## **API-Utente**

All’interno dell’architettura di SmarTrip, il modulo di gestione degli utenti rappresenta il punto d’ingresso per tutte le operazioni relative all’autenticazione e registrazione degli utenti dell’app mobile.

Questa funzionalità è realizzata in Java tramite il framework Spring Boot, che permette di esporre in modo semplice e standard le API REST necessarie per la comunicazione tra client e server.

Le operazioni principali gestite da queste API sono due:

* **SignIn**: registrazione di un nuovo utente.
* **LogIn**: verifica delle credenziali di un utente esistente per consentire l’accesso all’applicazione.

**SignIn**

Questa operazione consente ad un utente di registrarsi al sistema.

Il metodo *signInUtente* riceve in ingresso un oggetto *Utente* contenente username e password, e chiama il metodo *registraUtente* del service.

Se l’utente non esiste già, viene salvato nel database (tramite il repository), e il sistema restituisce l’oggetto utente in risposta.

In caso contrario, viene restituito un messaggio d’errore (HTTP 400) che indica il fallimento dell’operazione.

Immagine che contiene testo, diagramma, Parallelo, linea

Il contenuto generato dall'IA potrebbe non essere corretto.

**LogIn**

Permette ad un utente già registrato di effettuare il login nell’app.

Il metodo *logInUtente* riceve l’oggetto *Utente* e chiama il metodo *accediUtente* del service per verificare che le credenziali siano corrette.

Se la password corrisponde a quella presente nel database, viene restituito l’esito positivo ("esito": true); in caso contrario, un errore HTTP 400 con "esito": false.

Immagine che contiene testo, diagramma, schermata, Parallelo

Il contenuto generato dall'IA potrebbe non essere corretto.

Entrambe le API poggiano sulla stessa struttura delle classi:

Immagine che contiene testo, schermata, diagramma, linea

Il contenuto generato dall'IA potrebbe non essere corretto.

## **API-Itinerario**

**AddItinerario**

Questa è l’API cuore del progetto, l’idea è di permettere all’utente di costruire una tabella di marcia coerente con i suoi interessi e i suoi bisogni.

Durante l’iterazione 0 ci preoccupiamo solo di scrivere lo pseudocodice e la complessità della stessa:

*if (itinerario è nullo oppure itinerario.luoghi è vuoto):*

*return (mappa vuota);*

*grafo = nuovo grafo pesato;*

*nodoAlloggio = nuovo luogo con coordinate dell’alloggio;*

*grafo.add(nodoAlloggio);*

*for each luogo in itinerario.luoghi:*

*grafo.add(luogo);*

*for (i da 0 a numero dei luoghi - 1){*

*for j da i+1 a numero dei luoghi:*

*luogo1 = itinerario.luoghi[i]*

*luogo2 = itinerario.luoghi[j]*

*distanza = calcolaDistanza(luogo1, luogo2);*

*tempoPercorrenza = calcolaTempoPercorrenza(distanza,velocità);*

*if (non esiste ancora un arco tra luogo1 e luogo2):*

*grafo.addArco(luogo1,luogo2);*

*grafo.setPeso(arco tra luogo1 e luogo2, tempoPercorrenza);*

*}*

*tabellaDiMarcia = nuova mappa vuota*

*for (nGiorno da 0 a itinerario.giorni.size - 1) {*

*giornoAttuale = itinerario.getGiorno(nGiorno-1);*

*tempoInizio = convertiInSecondi(giornoAttuale.Orario\_di\_inizio\_visita);*

*devePranzare = giornoAttuale.deve\_pranzare;*

*pranzoTroppoLungo = false;*

*orarioPranzo = convertiInSecondi(giornoAttuale.Orario\_di\_pranzo);*

*tempoCorrente = tempoInizio;*

*percorso = new lista di luoghi;*

*percorso.add(nodoAlloggio,tempoCorrente);*

*nodoCorrente = nodoAlloggio;*

*if(rimasto solo il nodoAlloggio nel grafo):*

*break;*

*while (true){*

*prossimoLuogo = nodo più vicino a nodoCorrente non ancora visitato e presente nel grafo che minimizza (tempoPercorrenza + tempoDiVisita);*

*if (prossimoLuogo.isEmpty()):*

*break;*

*luogoScelto = prossimoLuogo;*

*tempoPercorrenza = grafo.getPeso(nodoCorrente,luogoScelto);*

*if (devePranzare && tempoCorrente >= orarioPranzo):*

*if (tempoCorrente + durataPranzo < mezzanotte):*

*percorso.add(ricercaRistorante,tempoCorrente);*

*tempoCorrente += durataPranzo;*

*devePranzare = false;*

*else:*

*pranzoTroppoLungo = true;*

*if (tempoCorrente + tempoPercorrenza + luogoScelto.TempoDiVisita > giornoAttuale.orarioFineVisite || pranzoTroppoLungo):*

*break;*

*tempoCorrente += tempoPercorrenza;*

*percorso.add(luogoScelto,tempoCorrente);*

*tempoCorrente += luogoScelto.TempoPerVisitare;*

*nodoCorrente = luogoScelto;*

*}*

*if (nodoCorrente diverso da nodoAlloggio):*

*tempoCorrente += grafo.getPeso(nodoCorrente,nodoAlloggio);*

*percorso.add(nodoAlloggio,tempoCorrente);*

*tabellaDiMarcia.put(nGiorno,percorso);*

*for (luogo in percorso) {*

*if (luogo diverso da nodoAlloggio):*

*grafo.rimuoviNodo(luogo);*

*}*

*}*

*return repository.salva(itinerario.NomeMappa, itinerario.Utente, tabellaDiMarcia);*

La complessità algoritmica è data da 2 parti in cui l’Input è dato da un itinerario con N luoghi e G giorni.

