzo trimestre dell'anno. Da questa informazione si deduce che il periodo preferito per attività escursionistiche sia quello dei mesi estivi.

Il motivo prevalente per queste escursioni risulta essere stato *piacere personale, svago e tempo libero*, con un numero di escursioni così orientate pari a 23223: questo dato risulta particolarmente significativo, pari a un ordine di grandezza in più rispetto agli altri motivi registrati. È importante anche notare quanto il numero di escursioni in paesi esteri sia di gran lunga minore rispetto a quelle nelle regioni italiane.

Le regioni del Nord Italia sembrano essere state le preferite per visite giornaliere (la più visitata è stata il Veneto, con una media di 6200), seguite immediatamente da quelle del centro-Sud. Il Molise, invece, non risulta aver raggiunto la minima cifra considerata nella raccolta dati.

¹Fonte: [sito ufficiale Istat](#).

1.3 Valutazione dell'usabilità a priori

1.3.1 Definizione delle Personas

Le *Personas* sono veri e propri identikit, creati sulla base di ricerche, che rappresentano un particolare gruppo di potenziali clienti. Delineare le caratteristiche di questi clienti permette di conoscere e comprendere bisogni, paure, obiettivi e motivazioni degli utenti. Saranno questi identikit a consentire lo sviluppo di una **UX** piacevole, funzionale ed elegante.

Liam Kumar



"Il successo non è finale come il fallimento non è fatale: è il coraggio di continuare che conta."

Età: 23
Lavoro: Consulente finanziario
Status: Single
Residenza: Dubai

Bio

Al momento lavoro come consulente finanziario presso un'azienda a Dubai dove sono anche residente. Per me tenersi in forma è una priorità ed adoro fare escursioni, soprattutto con persone con più esperienza.

Personalità

Introversione	Estroversione
Pensiero	Sentimento
Sensazione	Intuizione
Giudizio	Percezione

Obiettivi

- Essere sempre aggiornato sulle novità del mondo finanziario
- Scrivere una biografia
- Essere sempre al massimo della forma

Frustrazioni

- Saltare gli allenamenti quotidiani
- Rimanere indietro rispetto la massa

Motivazione

Paura	
Crescita	Potere
Potere	Socialità
Socialità	

Marchi & Influenze





Competenze

Tecnologiche	
Sociali	Apprendimento
Apprendimento	Varie
Varie	

5

Juanna Cheng



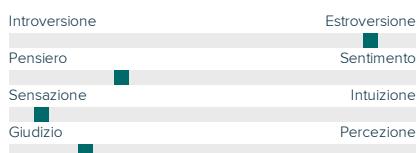
"Quando hai un sogno, lasciarlo andare è il più grande errore tu possa fare."

Età: 25
Lavoro: Capo reparto
Status: Single
Residenza: Milano

Bio

Correntemente vivo a Milano. Sono attualmente vice-leader di un reparto presso l'azienda nella quale lavoro. Amo molto fare lunghe passeggiate nei weekend sia da sola che in compagnia.

Personalità



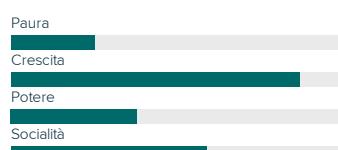
Obiettivi

- Bisogno di trovare persone con simili abilità per migliorarsi costantemente
- Diventare un modello da seguire
- Aumentare il suo bagaglio culturale

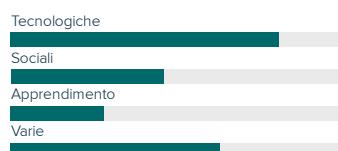
Frustrazioni

- Non essere in grado di rispettare delle scadenze
- Mancanza di confronto

Motivazione



Competenze



Alfredo Bianchi



"La cattiva notizia è che il tempo vola, la buona è che tu sei il pilota."

Età: 29

Lavoro: Consulente di viaggi

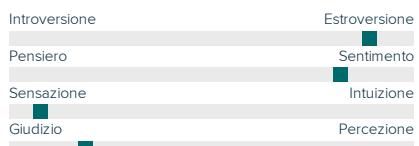
Status: Fidanzato

Residenza: New York

Bio

Vivo a New York con la mia compagna. Da 3 anni lavoro presso un'agenzia come consulente di viaggi. Dovendo consigliare dove viaggiare, spesso mi porto in tali destinazioni.

Personalità



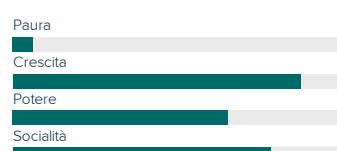
Obiettivi

- Aprire la sua agenzia di viaggi
- Toccare ogni continente
- Creare una famiglia

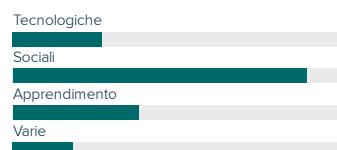
Frustrazioni

- Non esser preso in considerazione
- Doversi impostare dei limiti

Motivazione



Competenze



1.3.2 Progettazione dell'esperienza utente

Attraverso una fase di **Brainstorming** sono state generate idee per sfruttare le opportunità e risolvere i problemi emersi durante la creazione delle Personas. Sono state raccolte idee e i suggerimenti dei componenti del team di progettazione, raccogliendo anche le proposte più banali, così da valutare tutte le possibilità. Una volta raccolte le proposte, è stato deciso di realizzare dei prototipi dinamici con un tool di **CASE**, nello specifico, *Figma*.

Periodicamente ogni prototipo è stato discusso e creato coinvolgendo tutti i membri, svolgendo le seguenti analisi:

- Facilità di navigazione;
- Tempo di reperimento dei contenuti;
- Coerenza nella creazione di nuovi contenuti.

Progettare l'interazione tra utenti e prodotto è una materia complicata e spesso anche dispendiosa. Esiste però un modo, molto utilizzato in **UX Design**, per andare a ridurre al minimo possibile questi test utente: la valutazione euristica.

Le Euristiche di Nielsen sono il cardine dell'esperienza utente realizzata.

È stata posta particolare attenzione allo *stato del sistema*, specialmente agli stati di caricamento e di errore, dando eventualmente suggerimenti e feedback all'utente.

Nella maggior parte dei casi gli errori sono stati prevenuti, non è possibile procedere con una determinata operazione se non si soddisfano dei requisiti.

Le funzionalità più comuni sono rese disponibili sotto nomi ben noti, creando una corrispondenza tra il sistema e il mondo reale.

L'interfaccia non presenta limitazioni, non viene mai imposto all'utente di effettuare una determinata operazione non desiderata, eccezione fatta per operazioni *distruttive*, che richiedono una conferma. Tale interfaccia è estremamente minimalista, i contenuti più importanti sono posti in rilievo attraverso sfumature, colori, elevazione e posizionamento strategico delle informazioni.

L'ambiente è totalmente isolato, consistente, l'interfaccia non attraversa mai cambi di layout estremi, migliorando la velocità d'apprendimento degli utenti.

Prima di raffinare l'esperienza utente è stato necessario coinvolgere un ristretto bacino d'utenza per ottenere riscontri esterni rispetto al contesto di prototipazione e sviluppo.

1.3.3 Definizione di uno stile di design

Per raffinare l'esperienza utente è stata ritenuta una buona pratica affidarsi ad uno specifico **UI Design System**. La scelta è stata effettuata tra diversi sistemi concorrenti, ricadendo su **Material**².



Material Design si ispira al mondo fisico, includendo riflessi e ombre. Tipografie, griglie, spazi e colori creano gerarchie finalizzate a rendere l'utilizzatore totalmente immerso. Gli effetti di movimento mirano a focalizzare l'attenzione generando continuità spaziale attraverso feedback e transizioni. Le componenti sono altamente dinamiche, si trasformano e organizzano l'ambiente in base alle interazioni, generando altre trasformazioni. Le componenti Material sono blocchi interattivi che formano l'interfaccia utente. Queste includono stati built-in system per comunicare focus, selezione, attivazione, errore, hovering, pressione, trascinamento, e disabilitazione.

Ogni componente copre delle necessità, tra queste troviamo:

- **Esposizione** - Piazzare ed organizzare contenuti correlati usando componenti come *card* e *liste*;
- **Navigazione** - Consentire agli utenti di muoversi nel prodotto finito utilizzando *drawer* o *tabs*;
- **Azioni** - Consentire agli utenti di eseguire task utilizzando componenti di alto rilievo, come i *floating action button*;
- **Input** - Consentire agli utenti di inserire informazioni o selezionare tra un certo numero di opzioni, questo attraverso *campi di testo* dinamici e *chips*;
- **Comunicazione** - Avvertire gli utenti attraverso *snackbar*, *toast* e *dialog*.

²Material è un design system creato da Google per facilitare lo sviluppo di UX di altissima qualità per Android, iOS, Flutter, e Web-App.

2 Documento dei Requisiti Software

2.1 Requisiti funzionali

In questa sezione saranno esposti i requisiti *funzionali* dell'applicazione NaTour21, cioè le funzionalità richieste dai commissionanti.

2.1.1 Autenticazione

Nome	Descrizione
Registrazione	Il sistema deve consentire ad un utente di potersi registrare indicando e-mail e password, oppure utilizzando account di terze parti (Google o Facebook).

Tabella 1: RF.1

Nome	Descrizione
Accesso	Il sistema deve consentire ad un utente di poter effettuare l'accesso alla piattaforma avendo eseguito precedentemente una registrazione.

Tabella 2: RF.2

Nome	Descrizione
Reimpostazione password	Il sistema deve consentire ad un utente registrato di poter reimpostare la propria password se dimenticata o persa.

Tabella 3: RF.3

Nome	Descrizione
Uscita dall'account	Il sistema deve consentire ad un utente registrato di poter uscire dall'account con il quale ha eseguito l'accesso.

Tabella 4: RF.4

2.1.2 Interazione con un itinerario

Nome	Descrizione
Visualizzazione itinerari	Il sistema deve consentire a un utente autenticato di visualizzare i dettagli degli itinerari pubblicati e i post ad essi associati.

Tabella 5: RF.5

Nome	Descrizione
Inserimento itinerario	Il sistema deve consentire a un utente autenticato di inserire nuovi itinerari (sentieri) in piattaforma. Un sentiero è caratterizzato da un nome, una durata (max 16 ore), un livello di difficoltà, un punto di inizio e di fine, delle foto (max 5), una descrizione (opzionale), e un tracciato geografico (opzionale) che lo rappresenta su una mappa. Il tracciato geografico può essere inseribile manualmente (interagendo con una mappa interattiva) oppure tramite file in formato standard GPX.

Tabella 6: RF.6

Nome	Descrizione
Ricerca itinerari	Il sistema deve consentire a un utente autenticato di effettuare ricerche di itinerari tra quelli presenti in piattaforma, con possibilità di filtrare i risultati per area geografica (compresa in un raggio espresso in km scelto dall'utente), per livello di difficoltà, per durata e per accessibilità ai disabili.

Tabella 7: RF.7

Nome	Descrizione
Valutazione itinerario	Il sistema deve consentire a un utente autenticato di indicare un punteggio di difficoltà e/o un tempo di percorrenza diverso da quello indicato dall'utente che ha inserito il sentiero. In questo caso, il punteggio di difficoltà e il tempo di percorrenza per il sentiero saranno ri-calcolati come la media delle difficoltà e dei tempi indicati.

Tabella 8: RF.8

Nome	Descrizione
Salvataggio itinerario	Il sistema deve consentire a un utente autenticato di salvare gli itinerari nelle proprie compilation personalizzate.

Tabella 9: RF.9

Nome	Descrizione
Segnalazione itinerario	Il sistema deve consentire a un utente autenticato di segnalare degli itinerari le quali informazioni potrebbero non essere corrette e/o aggiornate.

Tabella 10: RF.10

Nome	Descrizione
Rimozione itinerario	Il sistema deve consentire a un utente autenticato di eliminare itinerari di cui è l'autore.

Tabella 11: RF.11

2.1.3 Interazione con un post

Nome	Descrizione
Visualizzazione post	Il sistema deve consentire a un utente autenticato di visualizzare i dettagli di un post.

Tabella 12: RF.12

Nome	Descrizione
Inserimento post	Il sistema deve consentire a un utente autenticato di inserire un post associato ad un itinerario, caratterizzato da foto (max 5) e una descrizione (opzionale).

Tabella 13: RF.13

Nome	Descrizione
Segnalazione post	Il sistema deve consentire a un utente autenticato di segnalare post con fotografie inappropriate.

Tabella 14: RF.14

Nome	Descrizione
Rimozione post	Il sistema deve consentire a un utente autenticato di eliminare post di cui è l'autore.

Tabella 15: RF.15

2.1.4 Gestione dati personali e conversazioni private

Nome	Descrizione
Gestione profilo	Il sistema deve consentire a un utente autenticato di gestire il suo profilo, egli potrà visitare i contenuti inseriti ed eliminarli. Inoltre potrà cambiare la foto profilo.

Tabella 16: RF.16

Nome	Descrizione
Gestione conversazioni private	Il sistema deve permettere all'utente autenticato lo scambio di informazioni con altri utenti. È possibile avviare la conversazione a partire dai contenuti pubblicati. Inoltre è possibile ricercare un destinatario.

Tabella 17: RF.17

2.1.5 Interazione con una compilation

Nome	Descrizione
Creazione compilation	Il sistema deve consentire a un utente autenticato di creare delle <i>compilation personalizzate</i> , caratterizzate da un titolo, una foto e da una descrizione.

Tabella 18: RF.18

Nome	Descrizione
Rimozione compilation	Il sistema deve consentire a un utente autenticato di eliminare le proprie compilation personalizzate

Tabella 19: RF.19

Nome	Descrizione
Visualizzazione compilation	Il sistema deve consentire a un utente autenticato di visualizzare i dettagli delle proprie compilation.

Tabella 20: RF.20

Nome	Descrizione
Rimozione itinerario da compilation	Il sistema deve consentire a un utente autenticato di rimuovere gli itinerari dalle proprie compilation personalizzate.

Tabella 21: RF.21

2.1.6 Requisiti amministratore

Nome	Descrizione
Visualizzazione segnalazioni	Il sistema deve permettere all'utente autenticato con i privilegi di amministratore di visualizzare la lista di segnalazioni inviate.

Tabella 22: RF.22

Nome	Descrizione
Amministrazione itinerario	Il sistema deve permettere all'utente autenticato con i privilegi di amministratore di modificare o rimuovere un itinerario segnalato.

Tabella 23: RF.23

2.2 Requisiti non funzionali

Questa sezione conterrà invece i requisiti *non funzionali* dell'applicazione, ovvero tutti i vincoli sui servizi offerti dal sistema .

Nome	Descrizione
Back-End scalabile	Il sistema deve essere scalabile e performante rispetto ai cambiamenti del Back-End, in modo da garantire un'agile manutenibilità nel tempo.

Tabella 24: RNF1

Nome	Descrizione
Prestazioni	Il sistema deve essere operativo entro 3 secondi dall'avvio e, inoltre, non deve ostacolare in alcun modo gli input dell'utente (e.g. presentando dei rallentamenti).

Tabella 25: RNF2

Nome	Descrizione
Reattività	Il sistema dovrebbe essere responsivo, sia nei confronti degli input da parte dell'utente, sia in caso di interruzioni esterne con alta priorità (e.g. chiamate in arrivo). Considerando il caso delle interruzioni: il sistema deve essere in grado di salvare il proprio stato e ripresentarlo all'utente.

Tabella 26: RNF3

Nome	Descrizione
Usabilità	L'applicazione dovrebbe presentare una UX semplice, che renda il flusso delle operazioni comprensibile, anche ad utenti meno esperti.

Tabella 27: RNF4

Nome	Descrizione
Usabilità	L'applicazione dovrebbe presentare una UX semplice, che renda il flusso delle operazioni comprensibile, anche ad utenti meno esperti.

Tabella 28: RNF4

Nome	Descrizione
Sicurezza	Il sistema deve essere sicuro, con dati criptati e protetti da attacchi esterni. Nel caso dell'autenticazione per i client, verrà salvato un token (nello specifico JWT) sul dispositivo.

Tabella 29: RNF5

Nome	Descrizione
Sicurezza password	Il sistema non consente di inserire password che non contengano almeno 8 caratteri.

Tabella 30: RNF6

Nome	Descrizione
Numero massimo di foto	Il sistema non consente di inserire più di 5 foto per itinerario o post.

Tabella 31: RNF7

2.3 Requisiti di dominio

Infine, questa sezione conterrà i requisiti *di dominio* dell'applicazione, ovvero tutti i vincoli generali a cui l'applicativo deve attenersi.

Nome	Descrizione
ISO/IEC 27018:2019	Il sistema deve essere conforme allo standard ISO/IEC 27018:2019, incentrato sulla protezione dei dati personali nel cloud.

Tabella 32: RD1

Nome	Descrizione
GDPR	Il sistema deve essere conforme al GDPR (Regolamento Generale sulla Protezione dei Dati), incentrato sul trattamento dei dati personali e sulla privacy dell'utente.

Tabella 33: RD2

2.4 Modellazione dei Casi d'Uso

In questa sezione sarà riportato il diagramma Use Case per l'applicazione.

Per garantire una maggiore leggibilità e data la molteplicità di funzioni garantite dal sistema, si è fatta la scelta di raggruppare le funzioni logicamente legate tra loro in packages.

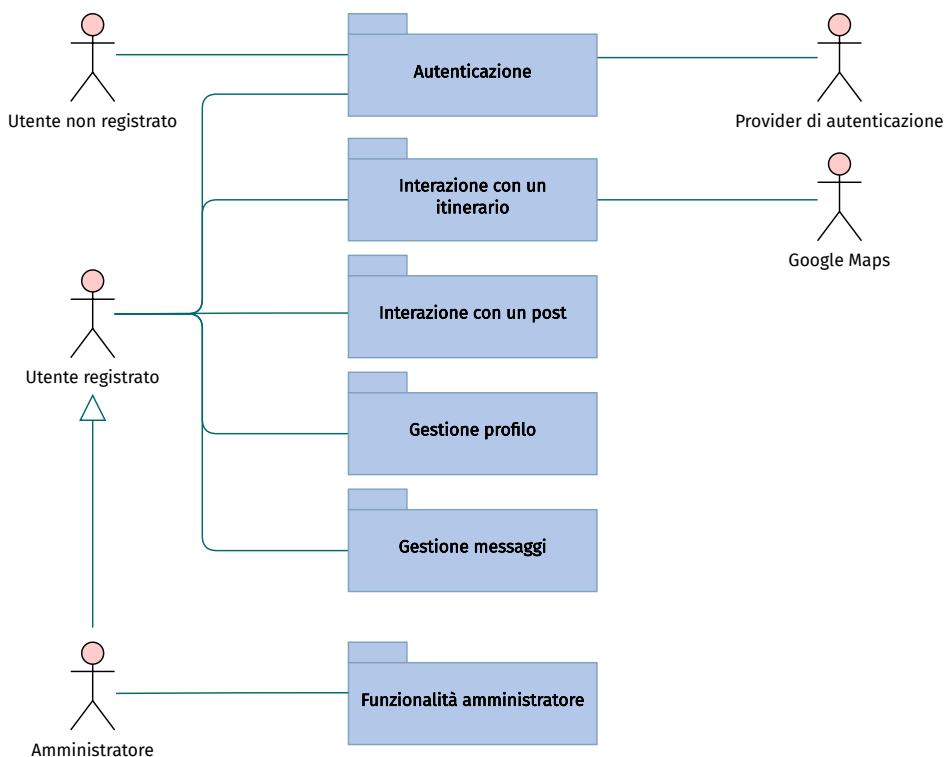


Figura 1: Use Case Diagram

2.4.1 Autenticazione

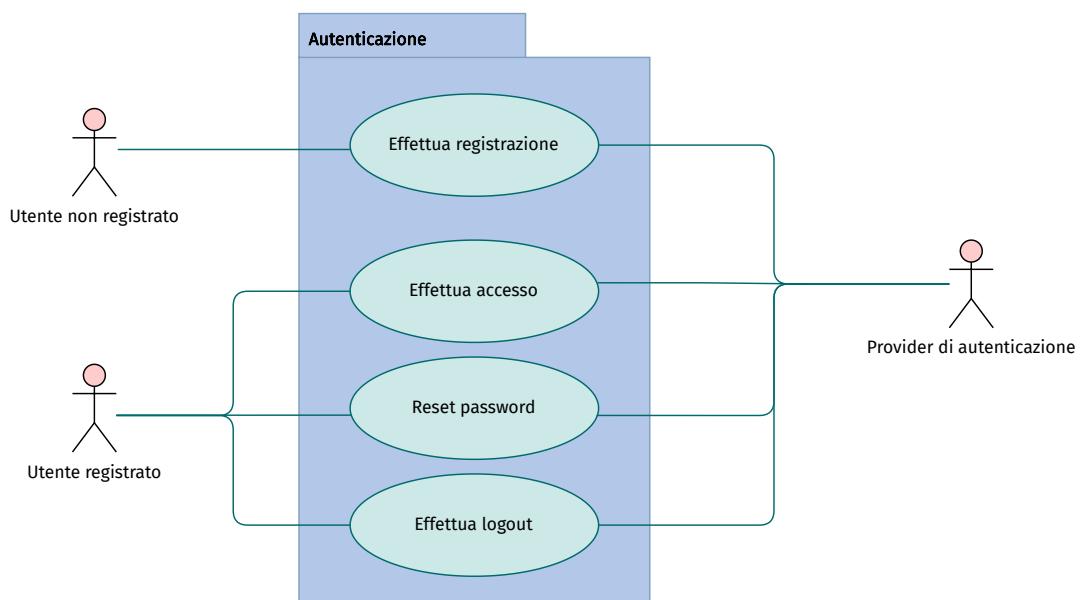


Figura 2: Package 1 - Autenticazione

2.4.2 Interazione con un itinerario

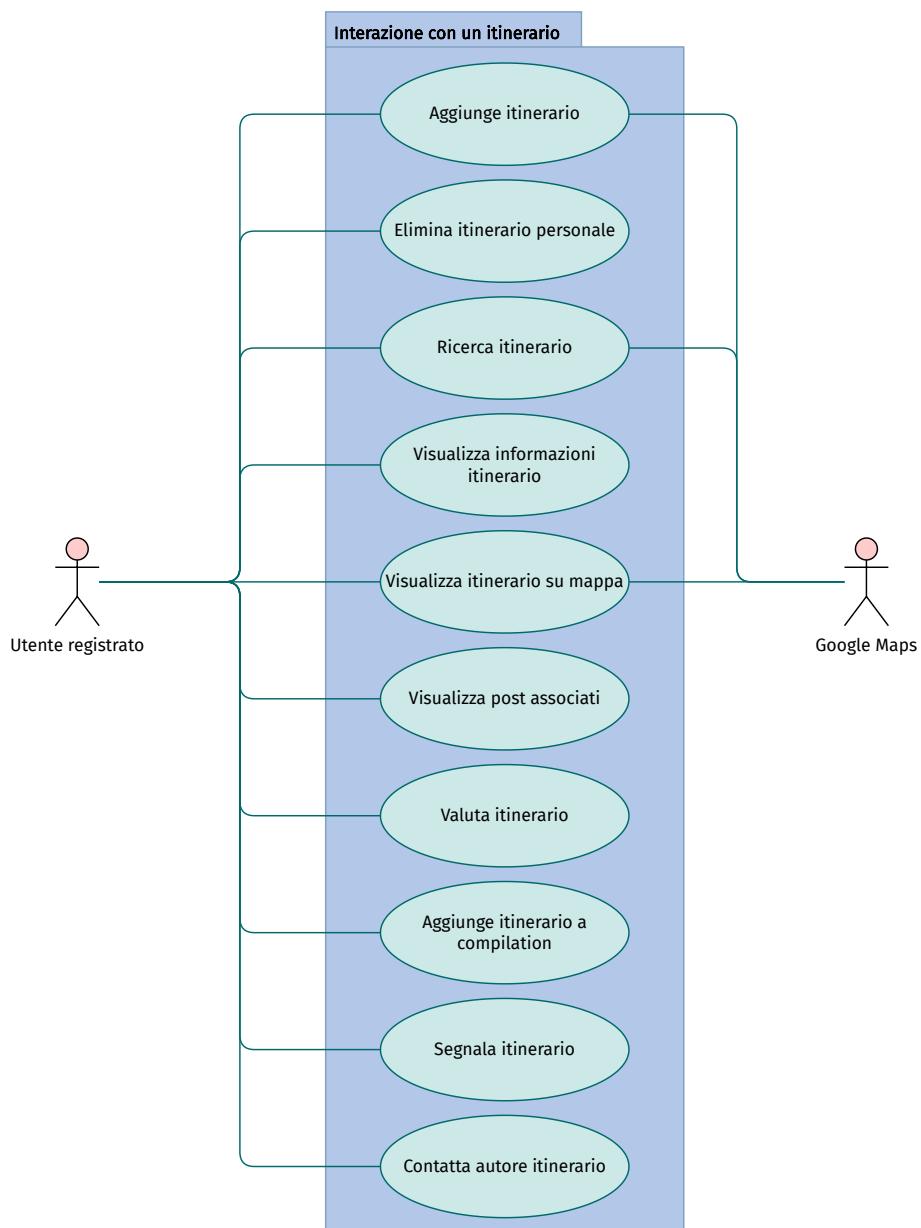


Figura 3: Package 2 - Interazione con un itinerario

2.4.3 Interazione con un post

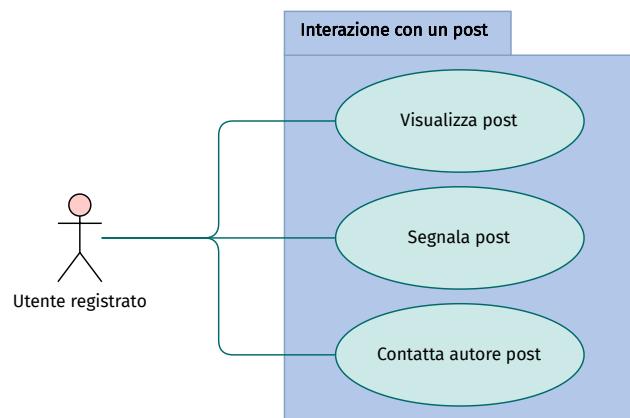


Figura 4: Package 3 - Interazione con un post

2.4.4 Gestione profilo

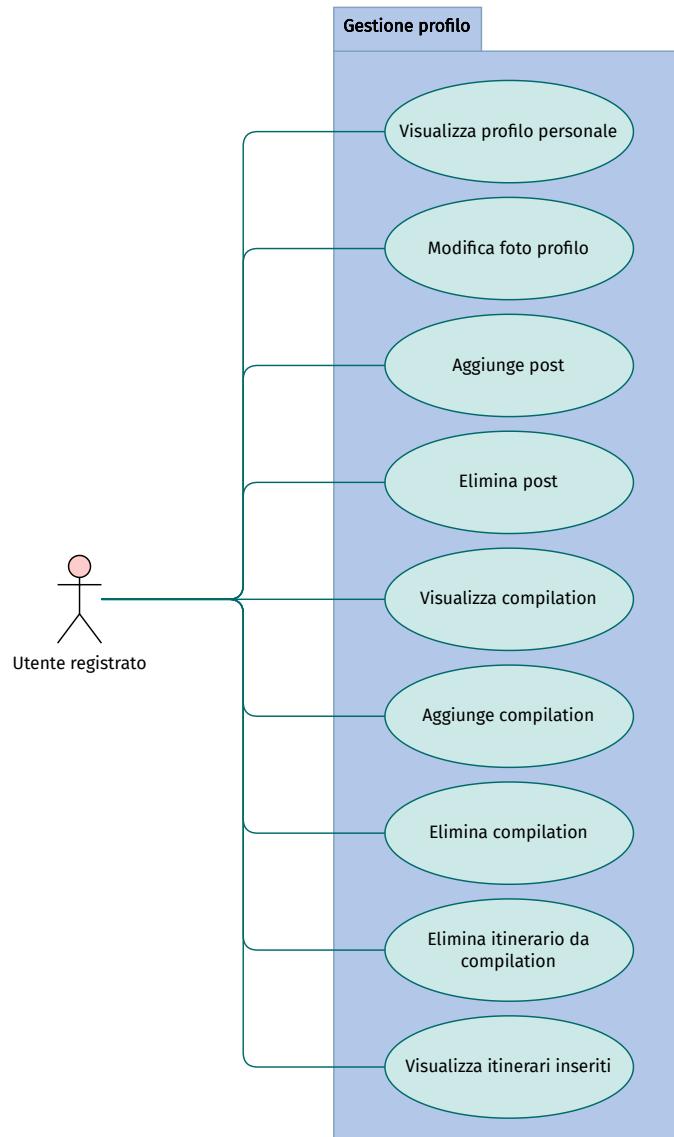


Figura 5: Package 4 - Gestione profilo

2.4.5 Gestione messaggi

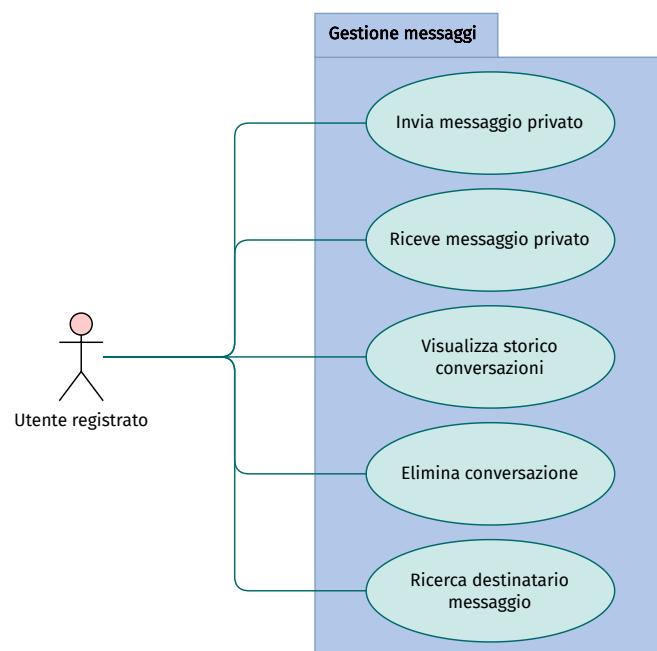


Figura 6: Package 5 - Gestione messaggi

2.4.6 Funzionalità amministratore

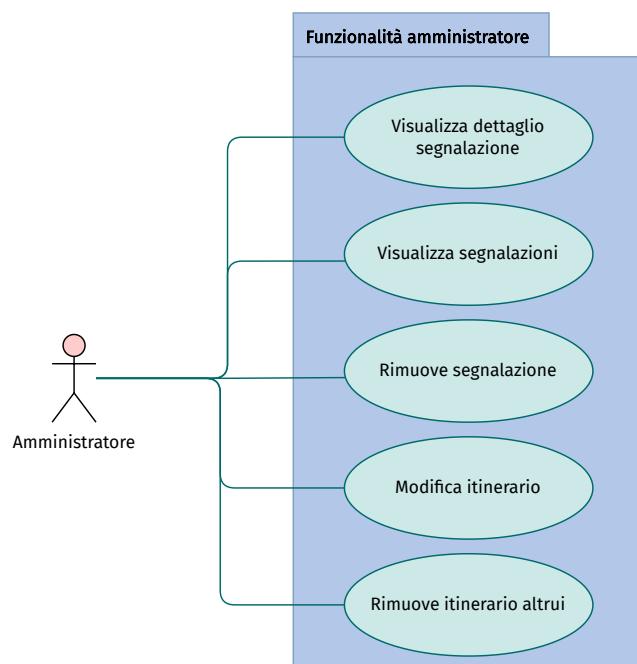


Figura 7: Package 6 - Funzionalità amministratore

2.5 Tabelle di Cockburn dei casi d'uso

Sono presentate in questa sezione le tabelle di Cockburn relative a due casi d'uso significativi dell'UCD.

2.5.1 Inserisce un itinerario

USE CASE #1	Inserisce un itinerario			
Goal in Context	L'utente vuole inserire un itinerario in piattaforma			
Preconditions	L'utente è autenticato			
Success End Conditions	L'utente riesce ad inserire il nuovo itinerario			
Failed End Conditions	L'utente non riesce ad inserire il nuovo itinerario			
Primary Actor	Utente registrato			
Secondary Actor	Google Maps			
Trigger	L'utente preme il Floating Action Button “newRouteFab” nella schermata “RoutesUI”			
	Step	Utente registrato	Sistema	Google Maps
	1	Preme il Floating Action Button “newRouteFab” nella schermata “RoutesUI”		
	2		Mostra la schermata “CreateRouteInfoUI”	
	3	Inserisce il nome dell'itinerario, (optionalmente) la descrizione, la durata approssimativa, il livello di difficoltà e la possibilità di accessibilità per disabili		
	4	Preme il Floating Action Button “nextFab”		
	5		Mostra la schermata “NewRoute-MapUI”	

	6	Preme sull'icona di ricerca		
	7	Inserisce la posizione della prima tappa del percorso per visualizzarla sulla mappa		
	8			Mostra la posizione desiderata sulla mappa
	9	Seleziona le tappe del percorso tramite la mappa interattiva		Recupera la posizione dei punti selezionati sulla mappa e traccia i percorsi tra essi
	10	Preme il Floating Action Button “nextFab”		
	11		Mostra la schermata “NewRoute-PhotosUI”	
	12	Preme il bottone insertPhotoButton		
	13		Mostra il photo picker per la selezione delle foto	
	14	Seleziona un numero di foto compreso tra 1 e 5		
	15	Preme il bottone “Fatto”		
	16		Mostra la schermata “NewRoute-PhotosUI”	

	17	Preme il Floating Action Button “confirmRouteFab”		
EXTENSION #1 Inserimento mappa con file GPX	Step	Utente registrato	Sistema	Google Maps
	9.1	Apre il menu a tendina in alto a destra della schermata e seleziona “Importa GPX”		
	10.1		Mostra la schermata per la selezione di un file GPX	
	11.1	Seleziona il file GPX		
	12.1		Mostra la schermata "NewRouteMapUI" aggiornata	

2.5.2 Segnala un itinerario

USE CASE #2	Segnala un itinerario		
Goal in Context	L'utente vuole segnalare un itinerario per informazioni inesatte o non accurate		
Preconditions	L'utente è autenticato		
Success End Conditions	L'utente riesce a segnalare l'itinerario correttamente		
Failed End Conditions	L'utente non riesce a segnalare l'itinerario		
Primary Actor	Utente registrato		
Trigger	L'utente preme sul pulsante di segnalazione nella schermata di dettaglio di un itinerario		
	Step	Utente registrato	Sistema
	1	Preme il pulsante di segnalazione nella schermata “ReportRouteUI”	
	2		Mostra la full dialog “ReportRoute-FullDialog”
	3	Inserisce un nome e una descrizione per la segnalazione	
	4	Preme il bottone “Fatto”	
	5		Mostra la schermata “RouteUI” e una snackbar di segnalazione andata a buon fine, aggiungendo all'itinerario un warning (se non è già presente) per possibili informazioni inesatte

2.6 Mock-Up dell'applicazione

Vengono ora mostrati i Mock-Up dei casi d'uso significativi dettagliati nella sottosezione precedente.

Sarà specificato per ogni Mock-Up il tipo e la funzionalità di ogni componente.

2.7 Prototipazione funzionale via statechart dell'interfaccia grafica

2.7.1 Segnalazione itinerario

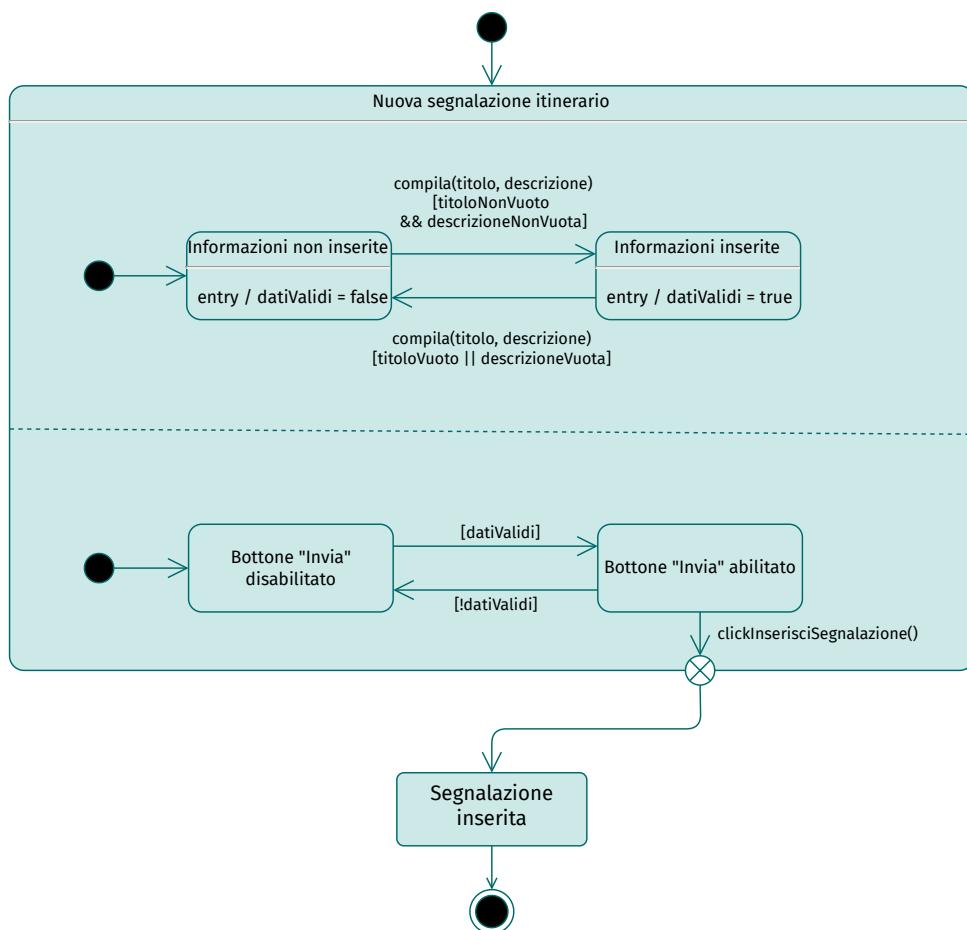


Figura 8: Nuova segnalazione itinerario

2.7.2 Inserimento itinerario

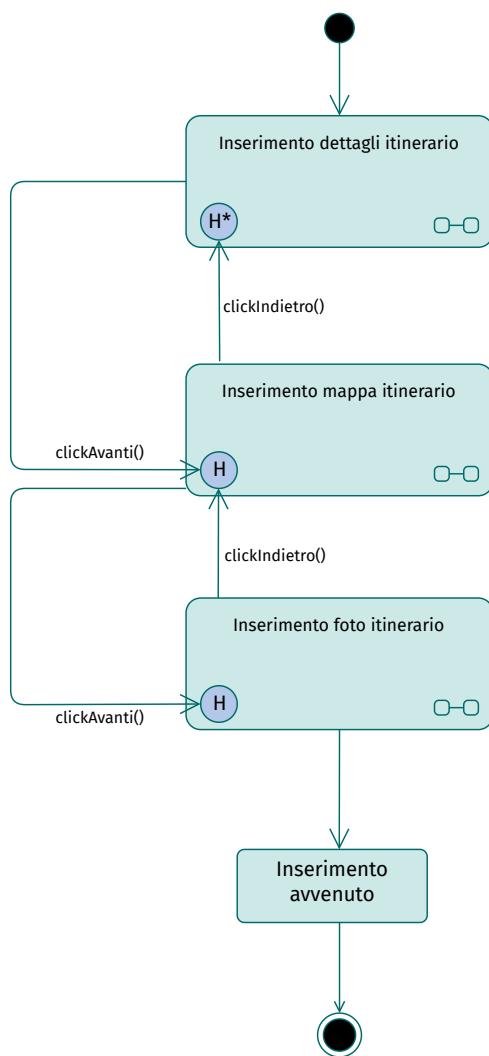


Figura 9: Nuovo itinerario

2.7.3 Inserimento dettagli itinerario

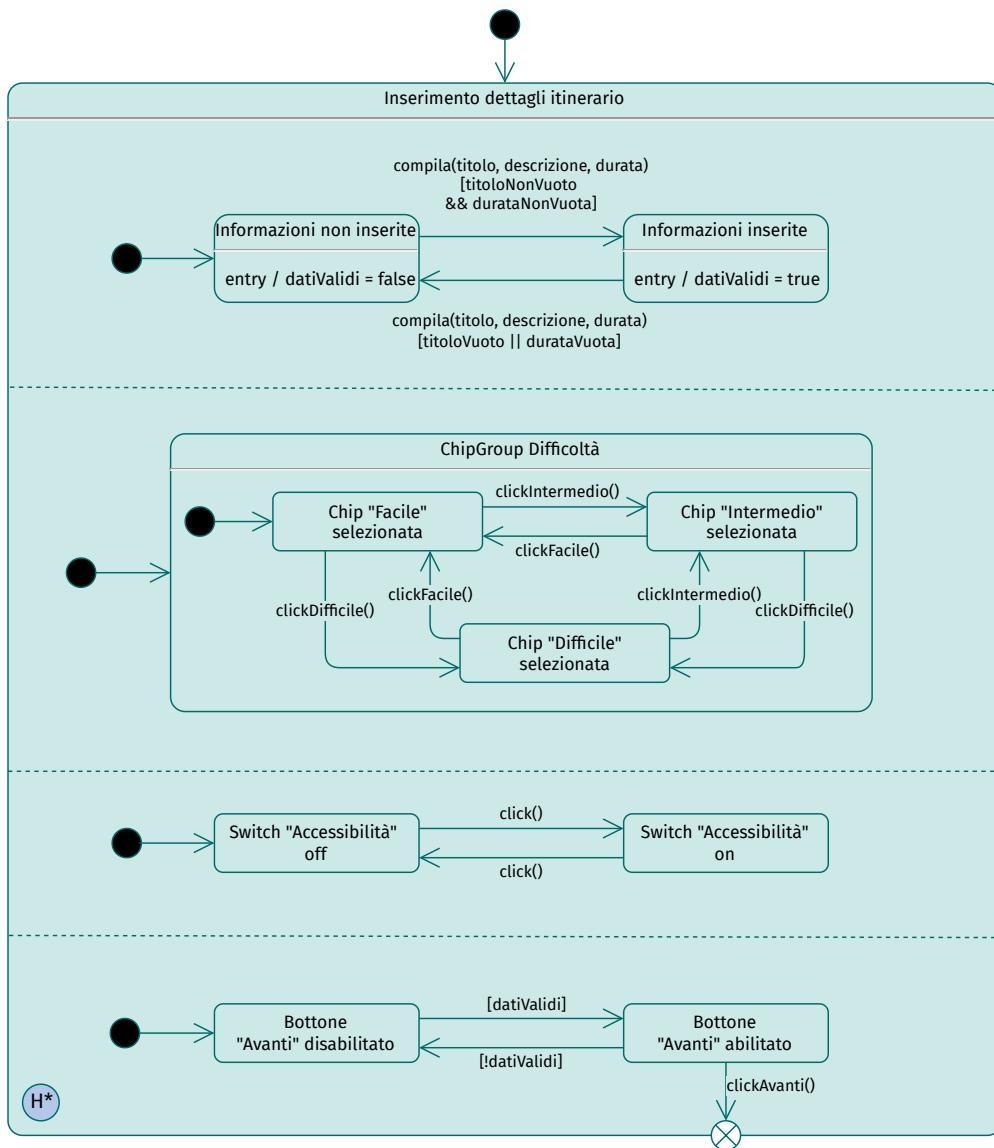


Figura 10: Regione interna - Inserimento dettagli itinerario

2.7.4 Inserimento mappa itinerario

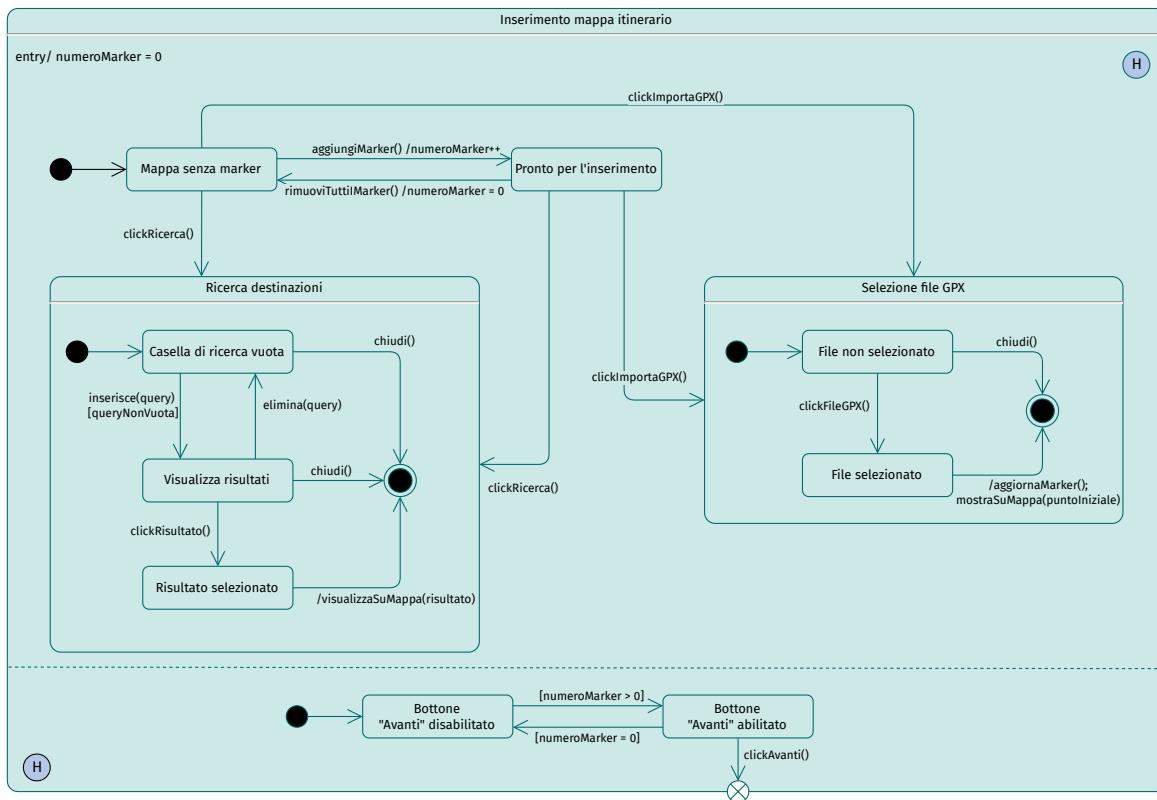


Figura 11: Regione interna - Inserimento mappa itinerario

2.7.5 Inserimento foto itinerario

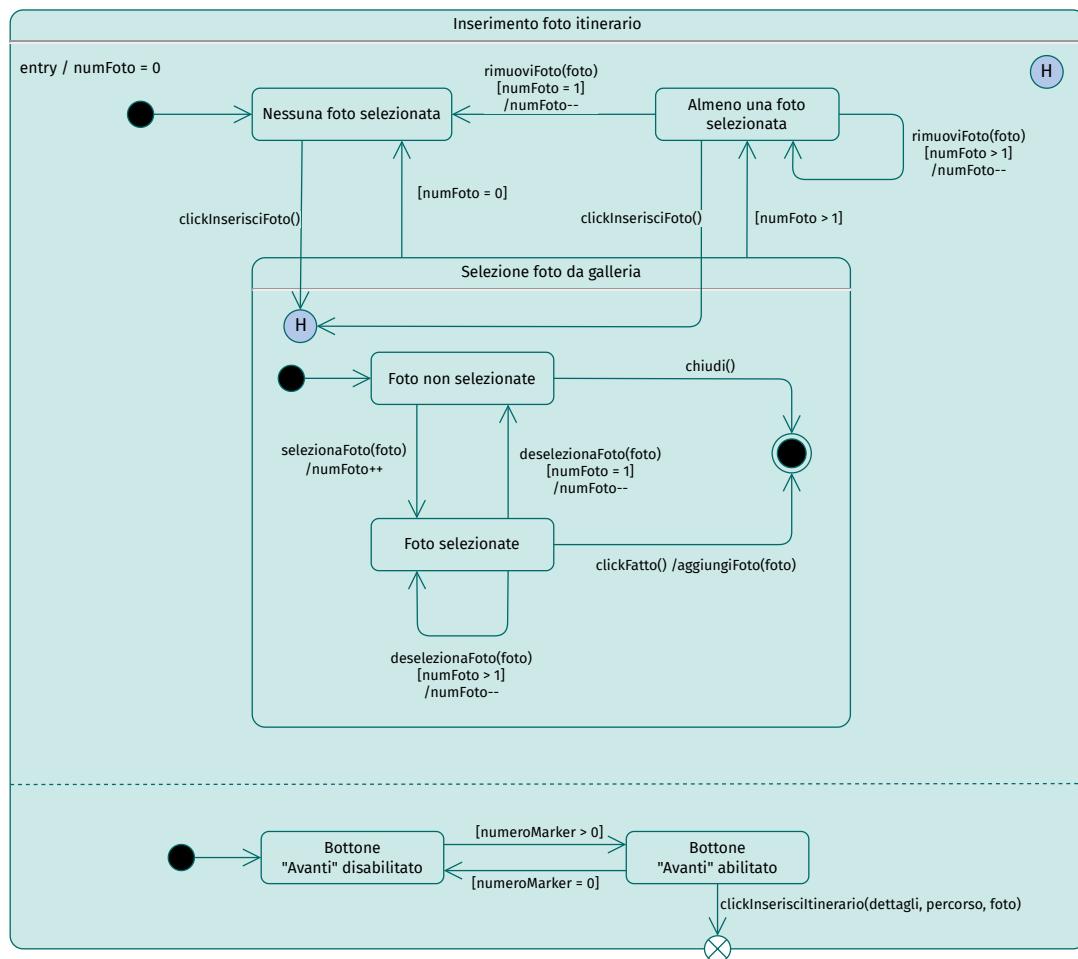


Figura 12: Regione interna - Inserimento foto itinerario

2.8 Classi, oggetti e relazioni di analisi

2.9 Diagrammi di sequenza di analisi

Sono presentati nella seguente sezione i Sequence Diagram relativi a due funzionalità offerte dall'applicazione: la segnalazione di un itinerario e la ricerca di un itinerario.

subsubsection Segnalazione itinerario

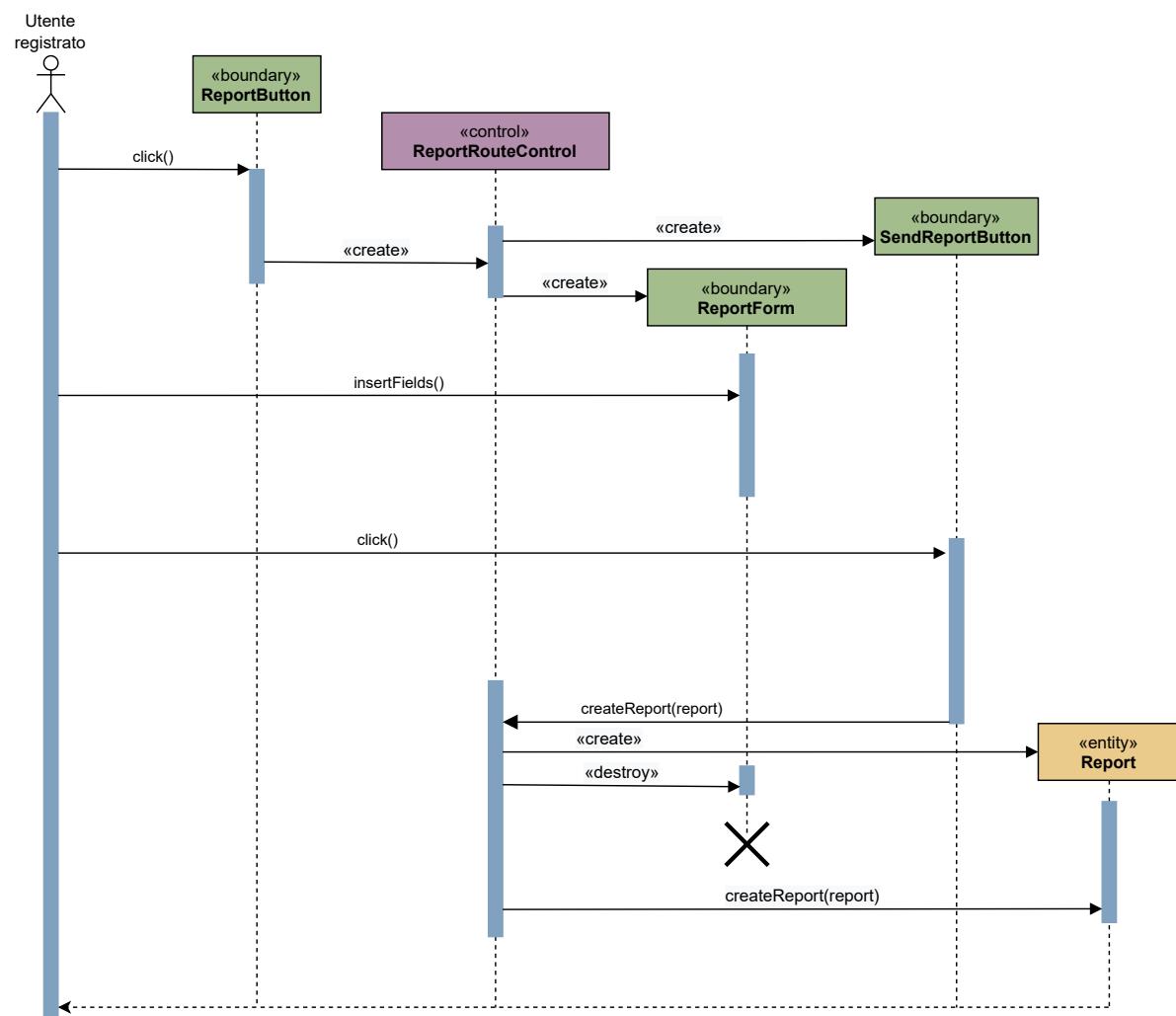


Figura 13: Sequence Diagram 1 - Segnalazione di un itinerario

2.9.1 Ricerca itinerario

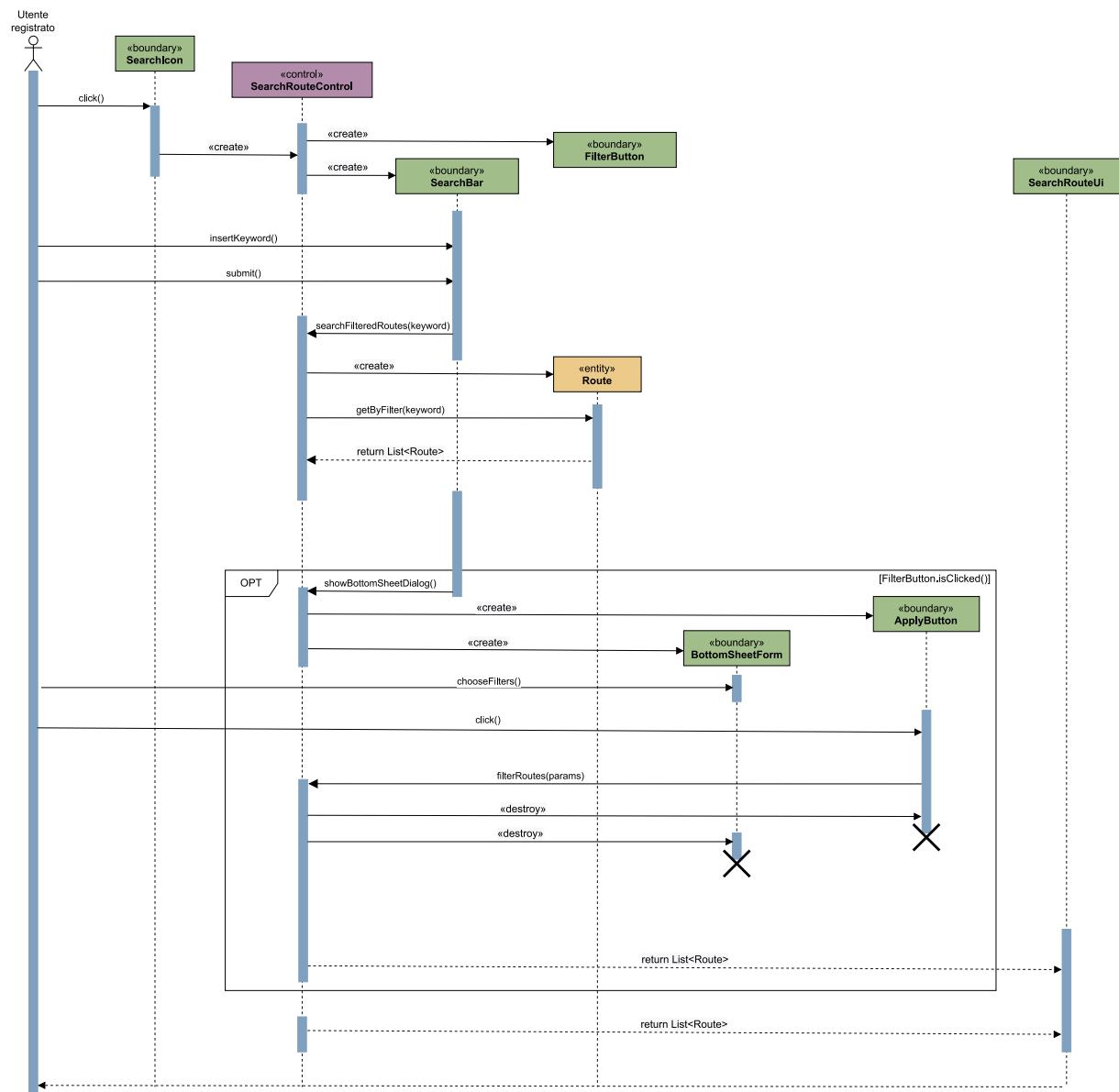


Figura 14: Sequence Diagram 2 - Ricerca di un itinerario

2.10 Diagrammi di attività

2.10.1 Autenticazione

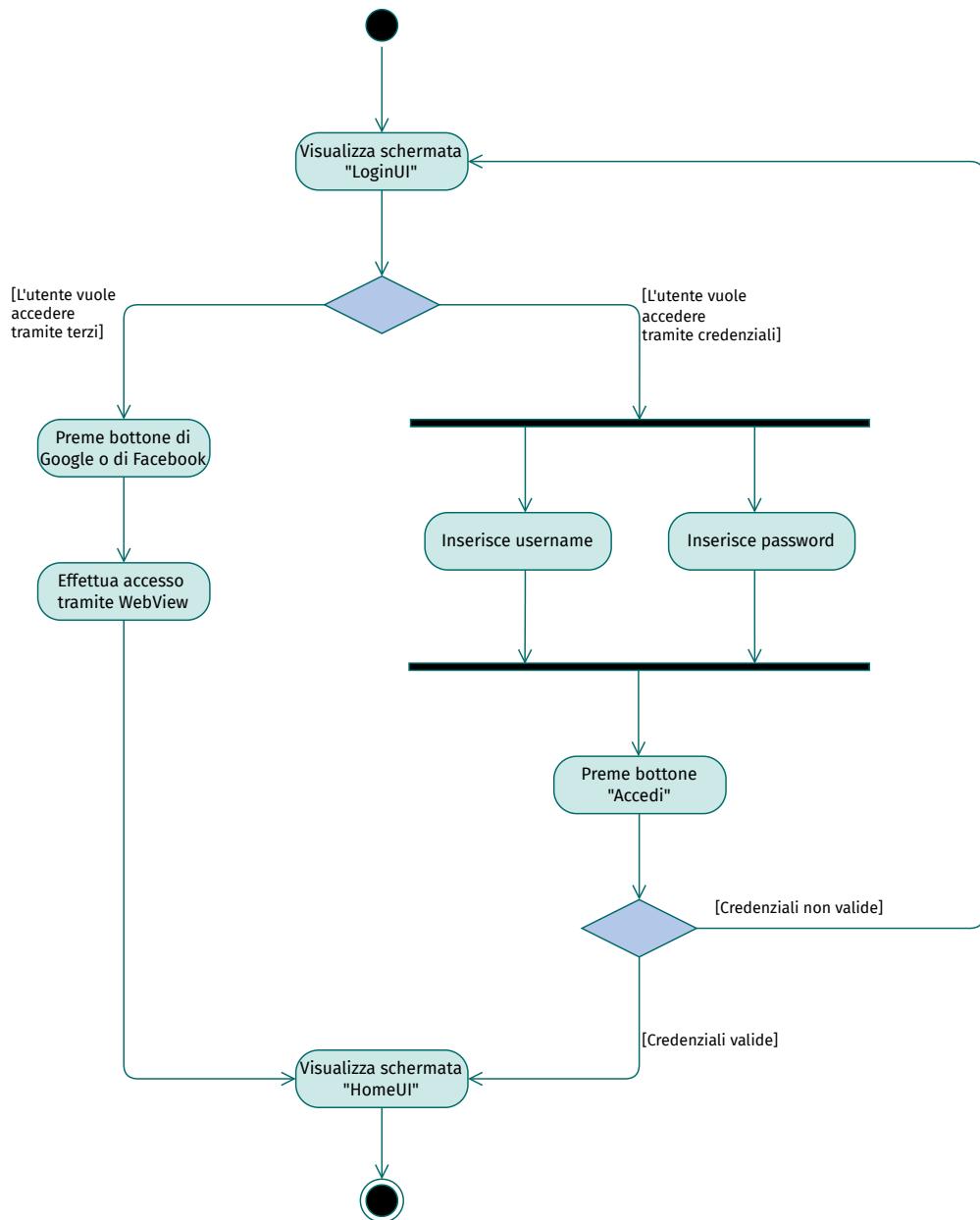


Figura 15: Login utente

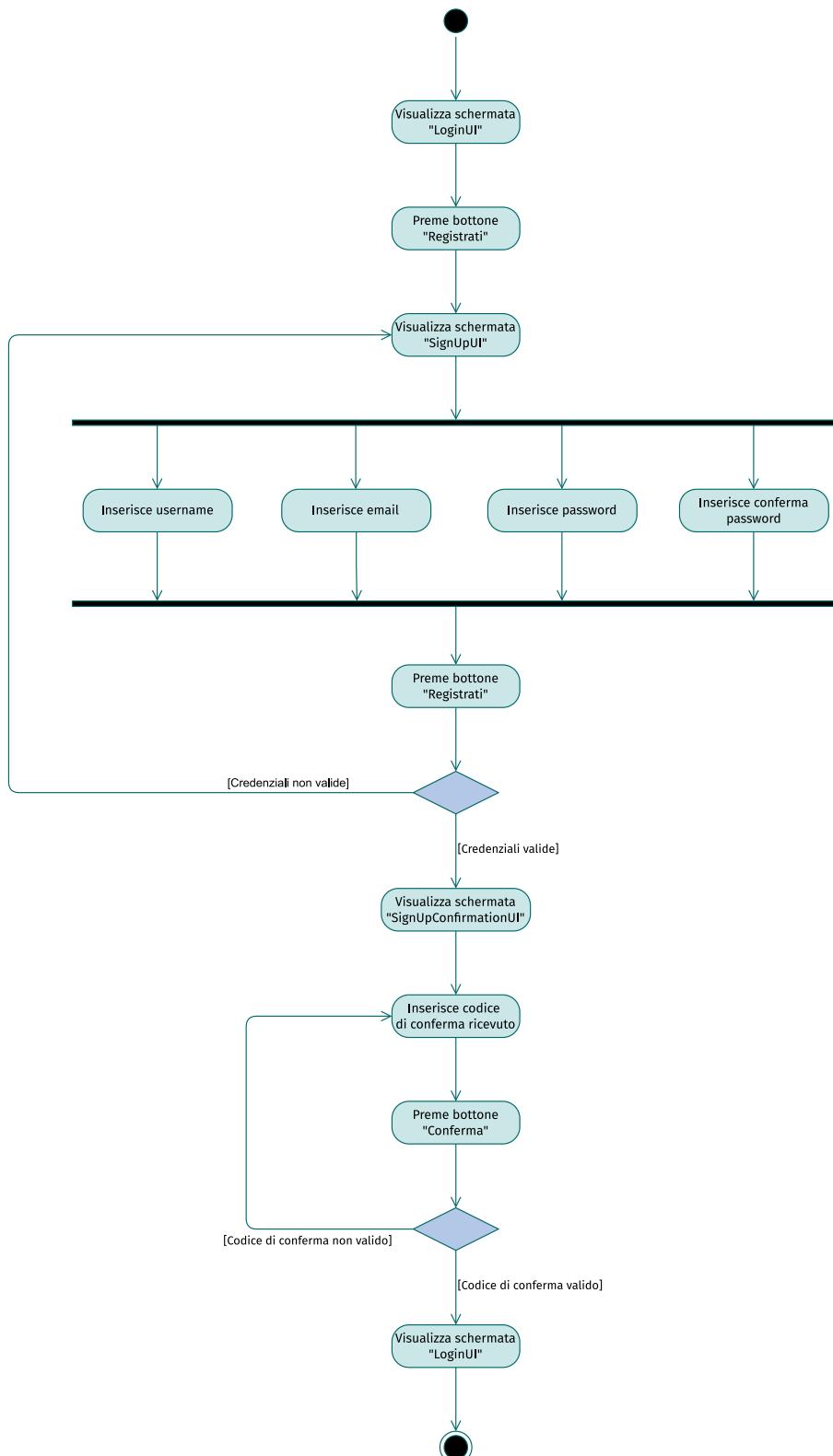


Figura 16: Registrazione utente

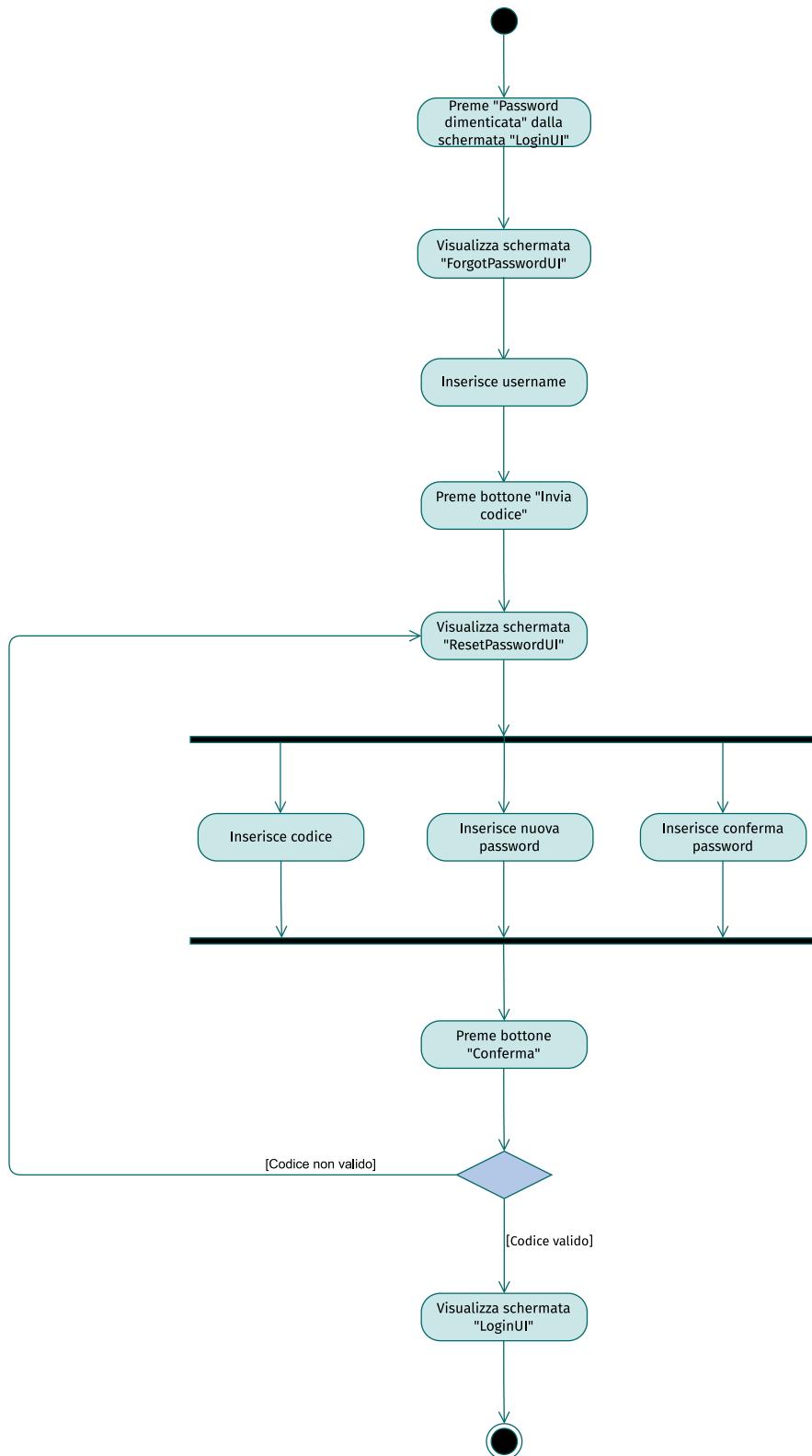


Figura 17: Reset password dimenticata

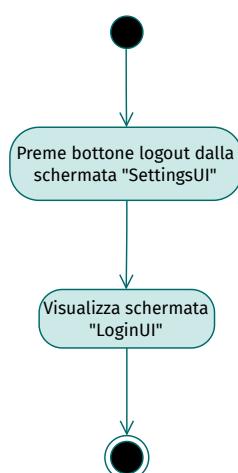


Figura 18: Logout

2.10.2 Interazione con un itinerario

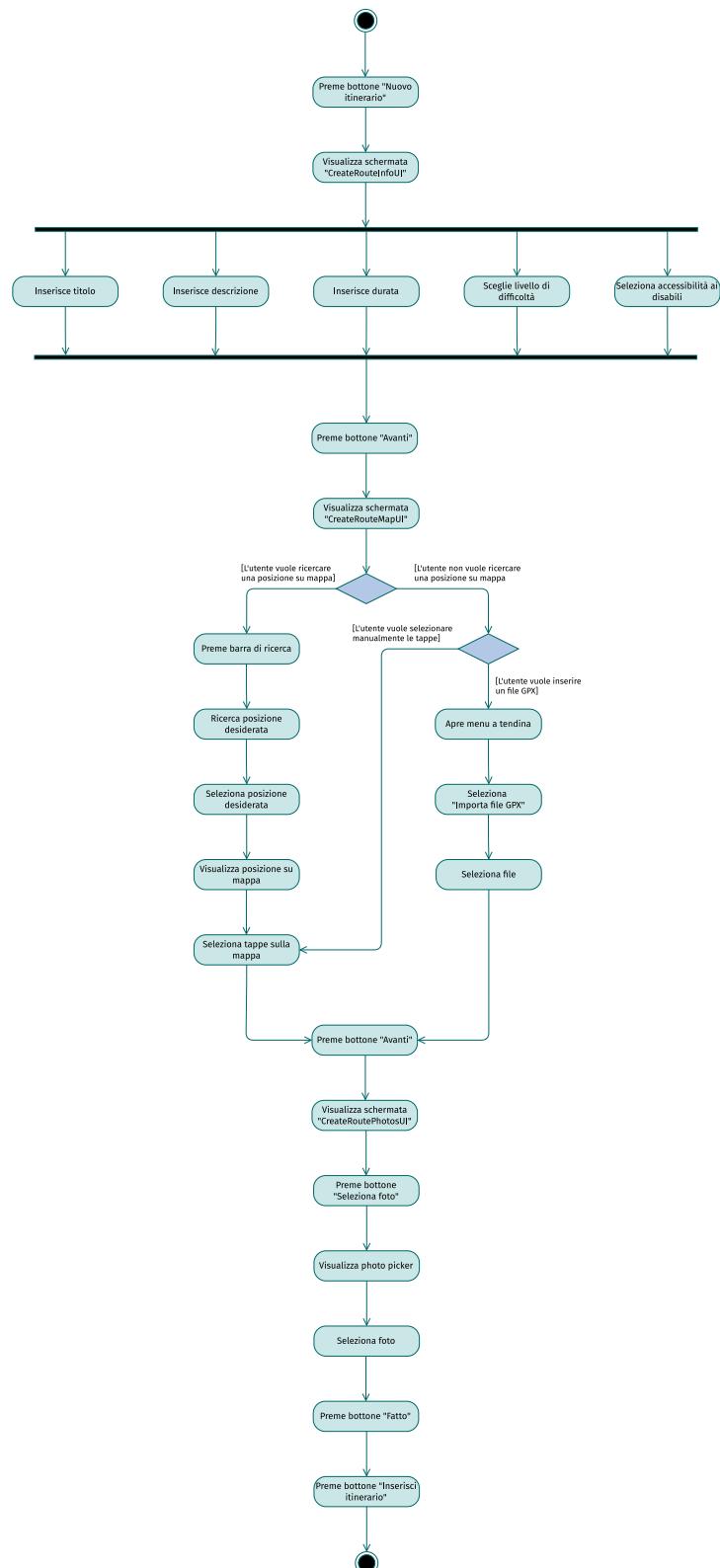


Figura 19: Aggiunta itinerario

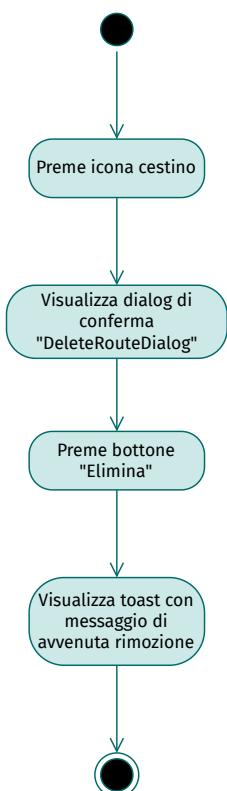


Figura 20: Rimozione itinerario

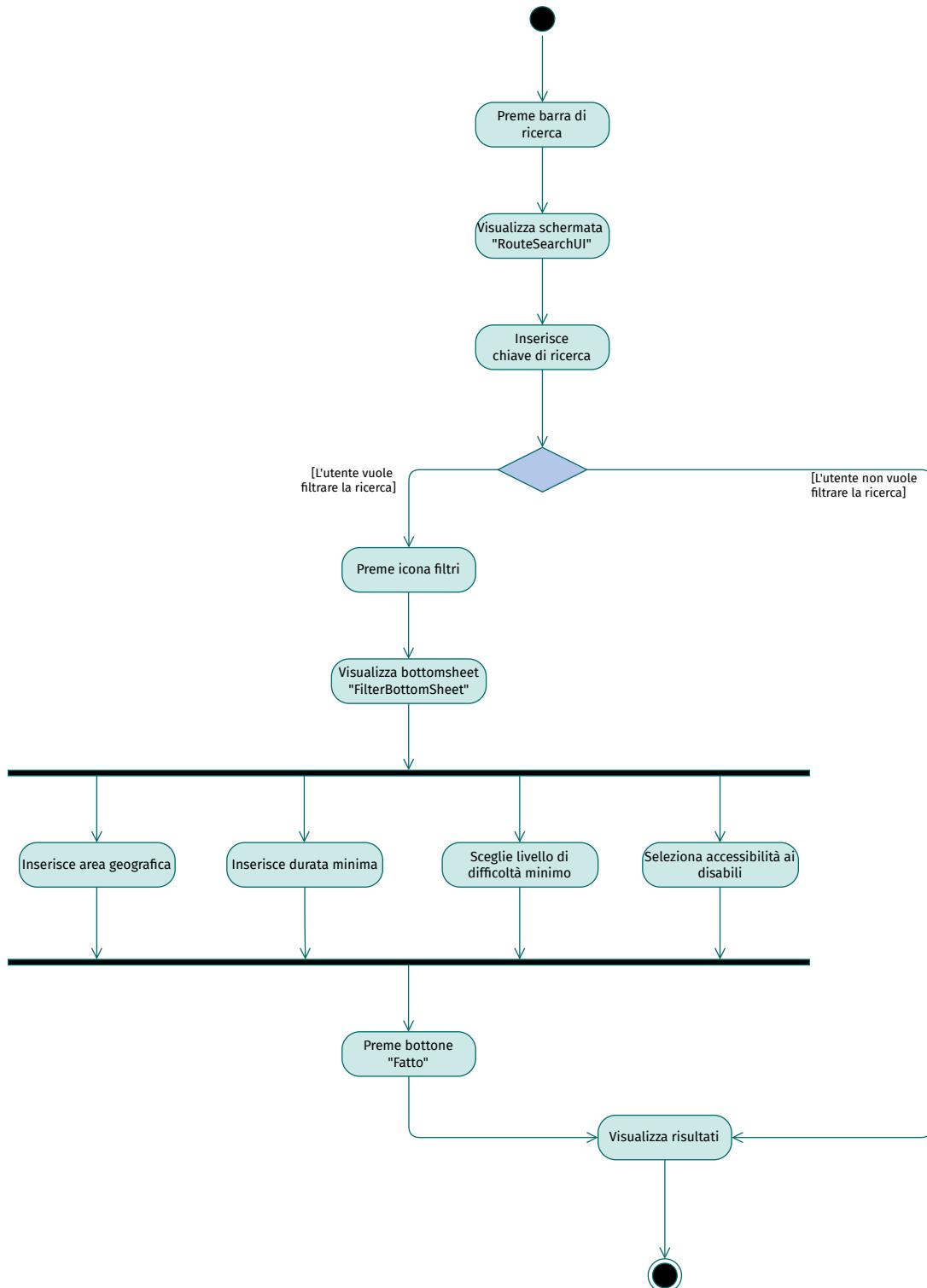


Figura 21: Ricerca itinerario

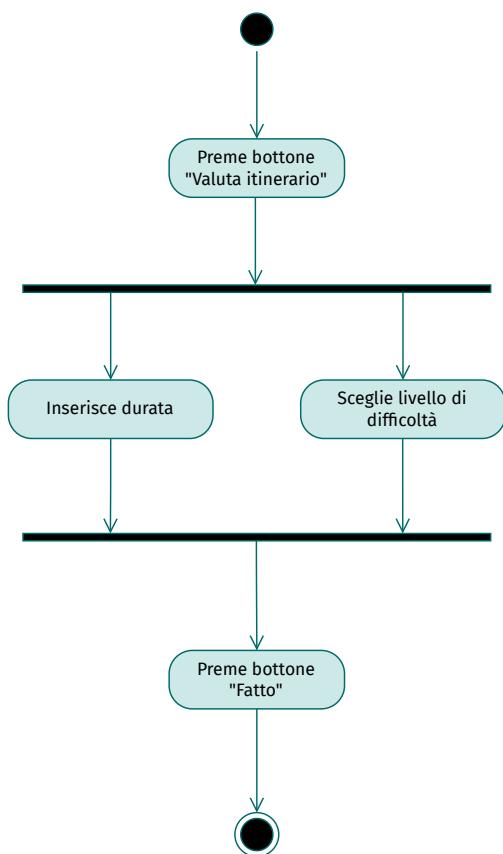


Figura 22: Valutazione itinerario



Figura 23: Salvataggio itinerario

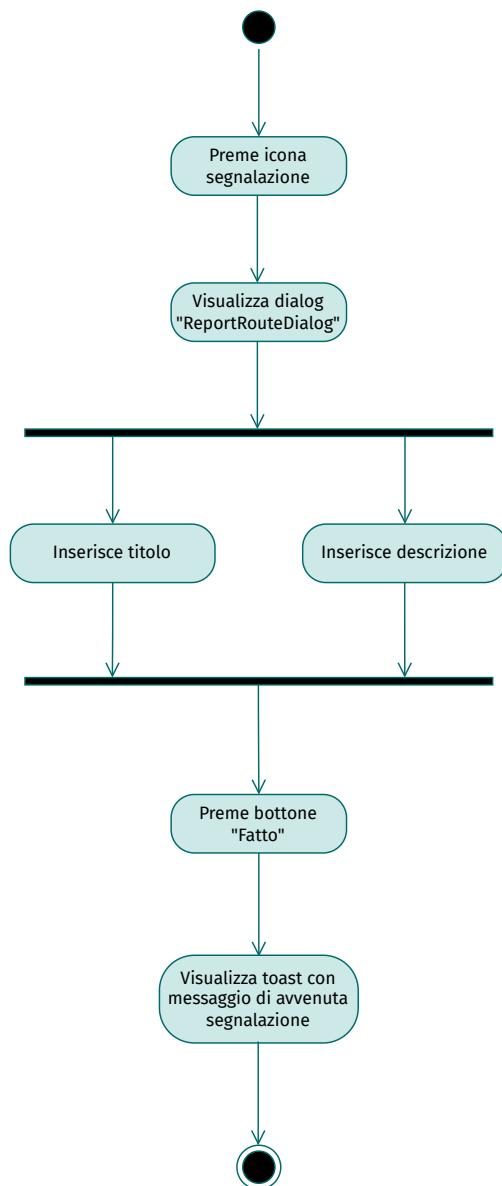


Figura 24: Segnalazione itinerario

2.10.3 Interazione con un post



Figura 25: Aggiunta post

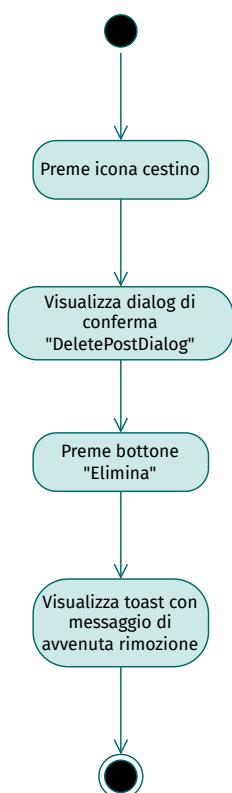


Figura 26: Rimozione post

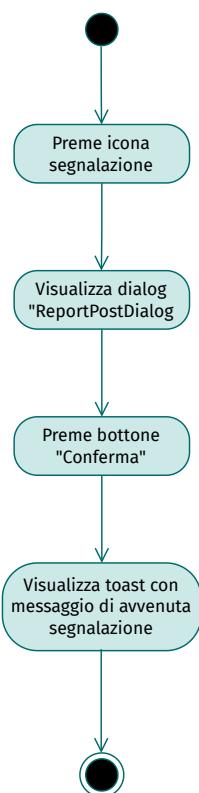


Figura 27: Segnalazione post

2.10.4 Interazione con una compilation

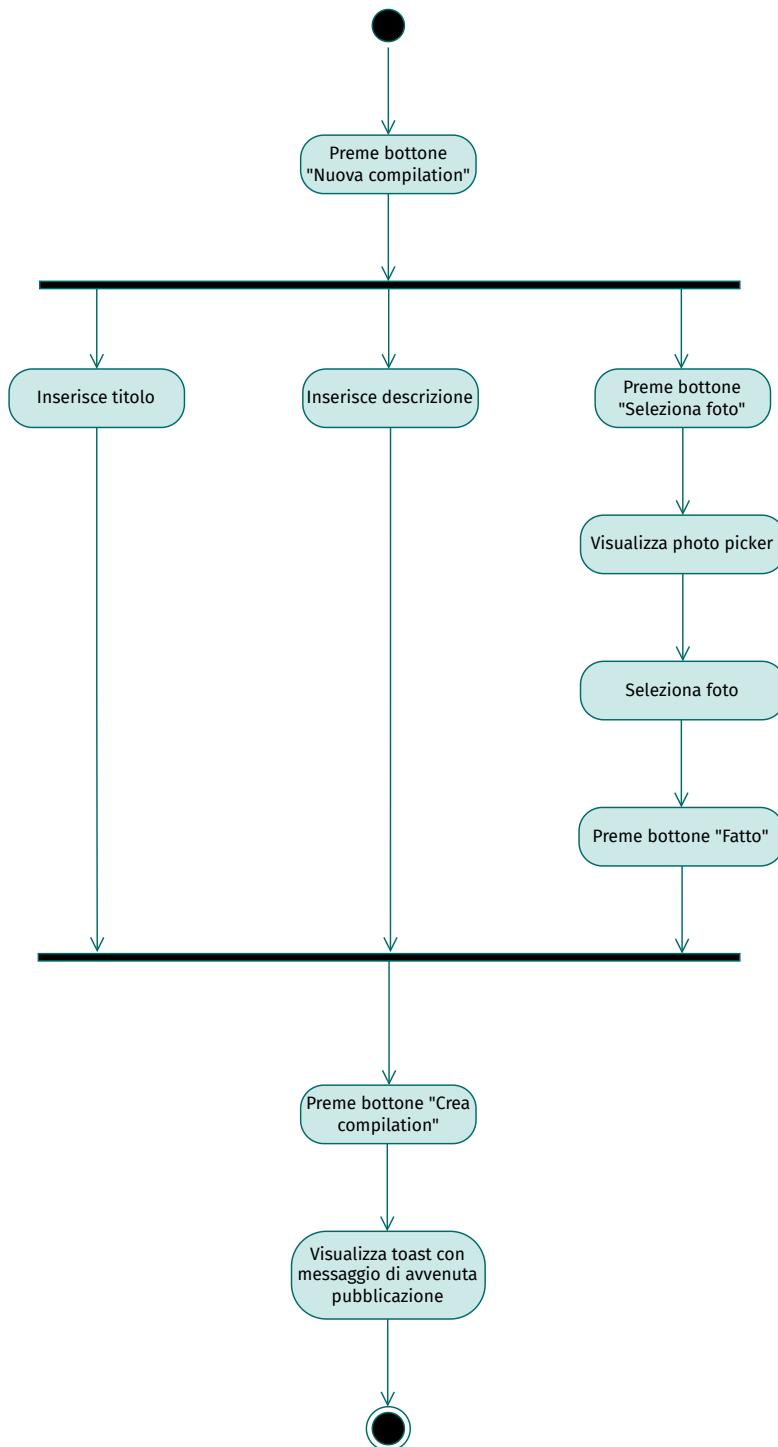


Figura 28: Aggiunta compilation

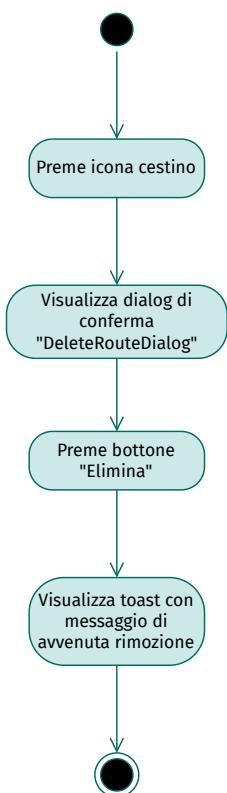


Figura 29: Rimozione compilazione

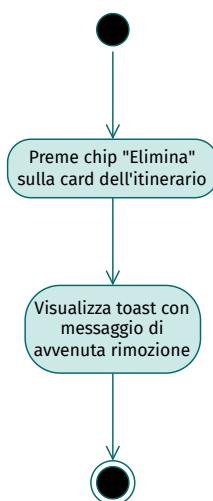


Figura 30: Rimozione itinerario da compilation

2.10.5 Gestione profilo e interazione con utenti

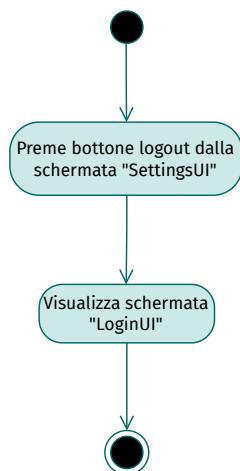


Figura 31: Invio messaggio

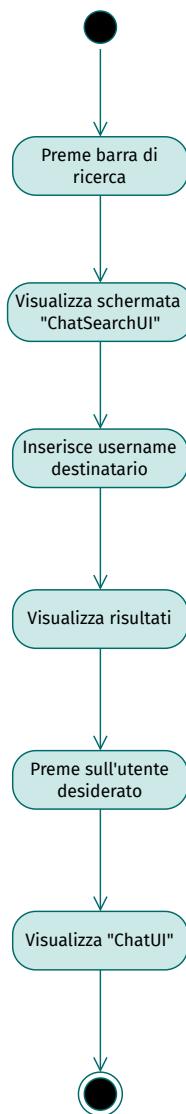


Figura 32: Ricerca destinatario messaggio

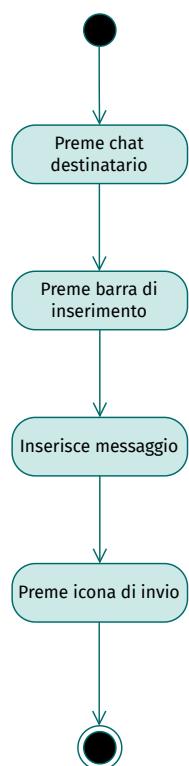


Figura 33: Avvio conversazione con l'autore di un post o itinerario

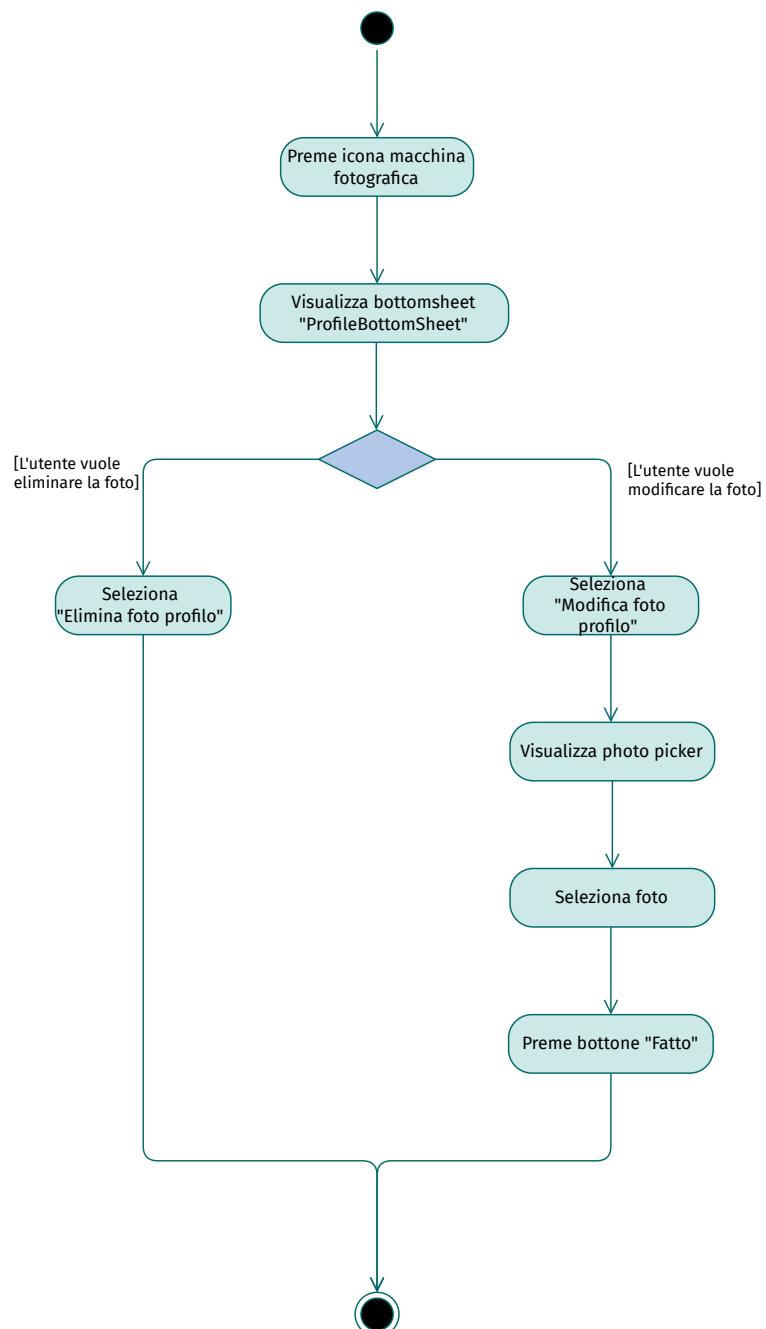


Figura 34: Modifica foto profilo

2.10.6 Funzionalità riservate agli amministratori



Figura 35: Rimozione segnalazione

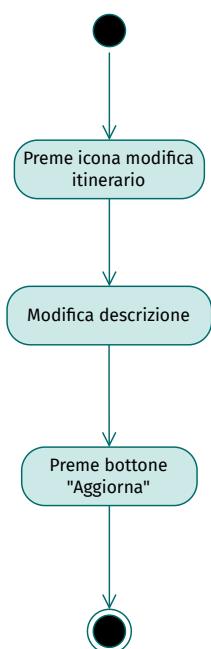


Figura 36: Modifica itinerario

3 Documento di Design del sistema

3.1 Analisi dell'architettura e criteri di design

3.1.1 Diagramma di design del sistema

In questa sezione viene presentato un diagramma realizzato con lo scopo di rappresentare la struttura generale dell'architettura di sistema realizzata; in esso sono stati messi in evidenza i servizi di cui si è usufruito nello sviluppo del software.

Le specifiche di ciascuna scelta implementativa sono dettagliate nelle sezioni successive.

