

# CINI CHALLENGE 2023

## co-located with iCities 2023



**TOR VERGATA**  
UNIVERSITÀ DEGLI STUDI DI ROMA

## Bus4You

**Supervisore: Prof.ssa Valeria Cardellini**

Team		
Nome e Cognome	Tipo di corso di studio	Indirizzo email
Matteo Conti	Laurea Magistrale in Ingegneria Informatica	matteo.conti.977@gmail.com
Luca Falasca	Laurea Magistrale in Ingegneria Informatica	lucafalasca08@gmail.com
Stefan Adrian Huma	Laurea Magistrale in Ingegneria Informatica	stefanadrianhuma@yahoo.it

## Indice

Definizione del sistema	3
Acronimi e abbreviazioni	3
Descrizione del sistema	4
Requisiti Funzionali	5
Requisiti non Funzionali	6
Descrizione dell'architettura	7
Interfacce dei servizi esposti dal sistema	9
Progettazione di dettaglio del sistema	24
Codifica RESTful delle interfacce dei servizi	36
Il sistema realizzato	63
Bibliografia	70

## Definizione del sistema

### Acronimi e abbreviazioni

- *Itinerario richiesto, booking, prenotazione, itinerario*: La richiesta dell'utente è composta dalle coordinate della fermata di partenza, le coordinate della fermata di arrivo e la data e l'orario della partenza o arrivo.
- *Itinerario proposto*: La proposta che il sistema presenta all'utente è composta dalle coordinate della fermata di partenza, le coordinate delle fermate di arrivo, la distanza per arrivare dalla fermata di partenza a quella di arrivo in km, il costo del biglietto in Euro, la data e l'orario di partenza e arrivo.
- *Percorso, route, corsa, rotta*: Insieme di fermate ordinate associate ad un insieme di itinerari proposti, calcolata attraverso un algoritmo.
- *Compatibilità temporale*: Proprietà necessaria affinché due o più prenotazioni possano essere aggregate per cercare di generare un percorso, due prenotazioni sono compatibili temporalmente se la loro data e orario di partenza/arrivo sono "vicini" temporalmente tra di loro, cioè hanno un orario simile, e.g. due prenotazioni che hanno date diverse non sono compatibili temporalmente.
- *Compatibilità spaziale*: Proprietà necessaria affinché due o più prenotazioni possano essere aggregate per cercare di generare un percorso, due prenotazioni sono compatibili spazialmente se le coordinate delle loro fermate di partenza e arrivo sono vicine, e.g., una fermata che si trova in zona Roma centro ed una che si trova nel litorale di Ostia non sono compatibili spazialmente.
- *Compatibilità*: Proprietà che indica se due o più prenotazioni possono essere aggregate per generare un percorso, due prenotazioni sono compatibili se sono compatibili sia spazialmente che temporalmente.
- *Recommended route*: È un percorso confermato per cui si ha che l'orario e la data di partenza dell'itinerario proposto associato alla prima fermata del percorso sono futuri, cioè un percorso per cui l'autobus non è ancora partito.
- *Deviazione*: È la distanza temporale di ritardo o anticipo rispetto all'orario richiesto dall'utente
- *Deviazione inaccettabile*: È una Deviazione non accettabile per l'utente (ad esempio, l'utente ha richiesto di arrivare entro le 10:00 e l'autobus arriva alle 10:10, quindi c'è una Deviazione inaccettabile di 10 minuti)
- *Deviazione accettabile*: È una Deviazione accettabile per l'utente (ad esempio, l'utente ha richiesto di arrivare entro le 10:00 e l'autobus arriva alle 09:50, quindi c'è una Deviazione accettabile di 10 minuti)
- *Clustering*: Meccanismo con cui vengono aggregate delle prenotazioni compatibili.

## Descrizione del sistema

Bus4You è un servizio di trasporto pubblico innovativo che consente agli utenti di prenotare un *itinerario* scegliendo la fermata di partenza e di arrivo (selezionabili da un insieme reso disponibile) con le quali il sistema crea una *route* personalizzata, che, una volta confermata, verrà percorsa utilizzando autobus dedicati.

L'obiettivo principale del servizio proposto è quello di ridurre l'inquinamento cittadino, incentivando l'utilizzo dei mezzi di trasporto condivisi per quella fascia di persone che non ne fanno uso per questioni di raggiungibilità, affidabilità, comodità, disabilità, fornendo inoltre innovazione al servizio di trasporto esistente.

Ad esempio, in luoghi poco raggiungibili o mal collegati, dove non è conveniente fornire una linea di trasporto fissa, può essere preferibile un mezzo dinamico il cui tragitto è più mirato rispetto a quello di un autobus di linea. Inoltre, il servizio aspira ad essere competitivo, tentando di fornire dei tempi di percorrenza simili a quelli che si avrebbero con un mezzo privato, incentivando, quindi, le persone che abitualmente utilizzano l'automobile, a servirsi di un autobus, con la conseguente riduzione delle emissioni di gas serra, dell'inquinamento cittadino e del traffico.

L'utente può richiedere un itinerario, il sistema lo elabora, individuando altre prenotazioni compatibili e scegliendole attraverso un processo di *clustering*, per poi generare un percorso utilizzando un algoritmo dedicato.

Quando il sistema riesce a generare il percorso, comunica ai vari utenti gli *itinerari proposti*; a sua volta, l'utente può accettare l'itinerario proposto e pagare, o declinarlo. Il servizio calcola il *percorso* migliore tenendo conto di diversi vincoli, come le fermate di partenza e di arrivo volute, gli orari richiesti dagli utenti, la gestione del carburante e le possibili strade alternative.

Il costo del biglietto parte da una tariffa fissa e varia a seconda del tratto percorso dall'utente del numero di persone che partecipano alla corsa. Il sistema può anche suggerire dei punti strategici in cui è più probabile che la richiesta venga accettata. L'utente può visualizzare i *recommended routes* ed aggiungersi ad essi; in tal caso, gli altri clienti, già prenotati, ricevono del *cashback* come incentivo, da utilizzare per richieste future.

Ogni *percorso* ha una scadenza entro la quale una soglia minima di persone deve accettarlo, altrimenti verrà scartato.

L'utente, una volta accettato l'itinerario proposto, può cambiare idea, venendo rimborsato integralmente, fino al momento della scadenza o fino a che non si raggiunge la soglia minima di accettazione e il percorso viene considerato confermato.

## Scenari

### 1. L'utente vuole richiedere un *itinerario*.

Attraverso questo scenario l'utente può fare una richiesta per un mezzo di trasporto che percorrerà la tratta da lui richiesta. Se la richiesta verrà accettata un mezzo di trasporto sarà incaricato di effettuare la tratta dell'utente.

Questo permette di avere un mezzo di trasporto pubblico anche in zone non coperte in certi orari, di ridurre i tempi di viaggio e di facilitare il viaggio anche per quella categoria di persone con disabilità motorie.

### 2. L'utente vuole vedere i suoi *itinerari proposti*.

In questo scenario l'utente, dopo aver inviato la richiesta di itinerario, vuole vedere se essa è elaborata dal servizio.

In particolare, il servizio soddisferà la richiesta dell'utente e mostrerà i dettagli della sua richiesta se, oltre a lui, anche altri utenti hanno effettuato richieste simili in modo da riempire per quanto possibile il mezzo di trasporto incaricato a percorrere le loro tratte.

### 3. L'utente vuole confermare o rifiutare un suo *itinerario proposto* dal servizio.

Se l'utente è soddisfatto della tratta proposta dal servizio, allora potrà confermarla e vedere se sarà effettivamente

effettuata da un mezzo di trasporto. Se tutti gli utenti coinvolti nella tratta confermano, allora il mezzo di trasporto sarà certamente adibito alla percorrenza di quella tratta.

Un utente potrebbe anche non essere soddisfatto della tratta proposta dal servizio e semplicemente potrebbe aver cambiato idea, in tal caso esso potrà rifiutarla.

4. **L'utente vuole aggiungersi ad un *recommended route* già esistente.**

L'utente potrebbe anche visualizzare le tratte già confermate e decidere di unirsi ad una di esse, in tal caso li verrà offerto un prezzo del biglietto ridotto e dando un piccolo cashback agli utenti già prenotati per quella tratta, incentivando così l'uso del suddetto servizio.

## Requisiti Funzionali

1. **Registrazione utente:** il sistema deve fornire un modo per permettere agli utenti di registrarsi inserendo un token.
2. **Mostrare fermate:** il sistema deve fornire un modo per ottenere la lista di tutte le fermate disponibili.
3. **Mostrare fermate in un'area circoscritta:** il sistema deve fornire un modo per ottenere la lista di tutte le fermate all'interno di un rettangolo di coordinate specificato.
4. **Mostrare itinerari richiesti compatibili:** il sistema deve fornire un modo per ottenere gli *itinerari richiesti* compatibili con un *itinerario richiesto* specificato.
5. **Mostrare tutti gli itinerari richiesti compatibili:** il sistema deve fornire un modo per ottenere un insieme casuale di *itinerari richiesti* compatibili.
6. **Mostrare tutti gli itinerari richiesti compatibili:** il sistema deve fornire un modo per ottenere tutti gli insiemi di *itinerari richiesti* compatibili.
7. **Raccomandazione di itinerari:** il sistema deve fornire un modo per ottenere i *recommended routes*.
8. **Aggiunta a percorso raccomandato:** il sistema deve fornire un modo per far aggiungere un utente ad una *recommended route*.
9. **Richiesta di un percorso:** il sistema deve fornire un modo per creare un percorso, calcolato sulla base delle coordinate delle fermate specificate dall'utente.
10. **Costo itinerari proposti che compongono il percorso:** il sistema deve fornire un modo per calcolare il costo in euro dei singoli itinerari proposti che compongono il percorso creato.
11. **Distanza itinerari proposti che compongono il percorso:** il sistema deve fornire un modo per calcolare la distanza percorsa in km dei singoli itinerari proposti che compongono il percorso creato.
12. **Creazione di un percorso:** il sistema deve fornire un algoritmo per calcolare il percorso di Deviazione minima.
13. **Statistiche:** il sistema deve fornire un modo per visualizzare le statistiche relative ai percorsi e alle fermate.
14. **Sistema di suggerimento personalizzato:** il sistema deve fornire un modo per esportare un sistema di suggerimento basato sui cluster di itinerari, utilizzando il nostro sistema per fare una raccomandazione personalizzata.
15. **Creazione di percorso:** il sistema deve fornire un modo per creare un percorso senza vincoli specificati dall'utente.

16. **Calcolo costo sottopercorso:** il sistema deve fornire un modo per calcolare il costo in euro di un sottopercorso di un *recommended route*.
17. **Calcolo distanza sottopercorso:** il sistema deve fornire un modo per calcolare la distanza percorsa in km di un sottopercorso di un *recommended route*.
18. **Mostrare itinerari richiesti:** il sistema deve fornire un modo per visualizzare tutti gli *itinerari proposti*.
19. **Mostrare itinerari proposti:** il sistema deve fornire un modo per visualizzare tutti gli *itinerari proposti*.
20. **Mostrare percorsi:** il sistema deve fornire un modo per visualizzare tutti i *percorsi* creati.
21. **Interfaccia richiesta percorsi:** il sistema deve fornire un'interfaccia in cui l'utente può effettuare una prenotazione scegliendo da una mappa le fermate in cui desidera partire e arrivare.
22. **Interfaccia itinerari proposti:** il sistema deve fornire un'interfaccia in cui l'utente può visualizzare, divisi per presenti e passati, tutti i suoi itinerari proposti che fanno parte di percorsi, indicandone costo, distanza percorsa, stato, orario di partenza, orario di arrivo, data di partenza, data di arrivo, nome della fermata di partenza, nome della fermata di arrivo, stato del percorso ed una mappa che indica il tragitto percorso dall'itinerario proposto.
23. **Interfaccia recommended routes:** il sistema deve fornire un'interfaccia in cui l'utente può visualizzare tutti i *recommended routes*, mostrando i nomi delle fermate, l'orario e la data in cui si passa nelle varie fermate che compongono il percorso, inoltre il percorso deve essere mostrato su mappa. Questa interfaccia deve inoltre permettere all'utente di selezionare due fermate A e B del percorso e ottenere la distanza percorsa in km ed il costo in Euro per andare da A e B. L'utente può quindi scegliere di unirsi al percorso premendo un bottone che genera un itinerario proposto da A e B che ha costo in € e distanza in km calcolata precedentemente.

## Requisiti non Funzionali

1. **Copertura:** Il sistema deve funzionare nella zona coperta dall'ATAC a Roma.
2. **Tolleranza Deviazione:** Il percorso non deve avere itinerari con una *Deviazione* superiore ai 30 minuti.
3. **Capienza Autobus:** L'autobus ha una capienza di 8 posti.
4. **Controllo Capienza:** Gli utenti non possono aggiungersi ad autobus già pieni.
5. **Scadenza Percorso:** Gli utenti possono accettare entro massimo 12h da quando gli viene proposta la rotta.
6. **Scalabilità:** Il sistema deve essere scalabile in base al numero di richieste.
7. **Tolleranza ai guasti:** Il sistema è in grado di tollerare

## Descrizione dell'architettura

L'architettura del sistema è basata su microservizi. La comunicazione tra i vari servizi avviene tramite RPC e tramite coda di messaggi (persistente) a seconda delle esigenze, in particolare le code vengono utilizzate per disaccoppiare i microservizi core che svolgono funzionalità più onerose dal punto di vista computazionale in modo da poterli replicare. La Figura 1 illustra l'architettura del sistema.