1. **Costruzione del grafo:**

Si crea un grafo pesato e si aggiunge il nodo dell’alloggio + tutti gli N luoghi (in totale N+1 nodi)

La costruzione degli archi è data da due cicli annidati

for i da 0 a N-1:

for j da i+1 a N:

Con complessità Temporale O(N^2)

Inoltre, essendo un grafo denso e completo tra luoghi, ogni coppia ha un arco.

Quindi ha complessità Spaziale O(N^2)

1. **Creazione della tabella di marcia:**

Per ogni giorno, si costruisce un percorso.

All’inizio del giorno si inizializza tempo, flags e percorso (lista vuota con il *nodoAlloggio* iniziale). Finché ci sono luoghi non visitati nel grafo, sceglie il prossimo luogo da visitare.

for *nGiorno* da 0 a G-1

Quindi, ogni volta si visita e rimuove un luogo → max N iterazioni.

Ad ogni iterazione per trovare il prossimo luogo più vicino si deve guardare i nodi rimanenti → peggio ≈ O(N).

Questo ci conduce a una complessità Temporale di O(N) \* O(N) = O(N^2)

La complessità Temporale totale dell’algoritmo è data allora da:

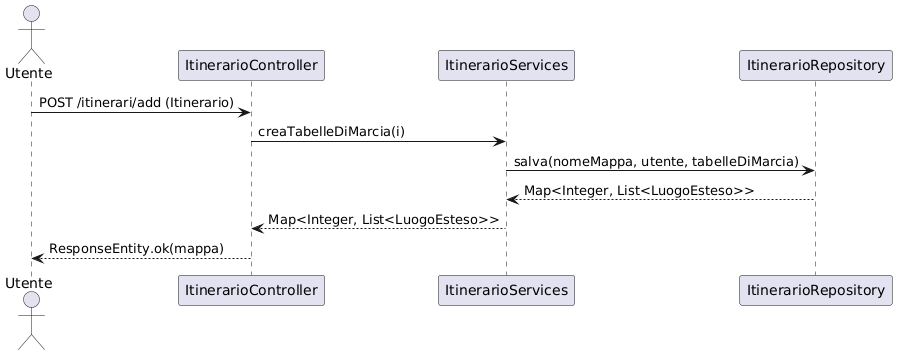
* Costruzione grafo: O(N^2)
* Per ogni giorno: O(N^2)
* Numero di giorni: G

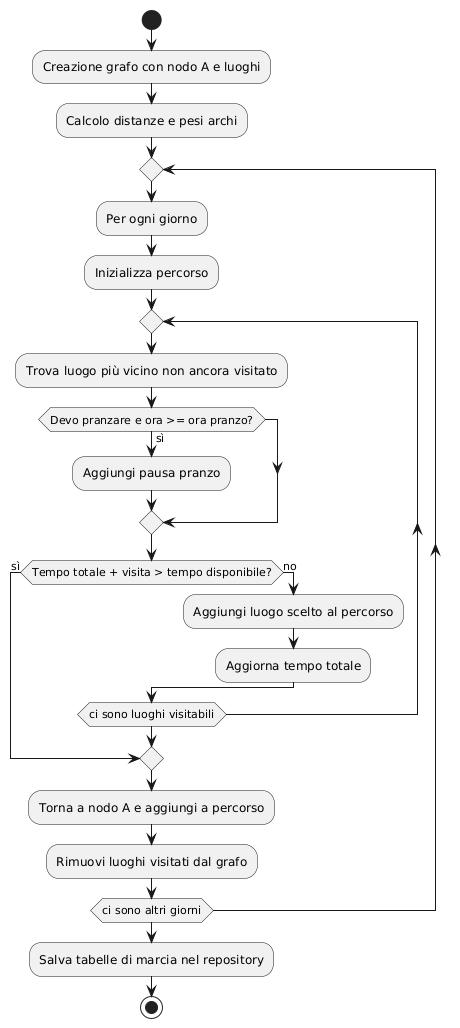
O(N^2 + G·N^2) = O(G·N^2) (termine G·N^2 domina se G cresce)

La complessità Spaziale totale dell’algoritmo è data da:

* Grafo: archi O(N^2)
* Tabella di marcia di return: O(G·N) (Ha G giorni come chiavi e per ogni giorno, la lista di luoghi è al massimo O(N) luoghi.)

O(N^2 + G·N) = O(N^2) (il termine N^2 domina per N grande)



Per comprendere meglio il funzionamento dell’API è stato costruito un diagramma delle attività: