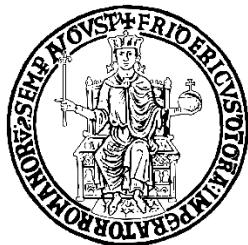


UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II
SCUOLA POLITECNICA E DELLE SCIENZE DI BASE
DIPARTIMENTO DI INGEGNERIA ELETTRICA E TECNOLOGIE DELL'INFORMAZIONE



CORSO DI LAUREA IN INFORMATICA
INSEGNAMENTO DI INGEGNERIA DEL SOFTWARE
ANNO ACCADEMICO 2021/2022

NaTour

Un nuovo modo di vivere il trekking

Contraenti:

Andrea PEPE

MATRICOLA N86003197

andrea.pepe2@studenti.unina.it

Marcello RUSSO

MATRICOLA N86003235

marcello.russo@studenti.unina.it

Committente:

SoftEngUniNA

Indice

Glossario	6
Introduzione	10
Cosa permette di fare NaTour?	10
Contenuto della documentazione	10
Contatti	11
Modello funzionale	12
Requisiti funzionali.....	12
Inserimento nuovo itinerario	12
Inserimento punti di interesse escursionistico	12
Registrazione nuovo percorso.....	13
Ricerca itinerari	13
Download percorso	13
Segnalazione percorso	13
Email promozionale.....	14
Requisiti non funzionali.....	14
Scalabilità	14
Sicurezza	14
Performance	15
Modellazione dei casi d'uso	16
Tabelle di Cockburn.....	17

Mockup	22
Presentazione dell'idea progettuale.....	46
Progettazione.....	48
Individuazione del target di utenti e Personas	49
Persona 1	50
Persona 2	51
Persona 3	52
Valutazione dell'usabilità a priori	54
Task 1	56
Task 2	57
Task 3	57
Riferimenti	58
Prototipazione funzionale via Statechart dell'interfaccia grafica	59
Modelli di Dominio Nella sezione seguente analizziamo i modelli di dominio durante la fase di analisi dei requisiti.....	61
Class Diagram di Analisi.....	62
Sequence Diagram di Analisi	67
Activity Diagram.....	69
Design del Sistema	75
Approccio API-Driven Development	75
Analisi Architetturale.....	75
Tecnologie utilizzate.....	76

Backend.....	77
Pattern MVC.....	77
Autenticazione	80
Salvataggio dei dati	81
Server.....	81
Gestione del server, manutenzione e rilascio aggiornamenti.....	82
Rest API	85
Front End.....	86
Scelta del design pattern.....	87
Pattern MVVM	88
Logging e analitica	92
Riferimenti	93
Class Diagram di Design	94
Sequence Diagram di Design.....	100
Gerarchie Funzionali	102
Codice sorgente sviluppato.....	103
Licenza	103
Testing	104
Middleware Auth	106
Registrazione utente	108
Login Utente.....	115
Aggiunta nuovo percorso	120

Valutazione dell'usabilità sul campo	137
Metodo euristico.....	137
Logging	140

Glossario

Area geografica: in NaTour si intende la città in cui si trovano le coordinate di inizio del percorso selezionato. In particolare, l'attributo è di tipo dinamico e viene definito in modo automatico dall'app.

Backend: infrastruttura che si interpone tra un client e un database. Si tratta a tutti gli effetti di un software che risiede in un server remoto con cui è possibile comunicare tramite delle API.

BYCRYPT: è una funzione di hashing di password basata sulla cifratura Blowfish, un algoritmo di crittografia basato su chiave simmetrica a blocchi. Oltre ad incorporare un salt per proteggere le password contro attacchi Rainbow Table è anche una funzione adattiva: col tempo, il conteggio dell'iterazione può essere aumentato per renderla più lenta, in modo da essere resistente ad attacchi di tipo Brute Force.

Data Breach: una violazione di sicurezza che comporta, accidentalmente o in modo illecito, la distruzione, la perdita, la modifica, la divulgazione non autorizzata o l'accesso ai dati personali trasmessi, conservati o comunque trattati. Una violazione dei dati personali può compromettere la riservatezza, l'integrità o la disponibilità di dati personali.

Database: un database è una raccolta organizzata di informazioni strutturate, o dati, generalmente archiviati elettronicamente in un sistema informatico. Un

database è solitamente controllato da un sistema di gestione del database (DBMS).

Difficoltà: in NaTour la difficoltà dei percorsi è espressa tramite la scala di valutazione delle difficoltà escursionistiche secondo le direttive del Club Alpino Italiano per quanto riguarda l'escursionismo. In particolare, la scala prevede le seguenti possibilità: T (turistico), E (escursionistico), EE (escursionisti esperti), EEA (escursionisti esperti con attrezzatura).

Durata: tempo necessario per completare il percorso selezionato. Questa informazione viene trasmessa all'utente nella schermata di dettaglio del percorso e durante la ricerca è possibile inserire un filtro riguardo la durata massima.

Framework: suite di strumenti utile allo sviluppatore software per ridurre i tempi di lavoro. Un framework, infatti, si occupa di tutti quei task necessari per lo sviluppo di ogni applicazione; dunque, al programmatore non resta che occuparsi delle funzioni richieste dall'applicativo.

GPX: o GPS Exchange Format, è uno schema XML progettato come formato di dati GPS comune per applicazioni software. Può essere utilizzato per descrivere waypoint, tracce e rotte. È un formato aperto e può essere utilizzato senza la necessità di pagare commissioni di utilizzo.

HTTPS: Hypertext Transfer Protocol Secure è un'estensione del Hypertext Transfer Protocol (HTTP). Viene utilizzato per comunicare in modo sicuro su una rete di computer ed è ampiamente utilizzato su Internet.

Lunghezza: lunghezza espressa in chilometri del percorso selezionato. Questa informazione viene trasmessa all'utente nella schermata di dettaglio del percorso e durante la ricerca è possibile inserire un filtro riguardo la lunghezza massima.

Milestone: indica un importante traguardo intermedio che deve essere raggiunto dal team di progetto.

Percorso: è l'entità fondamentale dell'applicativo NaTour rappresentata da informazioni quali titolo, descrizione, tracciato geografico, durata, lunghezza, area geografica, eventuali punti di interesse, difficoltà e attributo che indichi l'accessibilità a disabili.

Punto di interesse: è associato ad un percorso e rappresenta un luogo o un'attività ritenuta interessante da chi pubblica un percorso. Un punto di interesse è costituito da un titolo, una descrizione, una categoria e delle coordinate geografiche che ne identificano la posizione lungo il percorso.

Scalabilità: la scalabilità denota in genere la capacità di un sistema di aumentare o diminuire di scala in funzione delle necessità e disponibilità. Un sistema che gode di questa proprietà viene detto scalabile.

Stakeholder: sono tutti quei soggetti portatori di un interesse specifico in un'impresa e dunque interessati al buon andamento dell'impresa stessa.

Testing: è un metodo per verificare se il prodotto software realmente soddisfa i requisiti previsti e per garantire che il prodotto software sia privo di difetti.

Tracciato del percorso: rappresenta una serie di coordinate geografiche che identificano la sequenza delle posizioni nello spazio col passare del tempo.

UI: rappresenta il design degli elementi interattivi di un sito Web o di una app, inclusi pulsanti, cursori e altri elementi di design (user interface).

UX: si riferisce all'esperienza utente (user experience) e alla "sensazione" complessive che un utente sperimenta quando interagisce con un prodotto o servizio.

Introduzione

NaTour è il nuovo social network dedicato agli escursionisti commissionato dalla società SoftEngUniNA.

La realizzazione di tale progetto è stata affidato al gruppo INGSW2122_N_30 di cui fanno parte gli studenti Andrea Pepe, matricola N86003197, e Marcello Russo, matricola N86003235.

Cosa permette di fare NaTour?

Il progetto nasce come una soluzione software cloud based e verranno col tempo sviluppati i client per le diverse piattaforme. Al momento del lancio, il progetto avrà la sua app ufficiale Android.

Contenuto della documentazione

La documentazione è suddivisa in quattro macro-blocchi, ossia:

- Modello funzionale (documento dei requisiti software)
- Design del sistema
- Codice sorgente
- Testing
- Valutazione dell'usabilità sul campo

Contatti

Il software rilasciato al committente rappresenta una versione pronta alla commercializzazione.

Per richiedere assistenza in caso di malfunzionamenti o per richiedere lo sviluppo di nuove funzionalità, potete contattare uno dei membri del team di sviluppo

Andrea Pepe

N86003197

andrea.pepe2@studenti.unina.it

Marcello Russo

N86003235

marcello.russo@studenti.unina.it

Modello funzionale

In questa sezione descriveremo il modello funzionale dell'applicativo analizzando i requisiti funzionali e non funzionali definiti con gli stakeholder in fase di specifica dei requisiti.

Requisiti funzionali

Autenticazione

Un utente può registrarsi/autenticarsi. È apprezzata la possibilità di autenticarsi utilizzando account su altre piattaforme come Google o Facebook.

Inserimento nuovo itinerario

Un utente autenticato può inserire nuovi itinerari (sentieri) in piattaforma. Un sentiero è caratterizzato da un nome, una durata, un livello di difficoltà, un punto di inizio, una descrizione e un tracciato geografico che lo rappresenta su una mappa. Il tracciato geografico deve essere inseribile manualmente (interagendo con una mappa interattiva) oppure tramite file in formato standard GPX.

Inserimento punti di interesse escursionistico

Un utente può inserire punti di interesse escursionistico/naturalistico pertinenti un certo itinerario. Questi punti sono caratterizzati da una tipologia

(e.g.: sorgente, punto panoramico, area pic-nic, baita, flora, grotte, luoghi di interesse artistico, altro), e una posizione geografica, e vengono mostrati sulla mappa nel dettaglio del sentiero.

Registrazione nuovo percorso

Un utente può inserire un tracciato geografico per un sentiero anche registrando una sequenza di posizioni GPS con il proprio dispositivo mobile, all'interno dell'app NaTour.

Ricerca itinerari

Effettuare ricerche di itinerari tra quelli presenti in piattaforma, con possibilità di filtrare i risultati per area geografica, per livello di difficoltà, per durata, e per accessibilità a disabili.

Download percorso

Un utente può scaricare le informazioni riguardo un sentiero in formato GPX, e stampare le informazioni riepilogative in formato PDF.

Segnalazione percorso

Un utente può segnalare informazioni inesatte/non aggiornate riguardo un sentiero. Una segnalazione è caratterizzata da un titolo e da una descrizione. I sentieri per cui sono presenti segnalazioni di inesattezza mostrano un warning nella schermata di dettaglio relativa.

Email promozionale

Gli amministratori possono inviare email promozionali su accessori per escursionisti a tutti gli iscritti.

Requisiti non funzionali

Usabilità

Abbiamo impiegato molto tempo per studiare la UX e offrire all'utente il miglior prodotto dal punto di vista della semplicità e piacevolezza d'uso. L'obiettivo è stato quello di realizzare una UI veloce, affidabile e che potesse essere utilizzata da tutto il target degli utenti senza difficoltà o dubbi.

Scalabilità

L'obiettivo principale che ci siamo prefissati è stato quello di realizzare un backend scalabile in modo che con il crescere dell'app e degli utenti potesse essere sempre performante ed efficiente nel gestire le richieste dei client. Per raggiungere questo obiettivo è stato necessario adoperare un Framework versatile e modulare che potesse adattarsi ad ogni esigenza futura.

Sicurezza

La privacy dei nostri utenti viene prima di tutto e per questo motivo l'intera infrastruttura è stata realizzata seguendo solidi principi di sicurezza. In particolare, le comunicazioni tra client e server sono trasmesse attraverso il

protocollo HTTPS per evitare che i dati trasmessi possano essere letti da malintenzionati. Per quanto riguarda invece il salvataggio delle password, abbiamo adoperato la tecnica di criptografia BYCRYPT così da non poter mai risalire alla password. In questo modo malauguratamente si dovesse verificare una Data Breach del Database le password resteranno sicure.

Performance

Il tempo di risposta per effettuare ogni operazione all'interno della nostra applicazione mobile è nell'ordine dei decimi del secondo, se non inferiore. Le animazioni e la loro durata sono state studiate in modo da fare della fluidità uno dei pregi di questo software. Tali performance sono state ottenute adottando e ottimizzando il pattern MVVM(Model-View-ViewModel) e progettando l'app per avere il minor impatto possibile sulle risorse del dispositivo su cui è in uso.

Modellazione dei casi d'uso

Gli Use Case Diagram, nello standard UML, sono diagrammi che illustrano le funzioni o i servizi offerti da un sistema, così come sono percepiti e utilizzati dagli attori che interagiscono con il sistema stesso.

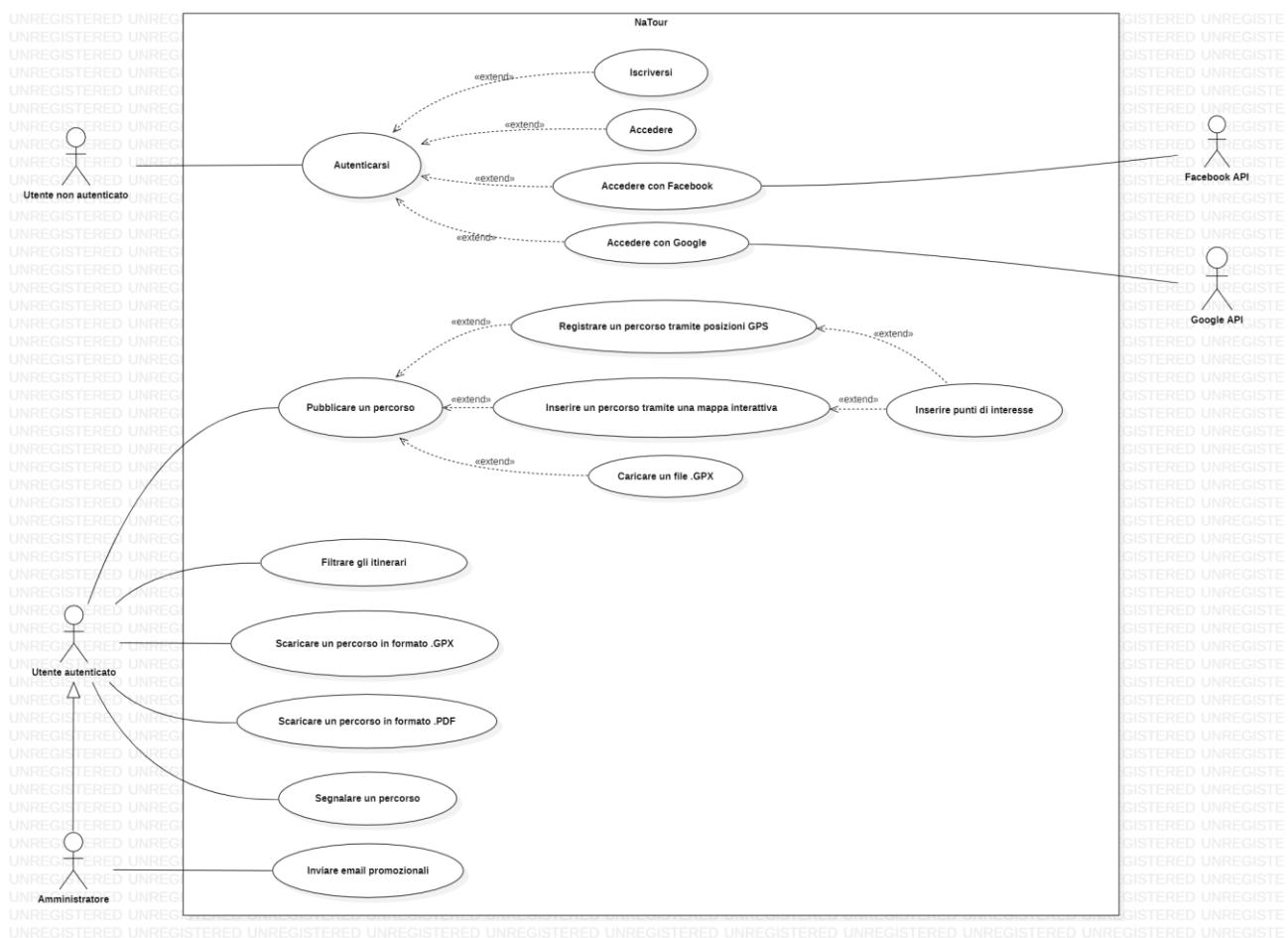


Tabelle di Cockburn

Le tabelle di Cockburn sono una tecnica di modellazione dei casi d'uso in cui vengono descritti i possibili passaggi che possono essere compiuti durante l'esecuzione del caso d'uso in questione. È possibile ampliare la tabella con estensioni e variazioni.

Di seguito sono riportati due casi d'uso come richiesto dai committenti.

Use Case	Registrazione percorso		
Obiettivo	Un utente vuole inserire un tracciato geografico per un sentiero registrando una sequenza di posizioni GPS con il proprio dispositivo mobile		
Precondizioni	L'utente deve essere autenticato, aver fornito l'autorizzazione alla geolocalizzazione e averla attivata		
Attore	Utente autenticato		
Condizione di Successo	L'utente ha caricato il suo percorso in piattaforma		
Trigger	L'utente clicca sul bottone “Registra percorso” nel mockup “Nuovo percorso 2”		
Descrizione	Step n°	Attore	Sistema
	1	Preme il bottone “Registra percorso”	
	2		Visualizza mockup “Registra percorso 1”
	3	Preme il primo bottone da sinistra per terminare la sessione	
	4		Visualizza mockup “Riepilogo 1”
	5	Riempie i campi obbligatori ("Nome" e "Descrizione")	
	6	Preme la spunta verde in alto a destra	
	7		Mostra popup di successo
	8		Visualizza mockup “Nuovo percorso 1”
Estensione 1:	Step n°	Attore	Sistema

L'utente aggiunge un punto di interesse al proprio percorso	2.1	Preme il primo bottone da destra per aggiungere un punto di interesse	
	2.2		Visualizza mockup “Registra percorso 3”

Use Case	Login		
Obiettivo	Un utente vuole autenticarsi sulla piattaforma con il proprio account		
Precondizioni	Nessuna		
Attore	Utente non autenticato		
Condizione di Successo	L'utente riesce ad accedere		
Trigger	L'utente clicca sul bottone “Continua con l'email” nel mockup “Start”		
Descrizione	Step	Attore	Sistema
	1	Clicca sul bottone “Continua con l'email” nel mockup “Start”	
	2		Visualizza mockup “Sign In”
	3	Riempie i campi obbligatori (“Email” “Password”) e preme il bottone “Accedi”	
	4		Visualizza mockup “Homepage”
Estensione 1: L'utente inserisce un indirizzo email non registrato	Step	Attore	Sistema
	2.1	Riempie i campi obbligatori (“Email” “Password”) e preme il bottone “Accedi”	
	2.2		Visualizza mockup “Sign In – Email non registrata”
Estensione 2: L'utente inserisce una password errata	Step	Attore	Sistema
	2.1	Riempie i campi obbligatori (“Email” “Password”) e preme il bottone “Accedi”	

	2.2		Visualizza mockup “Sign In – Password errata”
Estensione 3: L'utente non inserisce alcuna password	Step 2.1	Riempie soltanto il campo “Email” e preme il bottone “Accedi”	
	2.2		Visualizza mockup “Sign In – Password obbligatoria”
Estensione 4: L'utente non inserisce alcuna email	Step 2.1	Riempie soltanto il campo “Password” e preme il bottone “Accedi”	
	2.2		Visualizza mockup “Sign In – Email obbligatoria”
Estensione 5: L'utente lascia i campi vuoti	Step 2.1	Preme il bottone “Accedi”	
	2.2		Visualizza mockup “Sign Up – Email e password obbligatorie”

Mockup

In ambito grafico, il termine “mockup” si riferisce a quei particolari elementi che servono a mostrare la resa finale di un elaborato grafico. È, in pratica, una simulazione della realtà che permette di capire come potrebbe essere prodotto, realizzato o stampato un progetto.

Ci è risultato più semplice sviluppare l'app realizzando prima tutti i mockup necessari, motivo per il quale li illustreremo tutti anziché solo quelli di due casi d'uso, saranno presenti anche i mockup “di supporto” alle tabelle di Cockburn menzionati nelle stesse. Facciamo notare che alcuni elementi potrebbero differire dal risultato ottenuto utilizzando Android Studio, avendo a disposizione elementi nativi di Android integrati con maggiore semplicità.



NaTour

Unisciti alla più grande
community di escursionisti!



Continua con Facebook



Continua con Google



Continua con l'email

Start



Bentornato

Email

Password

Accedi

oppure

Iscriviti

Sign In



Bentornato

Email

Email non registrata nei nostri sistemi

Password

[Accedi](#)

oppure

[Iscriviti](#)

[Sign In - Email non registrata](#)



Bentornato

Email

Password

Riprova inserendo le credenziali corrette

Accedi

oppure

Iscriviti

Sign In - Password errata



Bentornato

Email

Il campo email è obbligatorio

Password

Accedi

oppure

Iscriviti

Sign In - Email obbligatoria



Bentornato

Email

Password

Il campo password è obbligatorio

Accedi

oppure

Iscriviti

Sign In - Password obbligatoria



Bentornato

Email

Il campo email è obbligatorio

Password

Il campo password è obbligatorio

Accedi

oppure

Iscriviti

Sign In - Email e Password obbligatorie



Entra a far parte della nostra
community

Nickname

Email

Password

Conferma password

Iscriviti

oppure

Accedi

Sign Up



**Entra a far parte della nostra
community**

Nickname

Il nickname è stato già preso

Email

L'email è stata già utilizzata

Password

La password deve contenere almeno 8 caratteri

Conferma password

La password non corrisponde

Iscriviti

oppure

Accedi

Sign Up - Errori 1



**Entra a far parte della nostra
community**

Nickname

Il nickname è stato già preso

Email

Inserisci un'email valida

Password

La password deve contenere almeno 8 caratteri

Conferma password

La password non corrisponde

Iscriviti

oppure

Accedi

Sign Up - Errori 2

NaTour



@marcellorusso



Nome percorso molto
EEA originale
Napoli

@marcellorusso



Homepage

< @marcellorusso



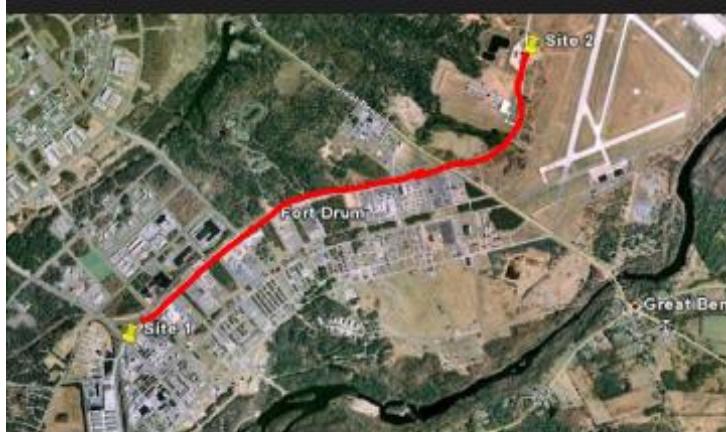
T Nome percorso molto originale

Napoli

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Distanza: 1,2 km

Durata: 0h:30min:12sec



GPX



PDF

Dettaglio percorso

 Segnala percorso 

Titolo

Inserisci un titolo

Corpo

Inserisci una descrizione



Segnala percorso



Filtri



Nel raggio di

km

Distanza

km

Durata

h

Accessibilità a disabili



Difficoltà



T



E



EE



EEA

Azzera

Aggiorna



Filtri

NaTour



@marcellorusso

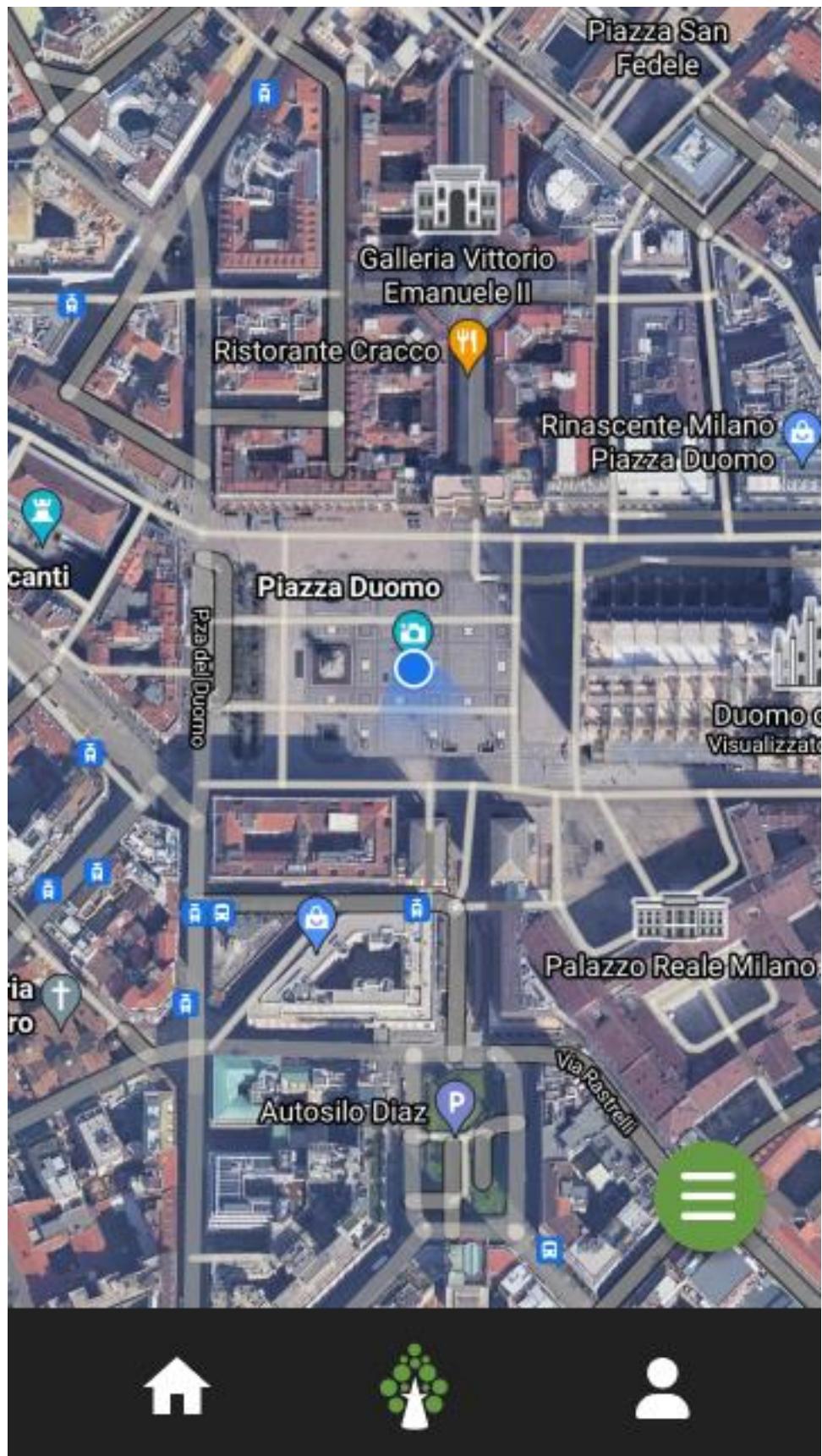


EE Nome percorso molto
originale
Napoli

@marcellorusso



Homepage filtrata



MapPage



MapPage 2



Registra percorso 1



Punto di interesse

Nome

Aggiungi un nome

Descrizione

Inserisci una descrizione

Categoria

Seleziona una categoria



Registra percorso 2



Registra percorso 3

X

Riepilogo

✓

Nome del percorso

Aggiungi un nome al percorso

Descrizione

Inserisci una descrizione

Difficoltà

Seleziona una difficoltà

Accessibilità a disabili



Riepilogo percorso

Account



Nome utente

username

Email

email@domain.com

Invia mail promozionale



Account

 Email promozionale 

Titolo

Inserisci un titolo

Corpo

Inserisci un corpo



Email promozionale

Presentazione dell'idea progettuale

Cos'è NaTour

NaTour nasce come hub per unire appassionati di trekking ed escursioni al fine di condividere in modo semplice e diretto le proprie avventure. Si configura quindi come un social network il cui contenuto principale sono i percorsi di tipo escursionistico, creati e condivisi dagli utenti registrati alla piattaforma sia che essi siano esperti o novizi alle prime armi. Attualmente il prodotto è disponibile soltanto per i dispositivi Android, mentre le versioni iOS e Web arriveranno nei prossimi mesi.

Per accedere alla piattaforma è prevista la registrazione dell'utente il quale dovrà fornire e-mail, username e password. In alternativa, gli utenti non interessati a creare una nuova utenza potranno procedere attraverso i loro account Google o Facebook.

Una volta effettuato l'accesso, gli utenti potranno navigare e scoprire i percorsi presenti in piattaforma attraverso un'interfaccia studiata con lo scopo di essere intuitiva e di facile utilizzo anche per i meno esperti. Per gli escursionisti più esigenti è possibile filtrare i risultati per difficoltà, lunghezza, durata e distanza dalla propria posizione in modo da poter trovare solo i percorsi di proprio interesse.

Per ogni percorso gli utenti hanno a disposizione diverse informazioni contenute nella pagina di dettaglio del percorso; oltre alle informazioni fondamentali come titolo, descrizione, durata, lunghezza e area geografica, sarà possibile visualizzare il tracciato del percorso direttamente su una mappa e una lista dei punti di interesse che si incontreranno lungo il cammino. Per gli utenti più esigenti, sarà possibile esportare i dettagli del percorso in formato PDF ed il tracciato in formato GPX. Infine, chiunque trovi delle inesattezze su un percorso in piattaforma può segnalarlo direttamente in app e qualunque percorso già segnalato in precedenza sarà riconoscibile grazie ad un badge visibile nella pagina di dettaglio.

Quanto all'inserimento dei percorsi è previsto procedere in tre modi così da soddisfare tutte le esigenze. Il primo è per coloro che amano addentrarsi nell'avventura senza un percorso ben definito e consiste nel registrare con il proprio smartphone il tracciato seguito attraverso il GPS. Vi è poi la possibilità di inserire un percorso, con i relativi punti di interesse, interagendo direttamente con una mappa; - in particolare - durante l'inserimento saranno visibili due bottoni, uno per entrare in "modalità punti geografici" del percorso e l'altro per entrare in "modalità punto di interesse". Il terzo metodo è previsto per quegli utenti che utilizzano altri dispositivi di registrazione del tracciato consentendo loro di poter caricare un file GPX con i dettagli del percorso.

Quanto alle utenze previste dalla piattaforma queste sono di due tipi: Utente normale e Admin. Un Admin può usare l'app allo stesso modo di un utente qualunque ma avrà in aggiunta alcune funzioni dedicate proprio alla gestione della piattaforma come, ad esempio, l'invio di materiale promozionale a tutti gli iscritti. In futuro è prevista la possibilità di avere report dettagliati e integrazione con altri strumenti di marketing.

Progettazione

Per quanto riguarda la fase progettuale seguiremo il modello di processo a cascata per la fase di progettazione, analisi dei requisiti e interfaccia utente e il modello di processo iterativo per la parte di coding e testing. In questo modo seguendo il processo a cascata possiamo procedere per milestone e distribuire il lavoro nel modo più efficiente, così da mantenere alta la produttività del team. Invece, il processo iterativo per la fase di coding e testing permette di avere sempre un software utilizzabile e testabile fin da subito.

Individuazione del target di utenti e Personas

Dall'analisi condotta si evince che le quote più cospicue di escursioni nei vari trimestri del 2007 corrispondono agli adulti tra i 35 e i 44 anni, che hanno realizzato il 21,6% del complesso delle escursioni dell'anno; se si aggiungono le escursioni effettuate da chi ha tra i 25 e i 34 anni si raggiunge quasi il 40% del totale delle escursioni. Le escursioni sono effettuate nella stessa misura da uomini e donne (nel 2007 quelle effettuate dagli uomini sono il 49,9%, quelle delle donne il 50,1%), con modeste variazioni nei vari trimestri. Quindi il profilo rilevato è quello di una persona che abbia dai 25 ai 44 anni circa, usi uno smartphone e sia interessata a condividere i propri percorsi su un social network realizzato ad hoc per gli escursionisti. L'interfaccia e le funzionalità sono state implementate in modo da soddisfare in maniera più efficiente possibile le esigenze del target individuato. Ad esempio, abbiamo preso in considerazione la struttura di altri social network utilizzati dal target così da rendere l'ambiente familiare. Inoltre, abbiamo posto particolare attenzione all'ottimizzazione delle main feature (registrazione e conseguente aggiunta di un percorso) così da avere un'esperienza semplice e intuitiva anche nelle funzionalità estranee all'utente, peculiarità di questo software.

I suddetti dati sono stati ricavati da "[ISTAT - Le escursioni per motivi personali in Italia](#)"

Di seguito riporteremo le Personas individuate.

Persona 1

Davide, Studente

Su di me

- 25
- Celibe
- Medio spendente
- Competenza tecnologica media
- Device usati: Smartphone e fitness tracker
- Social usati: Instagram e Tiktok

Biografia

Davide da un anno è diventato escursionista. Fino a oggi ha sempre percorso itinerari da solo, ma vorrebbe poter far avvicinare anche i suoi amici a questo mondo.

Bisogni

Per coinvolgere i suoi amici avrebbe bisogno di tracciare accuratamente i suoi percorsi, così da mostrarglieli.

Obiettivi

Davide vuole un'app che gli permetta di conoscere altre persone con la stessa sua passione.

Difficoltà

Ha spesso difficoltà a trovare percorsi nuovi, ma soprattutto validi, nella sua zona.

Persona 2

Carlotta, Impiegata e micro-influencer

Su di me

- 36
- Nubile
- Medio spendente
- Competenza tecnologica media
- Device usati: Smartphone e Smartwatch
- Social usati: Instagram, Facebook e Twitter

Biografia

Carlotta ama esplorare i percorsi naturali della sua zona. Ogni sua avventura è

ricca di foto ricordo che spesso pubblica sui social. Carlotta ha una piccola ma fedele community su Instagram.

Bisogni

Portare contenuti sempre nuovi sui social è difficile, per questo ha bisogno di un'app che raccolga i percorsi più interessanti da esplorare.

Obiettivi

Carlotta vuole un'app che sia incentrata esclusivamente sulla sua passione per le escursioni.

Difficoltà

Il problema che incontra più spesso è condividere i suoi percorsi sui social.

Persona 3

Simone, CEO

Su di me

- 40
- Sposato
- Medio - Alto spendente
- Competenza tecnologica bassa-media
- Device usati: Smartphone

- Social usati: Facebook, Twitter, LinkedIn

Biografia

Simone è escursionista da ormai 10 anni, ogni domenica mattina si raduna con i suoi amici storici e percorre spesso i soliti sentieri. È praticamente diventato un esperto, spesso infatti aiuta i novizi che incontra sul suo cammino.

Bisogni

Spesso con i suoi amici scommette su quanti chilometri abbiano percorso, ma non avendo un modo accurato per verificarlo non sanno rispondersi. Simone ha quindi bisogno di poter verificare l'estensione delle sue escursioni.

Obiettivi

Vorrebbe poter condividere la sua esperienza su un'app dedicata, tutti gli altri social che utilizza sono invasi da persone che conosce e che non hanno nulla a che fare con l'escursionismo.

Difficoltà

A volte è successo che nei suoi percorsi dovesse tornare indietro a causa di un problema naturale. Ha quindi bisogno di essere avvisato se un percorso è agibile.

Valutazione dell'usabilità a priori

Abbiamo investito molto tempo ed energie nell'usabilità del nostro software perché crediamo che anche il migliore dei progetti non sarà apprezzato su vasta scala senza l'adozione di tecniche di sviluppo che sono compatibili con l'ingegneria dell'usabilità. Pertanto, abbiamo realizzato l'applicativo seguendo le 8 regole d'oro di Ben Shneiderman, concentrandoci in particolare sulla prima (Consistenza prima di tutto), sulla quinta (Prevenire gli errori) e sull'ottava (Ridurre il carico di memoria a breve termine).

Per quanto riguarda la prima regola abbiamo reso il design e le azioni da svolgere simili per funzioni simili. Ad esempio, la segnalazione di un percorso e l'invio di una mail promozionale da parte di un admin sono quasi identiche dato che in entrambe è necessario un titolo e un corpo e necessitano degli stessi buttoni. Dicasì lo stesso per il popup di inserimento di un nuovo punto di interesse e la schermata di riepilogo di un nuovo percorso, entrambi hanno un nome, una descrizione e altri campi simili, inoltre hanno gli stessi buttoni di annullamento (x) o di conferma (✓).

Sappiamo bene che è meglio prevenire che curare, è questo il motivo per cui tutti gli errori che l'utente può commettere non sono irreversibili e gli basterà leggere il popup di errore e andare avanti. In ogni caso la prevenzione vera e

propria è stata attuata inserendo – dove possibile – checkbox, combobox e switch per facilitare l'input di dati standard.

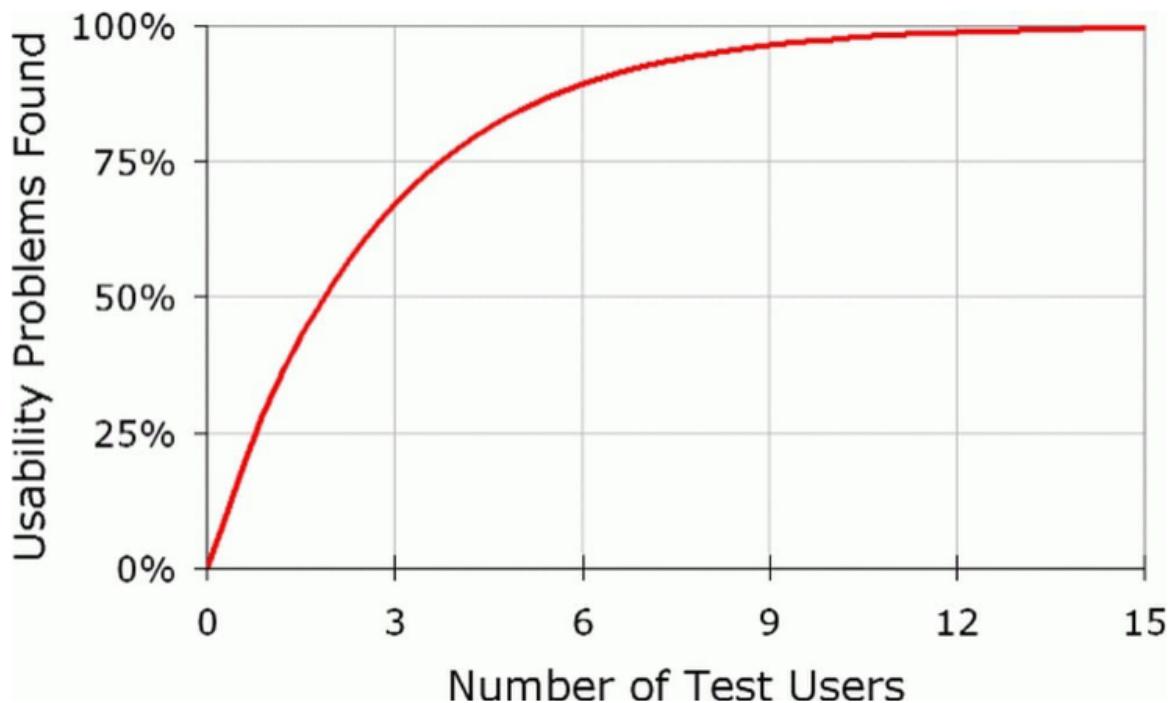
Circa il carico di memoria a breve termine, possiamo affermare che sia pari a zero dato che ogni schermata non ha nessun rimando alla precedente e non è necessario che l'utente ricordi niente di ciò che ha fatto prima.

Aver realizzato i mockup con Figma ci ha dato la possibilità di sfruttare la sua funzione “Prototype Mirror Share” per simulare il funzionamento del nostro applicativo. L'utente ha l'impressione che si tratti di una vera applicazione e può provare a completare dei task assegnatigli oppure valutarla soggettivamente (tecnica del Mago di Oz). È stato questo quello che abbiamo chiesto ai nostri tester quando gli abbiamo presentato il prototipo.

I task sono stati i seguenti:

1. Registrazione alla piattaforma;
2. Registrazione di un percorso;
3. Segnalazione di un percorso.

Per ognuno di essi è stato fissato un timer di cinque minuti.



La scelta di affidare la valutazione a cinque valutatori è stata fatta in base alla regola di Nielsen, che evidenzia il fatto che da cinque valutatori in poi la derivata della curva dei problemi di usabilità totali diminuisce notevolmente.

Task 1

Il primo task è stato completato da tutti i cinque valutatori con successo, sebbene uno dei tester ha avuto difficoltà perché i bottoni erano coperti dalla tastiera quando la quest'ultima era attiva, lo consideriamo un successo parziale.

Task 2

Uno dei valutatori non ha compreso appieno il funzionamento della registrazione del percorso tramite GPS e ha avuto difficoltà a completare l'obiettivo. Il fatto che non fosse chiaro quale bottone dovesse premere per avviare la registrazione non ha aiutato.

Task 3

Il terzo task è stato completato da tutti i cinque valutatori con successo.

	Task 1	Task 2	Task 3
Valutatore 1	S	S	S
Valutatore 2	S	F	S
Valutatore 3	S	S	S
Valutatore 4	S	S	S
Valutatore 5	P	S	S

- **S:** successo (1);
- **P:** successo parziale (0,5);
- **F:** fallimento (0).

Facendo i dovuti calcoli otteniamo un punteggio di 28,5/30 ovvero una percentuale di successo del 95%.

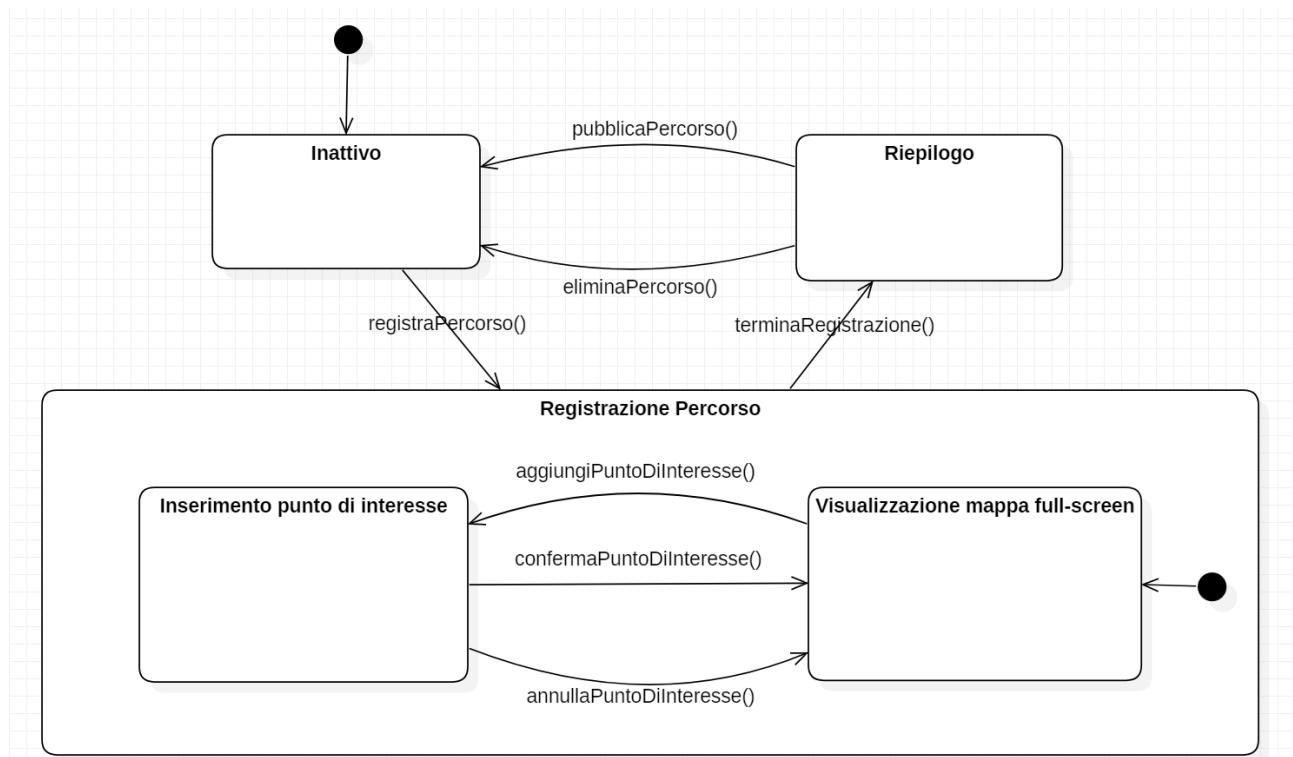
Sulla base dell'esito della valutazione siamo riusciti a riconoscere i principali problemi riscontrati. Difatti abbiamo da una parte spostato più in alto i bottoni necessari per la fase di registrazione e accesso, dall'altra abbiamo reso più intuitivo l'avvio e il riconoscimento della funzione di registrazione di un nuovo percorso.

Riferimenti

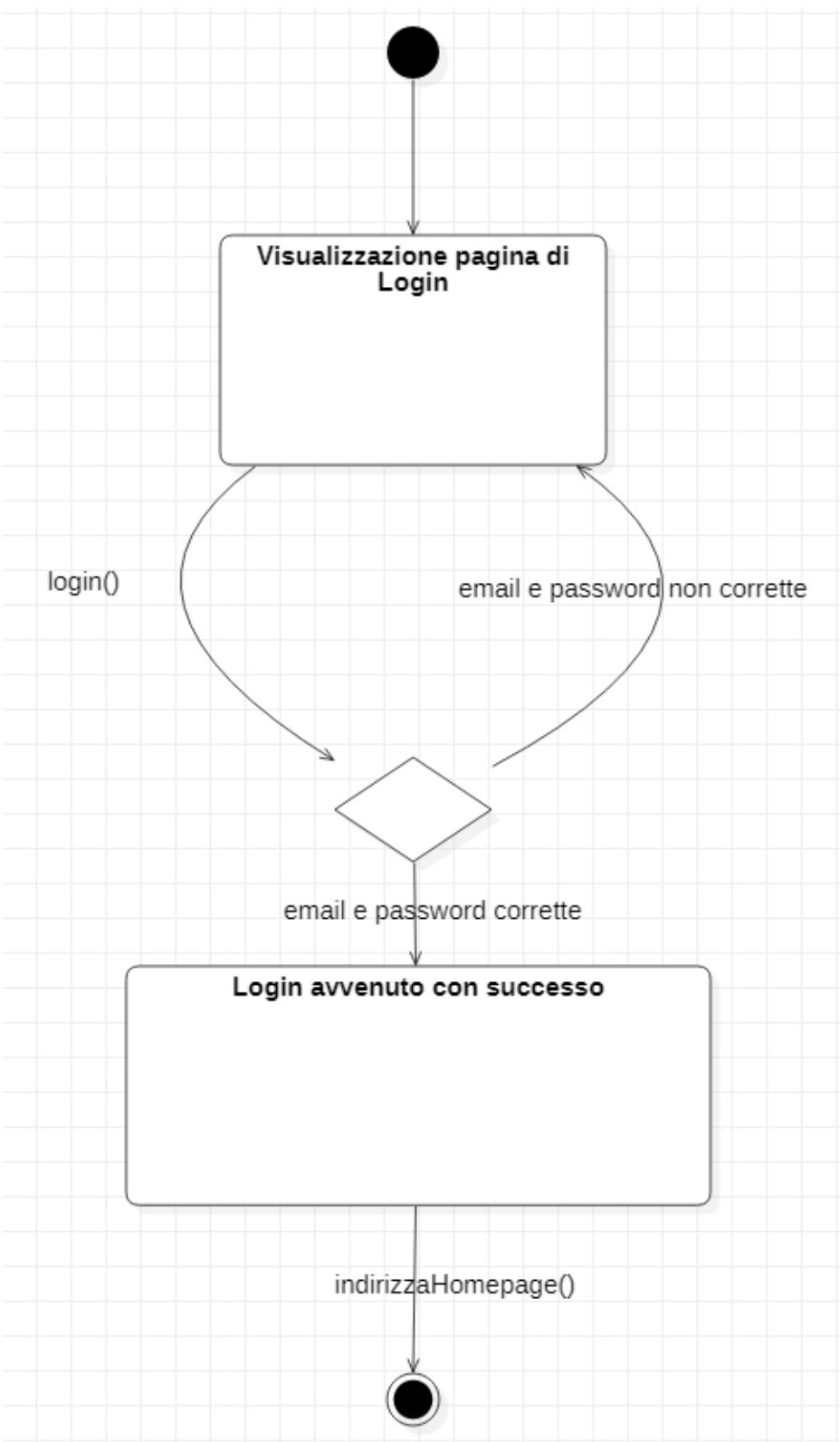
[Polillo R., Facile da usare – Una moderna introduzione alla ingegneria dell'usabilità, Edizioni Apogeo, 2010](#)

Prototipazione funzionale via Statechart dell'interfaccia grafica

La nostra scelta sui due casi d'uso di cui realizzare gli Statecharts è ricaduta sulla funzionalità di registrazione di un percorso e caricamento dello stesso e su quella di login.



Registrazione (tramite GPS) e pubblicazione di un percorso



Login

Modelli di Dominio

Nella sezione seguente analizziamo i modelli di dominio durante la fase di analisi dei requisiti.

Gli strumenti principali durante questa fase sono:

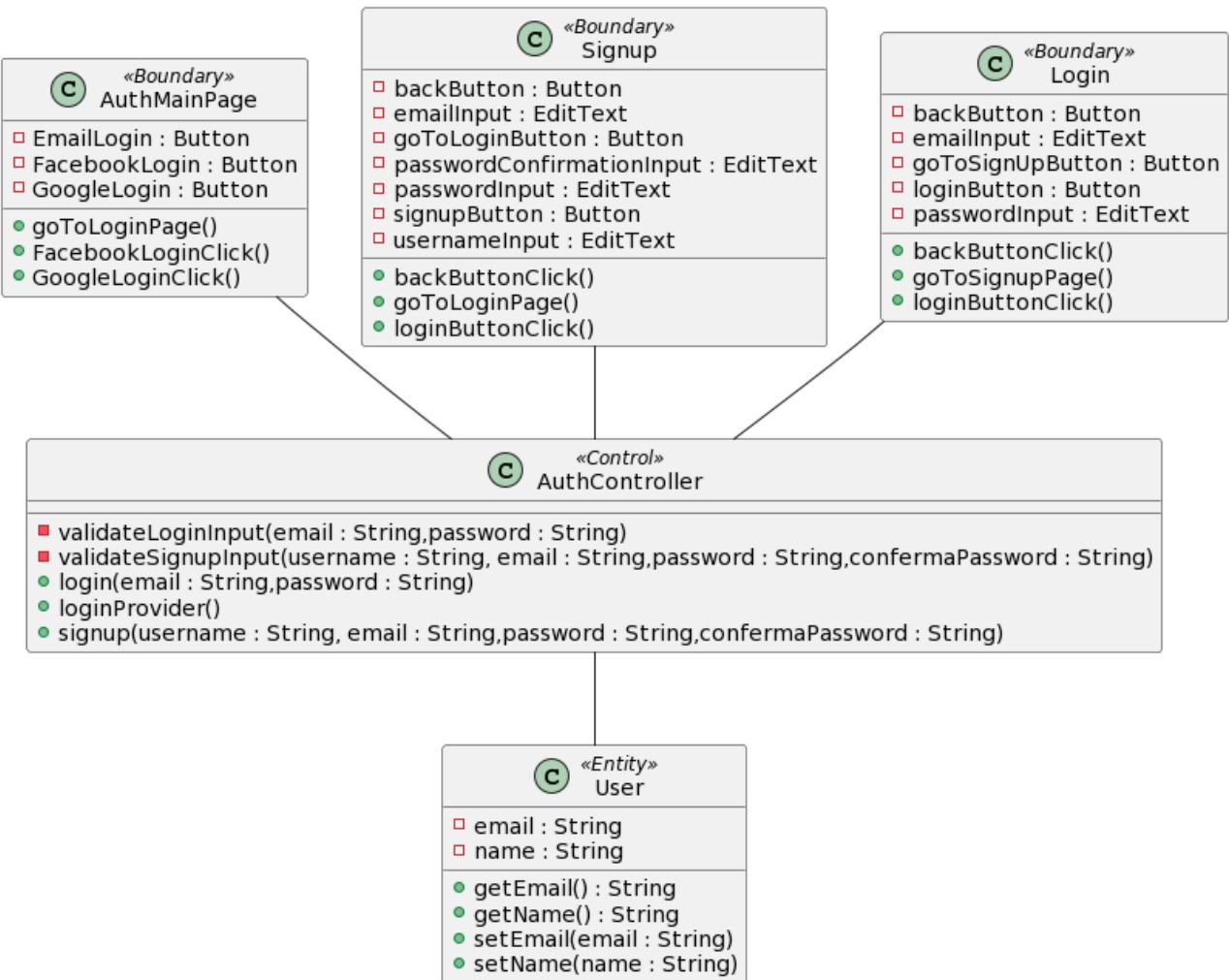
- Class Diagram
- Sequence diagram
- Activity Diagram

Per semplificare la comprensione e astrarre l'implementazione abbiamo usato l'euristica Three-Object-Type in modo da individuare classi e entità in questione.

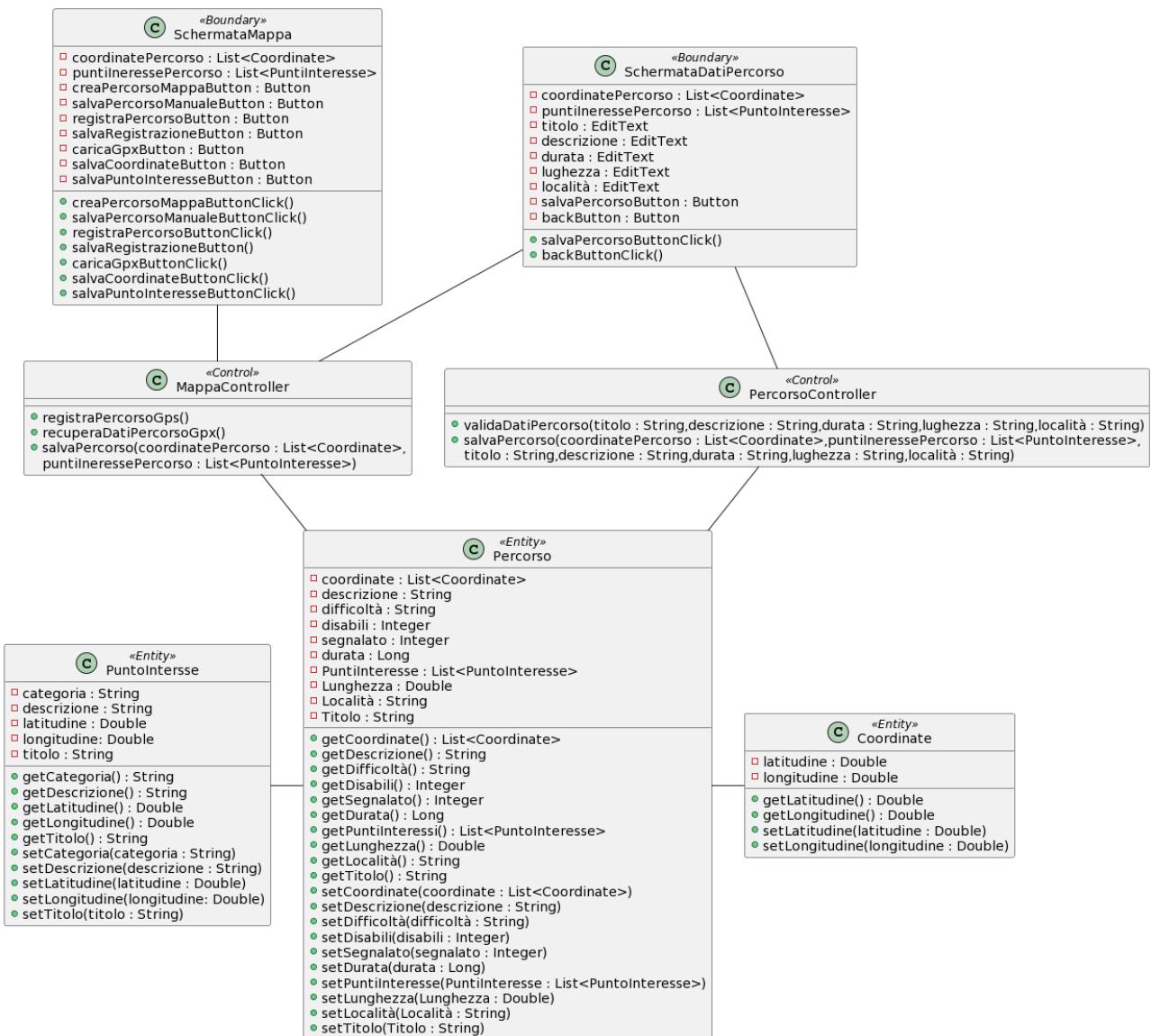
L'euristica Three-Object-Type è definita dai seguenti elementi:

- Entity: entità o modelli
- Boundary - rappresentano le interazioni tra utente e sistema (ad esempio la GUI).
- Control - rappresentano i controller o meglio la logica del sistema

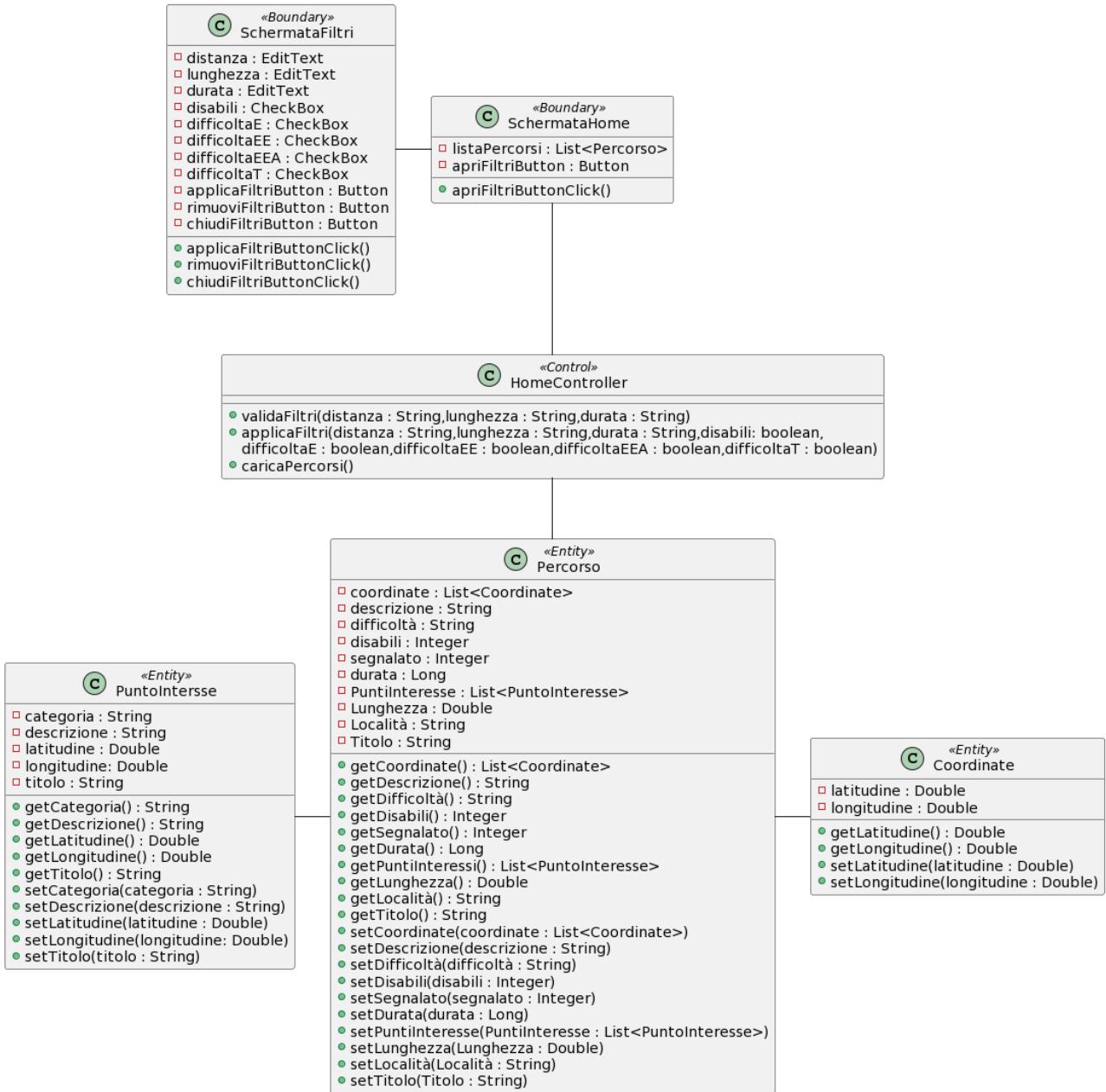
Class Diagram di Analisi



Autenticazione



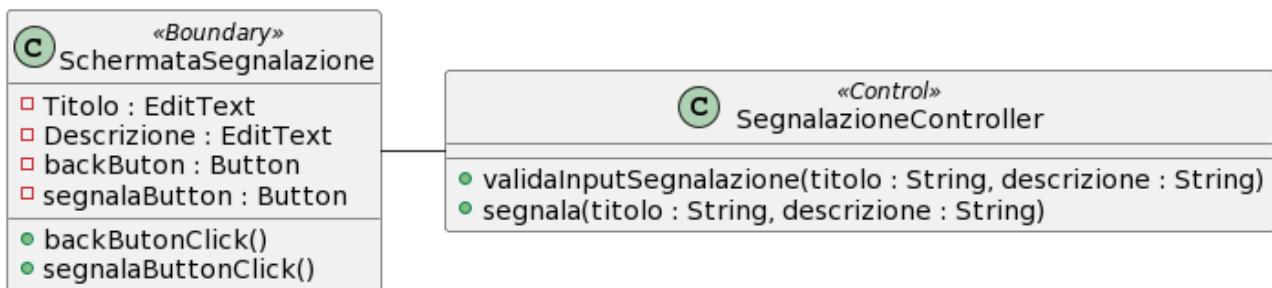
Pubblicazione percorso



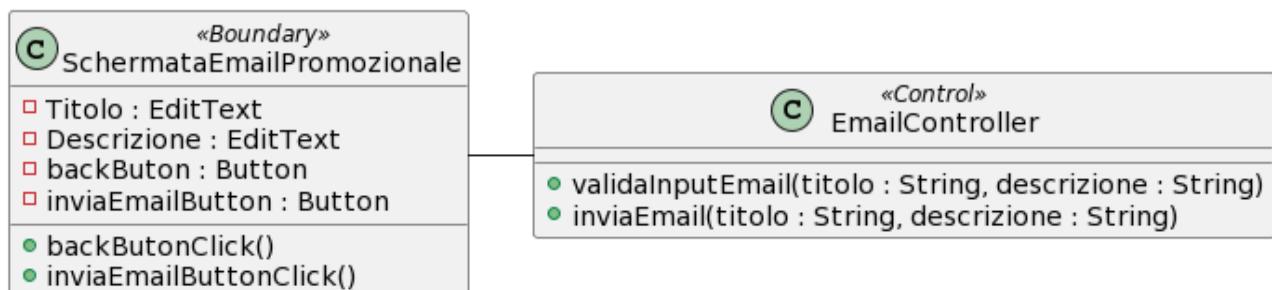
Filtro percorsi



Download GPX/PDF del percorso

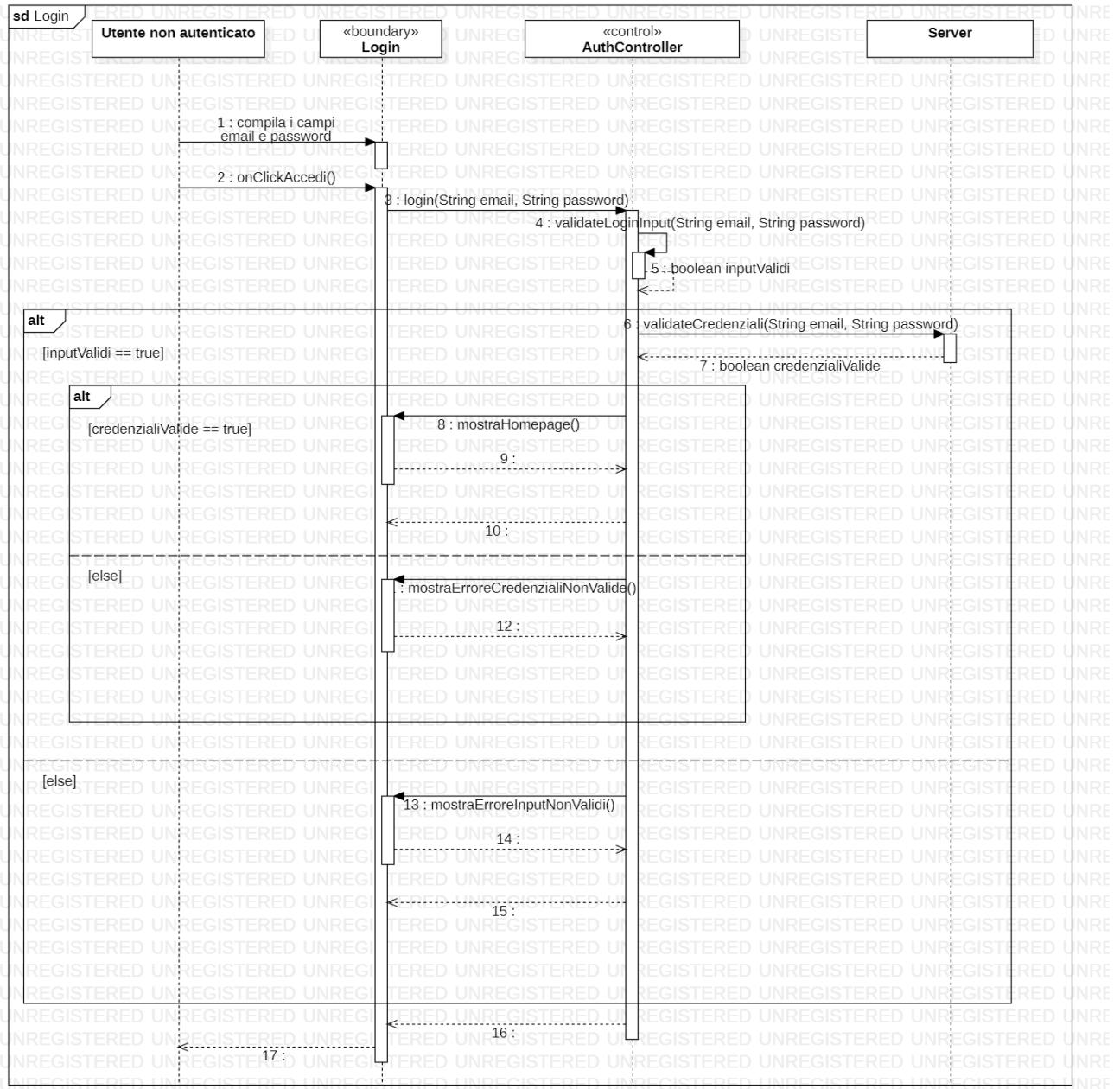


Segnalazione percorso

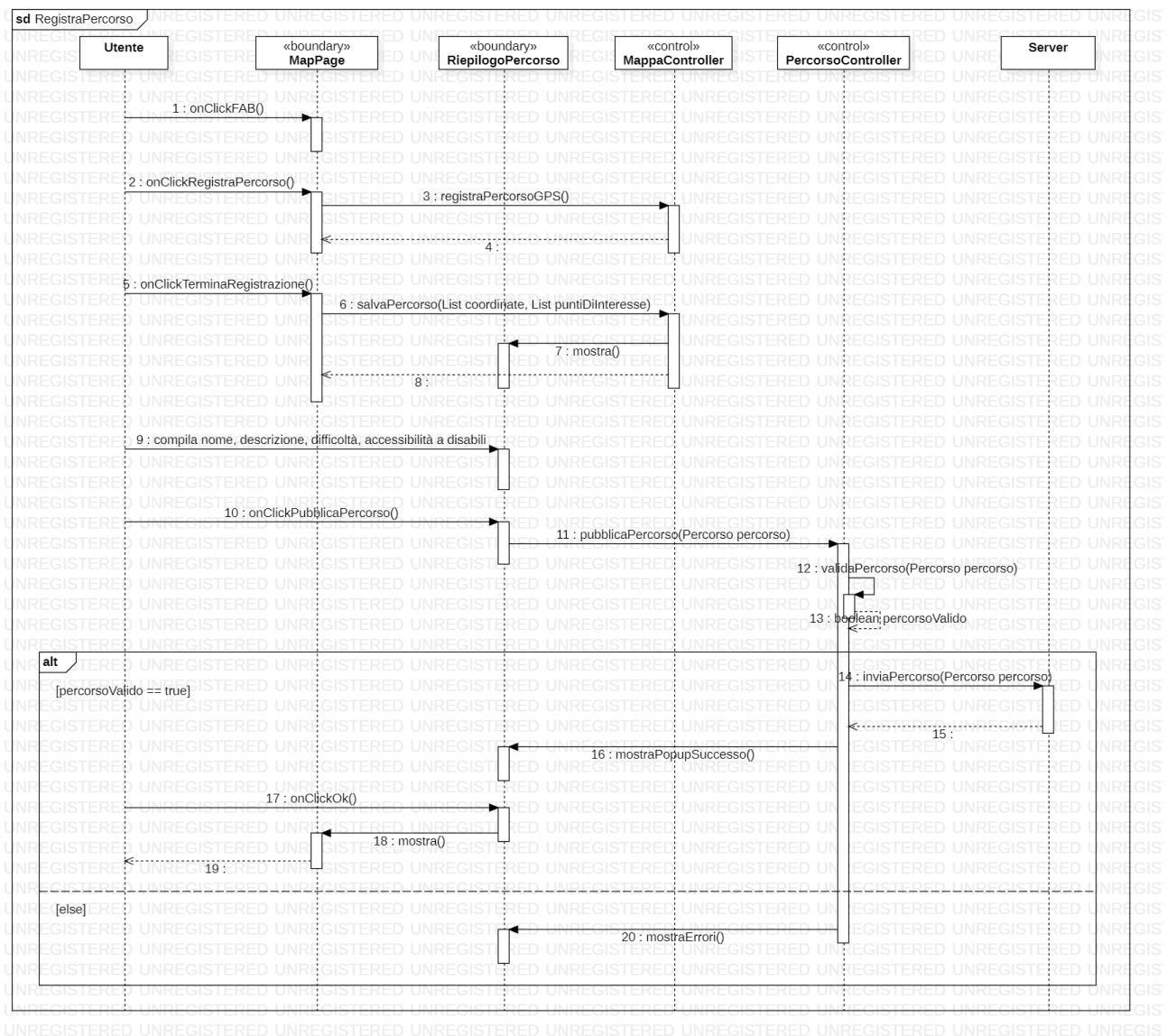


Email promozionale

Sequence Diagram di Analisi



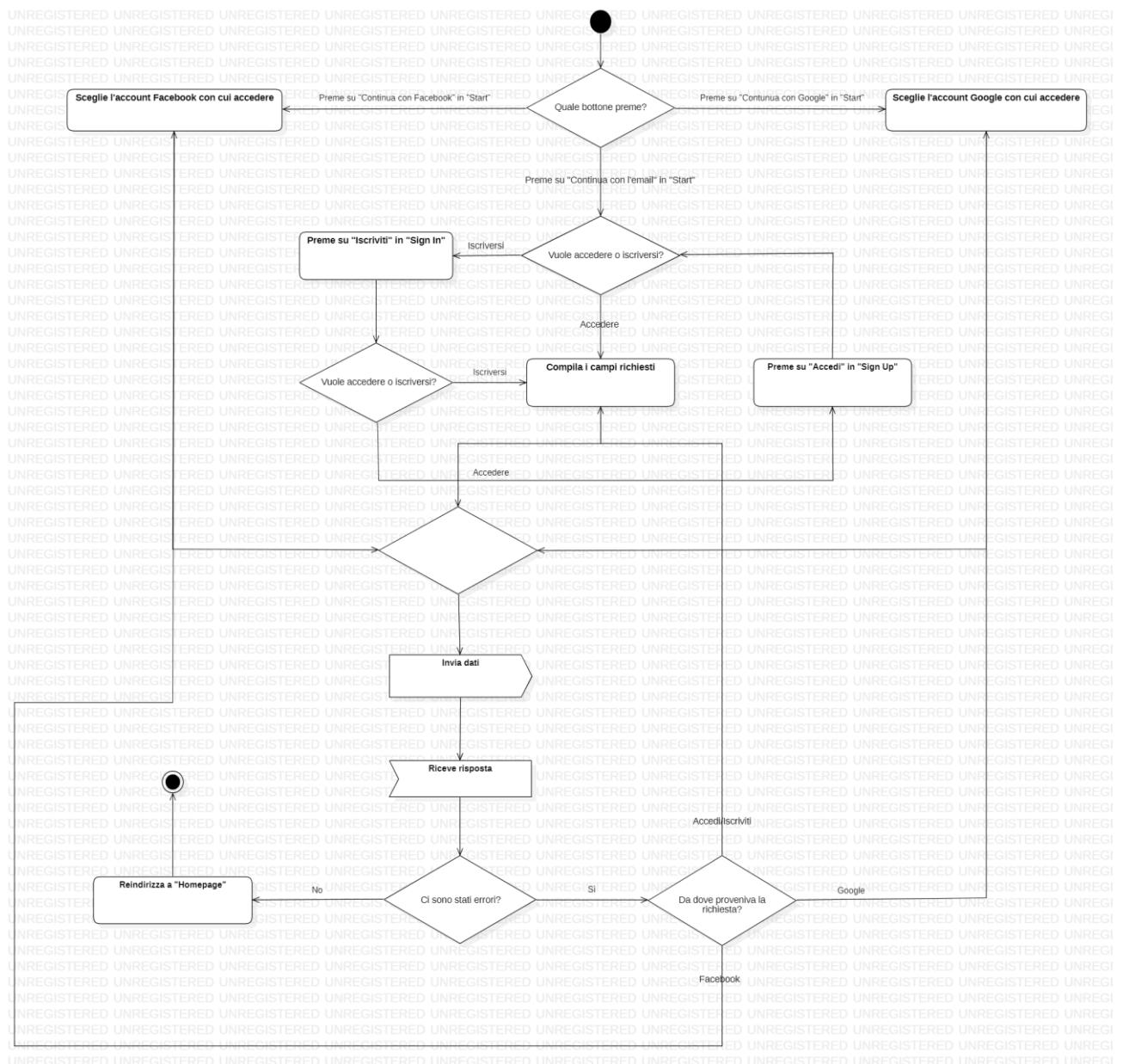
Login



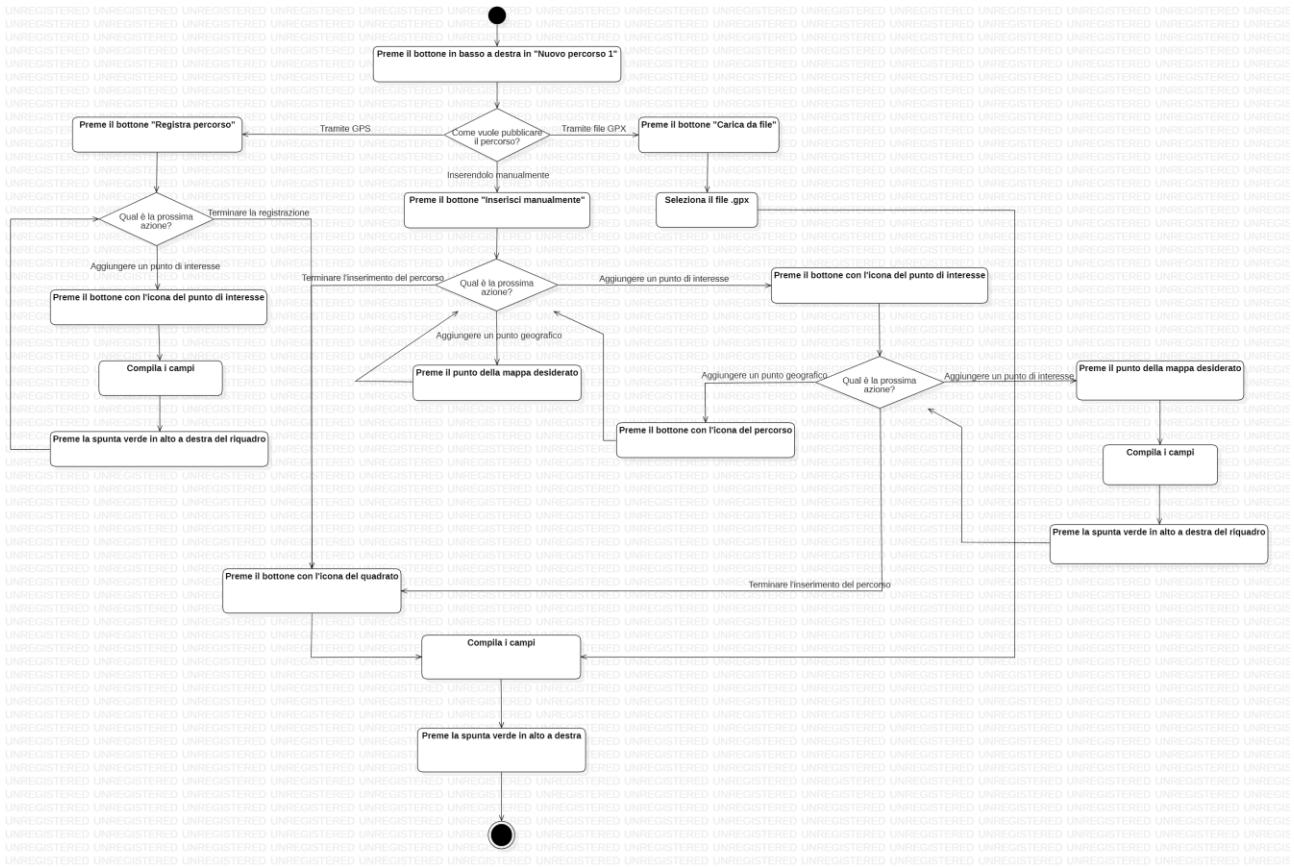
Registrazione (tramite GPS) e pubblicazione di un percorso

Activity Diagram

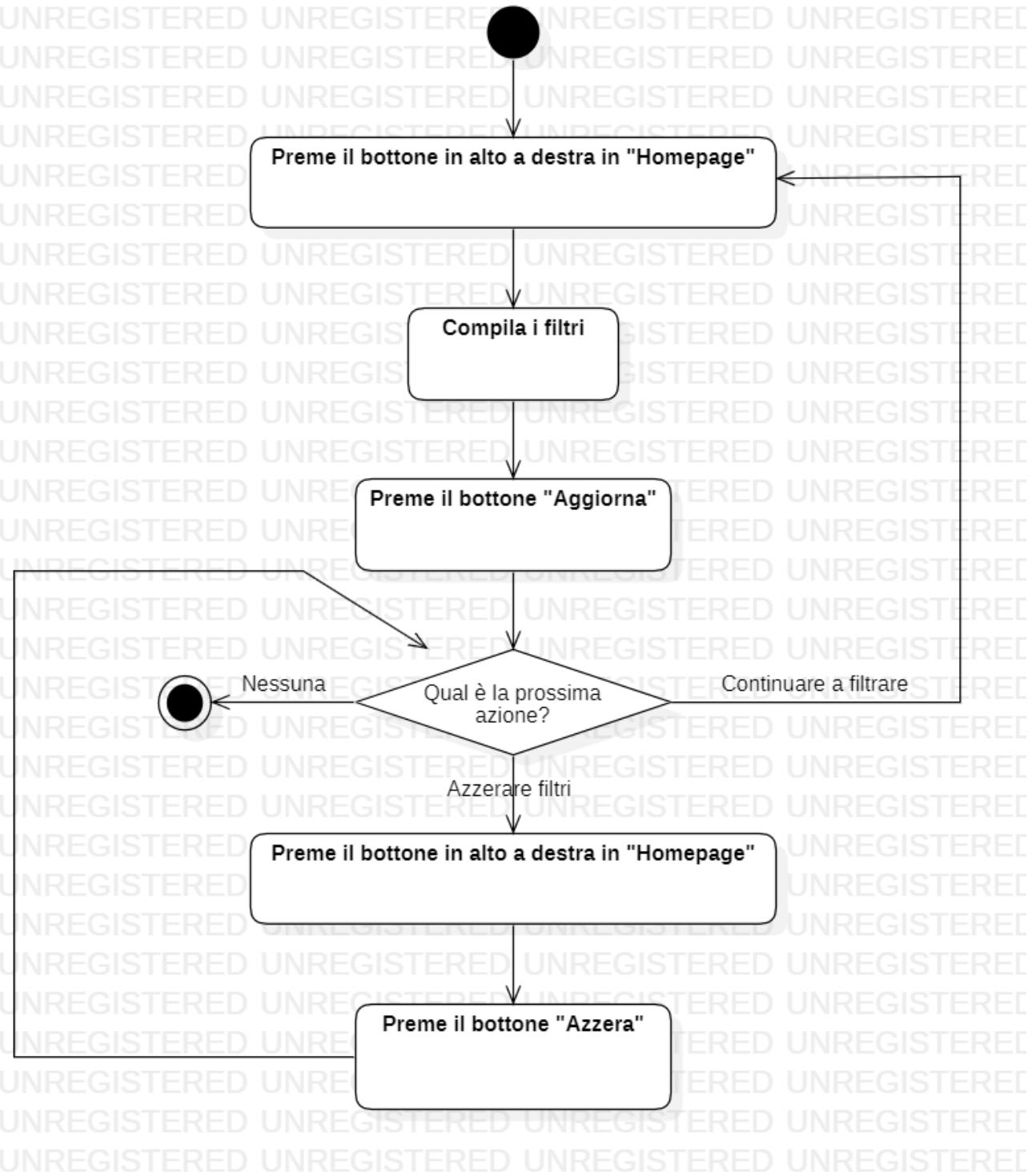
In questa sezione allegheremo i diagrammi di attività di tutti i casi d'uso dell'applicazione.



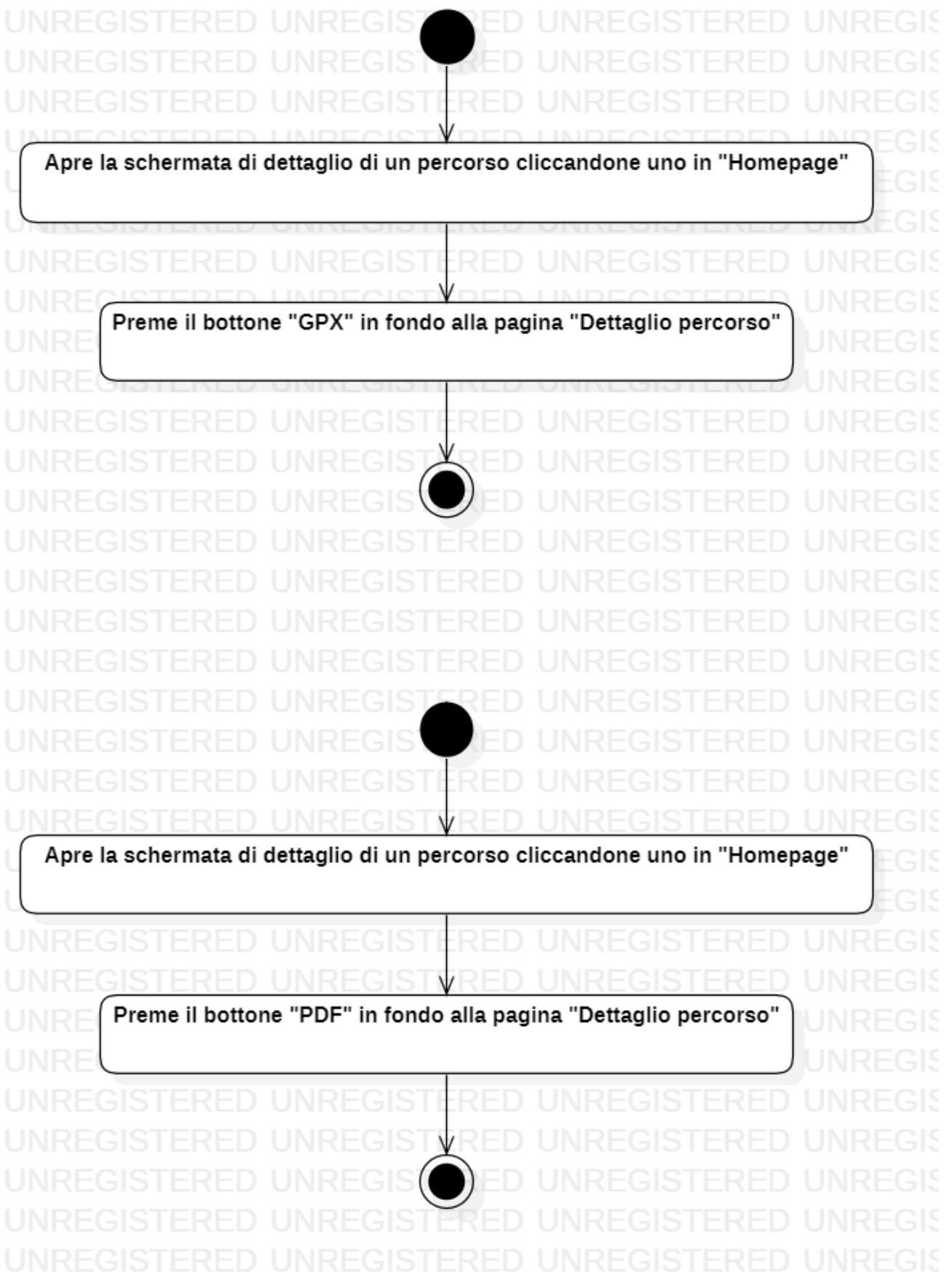
Autenticazione



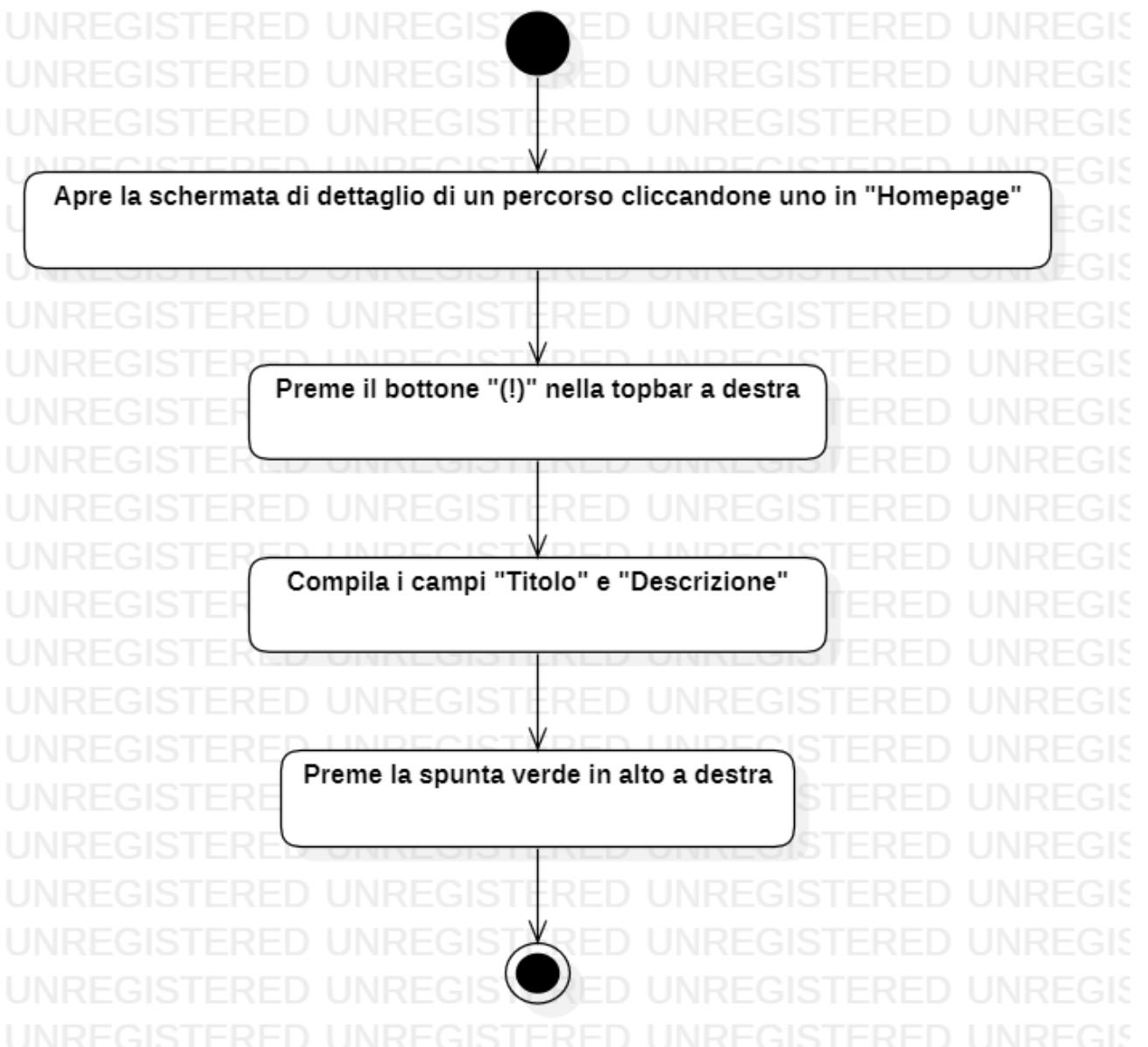
Pubblicazione percorso



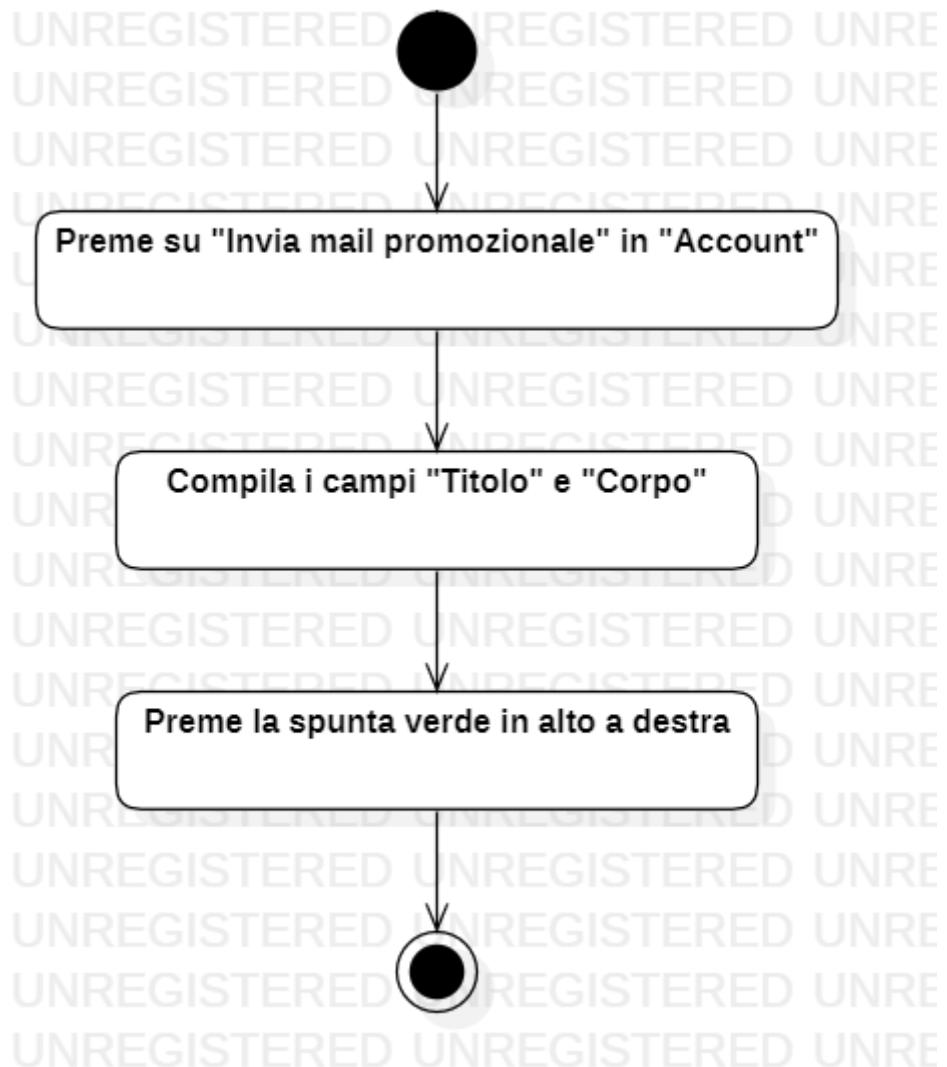
Filtro percorsi



Download GPX/PDF del percorso



Segnalazione percorso



Invio email promozionale

Design del Sistema

Analisi dell'Architettura

Introduzione

Nella seguente sezione andremo ad analizzare come è stata progettata l'applicazione e quali sono state le scelte implementative del team di sviluppo.

Approccio API-Driven Development

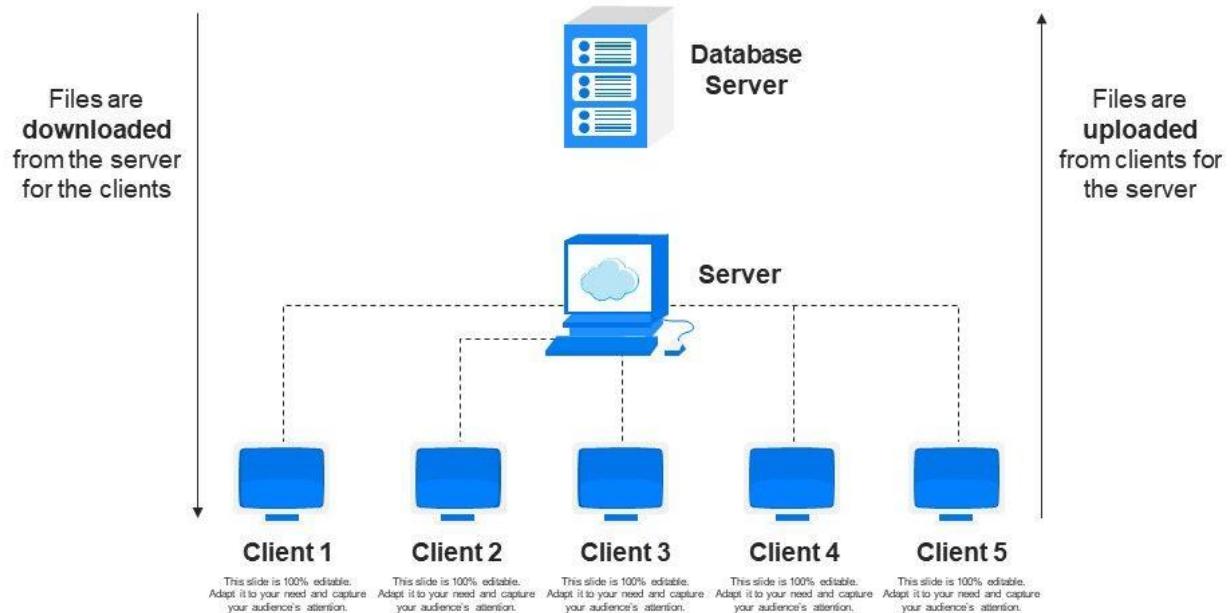
Data la natura dell'app si è optato per un approccio API-Driven, cioè una pratica di sviluppo in cui si vanno a realizzare prima le API e successivamente il resto dell'applicativo. Abbiamo deciso di muoverci in questa direzione in quanto l'obiettivo era quello di realizzare un sistema Cloud-Friendly e interamente accessibile tramite API così da non essere limitati, in futuro, di estendere i dispositivi supportati da NaTour. Lo scopo è dunque quello di avere un'app versatile che possa facilmente girare non solo sui dispositivi Android, ma anche su sistemi iOS, in ambiente Desktop o su Internet attraverso una web app. Inoltre, tale approccio ci consente anche di permettere ad altri developer di creare client alternativi per NaTour fornendo loro l'accesso e la documentazione dei nostri endpoint.

Analisi Architetturale

L'applicativo è stato sviluppato seguendo il pattern client-server; dunque, il

suo funzionamento è basato sulla comunicazione tra il frontend (client) ed il backend (server) attraverso il protocollo di comunicazione via internet HTTP secondo lo schema indicato di seguito.

Work of Client Server Model



Come visibile in foto, la comunicazione avviene tramite messaggi che prendono il nome di Request (Richiesta) e Response (Risposta). La prima va dal client verso il server, la seconda viceversa.

Tecnologie utilizzate

Nel seguente paragrafo analizziamo le tecnologie che sono state necessarie

per realizzare l'applicativo. Si specifica che le scelte prese dal team sono il frutto di uno studio approfondito non solo dei requisiti richiesti dal cliente ma anche degli obiettivi e delle evoluzioni future dell'applicativo. Inoltre, abbiamo tenuto conto anche dei costi da sostenere per la manutenzione e la scalabilità del software. Lavorando su questi fattori siamo riusciti a progettare NaTour abbattendo i costi di sviluppo e manutenzione senza rinunciare al fattore qualità del risultato ottenuto.

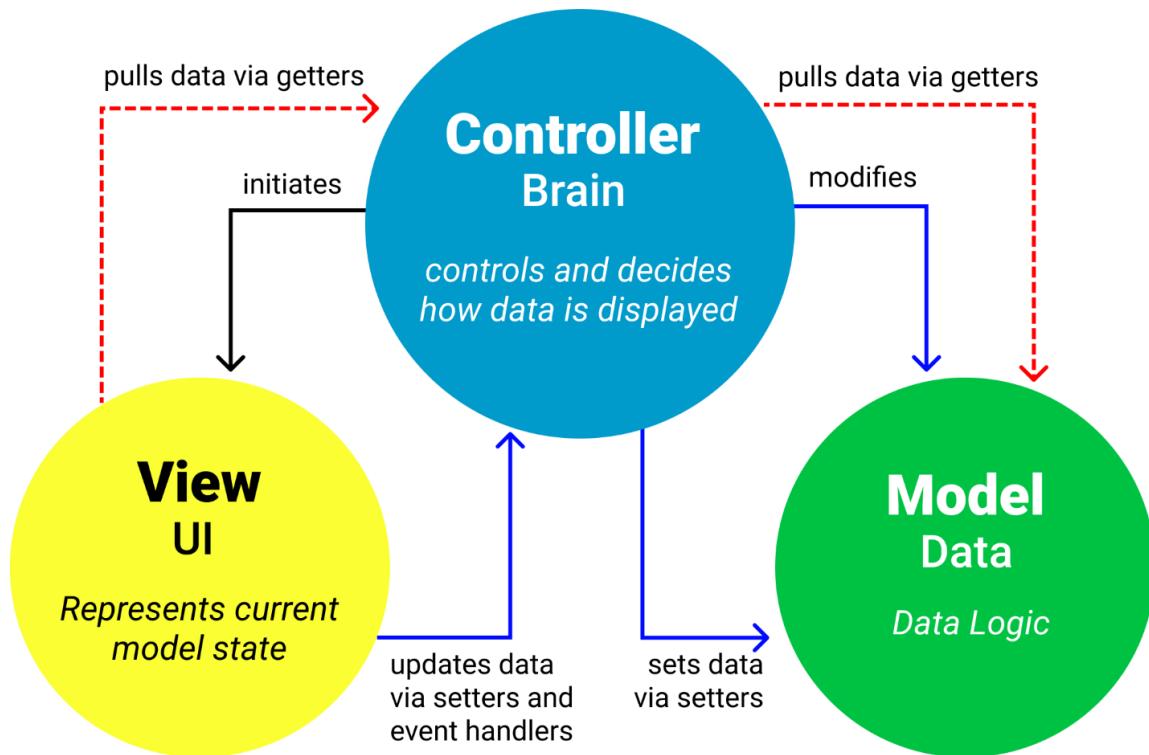
Backend

Per la realizzazione del Back End dell'applicativo abbiamo optato per il framework di sviluppo Laravel (versione 9). Si tratta di un framework PHP open source Object-Oriented basato sul design pattern MVC (Model-View-Controller) che negli ultimi anni si è guadagnato la fama di essere altamente efficiente, performante e affidabile. Inoltre, gode di una estrema flessibilità e modularità consentendo allo sviluppatore di usare solo le componenti di cui ha bisogno.

Pattern MVC

Come abbiamo anticipato, Laravel è realizzato su architettura Model-View-Controller il cui funzionamento è il seguente:

MVC Architecture Pattern



Analizzando lo schema possiamo dunque definire i componenti del pattern in questo modo:

- **Model**: sono le classi responsabili delle strutture dati per la rappresentazione degli oggetti e gestisce la business logic del software;
- **View**: sono responsabili della rappresentazione dei dati dei Model;
- **Controller**: rappresentano il cervello dell'applicazione facendo da intermediari tra i Model e le View. Inoltre, decidono quali operazioni compiere e come restituire i dati alle View.

Oltre alle classi proprie del pattern MVC, Laravel aggiunge alla lista innumerevoli Utility Class che consentono di incapsulare al meglio alcune componenti dell'applicativo.

Nel caso specifico di NaTour ci siamo serviti delle seguenti classi:

- Request: queste classi consentono di definire una particolare struttura per le richieste che vengono effettuate e consentono dunque di creare un sistema di validazione degli input efficace e sicuro;
- Trait: anche se si tratta di un componente del linguaggio PHP, Laravel tratta questi ultimi come le altre utility class. Le classi Trait rappresentano una soluzione per i linguaggi Object-Oriented a singola eredità. Nel nostro caso, ci siamo serviti dei Trait per creare una utility class in grado di formattare e uniformare le risposte JSON restituite dagli end point;
- Migration: classi utili per modellare il database definendo tabelle e relazioni;
- Middleware: come indicato dal nome si tratta di classi che lavorano come intermediari tra una richiesta e l'invocazione del metodo associato. Questi vengono di solito utilizzati per effettuare dei controlli preventivi e se non superati viene generata una risposta di errore.

Autenticazione

Per gestire l'autenticazione degli utenti sulla piattaforma abbiamo fatto uso del package ufficiale di Laravel "Sanctum". Questo componente consente di gestire l'autenticazione attraverso il rilascio di Token che inseriti opportunamente nelle richieste HTTP consentono di identificare l'utente. Dal punto di vista pratico, quando il client chiama l'endpoint di "Login" fornendo una valida coppia di credenziali (e-mail e password) il backend restituisce un token di accesso che il client dovrà utilizzare per poter autenticare le richieste.

Inoltre, abbiamo selezionato Sanctum come sistema di autenticazione in quanto permette di autenticare gli utenti anche tramite i meccanismi di Session Tracking, quindi, dal momento che NaTour avrà anche un client web, possiamo integrare quest'ultimo senza effettuare alcuna modifica al codice del backend.

NaTour consente ai propri utenti di autenticarsi anche attraverso provider quali Google e Facebook. Per questo motivo abbiamo affiancato Sanctum con Socialite, un altro package ufficiale dell'ecosistema Laravel. Quest'ultimo, lavora in sinergia con Sanctum, infatti una volta ricevuti i token dal provider attraverso il quali ci si vuole autenticare, permette di recuperare le informazioni associate all'account, vale a dire e-mail e nome utente. Una volta recuperate verifica la presenza nel database di un utente con le suddette credenziali, altrimenti lo crea.

Salvataggio dei dati

Per quanto riguarda il salvataggio dei dati abbiamo optato per il DBMS MySql con la consapevolezza che se dovesse essere necessario in qualunque momento cambiare il DBMS sottostante questa operazione potrà essere svolta senza alcun problema. Infatti, grazie alla flessibilità di Laravel è possibile sostituire l'attuale DBMS con un altro modificando soltanto il nome del driver da utilizzare.

Per quanto riguarda lo storage di file, abbiamo optato per il servizio S3 di AWS attraverso il quale possiamo recuperare e salvare file in modo efficiente e particolarmente economico.

N.B. Attualmente abbiamo pronta solo l'implementazione di S3 nel backend ma ancora devono essere implementate le funzioni che ne richiedono l'utilizzo.

Server

Per quanto riguarda il setup dell'infrastruttura lato server, ci siamo affidati al provider di servizi cloud computing EcoWebHosting con i quale il team ha avuto modo, nei diversi anni di utilizzo, di constatare la loro affidabilità, le performance dei servizi venduti ma soprattutto l'assistenza clienti immediata che comporta un enorme risparmio di tempo. Abbiamo dunque scelto di effettuare il deploy del backend su un VPS, vale a dire un ambiente

virtualizzato e altamente scalabile. La scelta di orientarci verso servizi di cloud computing è stata dettata dai vantaggi e i benefici che abbiamo individuato in questa soluzione piuttosto delle alternative. I vantaggi principali che abbiamo preferito per il nostro setup sono la scalabilità ed il pagamento a consumo. Per quanto riguarda il primo, un ambiente virtualizzato gode di una straordinaria scalabilità in quanto è possibile aggiungere o rimuovere risorse all'ambiente in qualunque momento e secondo le necessità e, collegandoci al secondo vantaggio, pagare dunque solo l'effettivo utilizzo di tali risorse.

Inoltre, la scelta del provider EcoWebHosting è dovuta anche alla “natura” dell'app. L'energia utilizzata per l'alimentazione dell'infrastruttura proviene completamente da fonti rinnovabili e ogni mese parte degli introiti è destinata alla ricostruzione della flora mondiale piantando migliaia di alberi ogni mese.

Gestione del server, manutenzione e rilascio aggiornamenti

Nel seguente paragrafo andiamo ad analizzare come abbiamo strutturato e gestito il mantenimento dell'infrastruttura facendo attenzione ai tool e servizi che abbiamo selezionato, in modo da abbattere i costi che la software house deve sostenere e potenzialmente aumentare i profitti. Possiamo individuare tre elementi essenziali in questo setup:

- Gestione del server: per questa categoria rientrano i task di aggiornamento dei servizi installati nel server, del database, meccanismo di load balancing per gestire i picchi di utenza, sicurezza e backup periodici in modo da avere

sempre una copia dei dati a disposizione. Per tutti questi compiti ci siamo affidati a Forge, un tool pensato per creare e gestire infrastrutture per applicazioni PHP ed è particolarmente ottimizzato per Laravel. L'utilizzo di Forge permette di delegare alla piattaforma l'intera amministrazione dell'infrastruttura, riducendo le ore di lavoro necessarie. Inoltre, Forge consente di switchare infrastruttura in tempo zero, dunque, se fosse necessario cambiare provider, ad esempio passare a macchine più performanti offerte da AWS, grazie a questo strumento l'operazione può essere eseguita in un tempo estremamente ridotto e perlopiù in modo automatico.

- Manutenzione: l'applicativo ha necessità di essere costantemente aggiornato per risolvere falle di sicurezza e in generale per non renderlo obsoleto. Per automatizzare questo lavoro abbiamo selezionato Shift, un servizio cloud pensato proprio per Laravel. Shift si collega direttamente al repository dell'app e su cadenza periodica esegue scansioni di tutto il codice in modo da performare i compiti per cui è pensato. Il compito fondamentale di Shift è quello di aggiornare costantemente le dipendenze ogni qualvolta è disponibile un aggiornamento. Altro compito molto utile è quello di convertire il codice scritto dallo sviluppatore nel “Laravel way” ovvero secondo la convenzione prevista dal Framework, in questo modo il software diventa più manutenibile e comprensibile ad altri developer che entrano a far parte dello sviluppo.

- Rilascio aggiornamenti: per quest'ultimo task ci siamo affidati ad Envoyer un cloud service responsabile per il deploy in tempo zero del software. Il suo compito è quello di avviarsi nel momento in cui individua un commit nuovo nel repository, innescando un meccanismo automatizzato rilasciando la versione aggiornata in produzione senza creare Downtime per gli utenti. In caso di errori, attraverso il servizio è possibile in qualunque momento fare Rollback e tornare all'ultima versione funzionante. Questo meccanismo può essere ulteriormente automatizzato; Envoyer controlla costantemente lo stato di salute dell'applicativo e in caso di errori invia comunicazioni al developer, quindi, in seguito ad un aggiornamento, se dovesse verificare errori gli utenti non avranno nessun disservizio facendo rollback automatico in attesa che vengano risolti gli errori segnalati.

In conclusione, far lavorare in sinergia questi tre servizi consente di delegare il grosso del lavoro di manutenzione del software e dell'infrastruttura sottostante e automatizzare i processi così che i developers possano concentrarsi particolarmente sullo sviluppo di nuove funzionalità e sulla scrittura di dest automatici. Quanto ai costi, per questo setup la software house deve sostenere una spesa di circa 300€ annui, una cifra decisamente irrisoria se paragonata al costo delle ore di lavoro straordinario.

Rest API

Per concludere la sezione dedicata al backend forniamo di seguito una tabella descrittiva con gli endpoint attualmente accessibili.

HTTP VERB	URL	Descrizione
POST	https://natour.pepeandrea.it/api/login/{provider}	Login tramite provider terzi
POST	https://natour.pepeandrea.it/api/login	Login con credenziali
POST	https://natour.pepeandrea.it/api/register	Registrazione
GET	https://natour.pepeandrea.it/api/export/pdf/{path}	Esporta PDF percorso
GET	https://natour.pepeandrea.it/api/export/gpx/{path}	Esporta GPX percorso
POST	https://natour.pepeandrea.it/api/logout	Distruggi token auth
GET	https://natour.pepeandrea.it/api/checkUser	Verifica token
POST	https://natour.pepeandrea.it/api/sendemail	Invia e-mail
GET	https://natour.pepeandrea.it/api/paths/filter	Filtrà percorsi

GET	https://natour.pepeandrea.it/api/paths	Recupera percorsi
GET	https://natour.pepeandrea.it/api/path/{path}	Recupera percorso
POST	https://natour.pepeandrea.it/api/path	Aggiungi percorso
POST	https://natour.pepeandrea.it/api/report/{path}	Segnala percorso

* {par} indica un parametro passato tramite url

Front End

In questa sezione andiamo ad analizzare nel dettaglio il client Android sviluppato, dedicando particolare attenzione al pattern di design utilizzato e i supporti che sono stati necessari per ottenere il risultato richiesto dal cliente.

Come anticipato, il client sviluppato con la prima versione di NaTour è una app Android scritta in codice nativo Java (versione 17.0.1). Il target SDK utilizzato è il più recente, cioè il 32 ma abbiamo abilitato la retrocompatibilità fino all'SDK 26, in questo modo da includere anche gli utenti con dispositivi leggermente datati. La scelta della retrocompatibilità fino alla versione 26 è stata dettata da una questione statistica. Alcune funzionalità dell'app prevedono un consumo maggiore di batteria e dal momento che fino a pochi anni fa le batterie

avevano mediamente una durata molto breve, usare l'app su questi dispositivi sarebbe stato limitante. Noi di NaTour non vogliamo di certo limitare il nostro bacino di utenza, motivo per il quale sulla Roadmap è presente un client più leggero con un numero ristretto di funzionalità.

Scelta del design pattern

Quanto alla scelta del design pattern da utilizzare, abbiamo preferito attenerci alla documentazione ufficiale di Android, dove il pattern raccomandato per lo sviluppo di app Android è il MVVM, acronimo di Model-View-ViewModel.

Riportando la documentazione ufficiale, le guide linea richiedono che l'app sia

- Reattiva
- Faccia uso dei ViewModel e dei LiveData
- Faccia uso di Repository per il recupero dei dati

Inoltre, abbiamo tenuto conto di altri fattori quali

- Flessibilità
- Semplicità
- Manutenibilità

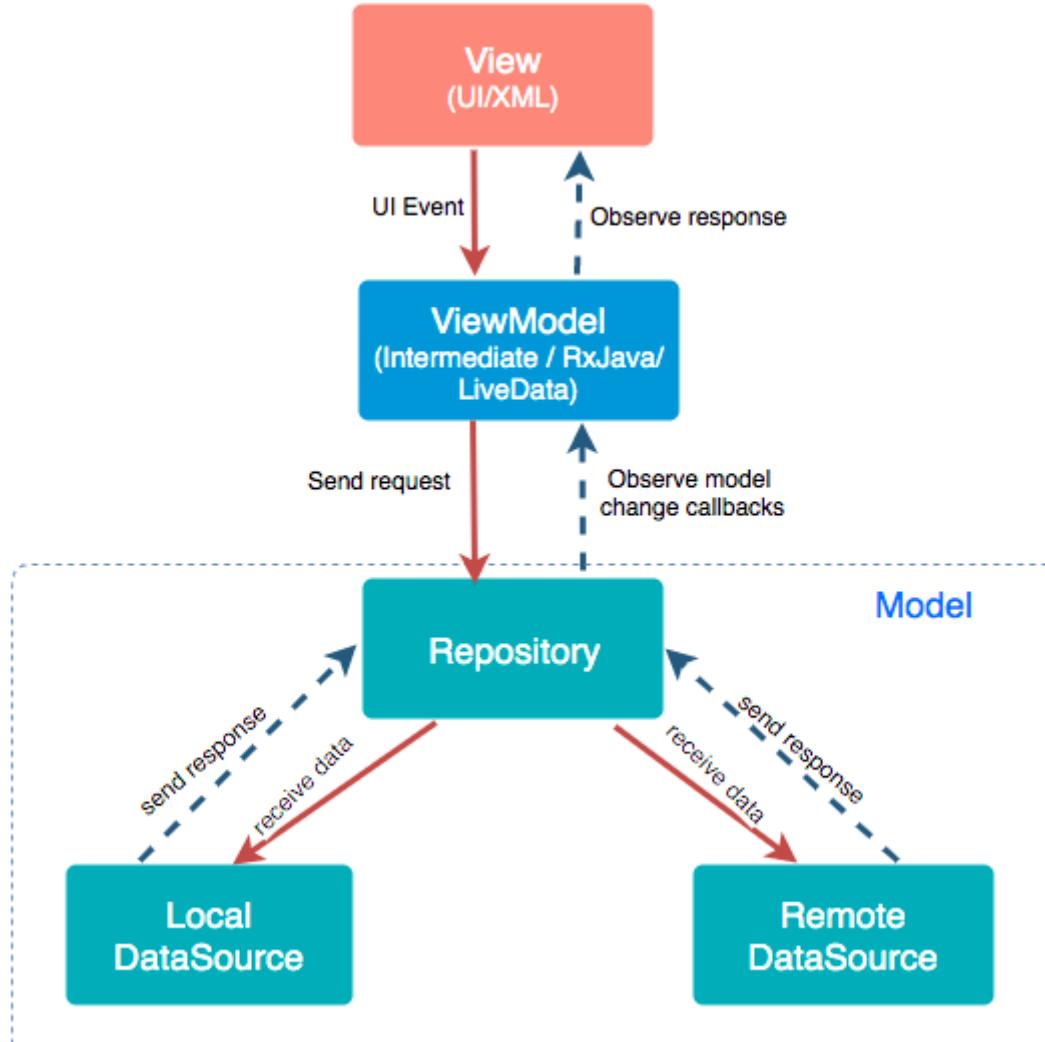
Analizziamo ora punto per punto come il pattern MVVM rispecchi tutti i requisiti richiesti e perché alla luce di quanto detto è risultata la scelta più idonea al progetto su cui stavamo lavorando.

- Reattività: per reattiva intendiamo una app che si adatta ai cambiamenti senza particolari problemi. La caratteristica principale del MVVM sta nel fatto che i dati non sono salvati nell'activity ma nel ViewModel motivo per il quale ad ogni aggiornamento delle classi di Activity non è necessario recuperare i dati dalla sorgente ma questi saranno immediatamente disponibili.
- Flessibilità: il pattern MVVM per il recupero dei dati fa uso delle classi Repository. Queste rappresentano un'astrazione della sorgente dati sottostante, infatti al loro interno è possibile implementare sorgenti dati in modo arbitrario senza che si vengano a creare problemi di compatibilità con il resto dell'applicativo. Per questo motivo, le classi Repository mettono a disposizione dei metodi pubblici per eseguire operazioni di recupero e inserimento dati.
- Semplicità e manutenibilità: come abbiamo iniziato ad accennare parlando delle classi proprie del pattern in questione, il fattore semplicità deriva dal fatto che le classi sono a singola responsabilità è quindi svolgono un insieme ridotto di funzioni. Si evince dunque che tale separazione e astrazione dei componenti consente sia di semplificare lo sviluppo ma anche di individuare immediatamente dove risiedono gli errori riscontrati.

Pattern MVVM

Come indicato precedentemente, tale nome sta per Model-View-ViewModel e identificano le tre classi essenziali per l'applicazione del pattern.

Di seguito un diagramma che mostra in che modo le classi collaborano tra di loro.



Analizziamo ora le componenti del pattern e successivamente individuiamo le classi aggiuntive di cui ci siamo serviti.

- **View**: le view corrispondono alle classi che definiscono l'interfaccia utente del nostro applicativo. Nel caso specifico di Android le classi View sono definite dalle Activity e dai Fragment.

- ViewModel: funge da collegamento tra il Model e la View. È responsabile del recupero e della trasformazione dei dati del Model e li fornisce alla View. Inoltre, fa uso di callback per aggiornare la View in caso di aggiornamento dei dati.
- Repository: fa parte del dominio Model e come accennato in precedenza è la classe responsabile per l'acquisizione di dati da sorgenti locali o remote ed esegue operazioni su questi ultimi.

- Remote DataSource: in questa categoria appartengono le classi che di fatto interagiscono con il backend e restituiscono le risposte ricevute dal repository.

Nel caso specifico di NaTour queste classi sono definite come ClientAPI proprio perché il loro ruolo è quello di interagire con le API esposte dal server e scambiarsi messaggi con quest'ultimo.

Al momento il client Android non ha implementazioni per una sorgente dati locale; quindi, per il momento escludiamo dall'analisi questa componente.

Come per il backend, anche nel frontend ci siamo serviti di ulteriori classi e componenti, nel dettaglio:

- Service: in Android un service è un task eseguito parallelamente all'applicazione e di solito lo si utilizza per compiere operazioni in background. Data la sua natura, abbiamo sviluppato un service dedicato alla registrazione delle posizioni raccolte tramite GPS nel tempo. Al momento che viene eseguito in background, tramite il service siamo stati in grado di

mantenere attiva la registrazione anche quando l'utente esce dall'app o blocca il dispositivo.

- RecyclerView: si tratta di un componente essenziale per la gestione di una lista di elementi orientato all'ottimizzazione delle risorse utilizzate. Infatti, quando viene passato una lista di elementi da mostrare in una View, il compito della RecyclerView è quello di non caricare in lista tutti i dati contemporaneamente, ma solo quelli visibili sullo schermo e a mano a mano che si scorre, libera dalla memoria quelli passati e aggiunge in lista i nuovi.
- Adapter: sono di supporto alle RecyclerView e servono per definire in che modo i dati passati dal ViewModel alla View devono essere adattati al layout e mostrati a schermo.

Supporti utilizzati

Retrofit: retrofit è un client HTTP per java attraverso il quale è possibile interagire con le Rest API di terze parti, nel nostro caso il backend di NaTour.

Google Maps SDK: attraverso questo SDK messo a disposizione Google è possibile implementare il sistema Google Maps all'interno del proprio applicativo. L'SDK oltre alla visualizzazione della mappa, mette a disposizione diverse API con i quali è possibile eseguire operazioni come calcolo di un percorso, posizionare dei markers o disegnare poligoni e linee. In particolare, abbiamo sfruttato il sistema di linee per tracciare i percorsi piuttosto che usare la specifica API che consente di calcolare il percorso in modo automatico per

un motivo in particolare, cioè avere un percorso valido quando l'utente registra o carica un percorso su un'area di territorio non registrata nella piattaforma di Google, cosa che accade spesso con i sentieri di montagna.

Altro elemento di cui abbiamo fatto uso sono i markers per indicare i punti di interesse lungo il percorso.

Logging e analitica

Un aspetto su cui abbiamo lavorato è stato quello del logging e monitoraggio continuo del client Android. Abbiamo ritenuto fondamentale aggiungere un sistema di metriche e segnalazione di errori automatico, in modo da poter tenere sotto controllo eventuali errori riscontrati dagli utenti. Inoltre, il sistema è in grado di fornire report anche sulle performance dell'app in modo da individuare i colli di bottiglia e risolverli nel minor tempo possibile.

Per questo scopo, oltre al sistema di logging integrato in Android, ci siamo serviti della suite Analytics di Firebase, un prodotto offerto da Google e per questo motivo offre un sistema di implementazione rapido nell'ecosistema Android.

Delle tante funzioni offerte dal prodotto, quelle che abbiamo implementato sono:

- Sistema di logging su eventi personalizzati: sistema di logging che consente di raccogliere informazioni su eventi specifici così da poter analizzare

statisticamente alcuni comportamenti dell'app. È particolarmente utile anche per svolgere analisi dettagliate sull'usabilità.

- Firebase Crashlytics: è un reporter di arresti anomali e in tempo reale che aiuta a monitorare, assegnare priorità e risolvere i problemi di stabilità che si presentano quando l'app è in esecuzione.
- Firebase Performance Monitoring: l'SDK di monitoraggio delle prestazioni consente di raccogliere i dati sulle prestazioni dell'app. Il monitoraggio delle prestazioni consente di capire in tempo reale dove è possibile apportare migliorie per risolvere i problemi di prestazioni.

Riferimenti

R. Martin, Clean Code: Guida per diventare bravi artigiani nello sviluppo agile di software. Maestri di programmazione, Feltrinelli Editore, 2018.

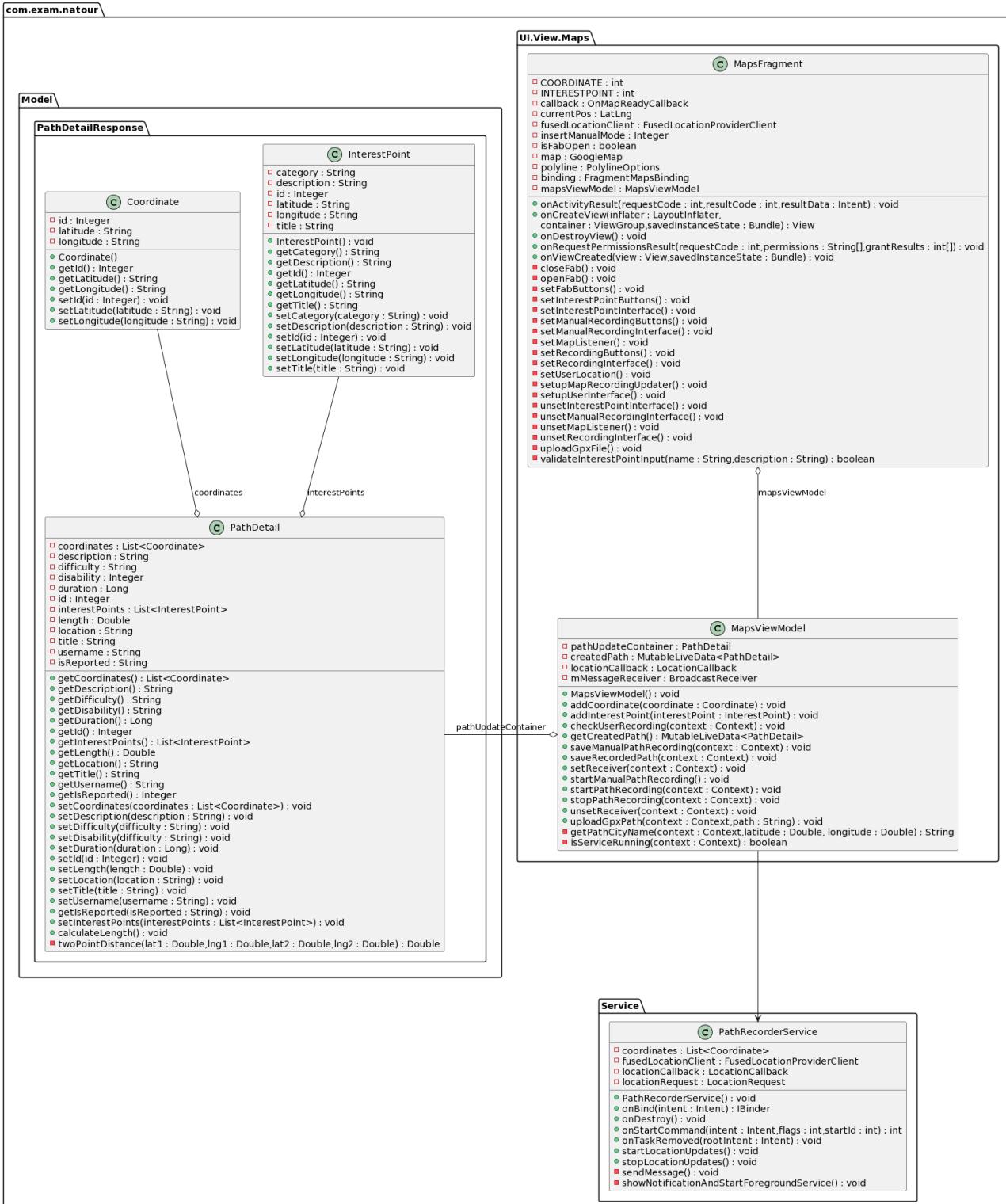
[Documentazione ufficiale di Laravel](#)

[Documentazione ufficiale di Android](#)

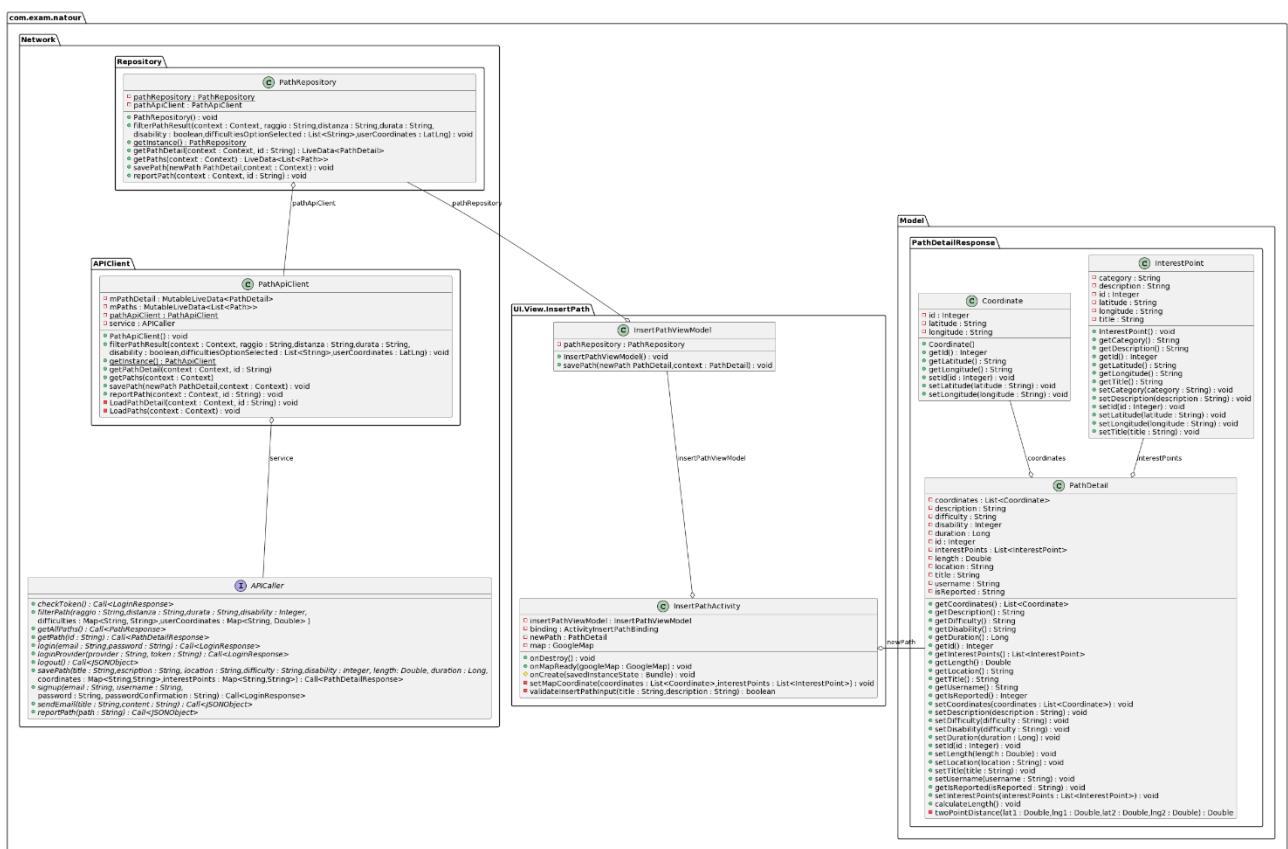
Class Diagram di Design



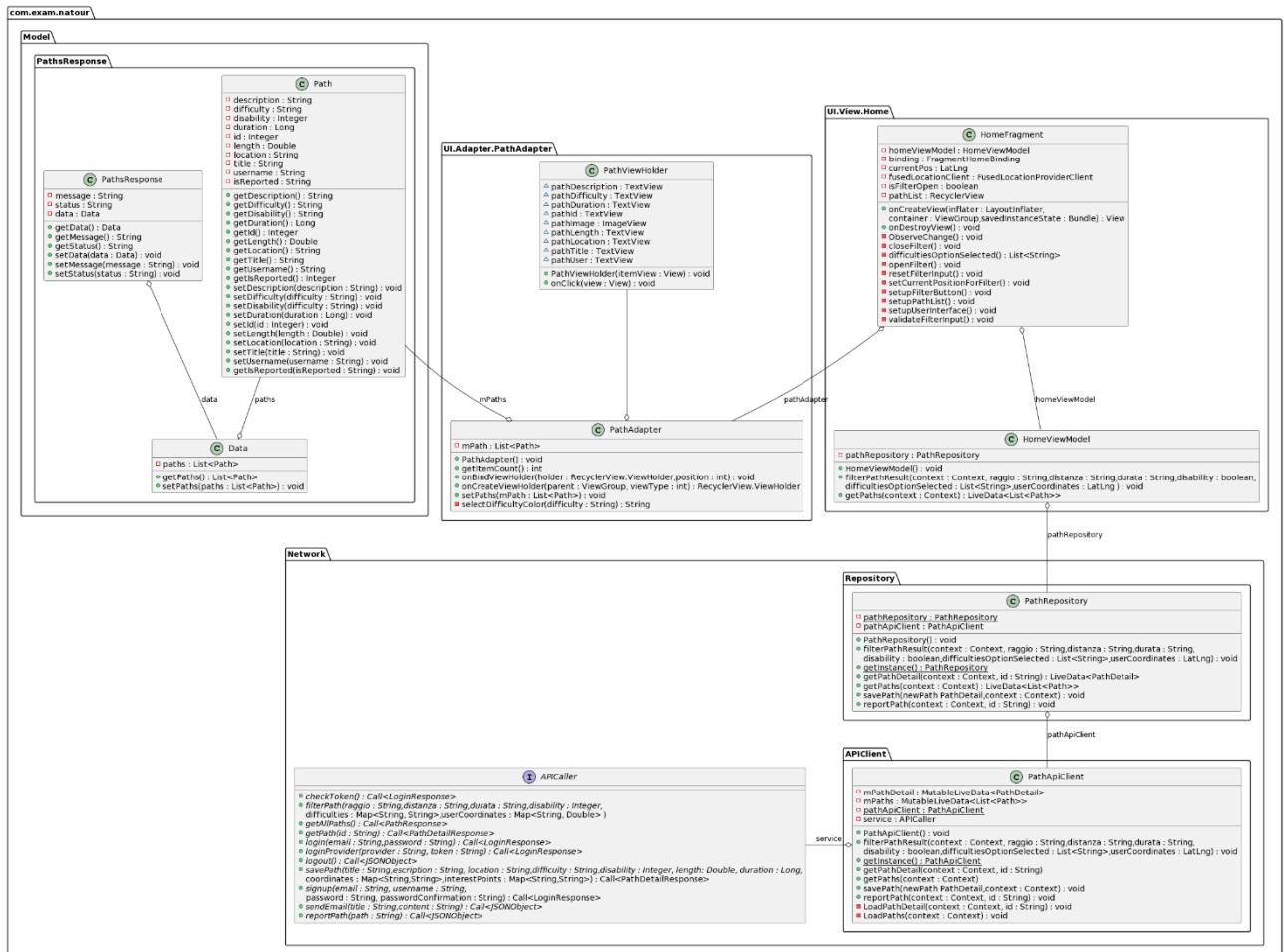
Autenticazione



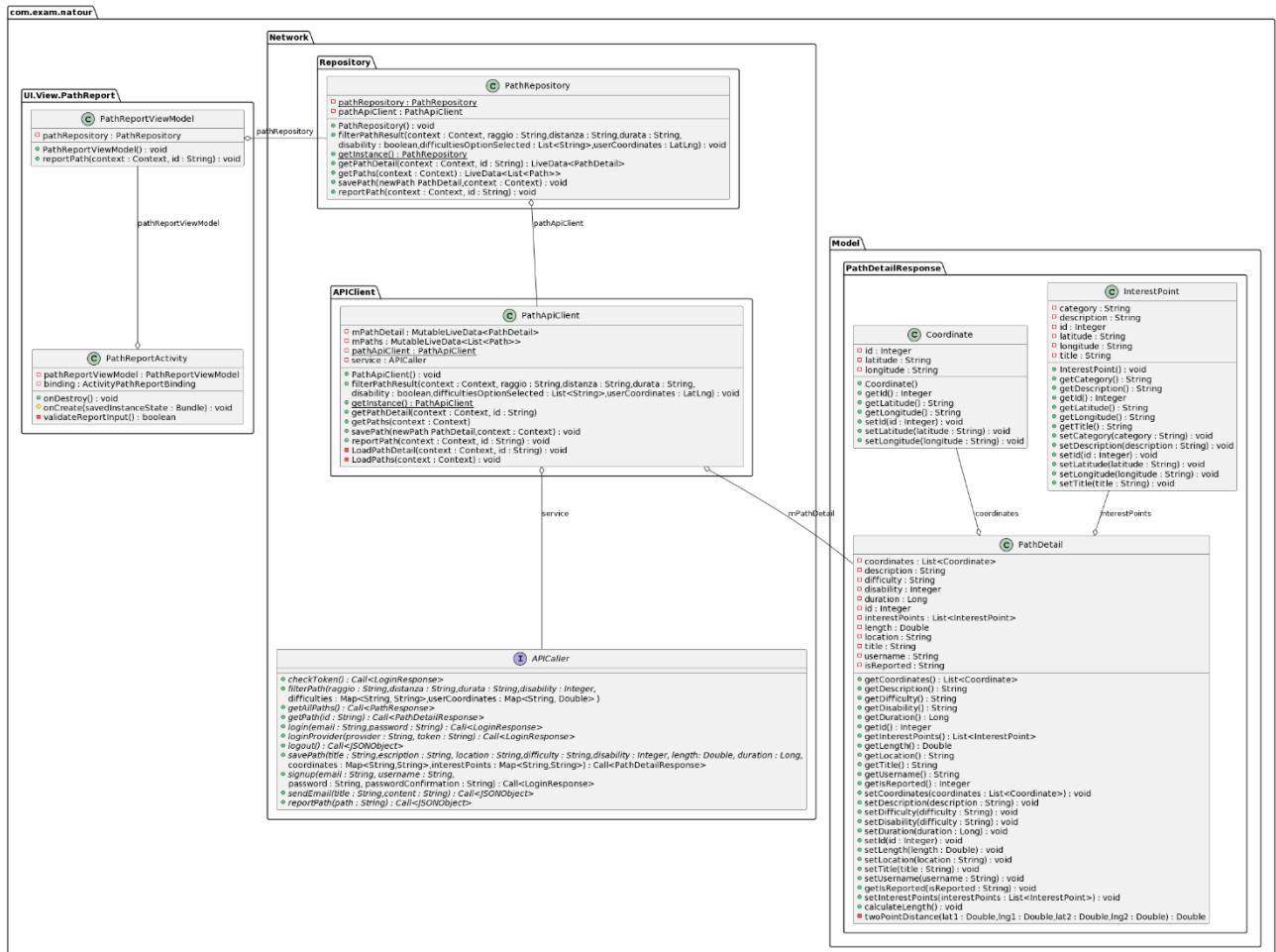
Creazione percorso



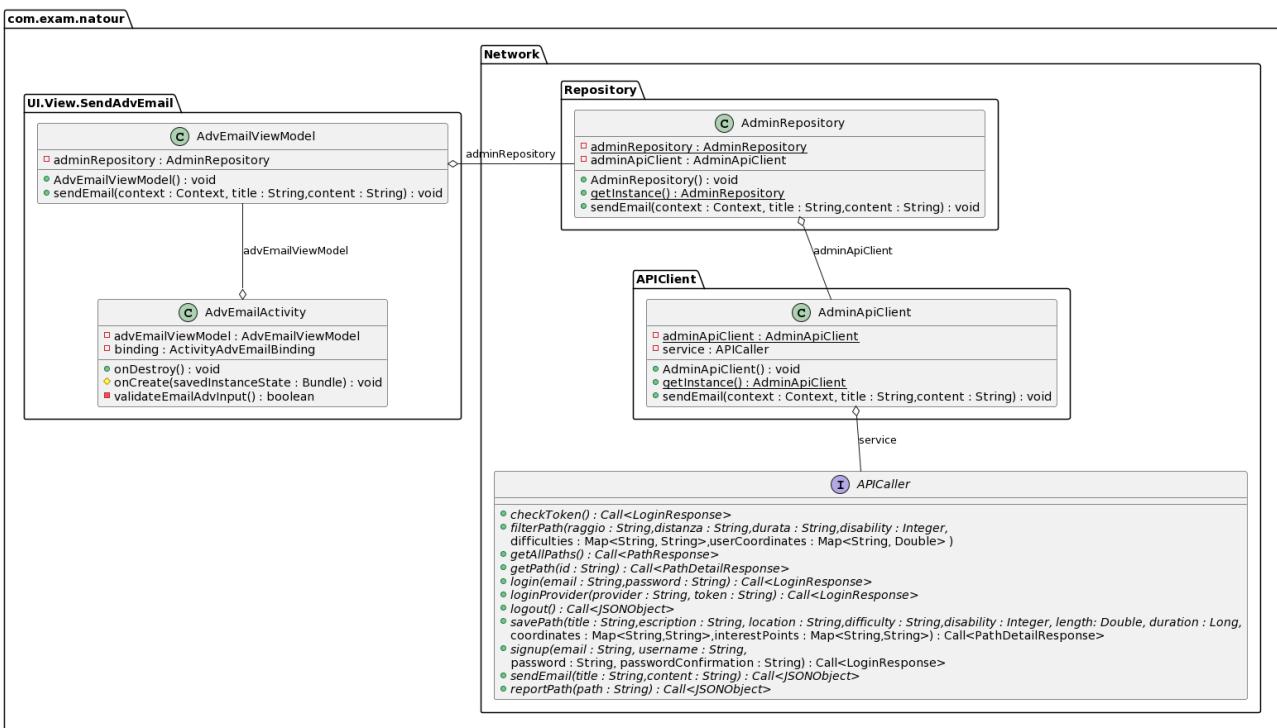
Upload percorso



Filtro percorsi

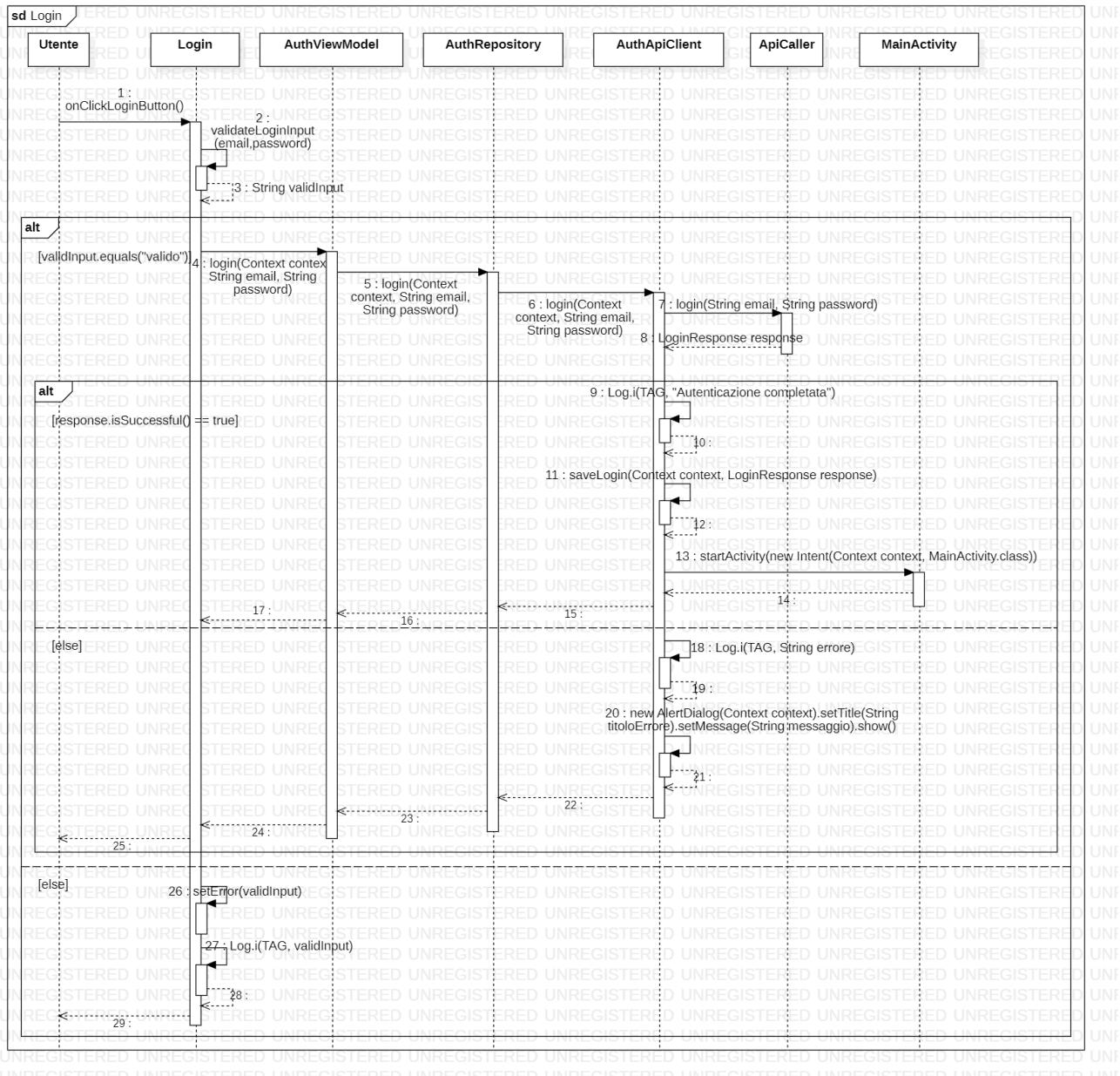


Segnalazione percorso

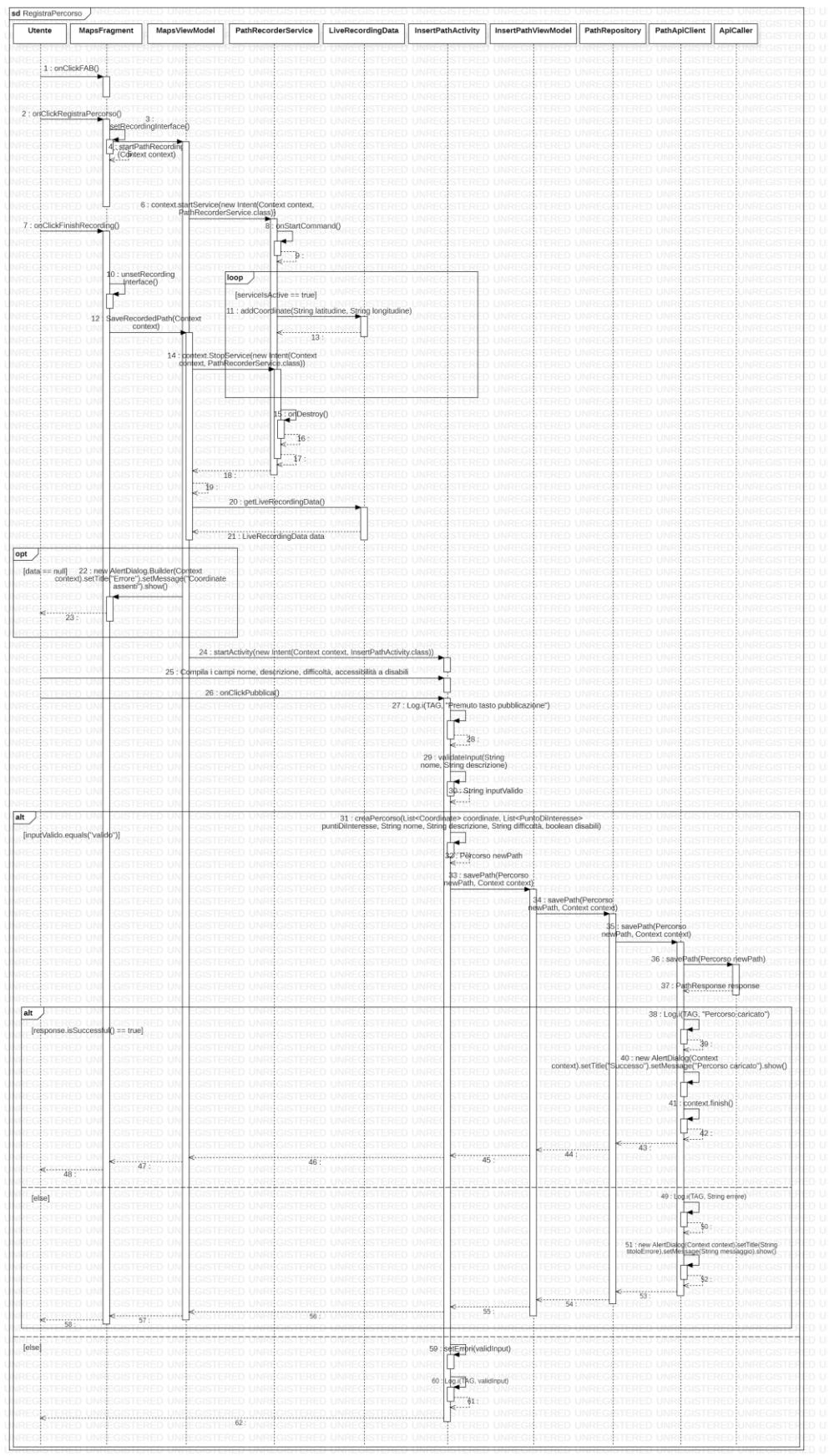


Email promozionale

Sequence Diagram di Design



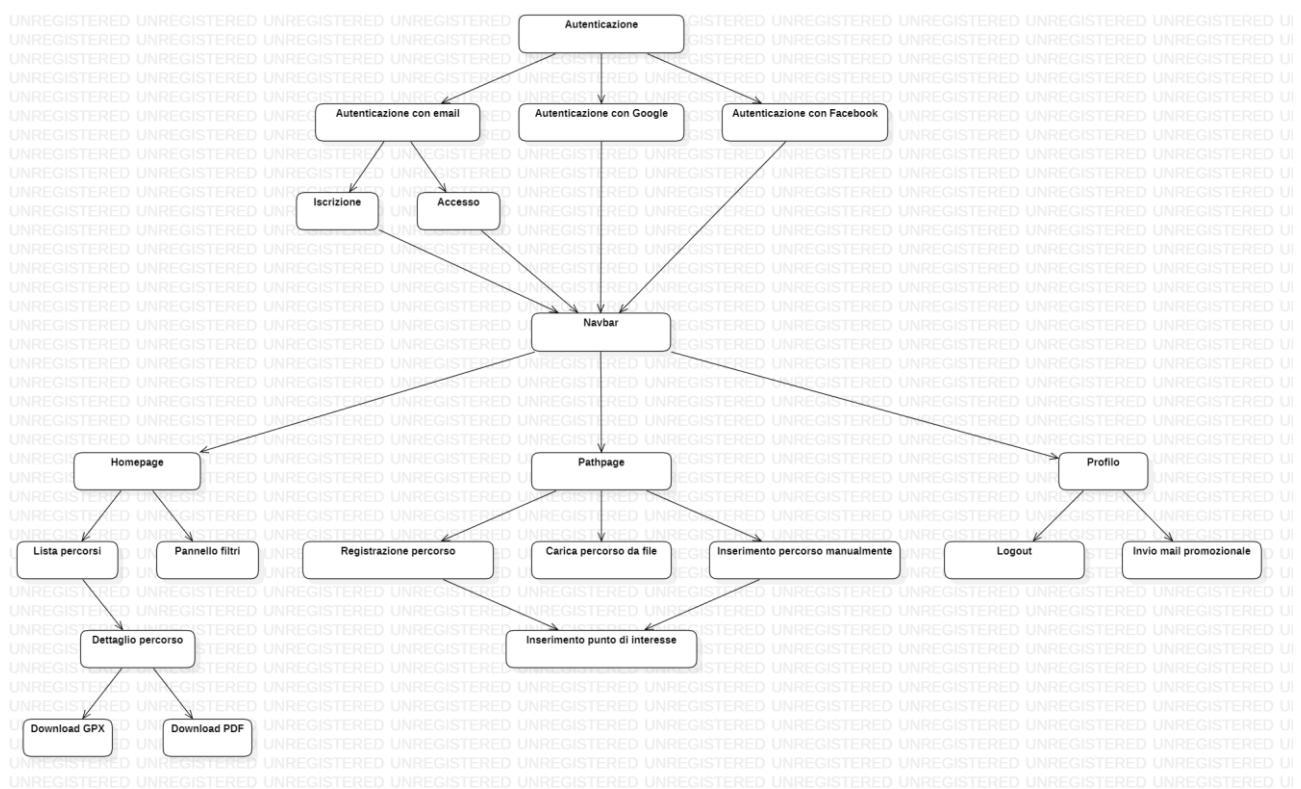
Login



Registrazione (tramite GPS) e pubblicazione di un percorso

Gerarchie Funzionali

Di seguito illustreremo le gerarchie funzionali dell'applicazione. Per semplicità, l'autenticazione dà accesso alla Navbar - dato che la nostra app ne è provvista - e quest'ultima dà accesso ai tre fragment.



Codice sorgente sviluppato

Codice Sorgente

Durante lo sviluppo il codice è stato versionato attraverso Git ed il provider GitHub dove è possibile visionare i sorgenti ai seguenti link:

- App Android: [premi qui](#)
- Backend: [premi qui](#)

Licenza

Tutti i sorgenti NaTour, presenti e futuri, sono sotto licenza *GNU - General Public License v3.0*

Testing

Come richiesto dai committenti abbiamo realizzato ed eseguito una suite di test automatici per la verifica della corretta funzionalità di alcuni metodi non banali.

Per i motivi indicati nella sezione “*Analisi dell’architettura*” abbiamo dato priorità al testing del backend verificando il corretto funzionamento delle Rest API e quindi dei metodi associati. In particolare, abbiamo testato tre metodi ed un middleware essenziale per il funzionamento dell’app.

Il middleware in questione è il responsabile della verifica dell’utente autenticato; il suo ruolo, quindi, è quello di analizzare la richiesta proveniente dal client, assicurarsi che questa sia autenticata e in caso di esito negativo restituire immediatamente un messaggio di errore con status HTTP 401 al client.

Quanto ai metodi questi sono:

- Registrazione utente
- Login utente
- Aggiunta nuovo percorso

La scelta è ricaduta su questi perché sono accomunati dal fatto che prevedono diversi parametri in ingresso e poiché questi ultimi vengono inseriti dall’utente

finale, si tratta di metodi che per ovvi motivi sono soggetti a tanti errori di input ed è dunque essenziale verificarne il corretto funzionamento per evitare di rovinare l'esperienza utente.

Per quanto riguarda la strategia di testing abbiamo optato per una strategia di tipo BlackBox con criterio di copertura WECT (Weak Equivalence Class Testing). Seguendo questo criterio, abbiamo proceduto scrivendo un singolo test per ogni classe di equivalenza su ognuno degli input ed un solo caso in cui vengono rispettate le classi di equivalenza valide di tutti gli input.

Abbiamo preferito il criterio WECT rispetto a criteri come SECT o WCT per evitare un numero di test potenzialmente grande data la quantità di parametri e classi di equivalenza su questi ultimi.

Middleware Auth

Il seguente test ha solamente due casi in quanto una richiesta può essere autenticata o meno.

```
<?php

namespace Tests\Unit;

use App\Models\User;

use Laravel\Sanctum\Sanctum;

use Tests\TestCase;

use Illuminate\Foundation\Testing\DatabaseTransactions;

class AuthUserTest extends TestCase

{
    use DatabaseTransactions;

    /**
     * A basic test example.

     *
     * @return void
     */

    public function test_not_logged_user()
    {
        $response = $this->json("GET", "/api/user");

        $response->assertStatus(401);
    }

    public function test_logged_user()
```

```
{  
    Sanctum::actingAs( User::factory() ->create());  
  
    $response = $this->json("GET", "/api/user");  
  
    $response->assertStatus(200);  
  
}  
  
}
```

Registrazione utente

Classi di equivalenza individuate:

- CEN01: Nome vuoto
- CEN02: Nome valido
- CEE01: Email vuota
- CEE02: Email valida
- CEE03: Email come stringa
- CEE04: Email senza @
- CEE05: Email senza parte prima di @
- CEE06: Email senza parte dopo di @
- CEP01: Password vuota
- CEP02: Password con meno di 8 caratteri
- CEP03: Password valida
- CECP01: Conferma password vuoto
- CECP02: Conferma password non corrispondente
- CECP03: Conferma password valido
- CER01: Registrazione utente
- CEL02: Registrazione email duplicata

```
<?php  
namespace Tests\Unit;
```

```
use App\Models\User;

use Illuminate\Foundation\Testing\DatabaseTransactions;

use Tests\TestCase;

class RegisterUserTest extends TestCase

{
    use DatabaseTransactions;

    private $dummyRequest = [
        'name' => '',
        'email' => '',
        'password' => '',
        'password_confirmation' => '',
    ];

    public function test_empty_request()
    {
        $response = $this->json("post", "api/register", []);
        $response->assertStatus(422);
    }

    public function test_request_empty_name()
    {
        $response = $this->json("post", "api/register", $this-
>dummyRequest);
        $this->assertArrayHasKey("name", $response["errors"]);
    }
}
```

```

public function test_request_not_empty_name()
{
    $this->dummyRequest["name"] = "test name";
    $response = $this->json("post", "api/register", $this->dummyRequest);
    $this->assertArrayNotHasKey("name", $response["errors"]);
}

public function test_request_empty_email()
{
    $response = $this->json("post", "api/register", $this->dummyRequest);
    $this->assertArrayHasKey("email", $response["errors"]);
}

public function test_request_valid_email()
{
    $this->dummyRequest["email"] = "test@test.it";
    $response = $this->json("post", "api/register", $this->dummyRequest);
    $this->assertArrayNotHasKey("email", $response["errors"]);
}

public function test_request_not_valid_email_string()
{
    $this->dummyRequest["email"] = "test email";
    $response = $this->json("post", "api/register", $this->dummyRequest);
    $this->assertArrayHasKey("email", $response["errors"]);
}

```

```

}

public function test_request_not_valid_email_no_at()
{
    $this->dummyRequest["email"] = "testemail.it";
    $response = $this->json("post", "api/register", $this->dummyRequest);
    $this->assertArrayHasKey("email", $response["errors"]);
}

public function test_request_not_valid_email_no_extension()
{
    $this->dummyRequest["email"] = "teste@mail";
    $response = $this->json("post", "api/register", $this->dummyRequest);
    $this->assertArrayNotHasKey("email", $response["errors"]);
}

public function test_request_not_valid_email_no_first_part()
{
    $this->dummyRequest["email"] = "@mail.it";
    $response = $this->json("post", "api/register", $this->dummyRequest);
    $this->assertArrayHasKey("email", $response["errors"]);
}

public function test_request_not_valid_email_no_domain()
{
    $this->dummyRequest["email"] = "test@";
}

```

```

    $response = $this->json("post", "api/register", $this-
>dummyRequest);

    $this->assertArrayHasKey("email", $response["errors"]);

}

public function test_request_empty_password()
{
    $response = $this->json("post", "api/register", $this-
>dummyRequest);

    $this->assertArrayHasKey("password", $response["errors"]);
}

public function test_request_password_without_confirmation()
{
    $this->dummyRequest["password"] = "abc";
    $response = $this->json("post", "api/register", $this-
>dummyRequest);

    $this->assertArrayHasKey("password", $response["errors"]);
}

public function test_request_not_minimum_length_password()
{
    $this->dummyRequest["password"] = "abc";
    $this->dummyRequest["password_confirmation"] = "abc";
    $response = $this->json("post", "api/register", $this-
>dummyRequest);

    $this->assertArrayHasKey("password", $response["errors"]);
}

```

```

public function test_request_valid_password()
{
    $this->dummyRequest["password"] = "testpassword";
    $this->dummyRequest["password_confirmation"] =
"testpassword";
    $response = $this->json("post", "api/register", $this-
>dummyRequest);
    $this-
>assertArrayNotHasKey("password", $response["errors"]);
}

public function test_valid_request()
{
    $this->dummyRequest["name"] = "test";
    $this->dummyRequest["email"] = "email@email.it";
    $this->dummyRequest["password"] = "password";
    $this->dummyRequest["password_confirmation"] =
"password";
    $response = $this->json("post", "api/register", $this-
>dummyRequest);
    $this->assertArrayNotHasKey("errors", $response);
    $response->assertStatus(200);
}

public function test_request_duplicated_email()
{
    $user = [
        'name' => "test",
        'email' => "test@email.it",

```

```
'password' => "password",
'password_confirmation' => "password",
];

$response = $this->json("post", "api/register", $user);
$response->assertStatus(200);
$response = $this->json("post", "api/register", $user);
$response->assertStatus(422);
$this->assertArrayHasKey("email", $response["errors"]);
}

}
```

Login Utente

Classi di equivalenza individuate:

- CEE01: Email vuota
- CEE02: Email valida
- CEE03: Email come stringa
- CEE04: Email senza @
- CEE05: Email senza parte prima di @
- CEE06: Email senza parte dopo di @
- CEP01: Password vuota
- CEP02: Password valida
- CEL01: Login utente inesistente
- CEL02: Login utente esistente

```
<?php  
namespace Tests\Unit;  
  
use App\Models\User;  
use Illuminate\Foundation\Testing\DatabaseTransactions;  
use Tests\TestCase;  
  
class UserLoginTest extends TestCase  
{  
    use DatabaseTransactions;
```

```
private $dummyRequest = [
    'name' => '',
    'email' => '',
    'password' => '',
    'password_confirmation' => '',
];

public function test_empty_request()
{
    $response = $this->json("post", "api/login", []);
    $response->assertStatus(422);
}

public function test_request_empty_email()
{
    $response = $this->json("post", "api/login", $this-
>dummyRequest);
    $this->assertArrayHasKey("email", $response["errors"]);
}

public function test_request_valid_email()
{
    $this->dummyRequest["email"] = "test@test.it";
    $response = $this->json("post", "api/login", $this-
>dummyRequest);
    $this->assertArrayNotHasKey("email", $response["errors"]);
}
```

```

public function test_request_not_valid_email_string()
{
    $this->dummyRequest["email"] = "test email";
    $response = $this->json("post", "api/login", $this-
>dummyRequest);
    $this->assertArrayHasKey("email", $response["errors"]);
}

public function test_request_not_valid_email_no_at()
{
    $this->dummyRequest["email"] = "testemail.it";
    $response = $this->json("post", "api/login", $this-
>dummyRequest);
    $this->assertArrayHasKey("email", $response["errors"]);
}

public function test_request_not_valid_email_no_extension()
{
    $this->dummyRequest["email"] = "teste@mail";
    $response = $this->json("post", "api/login", $this-
>dummyRequest);
    $this->assertArrayNotHasKey("email", $response["errors"]);
}

public function test_request_not_valid_email_no_first_part()
{
    $this->dummyRequest["email"] = "@mail.it";
    $response = $this->json("post", "api/login", $this-
>dummyRequest);
}

```

```

        $this->assertArrayHasKey("email", $response["errors"]);

    }

public function test_request_not_valid_email_no_domain()
{
    $this->dummyRequest["email"] = "test@";
    $response = $this->json("post", "api/login", $this-
>dummyRequest);
    $this->assertArrayHasKey("email", $response["errors"]);
}

public function test_request_empty_password()
{
    $response = $this->json("post", "api/login", $this-
>dummyRequest);
    $this->assertArrayHasKey("password", $response["errors"]);
}

public function test_request_not_empty_password()
{
    $this->dummyRequest["password"] = "testpassword";
    $response = $this->json("post", "api/login", $this-
>dummyRequest);
    $this-
>assertArrayNotHasKey("password", $response["errors"]);
}

public function test_request_not_registered_user()
{
}

```

```

        $this->dummyRequest["email"] = "email@email.it";
        $this->dummyRequest["password"] = "password";
        $response = $this->json("post", "api/login", $this-
>dummyRequest);
        $this->assertArrayNotHasKey("errors", $response);
        $response->assertStatus(401);
    }

public function test_request_registered_user()
{
    User::create([
        'name' => "testname",
        'email' => "user@email.it",
        'password' =>
        '$2y$10$92IXUNpkjO0rQ5byMi.Ye4oKoEa3R09llC/.og/at2.uheWG/igi', // password
    ]);
}

        $this->dummyRequest["email"] = "user@email.it";
        $this->dummyRequest["password"] = "password";
        $response = $this->json("post", "api/login", $this-
>dummyRequest);
        $this->assertArrayNotHasKey("errors", $response);
        $response->assertStatus(200);
    }
}

```

Aggiunta nuovo percorso

Classi di equivalenza individuate:

- CET01: Titolo vuoto
- CET02: Titolo valido
- CED01: Descrizione vuota
- CED02: Descrizione valida
- CEL01: Località vuota
- CEL02: Località valida
- CEDI01: Difficoltà vuota
- CEDI02: Difficoltà valida
- CEH01: Disabilità vuota
- CEH02: Disabilità con stringa
- CEH03: Disabilità valida
- CEL01: Lunghezza vuota
- CEL02: Lunghezza con stringa
- CEL03: Lunghezza valida
- CEDU01: Durata vuota
- CEDU02: Durata con stringa
- CEDU03: Durata valida
- CECA01: Array coordinate vuoto
- CECA02: Array coordinate con latitudine vuota

- CECA03: Array coordinate con longitudine vuota
- CEPIA04: Array coordinate valido
- CEPI01: Array punto interesse vuoto
- CEPI02: Titolo punto interesse vuoto
- CEPI03: Titolo punto interesse valido
- CEPI04: Descrizione punto interesse vuota
- CEPI05: Descrizione punto interesse valida
- CEPI06: Categoria punto interesse vuota
- CEPI07: Categoria punto interesse valida
- CEPI08: Latitudine punto interesse vuota
- CEPI09: Latitudine punto interesse valida
- CEPI10: Longitudine punto interesse vuota
- CEPI11: Longitudine punto interesse valido

```

<?php

namespace Tests\Unit;

use App\Models\User;

use Illuminate\Foundation\Testing\DatabaseTransactions;
use Laravel\Sanctum\Sanctum;
use Tests\TestCase;

class AddNewPathTest extends TestCase
{
    use DatabaseTransactions;
}

```

```

private $dummyRequest = [
    //Path data
    'title' => '',
    'description' => '',
    'location' => '',
    'difficulty' => '',
    'disability' => '',
    'length' => '',
    'duration' => '',
    //Coordinates data
    'coordinates' => [],
    //Interest Point Data
    'interest_points' => []
];

public function test_not_authenticated_request()
{
    $response = $this->json("post", "api/path");
    $response->assertStatus(401);
}

public function test_authenticated_request_without_payload()
{
    Sanctum::actingAs( User::factory()->create());
    $response = $this->json("post", "api/path", $this-
>dummyRequest);
    $response->assertStatus(422);
}

```

```
public function test_authenticated_request_empty_title()
{
    Sanctum::actingAs( User::factory() ->create());
    $response = $this->json("post", "api/path", $this->dummyRequest);
    $this->assertArrayHasKey("title", $response["errors"]);
}

public function test_authenticated_request_not_empty_title()
{
    Sanctum::actingAs( User::factory() ->create());
    $this->dummyRequest["title"] = "test title";
    $response = $this->json("post", "api/path", $this->dummyRequest);
    $this->assertArrayNotHasKey("title", $response["errors"]);
}

public function
test_authenticated_request_empty_description()
{
    Sanctum::actingAs( User::factory() ->create());
    $response = $this->json("post", "api/path", $this->dummyRequest);
    $this->assertArrayHasKey("description", $response["errors"]);
}
```

```
public function test_authenticated_request_not_empty_description()

{
    Sanctum::actingAs( User::factory() ->create());
    $this->dummyRequest["description"] = "test description";
    $response = $this->json("post", "api/path", $this-
>dummyRequest);
    $this-
>assertArrayNotHasKey("description", $response["errors"]);
}

public function test_authenticated_request_empty_location()

{
    Sanctum::actingAs( User::factory() ->create());
    $response = $this->json("post", "api/path", $this-
>dummyRequest);
    $this->assertArrayHasKey("location", $response["errors"]);
}

public function test_authenticated_request_not_empty_location()

{
    Sanctum::actingAs( User::factory() ->create());
    $this->dummyRequest["location"] = "test location";
    $response = $this->json("post", "api/path", $this-
>dummyRequest);
    $this-
>assertArrayNotHasKey("location", $response["errors"]);
}
```

```

public function test_authenticated_request_empty_difficulty()
{
    Sanctum::actingAs( User::factory() ->create());
    $response = $this->json("post", "api/path", $this-
>dummyRequest);
    $this-
>assertArrayHasKey("difficulty", $response["errors"]);
}

public function
test_authenticated_request_not_empty_difficulty()
{
    Sanctum::actingAs( User::factory() ->create());
    $this->dummyRequest["difficulty"] = "test difficulty";
    $response = $this->json("post", "api/path", $this-
>dummyRequest);
    $this-
>assertArrayNotHasKey("difficulty", $response["errors"]);
}

public function test_authenticated_request_empty_disability()
{
    Sanctum::actingAs( User::factory() ->create());
    $response = $this->json("post", "api/path", $this-
>dummyRequest);
    $this-
>assertArrayHasKey("disability", $response["errors"]);
}

```

```

public function

test_authenticated_request_not_empty_disability_with_string()

{
    Sanctum::actingAs( User::factory() ->create());
    $this->dummyRequest["disability"] = "test disability";
    $response = $this->json("post", "api/path", $this-
>dummyRequest);
    $this-
>assertArrayHasKey("disability", $response["errors"]);
}

public function

test_authenticated_request_not_empty_disability_with_boolean()

{
    Sanctum::actingAs( User::factory() ->create());
    $this->dummyRequest["disability"] = true;
    $response = $this->json("post", "api/path", $this-
>dummyRequest);
    $this-
>assertArrayNotHasKey("disability", $response["errors"]);
}

public function test_authenticated_request_empty_length()

{
    Sanctum::actingAs( User::factory() ->create());
    $response = $this->json("post", "api/path", $this-
>dummyRequest);
    $this->assertArrayHasKey("length", $response["errors"]);
}

```

```
}

public function
test_authenticated_request_not_empty_length_with_string()
{
    Sanctum::actingAs( User::factory() ->create());
    $this->dummyRequest["length"] = "test length";
    $response = $this->json("post", "api/path", $this-
>dummyRequest);
    $this->assertArrayHasKey("length", $response["errors"]);
}

public function
test_authenticated_request_not_empty_length_with_numeric()
{
    Sanctum::actingAs( User::factory() ->create());
    $this->dummyRequest["length"] = 2000;
    $response = $this->json("post", "api/path", $this-
>dummyRequest);
    $this-
>assertArrayNotHasKey("length", $response["errors"]);
}

public function test_authenticated_request_empty_duration()
{
    Sanctum::actingAs( User::factory() ->create());
    $response = $this->json("post", "api/path", $this-
>dummyRequest);
    $this->assertArrayHasKey("duration", $response["errors"]);
}
```

```
}

public function
test_authenticated_request_not_empty_duration_with_string()
{
    Sanctum::actingAs( User::factory() ->create());
    $this->dummyRequest["duration"] = "test duration";
    $response = $this->json("post", "api/path", $this-
>dummyRequest);
    $this->assertArrayHasKey("duration", $response["errors"]);
}

public function
test_authenticated_request_not_empty_duration_with_numeric()
{
    Sanctum::actingAs( User::factory() ->create());
    $this->dummyRequest["length"] = 2000;
    $response = $this->json("post", "api/path", $this-
>dummyRequest);
    $this-
>assertArrayNotHasKey("length", $response["errors"]);
}

public function
test_authenticated_request_empty_coordinates_array()
{
    Sanctum::actingAs( User::factory() ->create());
    $response = $this->json("post", "api/path", $this-
>dummyRequest);
```

```

    $this-

>assertArrayHasKey("coordinates", $response["errors"]);

}

public function

test_authenticated_request_coordinates_empty_latitude_not_empty_lo
ngitude()

{
    $this->dummyRequest["coordinates"][0]["longitude"] =
20.93004;

    Sanctum::actingAs( User::factory()->create());

    $response = $this->json("post", "api/path", $this-
>dummyRequest);

    $this-

>assertArrayHasKey("coordinates.0.latitude", $response["errors"]);

}

public function

test_authenticated_request_coordinates_empty_longitude_not_empty_l
atitude()

{
    $this->dummyRequest["coordinates"][0]["latitude"] =
20.93004;

    Sanctum::actingAs( User::factory()->create());

    $response = $this->json("post", "api/path", $this-
>dummyRequest);

    $this-

>assertArrayHasKey("coordinates.0.longitude", $response["errors"]);

```

```

    }

public function
test_authenticated_request_coordinates_not_empty_longitude_not_emp
ty_latitude()
{
    $this->dummyRequest["coordinates"][0]["latitude"] =
20.93004;
    $this->dummyRequest["coordinates"][0]["longitude"] =
20.93004;
    Sanctum::actingAs( User::factory()->create());
    $response = $this->json("post", "api/path", $this-
>dummyRequest);
    $this-
>assertArrayHasKey("coordinates.0.longitude", $response["errors"]);
    $this-
>assertArrayHasKey("coordinates.0.latitude", $response["errors"]);
}

public function
test_authenticated_request_empty_interest_points_array()
{
    Sanctum::actingAs( User::factory()->create());
    $response = $this->json("post", "api/path", $this-
>dummyRequest);
    $this-
>assertArrayHasKey("interest_points", $response["errors"]);
}

```

```

}

public function
test_authenticated_request_empty_interest_point_title()
{
    $this->dummyRequest["interest_points"][0]["title"] = "";
    Sanctum::actingAs( User::factory()->create());
    $response = $this->json("post", "api/path", $this-
>dummyRequest);
    $this-
>assertArrayHasKey("interest_points.0.title", $response["errors"]);
}

public function
test_authenticated_request_not_empty_interest_point_title()
{
    $this->dummyRequest["interest_points"][0]["title"] = "test
title";
    Sanctum::actingAs( User::factory()->create());
    $response = $this->json("post", "api/path", $this-
>dummyRequest);
    $this-
>assertArrayNotHasKey("interest_points.0.title", $response["errors"
]);
}

public function
test_authenticated_request_empty_interest_point_description()

```

```

    {

        $this->dummyRequest["interest_points"][0]["description"] =
        "";

        Sanctum::actingAs( User::factory()->create());

        $response = $this->json("post", "api/path", $this-
>dummyRequest);

        $this-

>assertArrayHasKey("interest_points.0.description", $response["erro
rs"]);

    }

    public function

test_authenticated_request_not_empty_interest_point_description()

{

    $this->dummyRequest["interest_points"][0]["description"] =
"test description";

    Sanctum::actingAs( User::factory()->create());

    $response = $this->json("post", "api/path", $this-
>dummyRequest);

    $this-

>assertArrayNotHasKey("interest_points.0.description", $response["e
rrors"]);

}

    public function

test_authenticated_request_empty_interest_point_category()

{
    $this->dummyRequest["interest_points"][0]["category"] =
"";
```

```

        Sanctum::actingAs( User::factory() ->create() );

        $response = $this->json("post", "api/path", $this-
>dummyRequest);

        $this-
>assertArrayHasKey("interest_points.0.category", $response["errors"]
]);

}

public function
test_authenticated_request_not_empty_interest_point_category()
{
    $this->dummyRequest["interest_points"][0]["category"] =
"test category";

    Sanctum::actingAs( User::factory() ->create() );

    $response = $this->json("post", "api/path", $this-
>dummyRequest);

    $this-
>assertArrayNotHasKey("interest_points.0.category", $response["erro
rs"]);
}

public function
test_authenticated_request_empty_interest_point_latitude()
{
    $this->dummyRequest["interest_points"][0]["latitude"] =
"";

    Sanctum::actingAs( User::factory() ->create() );

    $response = $this->json("post", "api/path", $this-
>dummyRequest);
}

```

```

    $this-

>assertArrayHasKey("interest_points.0.latitude", $response["errors"]
]) ;

}

public function

test_authenticated_request_not_empty_interest_point_latitude()

{
    $this->dummyRequest["interest_points"][0]["latitude"] =
20.000;

    Sanctum::actingAs( User::factory()->create());

    $response = $this->json("post", "api/path", $this-
>dummyRequest);

    $this-

>assertArrayNotHasKey("interest_points.0.latitude", $response["erro
rs"]);
}

public function

test_authenticated_request_empty_interest_point_longitude()

{
    $this->dummyRequest["interest_points"][0]["longitude"] =
"";

    Sanctum::actingAs( User::factory()->create());

    $response = $this->json("post", "api/path", $this-
>dummyRequest);

    $this-

>assertArrayHasKey("interest_points.0.longitude", $response["errors
"]);
}

```

```

}

public function
test_authenticated_request_not_empty_interest_point_longitude()
{
    $this->dummyRequest["interest_points"][0]["longitude"] =
20.000;

    Sanctum::actingAs( User::factory()->create());
    $response = $this->json("post", "api/path", $this-
>dummyRequest);
    $this-
>assertArrayNotHasKey("interest_points.0.longitude", $response["err
ors"]);
}

public function test_authenticated_complete_request()
{
    $request = [
        //Path data
        'title' => "test",
        'description' => "test",
        'location' => "test",
        'difficulty' => "test",
        'disability' => false,
        'length' => 200,
        'duration' => 200,
        //Coordinates data
        'coordinates' => [
            [

```

```
        'longitude' => 20.0004,
        'latitude' => 20.0000,
    ],
],
//Interest Point Data
'interest_points' => [
[
    'title' => "test",
    'description' => "test",
    'category' => "test",
    'longitude' => 20.0004,
    'latitude' => 20.0000,
]
];
Sanctum::actingAs( User::factory() ->create());
$response = $this->json("post", "api/path", $request);
$this->assertArrayNotHasKey("errors", $response);
$response->assertStatus(200);
}
}
```

Valutazione dell'usabilità sul campo

Usabilità sul campo

Oltre ad aver eseguito uno studio dell'usabilità sui prototipi del prodotto, abbiamo proceduto con la valutazione sul campo dell'usabilità una volta terminato lo sviluppo. Quanto alle tecniche utilizzate, abbiamo seguito due approcci: da una parte abbiamo applicato il metodo euristico, dall'altra abbiamo fatto uso di sistemi di logging automatico.

Metodo euristico

Per la valutazione attraverso il metodo euristico abbiamo intervistato quattro utenti, invitandoli ad usare l'app per un periodo di almeno trenta minuti. Successivamente abbiamo chiesto loro di rispondere ad alcune domande formulate secondo le euristiche di Nielsen e di svolgere tre task.

Le domande che abbiamo posto sono:

1. Il linguaggio utilizzato è di facile comprensione?
2. Il carico di memoria a breve termine è basso?
3. Il sistema è coerente per quanto riguarda attività simili tra loro?
4. Il sistema rende chiaro all'utente cosa è successo in seguito alle sue azioni?
5. Si riesce sempre a tornare indietro in caso di errori?
6. Nel caso in cui l'utente commetta un errore, questo è segnalato assieme al perché e come risolverlo?

Quanto ai task richiesti:

1. Registrare un percorso sulla mappa interattiva
2. Aggiungere un punto di interesse
3. Segnalare un percorso

Per avere un risultato quanto più veritiero abbiamo selezionato i quattro utenti con grado di familiarità e conoscenza tecnologica molto diversa tra loro.

	Basso dominio	Alto dominio
Bassa tecnologia	Vincenzo Russo	Claudia Pepe
Alta tecnologia	Melania Naso	Naara Soledad Camardella

Come anticipato poche righe sopra, la prima fase di valutazione prevede un utilizzo di almeno trenta minuti. Durante questa fase l'unico errore è stato riscontrato da Melania, la quale non riusciva a sfruttare la feature di registrazione GPS. Dopo aver analizzato il problema, ci siamo resi conto che il suo dispositivo bloccava l'accesso GPS alla nostra app. Abbiamo dunque inserito un popup in cui viene spiegato all'utente come risolvere il problema relativo alle autorizzazioni.

Analizziamo ora gli esiti dei task richiesti:

- Vincenzo ha riscontrato un problema durante la segnalazione di un percorso in quanto non riusciva a trovare il bottone per procedere. Grazie alla sua segnalazione abbiamo deciso di spostare il tasto nella barra in alto, in questo modo sarà sempre visibile e l'utente può effettuare la segnalazione non appena individua l'errore senza dover andare alla ricerca del tasto;
- Melania è riuscita a portare a termine i task, seppur impiegando più tempo rispetto agli altri;
- Claudia ha portato avanti il test non riscontrando alcun errore e ha aggiunto che utilizzare l'app è stato davvero piacevole in quanto non ha dovuto effettuare alcuno sforzo per comprenderne il funzionamento.

“Sembra quasi che utilizzi NaTour da anni”, ci ha riferito;

- Naara è stato l'unica in grado di fornirci un riscontro su un bug che coinvolgesse anche il backend. In particolare, si è resa conto che durante l'inserimento di un punto di interesse era possibile salvarlo senza indicare alcuna descrizione. Successivamente quando si tentava di salvare il percorso il server restituiva un messaggio di errore in quanto alcuni dati mancavano (la descrizione del punto di interesse) e infine l'app tornava indietro facendo così perdere il percorso creato.

Infine, i quattro utenti hanno risposto in modo pressoché unanime alle domande che abbiamo posto loro. Possiamo riassumere le risposte in questo modo:

1. Assolutamente sì, il linguaggio utilizzato si adatta a qualunque categoria di utenti.
2. Potremmo definire il carico di memoria nullo.
3. La coerenza è alla base di NaTour: tutte le operazioni simili si eseguono compiendo le stesse azioni.
4. Sì, ad ogni azione, con esito positivo o negativo viene sempre mostrato un messaggio che indica cosa è appena successo.
5. Se si esegue un'azione diversa da quella prevista è sempre possibile annullarla e tornare indietro.
6. Qualunque errore commesso viene prontamente segnalato assieme alla sua soluzione.

Logging

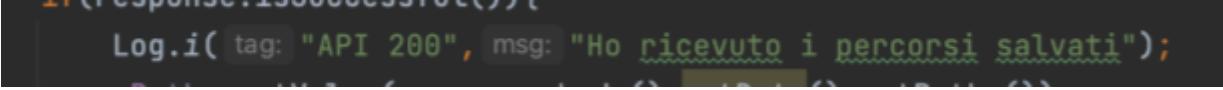
Come indicato nella sezione dei criteri di design dell'applicativo, per tenere traccia degli errori e per verificare il corretto funzionamento dell'app NaTour ci siamo serviti degli strumenti di logging messi a disposizione da Google attraverso la suite Firebase.

In particolare, abbiamo utilizzato:

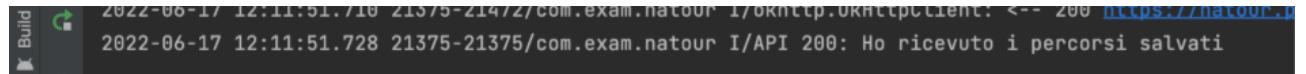
- Firebase Analytics
- Firebase Crashlytics
- Firebase Performance

Durante la fase di sviluppo, inoltre, abbiamo anche fatto uso della libreria di debug di Android per verificare che l'app eseguisse in modo corretto il codice.

Mostriamo di seguito un esempio della libreria di debug Android:



```
Log.i( tag: "API 200", msg: "Ho ricevuto i percorsi salvati");
```

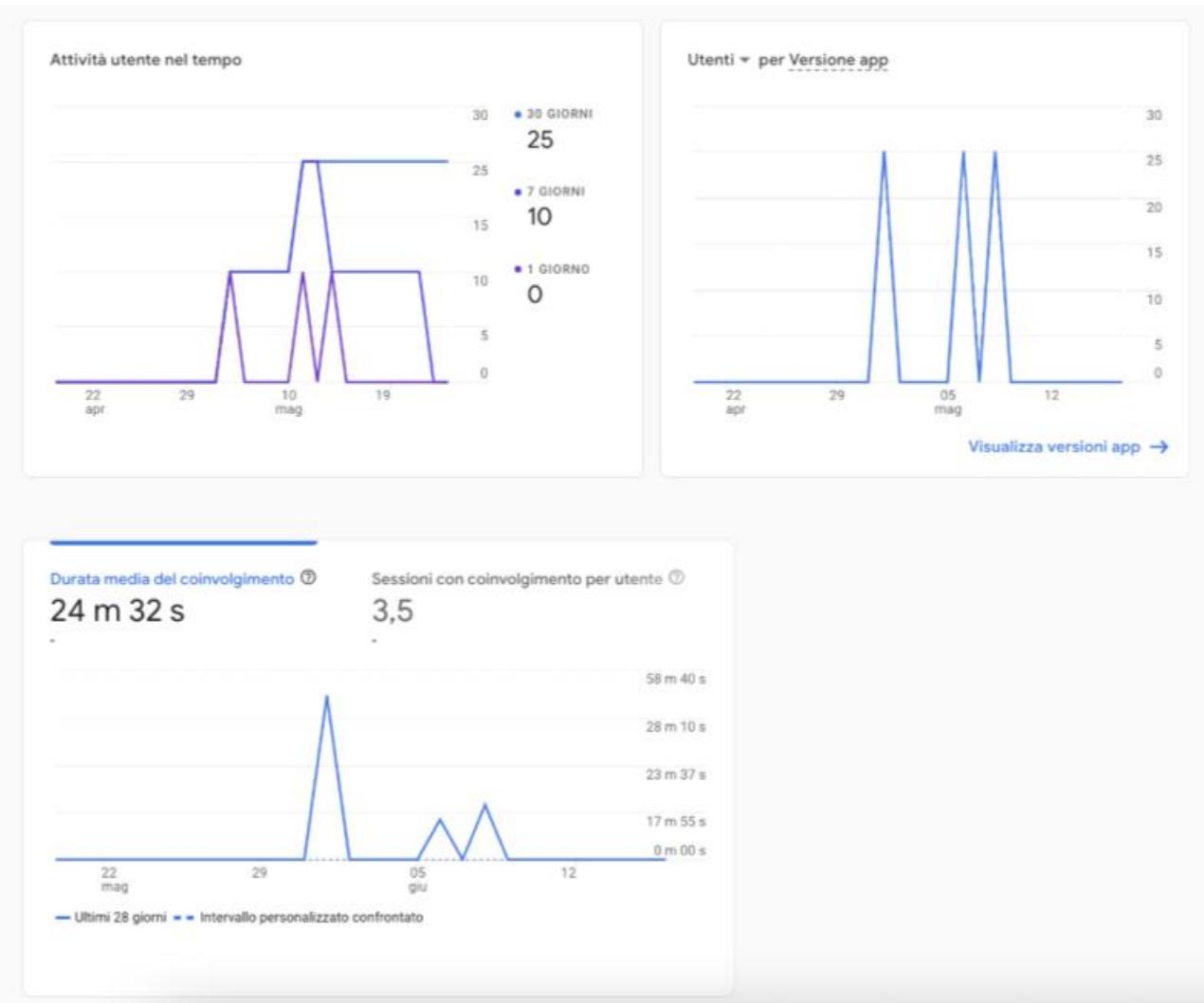


```
Build 2022-06-17 12:11:51.710 21375-21472/com.exam.natour I/OKHttpClient: <-- 200 https://natour.p  
2022-06-17 12:11:51.728 21375-21375/com.exam.natour I/API 200: Ho ricevuto i percorsi salvati
```

Analizziamo i singoli moduli Firebase e in che modo le informazioni fornite si sono rivelate utili per apportare migliorie e risolvere i problemi riscontrati.

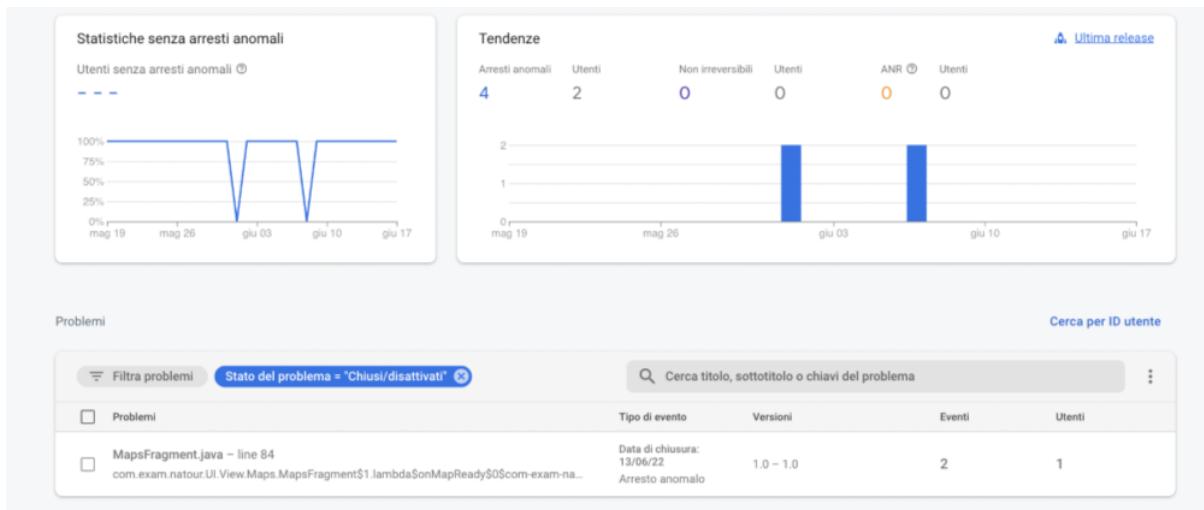
Firebase Analytics consente di tener traccia dell'attività degli utenti all'interno dell'applicazione. Nello specifico fornisce metriche in merito al numero di utenti, il loro coinvolgimento, quante sessioni riguardano un singolo utente, la loro durata e in che modo impiegano quel tempo.

Di seguito alcuni dati provenienti dalla dashboard di Firebase.



Firebase Crashlytics è un tool che consente di monitorare gli arresti anomali che si possono verificare durante l'esecuzione dell'app. In particolare, al verificarsi di un evento di questo tipo, Crashlytics memorizza non solo l'evento in sé, ma salva anche informazioni quali il modello del telefono, memoria di archiviazione disponibile, memoria RAM libera nel momento in cui si è verificato l'errore, versione del sistema operativo e lo stack trace per la verifica dei thread. Attraverso queste informazioni è dunque possibile individuare il problema e le cause così da risolverlo quanto prima.

Di seguito degli screenshot della console Crashlytics.



Firebase Performance, infine, tiene traccia delle prestazioni del software durante l'esecuzione. Attraverso questo tool, infatti, possiamo tenere traccia del tempo necessario all'applicativo per avviarsi, il tempo richiesto per effettuare e ricevere risposta dalle chiamate HTTP e le dimensioni dei payload inviati e ricevuti durante una richiesta HTTP.

Di seguito le metriche relative all'app NaTour.