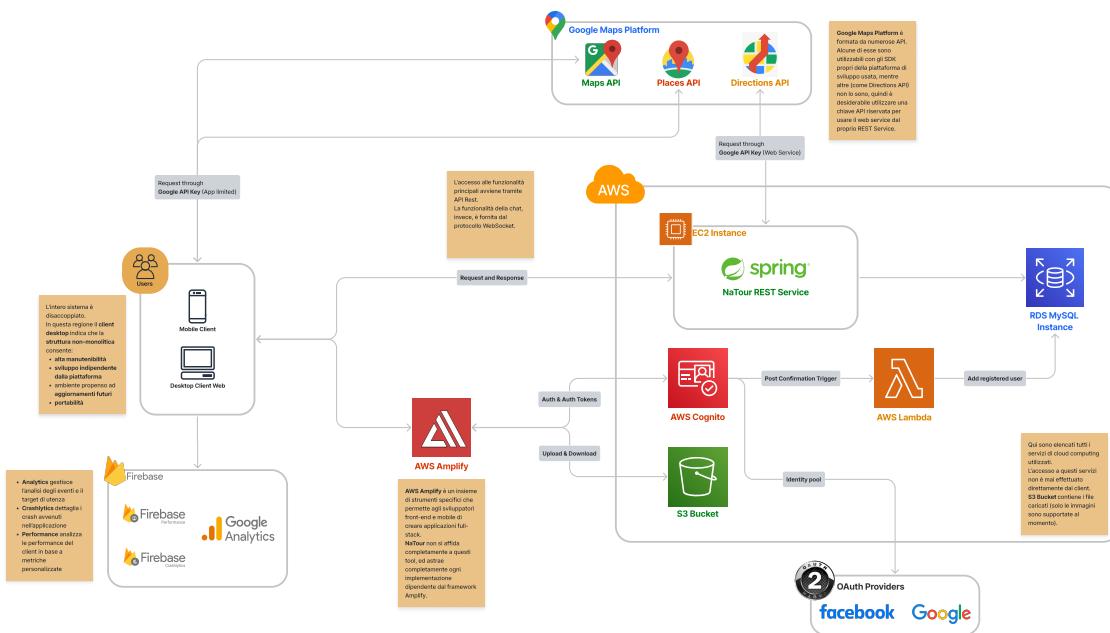


Figura 37: Diagramma del sistema

3.1.2 Software deployment

L'architettura del software realizzato pone le sue fondamenta nel concetto di **cloud computing**.

Il cloud computing, o computazione cloud, consiste nella distribuzione on-demand di risorse IT, con una tariffazione basata sul consumo.

Tra i diversi provider, è stato scelto di usufruire dei servizi offerti dalla piattaforma proprietaria del gruppo Amazon **AWS, Amazon Web Services**. AWS offre infatti cloud services ideali per creare applicazioni in modo scalabile, flessibile e affidabile; esso presenta inoltre notevoli vantaggi, ritenuti come caratteristiche desiderabili per la progettazione di qualunque software.

La scelta della piattaforma AWS è stata compiuta, inoltre, per le seguenti qualità:

- **Agilità** - la possibilità di aumentare a seconda delle necessità risorse quali servizi infrastrutturali, calcolo, storage e database;
- **Elasticità** - la possibilità di evitare l'allocazione anticipata di una quantità maggiore di risorse di quante siano necessarie, così da gestire i picchi nei livelli di attività future;
- **Distribuzione globale di contenuti** - l'infrastruttura AWS offre una copertura globale: fare in modo che le applicazioni siano vicino agli utenti finali riduce la latenza e migliora la loro esperienza.

AWS offre una grande varietà di servizi; ciascuno di essi va incontro a una specifica necessità dello sviluppatore.

Nello sviluppo del software, per implementare determinate funzionalità si è scelto di ricorrere all'utilizzo dei seguenti servizi:

- **Cognito** - Amazon Cognito permette di aggiungere strumenti di registrazione degli utenti, accesso e controllo degli accessi alle app Web e per dispositivi mobili. Esso supporta l'accesso degli utenti tramite l'uso di provider di identità social quali Facebook, Google. Tale risorsa è stata sfruttata realtivamente all'autenticazione utente, tenendo particolarmente in considerazione l'alto fattore di sicurezza che essa conferisce;
- **EC2** - l'**Elastic Compute Cloud** (Amazon EC2) fornisce capacità di calcolo scalabile in AWS Cloud. Esso offre ambienti di elaborazione virtuale, noti come *istanze*, varie configurazioni di CPU, memoria, archiviazione e capacità di rete per le istanza note come *tipi di istanza*. La scelta di tale servizio è stata finalizzata al deploy dell'applicativo Spring, nella realizzazione del Rest Service.
- **RDS** - Amazon RDS ha permesso la configurazione e l'utilizzo del database relazionale alla base del software prodotto;
- **S3** - il **Simple Storage Service** (Amazon S3) permette l'archiviazione di oggetti in modo scalabile. Esso è stato adoperato in merito alla preservazione permanente dei file immagine caricati dagli utenti;

- **Lambda** - AWS Lambda è un servizio di calcolo basato su eventi serverless. La scelta del servizio Lambda è stata incentivata dalla sua possibilità di integrazione con il servizio **Cognito**, e dettata dalla volontà di conferire *consistenza* al pool utenti del sistema. Nello specifico, si è sfruttata la possibilità offerta dal servizio di creare dei *trigger* (in particolare trigger di *post conferma*), al fine di evitare - in seguito al processo di registrazione - la possibile presenza di inconsistenze tra database e registrazioni effettivamente portate a termine.

Si è ritenuto inoltre fondamentale l'utilizzo del framework **Amplify**.

In particolare, Amplify è stato impiegato per realizzare:

- Un servizio di autenticazione particolarmente sicuro tramite, come citato, il servizio *Cognito*;
- L'archiviazione di file (nella prima versione del software solo di file immagine) tramite il servizio *S3*.

È importante specificare che il framework Amplify consente anche di creare back-end o applicativi serverless. Ciononostante, si sottolinea che nella realizzazione del software è stata fatta la scelta di utilizzarlo *solo* in funzione dei servizi da essi offerti, soprattutto in relazione alla deprecazione dell'SDK standard AWS per Android.

3.1.3 Google Maps Platform

Una delle caratteristiche principali del software risulta essere la forte componente geolocalizzata degli itinerari presenti in piattaforma.

Gli utenti, infatti, nell'inserimento e nella visualizzazione dei sentieri si trovano ad interfacciarsi direttamente con mappe interattive. Per garantire un'esperienza ottimale da questo punto di vista sono stati utilizzati i servizi della piattaforma Google Maps.

La **Google Maps Platform** è un insieme di API e SDK che permette di integrare in applicazioni mobile Google Maps, e di recuperare dati da esso stesso. L'esperienza utente con le funzionalità sopracitate è stata supportata dall'utilizzo delle seguenti API:

- MapsAPI - per la visualizzazione interattiva di mappe statiche e dinamiche;
- PlacesAPI - per il recupero di informazioni sui posti tramite richieste HTTP;
- DirectionsAPI - per il calcolo del percorso tra diverse tappe; utilizzano una richiesta HTTP per ritornare le direzioni tra località in formato JSON o XML. Le DirectionsAPI, in quanto web service, sono state integrate nel Rest Service proprio del software.

3.1.4 Architettura del REST Service

Nella realizzazione del REST Service è stato scelto di utilizzare il framework **Spring**. Spring fornisce un'*infrastruttura di supporto* per lo sviluppo di applicazioni: esso prevede una *modularizzazione* dell'architettura come segue:

- Presentation layer - strato più esterno, che gestisce la presentazione del contenuto e l'interazione con l'utente;

- Business logic layer - strato centrale, che gestisce la logica;
- Data access layer - strato più interno, che gestisce il recupero di dati dalle diverse sorgenti.

Ciascuno di questi strati dipende da quello sottostante per far sì che l'applicazione funzioni. In altre parole, lo strato di presentazione comunica con quello della business logic, che a sua volta comunica con lo strato di data access. Ogni strato ha quindi bisogno di questa *dipendenza* per eseguire il proprio compito correttamente

La scelta di utilizzare il framework Spring è stata fatta in relazione alla volontà di ottenere un *loose coupling*, ossia un accoppiamento largo. Senza l'utilizzo di Spring, ci sarebbe stata la possibilità che il codice avesse potuto causare *tight coupling*, che non è considerato essere una buona pratica di programmazione. Il loose coupling è ideale, in quanto le componenti largamente accoppiate sono indipendenti: ciò implica che, a seguito di possibili cambiamenti futuri, questi non influenzano le altre componenti.

Al cuore del framework Spring si trova la **dependency injection**.

La dependency injection è un pattern di programmazione che permette agli sviluppatori di costruire architetture disaccoppiate. Ciò vuol dire che Spring comprende le *annotazioni* poste in capo alle classi; in seguito alla creazione di un'istanza, dunque, il framework si assicura che le istanze siano state create con le opportune dependencies.

Nello specifico, per l'implementazione del software, è stata utilizzata un'estensione del framework Spring, chiamata **Spring Boot**.

Spring Boot ha delle caratteristiche specifiche che rendono la gestione dell'applicazione più semplice.

Tra alcuni dei vantaggi che hanno portato alla scelta di Spring Boot, si ricordano:

- Creazione di applicazioni Spring stand-alone;
- Dotazione di dependencies "starter" per semplificare la configurazione di build;
- Configurazione automatica di Spring e di librerie di terze parti, ove possibile;
- Nessuna generazione di codice e nessun requisito per la configurazione XML;
- Inclusione di un web server embedded (nello specifico *Apache Tomcat*) senza la necessità di ulteriori configurazioni.

Di seguito viene presentato un diagramma che dettaglia la struttura e il funzionamento del REST Service.

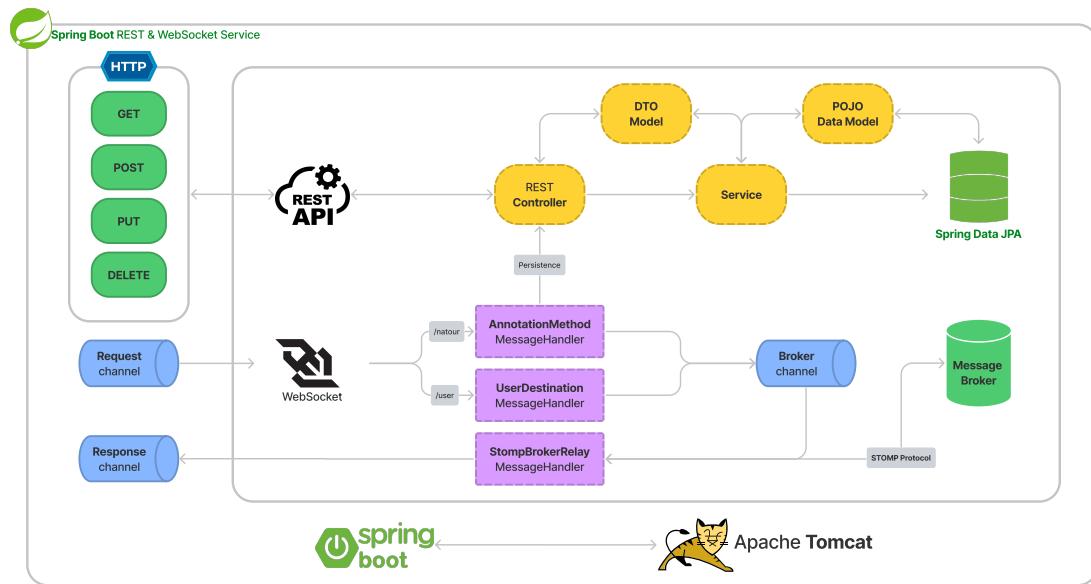


Figura 38: REST + WS Service realizzato con Spring Boot

3.2 Architettura dell'applicativo

L'intera architettura del software è stata progettata seguendo le linee guida della cosiddetta **Clean Architecture**.

Il concetto di architettura *pulita* si basa sui principi enunciati da Robert C. Martin nel libro "Clean Architecture". In seguito alle informazioni acquisite dalla lettura del libro, infatti, questo approccio è stato ritenuto quello più valido.

L'idea chiave è quella di utilizzare il principio di inversione di dipendenza per porre dei boundaries tra componenti di alto livello e componenti di basso livello. Ciò crea un'architettura *plug-in*, che conferisce al sistema un'elevata flessibilità e un'elevata manutenibilità. Un'architettura *pulita* inizia da un codice *pulito*. Classi pulite derivano da componenti puliti, che, a loro volta, determinano un *sistema pulito*.

È stato quindi ritenuto di fondamentale importanza applicare i principi della Clean Architecture e del Clean Code in maniera uniforme e costante, nell'implementazione di qualsiasi componente.

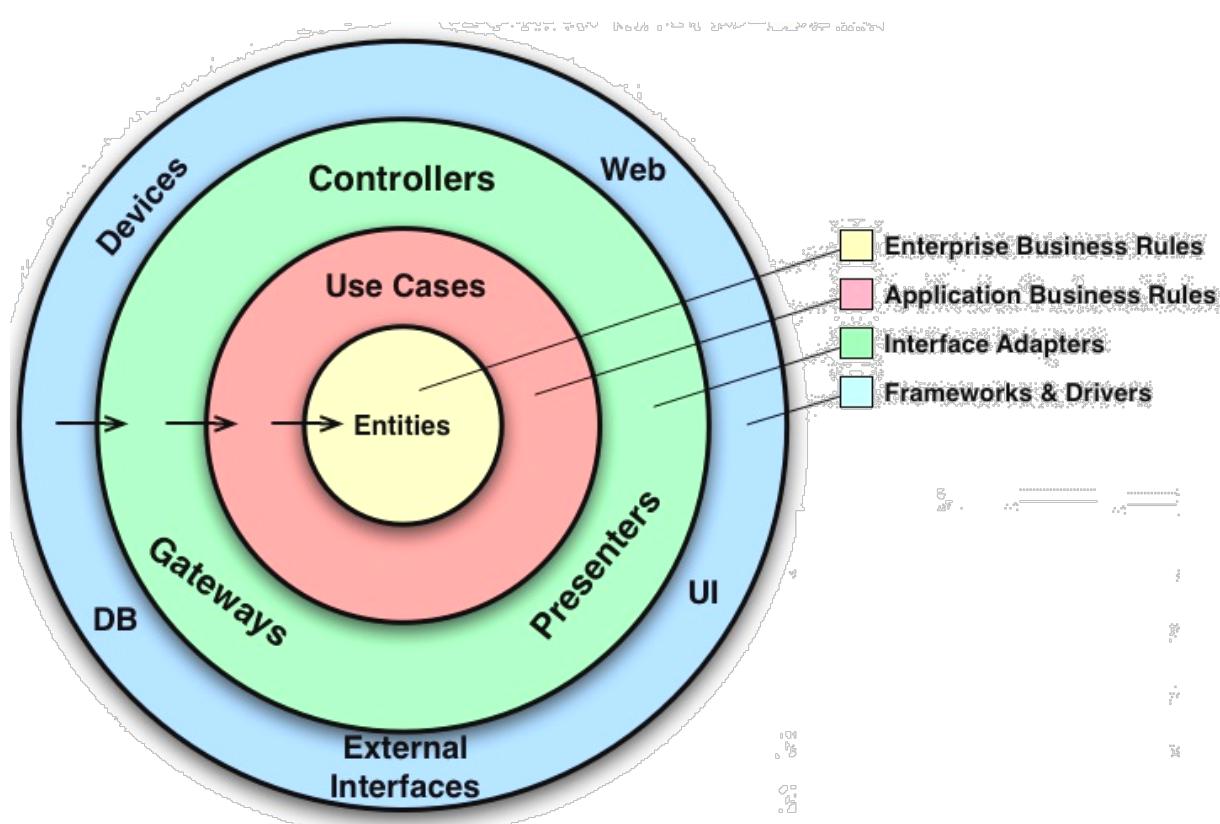


Figura 39: Clean Architecture

Il concetto di *Clean Code* sfruttato nello sviluppo del software segue i principi **SOLID**. Tali principi sono stati applicati in modo da ottimizzare l'architettura e ciascuna delle sue componenti, le quali risultano dunque essere caratterizzate dal:

- **Single responsibility principle** – una classe dovrebbe avere uno ed un solo motivo per cambiare;
- **Open-closed principle** – una classe dovrebbe essere aperta all'estensione ma chiusa alle modifiche;

- **Liskov's substitution principle** – gli oggetti di un programma dovrebbero essere sostituibili con istanze dei propri sottotipi senza alterare la correttezza del programma stesso;
- **Interface segregation principle** – più interfacce client-specific sono meglio di un'unica interfaccia general-purpose;
- **Dependency inversion principle³** – i moduli di alto livello non dovrebbero dipendere da moduli di basso livello; entrambi dovrebbero dipendere da *astrazioni*.

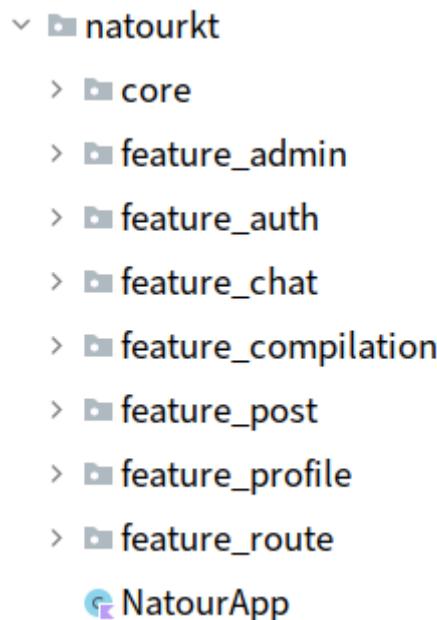
Il concetto di Clean Architecture si basa sulla *modularità* dell'architettura stessa.

Nel caso della progettazione del software prodotto è stata seguita una suddivisione del seguente tipo:

- Presentation layer – responsabile della presentazione a schermo e della gestione delle interazioni degli utenti;
- Domain layer – rappresentazione formale del dominio applicativo
- Data layer – contenitore delle implementazioni delle sorgenti di dati e dei repository, i quali coordinano i dati provenienti da esse.

È bene sottolineare, però, che nell'effettiva implementazione dell'applicativo non è stato pedissequamente seguito lo schema appena proposto.

È stato ritenuto più opportuno adottare un approccio *per funzionalità*; ciò è risultato nell'applicazione del concetto di **Screaming Architecture**. Di seguito ne è mostrato un esempio di implementazione, tratto dal software stesso.



³Il principio di inversione di dipendenza è stato applicato, in particolare, in relazione agli UseCase (nel Domain Layer): la loro indipendenza dagli altri moduli fa in modo che essi possano dipendere solo da *astrazioni* dei loro strati.

L'espressione "Screaming Architecture" è stata coniata dal sopracitato Robert C. Martin; essa viene utilizzata nelle situazioni in cui, rivolgendo un solo sguardo ad un progetto, si riesce ad avere un'idea di base su cosa esso faccia e su cosa riguardi.

Da ciò deriva la possibilità di comprendere esattamente la struttura al cuore del codice implementato: la suddivisione nei diversi package "urla" al lettore l'approccio utilizzato, che risulta dunque chiaro e comprensibile già a primo impatto.

La Clean Architecture non preclude, però, la possibilità di adattare all'applicativo un proprio design pattern architettonicale.

Come descritto nella sezione successiva, tale approccio è stato ritenuto adatto alle esigenze di implementazione.

3.3 Le scelte implementative

3.3.1 Applicazioni mobile native e ibride

Un'applicazione mobile nativa è un'applicazione software sviluppata per funzionare su uno specifico tipo di dispositivo o piattaforma. Le app native consentono performance migliori, un'esecuzione veloce e un alto grado di precisione. Nonostante ciò, sono noti diversi problemi:

- Le app native richiedono codice sorgente *esclusivo*, poiché ogni dispositivo deve avere la sua versione specifica dell'app;
- Richiede un costo maggiore, e sono richiesti più sviluppatori per creare e gestire il codice per ogni piattaforma;
- È richiesto molto tempo per la creazione differenziata per i diversi sistemi operativi in ogni aggiornamento delle funzionalità.

Un'app ibrida combina gli elementi di applicazioni web e native, i linguaggi utilizzati fanno parte di tecnologie web come HTML, CSS e JavaScript.

- Hanno un'interfaccia utente multipiattaforma;
- Possono essere sviluppate più velocemente e richiedono meno costi di sviluppo e manutenzione;
- Non hanno bisogno di codice sorgente specifico.
- Penalizzano la UX in alcuni casi.

La scelta stata è influenzata da diversi fattori. Tra i requisiti del sistema sono richieste prestazioni di buon livello, e il target di clienti è particolarmente esigente. È stato adottato un approccio nativo, precisamente su sistema operativo **Android**, scegliendo **Kotlin** come linguaggio di programmazione.

3.3.2 Perchè Kotlin

Java detiene da anni un ruolo chiave nello sviluppo di applicazioni, tuttavia alcuni requisiti *platform-dependant* possono rendere tedioso l'utilizzo di tale linguaggio. Esistono altri linguaggi che operano sulla JVM, tra questi Kotlin ha guadagnato molti punti a favore nello sviluppo di applicazioni native Android.

Tra i vantaggi di Kotlin c'è sicuramente la concisione attraverso l'inferenza di tipi, lo *smart-cast* e le *data class*: potendo risolvere lo stesso problema in meno righe di codice ne giovano leggibilità e manutenibilità.

Il type system di Kotlin mira ad eliminare le *NullPointerException*, garantendo controlli a tempo di compilazione e offrendo operatori di *null-safety*.

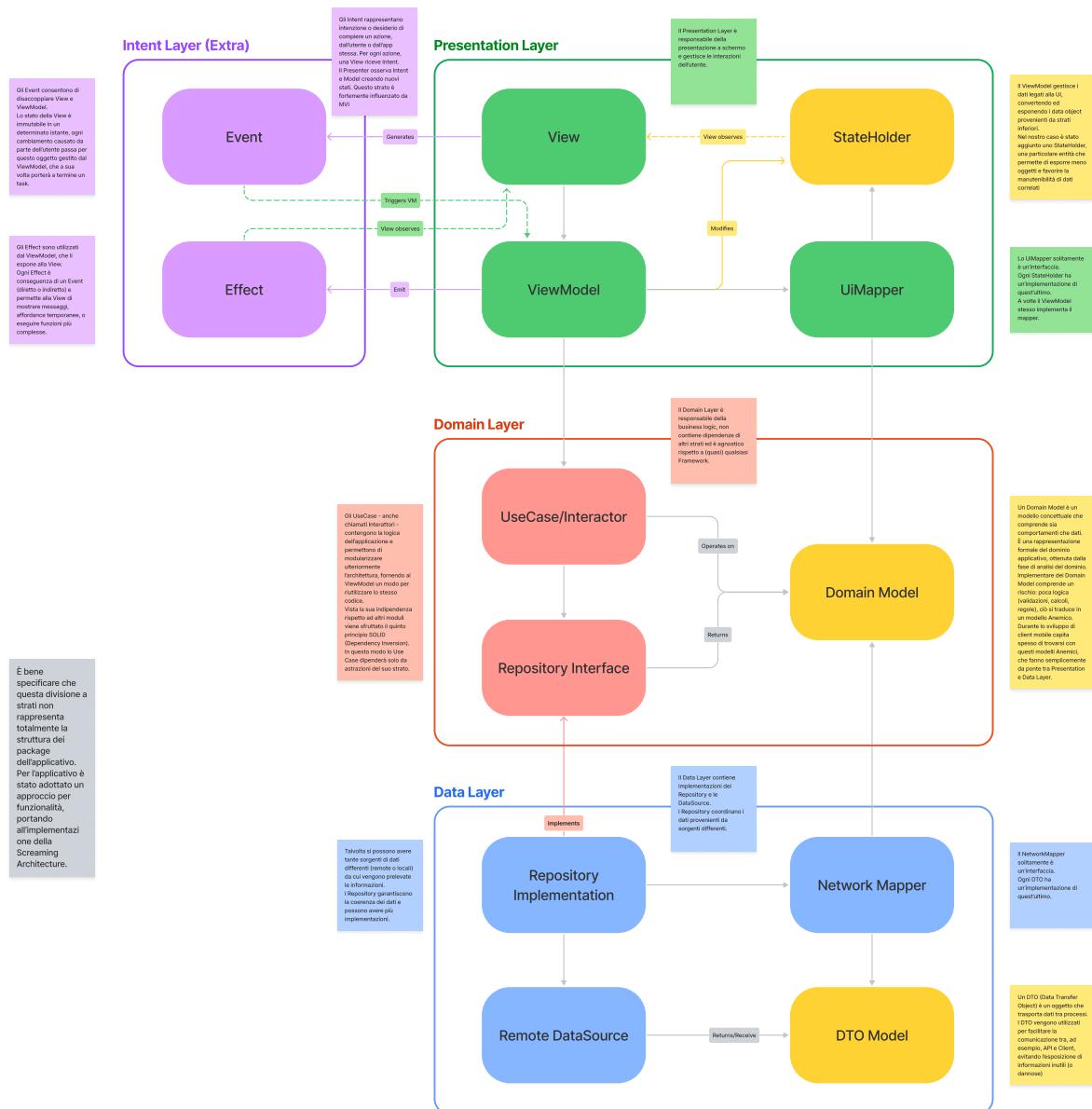
Tra le funzionalità più interessanti volte a migliorare la produttività dello sviluppatore ritroviamo una forte integrazione del paradigma di programmazione funzionale, oltre alla possibilità di creare funzioni particolari note come *estensioni*, che consentono l'aggiunta di nuovi metodi a classi pre-esistenti.

Inoltre Kotlin non preclude la possibilità di utilizzare codice Java, ma questa possibilità è bi-direzionale: Java e Kotlin possono coesistere all'interno dello stesso progetto.

3.3.3 Diagramma di design dell'applicativo

In questa sezione viene presentato un diagramma realizzato con lo scopo di rappresentare la struttura generale dell'applicativo.

Le specifiche di ciascuna scelta implementativa sono dettagliate nelle sezioni successive.



3.3.4 Pattern architetturale utilizzato

Il design pattern architetturale potrebbe essere definito come *ibrido*: ciò è dovuto alla *fusione* di due pattern differenti utilizzati nella progettazione.

Tali pattern sono **MVVM** e **MVI**; quest'ultimo è stato utilizzato per ovviare ai problemi sofferti dal primo. Procediamo ad un'analisi più dettagliata.

MVVM è un'architettura del tipo Model-View-ViewModel che evita l'accoppiamento stretto tra ciascuna componente. Più nello specifico, in questo tipo di architettura, i figli non hanno un riferimento diretto al padre, ma hanno solo riferimenti tramite *observables*. Esso è organizzato secondo tre strati:

- Model - rappresenta i dati e la business logic dell'applicazione;
- View - consiste del codice relativo alle interfacce utente;
- ViewModel - definisce una sorta di ponte tra i due modelli sopracitati.

Il modello MVVM, per quanto affermato, pone però alcune problematiche, quali ad esempio:

- la difficoltà di riusabilità sia per le View che per i ViewModel;
- la registrazione a molteplici observables nello stesso ViewModel, che ne aumenta le relative responsabilità;
- View e ViewModel possono essere soggetti ad avere un coupling stretto.

A questo scopo, è stata realizzata un'integrazione con il modello MVI, la cui differenza principale si risolve nell'assenza della moltitudine di observables citati in precedenza.

MVI sta per Model-View-Intent. Si tratta di uno dei pattern architetturali più recenti per applicazioni Android.

- Model - rappresenta uno stato. I Model dovrebbero essere immutabili, in modo da assicurare un flow di dati unidirezionale con gli altri strati;
- View - rappresenta le View, che possono essere implementate in Activity o Fragment
- Intent - rappresenta un'intenzione o una volontà di eseguire un'azione, sia dell'utente che dell'app stessa. Gli Intent si traducono in *Event* ed *Effect*.

L'integrazione dei due pattern architetturali sopracitati culmina nella nascita di un nuovo pattern, chiamato **UDF** dagli sviluppatori Android.

In sintesi:

- Presenza di *StateHolder*, necessari per ogni ViewModel nella maggior parte dei casi, non necessariamente unici;
- Richiamo alle macchine di stato e agli **Statechart** definiti in precedenza;
- *Event* e *Effect*, che si interpongono tra View e ViewModel.

3.4 Diagramma delle classi di design

3.4.1 Autenticazione

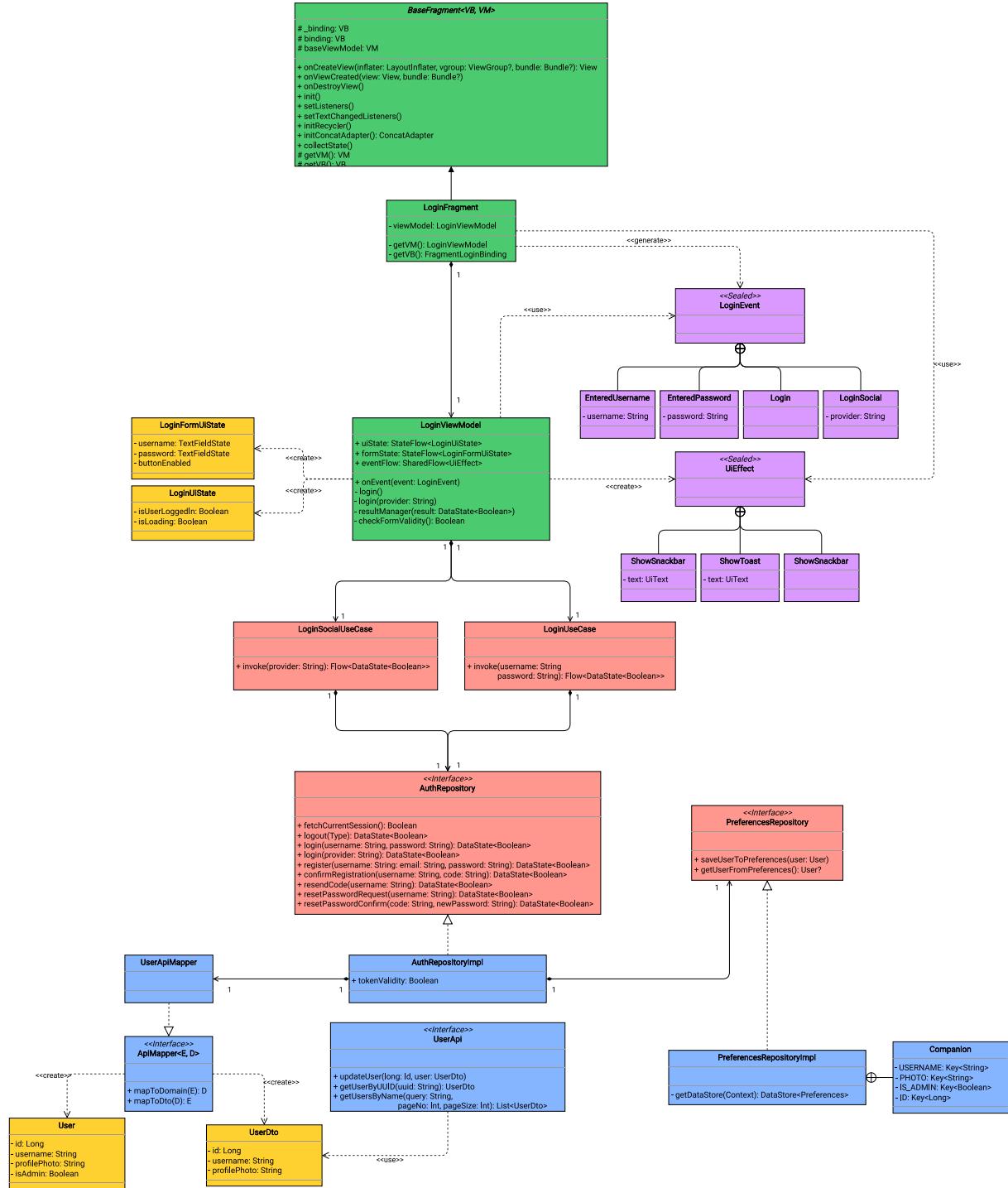


Figura 40: Login e Login con social

3.4.2 Interazione con una compilation

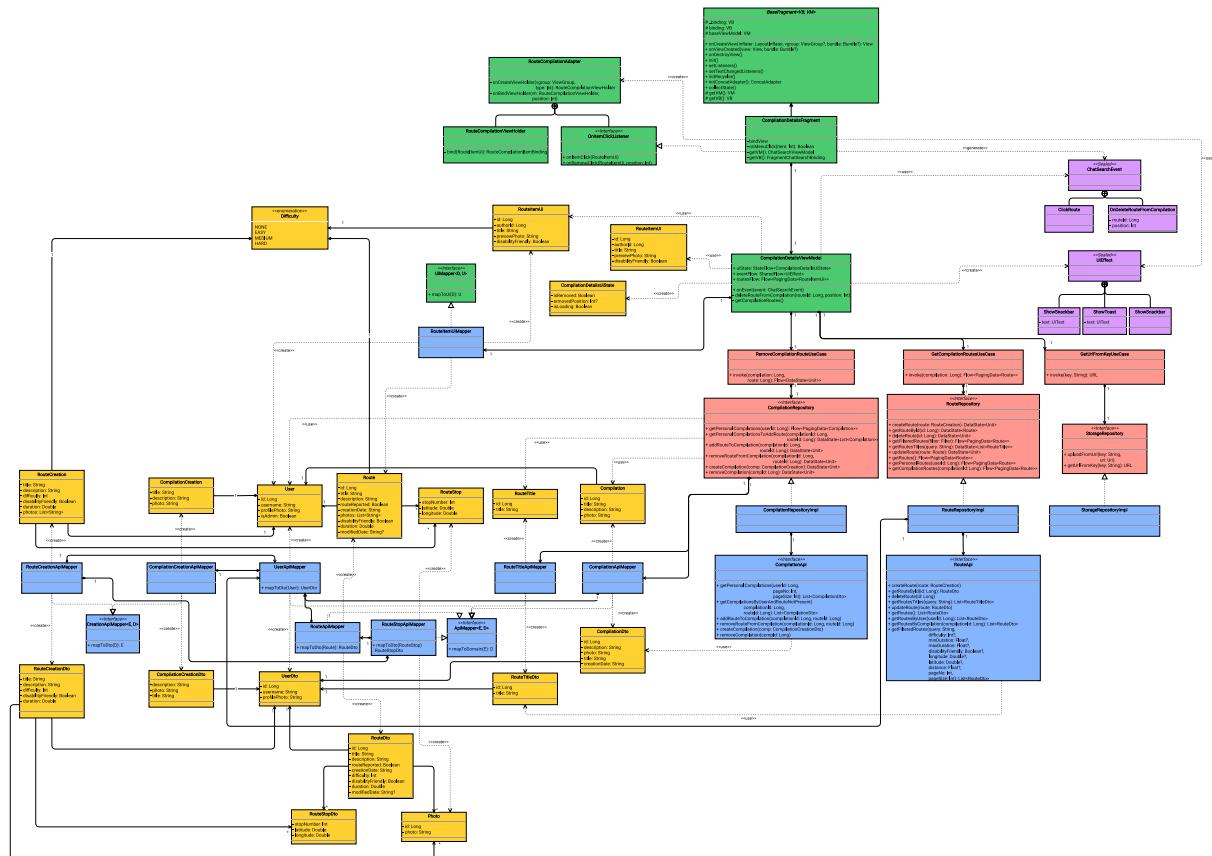


Figura 41: Dettaglio compilation e rimozione itinerario da compilation

3.4.3 Aggiunta di un itinerario

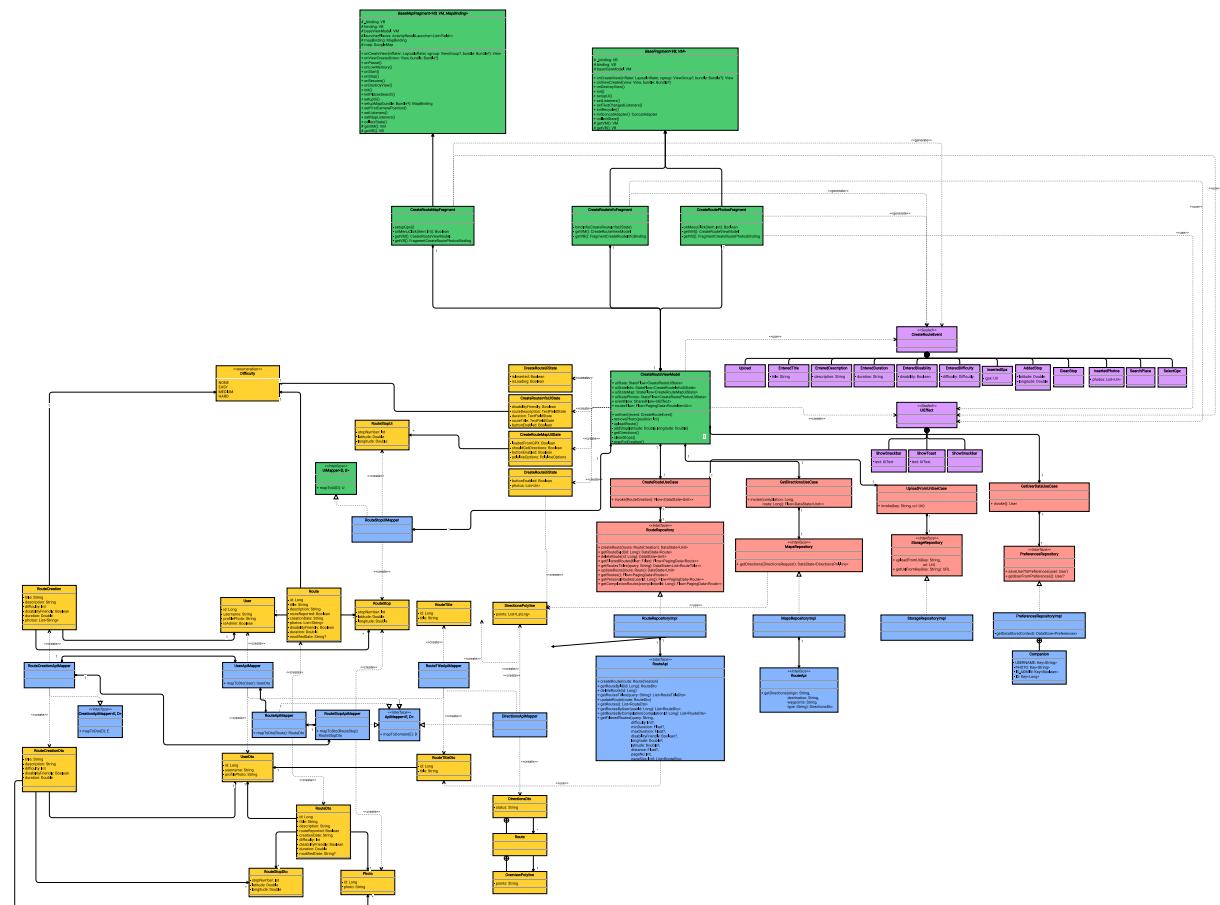


Figura 42: Aggiunta di un nuovo itinerario

3.5 Diagrammi di sequenza di design

Sono di seguito presentati i diagrammi di sequenza di design per due casi d'uso significativi.

3.6 Definizione delle gerarchie funzionali

4 Definizione di un piano di testing e valutazione sul campo dell’usabilità

4.1 Valutazione dell’usabilità

Parte fondamentale nel processo creativo dell’applicazione è sempre capire in che direzione andare per migliorare l’esperienza dei clienti. Per realizzare ciò siamo ricorsi ad alcune tecniche per testare l’usabilità del nostro prodotto. L’approccio da noi adottato richiede di far eseguire dei task a un gruppo di utenti con differenti retroscena.

Tali task possono avere un riscontro differente a seconda dell’utente:

- L’utente è stato in grado di compiere il task tramite gli strumenti offerti dall’app.
- L’utente non è stato in grado di compiere il task tramite gli strumenti offerti dall’app.
- L’utente è stato in grado di compiere il task, ma ha trovato difficoltà oppure trova l’interfaccia poco intuitiva.

Verranno fornite solo indicazioni di base, il sistema dovrebbe essere intuitivo, in caso di una percentuale di riscontri altamente negativa il sistema sarà modificato.