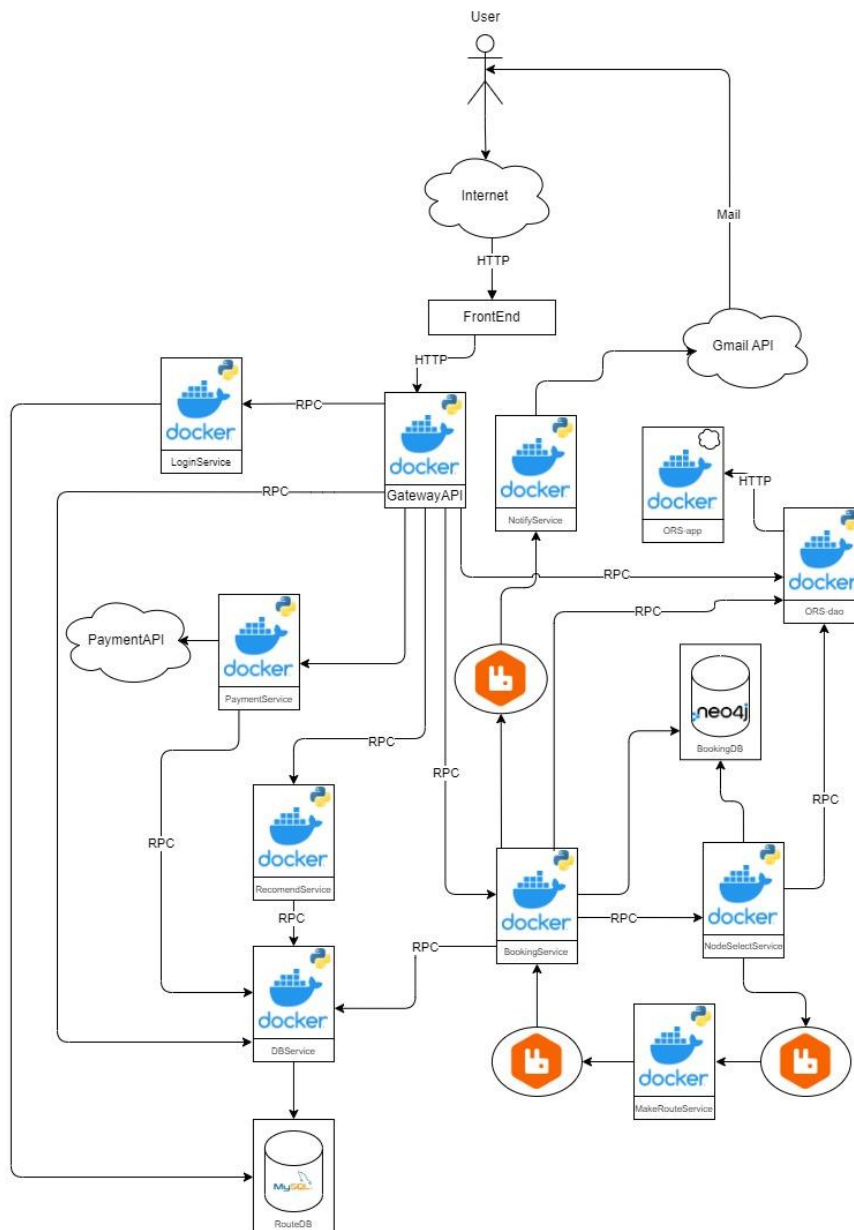


Figura 1

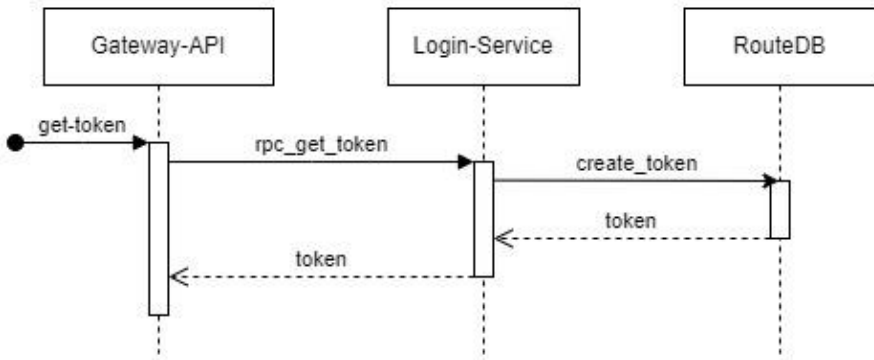
Tutti i container sono stateless in modo da poter essere facilmente replicati. In particolare, è stato opportuno scalare orizzontalmente i tre microservizi principali del sistema, che sono BookingService, NodeSelectorService e MakeRouteService; essi sono il core dell'applicazione, in quanto la creazione di un *percorso* avviene tramite la collaborazione tra questi tre microservizi. Per fare ciò sono state introdotte delle code di messaggi, le quali introducono disaccoppiamento spaziale e di sincronia tra questi tre microservizi, permettendo di scalarli orizzontalmente senza intaccare la logica dell'applicazione; infatti, ai fini della correttezza è importante che tutti i messaggi in coda vengano consumati, ma non importa quale istanza tra quelle scalate li consuma.

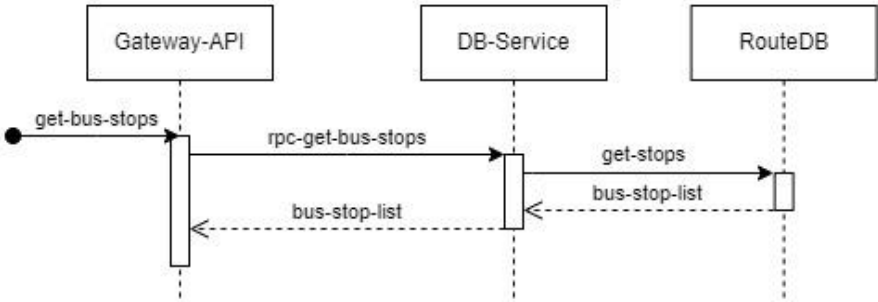
In una esecuzione dello scenario di richiesta di un *itinerario* il workflow dell'applicazione segue questo pattern:

1. La richiesta arriva a BookingService il quale inserisce un *itinerario richiesto* nel RouteDB e nel database a grafo BookingDB andando ad inserire le *compatibilità* per permettere la *clusterizzazione* degli *itinerari richiesti*.
2. Una volta fatto ciò BookingService richiede tramite chiamata RPC a NodeSelectorService di selezionare dal BookingDB un certo numero di nodi compatibili con quello appena inserito (il numero di nodi dipende dalla capienza dell'autobus).
3. NodeSelectorService recupera questi nodi e prepara le informazioni necessarie a poter eseguire l'algoritmo (ad esempio la matrice delle distanze dei nodi andando a chiederla ad OpenRouteService), e lo manda MakeRouteService tramite una coda di messaggi RabbitMQ.
4. MakeRouteService, che si occupa di eseguire l'algoritmo, è in costante attesa di messaggi dalla coda. Quando arriva un messaggio lo elabora andando a creare, tramite l'algoritmo, una *rotta* di *deviazione* minima per ogni utente. Una volta trovata la soluzione, la manda a BookingService tramite un'altra coda di messaggi e passa ad elaborare la prossima richiesta, se presente, altrimenti rimane in attesa di una richiesta.
5. BookingService riceve la *rotta* completata, calcola i vari costi e le distanze percorsa da associare ai singoli *itinerari proposti* che compongono la *rotta* e li inserisce nel RouteDB, infine invierà le informazioni degli *itinerari proposti* a NotifyService tramite coda di messaggi, il quale poi invierà una mail agli utenti associati agli *itinerari proposti* ricevuti, informandoli che la loro *prenotazione* è stata elaborata ed è stata definita una *corsa* per servirla.



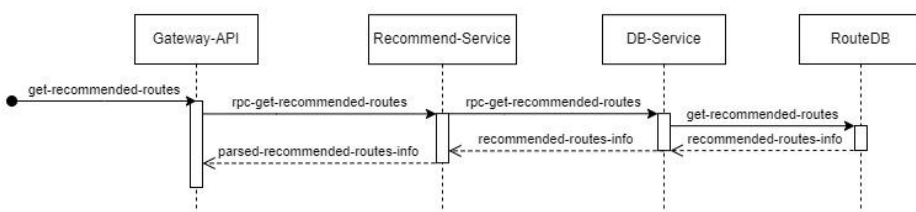
## Interfacce dei servizi esposti dal sistema

Nome	get-token
Requisiti di riferimento	Registrazione utente
Descrizione	Generazione di un token per poter utilizzare le altre operazioni dell'API.
Input	
Output	<div> <div>Descrizione dell'output</div> <div>Stringa contenente il token generato</div> </div>
Diagrammi di sequenza ed interazioni tra componenti del sistema	 <pre> sequenceDiagram     participant Client     participant Gateway-API     participant Login-Service     participant RouteDB      Client-&gt;&gt;Gateway-API: get-token     activate Gateway-API     Gateway-API-&gt;&gt;Login-Service: rpc_get_token     activate Login-Service     Login-Service-&gt;&gt;RouteDB: create_token     activate RouteDB     RouteDB--&gt;&gt;Login-Service: token     deactivate RouteDB     Login-Service--&gt;&gt;Gateway-API: token     deactivate Login-Service     Gateway-API--&gt;&gt;Client: token     deactivate Gateway-API           </pre>

Nome	get-bus-stops
Requisiti di riferimento	Mostrare fermate
Descrizione	Get della lista di tutte le fermate gestite dal servizio.
Input	
Output	<div> <b>Descrizione dell'output</b>            Lista contenente tutte le informazioni di tutte le fermate, in particolare per ogni fermata si hanno nome, latitudine e longitudine.         </div>
Diagrammi di sequenza ed interazioni tra componenti del sistema	 <pre> sequenceDiagram     participant Gateway-API     participant DB-Service     participant RouteDB     Gateway-API-&gt;&gt;Gateway-API: get-bus-stops     activate Gateway-API     Gateway-API-&gt;&gt;DB-Service: rpc-get-bus-stops     activate DB-Service     DB-Service-&gt;&gt;RouteDB: get-stops     activate RouteDB     RouteDB--&gt;&gt;DB-Service: bus-stop-list     deactivate RouteDB     DB-Service--&gt;&gt;Gateway-API: bus-stop-list     deactivate DB-Service     deactivate Gateway-API           </pre>

Nome	get-bus-stops-in-rect											
Requisiti di riferimento	Mostrare fermate in area circoscritta.											
Descrizione	Get della lista di tutte le fermate le cui coordinate si trovano all'interno di un rettangolo di coordinate.											
Input	<table><tr><th>Nome parametro</th><th>Descrizione del parametro</th></tr><tr><td>X (*)</td><td>Latitudine dell'angolo in basso a sinistra del rettangolo.</td></tr><tr><td>Y (*)</td><td>Longitudine dell'angolo in basso a sinistra del rettangolo.</td></tr><tr><td>Width (*)</td><td>Larghezza del rettangolo.</td></tr><tr><td>Height (*)</td><td>Altezza del rettangolo.</td></tr></table>		Nome parametro	Descrizione del parametro	X (*)	Latitudine dell'angolo in basso a sinistra del rettangolo.	Y (*)	Longitudine dell'angolo in basso a sinistra del rettangolo.	Width (*)	Larghezza del rettangolo.	Height (*)	Altezza del rettangolo.
Nome parametro	Descrizione del parametro											
X (*)	Latitudine dell'angolo in basso a sinistra del rettangolo.											
Y (*)	Longitudine dell'angolo in basso a sinistra del rettangolo.											
Width (*)	Larghezza del rettangolo.											
Height (*)	Altezza del rettangolo.											
Output	<table><tr><th>Descrizione dell'output</th></tr><tr><td>Lista contenente tutte le informazioni di tutte le fermate le cui coordinate si trovano all'interno del rettangolo di coordinate specificato (il quale ha area <math>(x + width) \times (y + height)</math>), in particolare per ogni fermata si hanno nome, latitudine e longitudine.</td></tr></table>		Descrizione dell'output	Lista contenente tutte le informazioni di tutte le fermate le cui coordinate si trovano all'interno del rettangolo di coordinate specificato (il quale ha area $(x + width) \times (y + height)$ ), in particolare per ogni fermata si hanno nome, latitudine e longitudine.								
Descrizione dell'output												
Lista contenente tutte le informazioni di tutte le fermate le cui coordinate si trovano all'interno del rettangolo di coordinate specificato (il quale ha area $(x + width) \times (y + height)$ ), in particolare per ogni fermata si hanno nome, latitudine e longitudine.												
Diagrammi di sequenza ed interazioni tra componenti del sistema	<pre>sequenceDiagram     participant G as Gateway-API     participant D as DB-Service     participant R as RouteDB     G-&gt;&gt;G: get-bus-stops-in-rect     G-&gt;&gt;D: rpc-get-bus-stops-in-rect     D-&gt;&gt;R: get-stops     R--&gt;&gt;D: bus-stop-list     D--&gt;&gt;G: bus-stop-list</pre>											

Nome	route-from-map																							
Requisiti di riferimento	Richiesta di un percorso.																							
Descrizione	Richiesta di un itinerario richiesto, per avviare la procedura di prenotazione.																							
Input	<table><tr><th>Nome parametro</th><th>Descrizione del parametro</th></tr><tr><td>User (*)</td><td>Token generato con get-token o mail di un utente registrato.</td></tr><tr><td>Starting point (*)</td><td>Nome della fermata di partenza.</td></tr><tr><td>Starting latitude (*)</td><td>Latitudine della fermata di partenza.</td></tr><tr><td>Starting longitude (*)</td><td>Longitudine della fermata di partenza.</td></tr><tr><td>Ending point (*)</td><td>Nome della fermata di arrivo.</td></tr><tr><td>Ending latitude (*)</td><td>Latitudine della fermata di arrivo.</td></tr><tr><td>Ending longitude (*)</td><td>Longitudine della fermata di arrivo.</td></tr><tr><td>Start or Finish (*)</td><td>Flag che indica se l’orario inserito è l’orario desiderato di arrivo o di partenza.</td></tr><tr><td>Date (*)</td><td>Data desiderata di partenza o arrivo, essere interpretata come partenza o arrivo dipende dal parametro Start or Finish.</td></tr><tr><td>Time (*)</td><td>Orario desiderato di partenza o arrivo, essere interpretata come partenza o arrivo dipende dal parametro Start or Finish.</td></tr></table> <p>(*) = obbligatorio</p>		Nome parametro	Descrizione del parametro	User (*)	Token generato con get-token o mail di un utente registrato.	Starting point (*)	Nome della fermata di partenza.	Starting latitude (*)	Latitudine della fermata di partenza.	Starting longitude (*)	Longitudine della fermata di partenza.	Ending point (*)	Nome della fermata di arrivo.	Ending latitude (*)	Latitudine della fermata di arrivo.	Ending longitude (*)	Longitudine della fermata di arrivo.	Start or Finish (*)	Flag che indica se l’orario inserito è l’orario desiderato di arrivo o di partenza.	Date (*)	Data desiderata di partenza o arrivo, essere interpretata come partenza o arrivo dipende dal parametro Start or Finish.	Time (*)	Orario desiderato di partenza o arrivo, essere interpretata come partenza o arrivo dipende dal parametro Start or Finish.
Nome parametro	Descrizione del parametro																							
User (*)	Token generato con get-token o mail di un utente registrato.																							
Starting point (*)	Nome della fermata di partenza.																							
Starting latitude (*)	Latitudine della fermata di partenza.																							
Starting longitude (*)	Longitudine della fermata di partenza.																							
Ending point (*)	Nome della fermata di arrivo.																							
Ending latitude (*)	Latitudine della fermata di arrivo.																							
Ending longitude (*)	Longitudine della fermata di arrivo.																							
Start or Finish (*)	Flag che indica se l’orario inserito è l’orario desiderato di arrivo o di partenza.																							
Date (*)	Data desiderata di partenza o arrivo, essere interpretata come partenza o arrivo dipende dal parametro Start or Finish.																							
Time (*)	Orario desiderato di partenza o arrivo, essere interpretata come partenza o arrivo dipende dal parametro Start or Finish.																							
Output	<table><tr><th>Descrizione dell’output</th></tr><tr><td>Id dell’itinerario richiesto inserito.</td></tr></table>		Descrizione dell’output	Id dell’itinerario richiesto inserito.																				
Descrizione dell’output																								
Id dell’itinerario richiesto inserito.																								
Diagrammi di sequenza ed interazioni tra componenti del sistema	<pre>sequenceDiagram     participant Gateway-API     participant Booking-Service     participant DB-Service     participant RouteDB     participant BookingDB     participant Node Selector Service     participant ORS-Geo     participant ORS-App     participant Main Route Service     participant Notify-Service      Gateway-API-&gt;&gt;Booking-Service: rpc-insert-booking     Booking-Service-&gt;&gt;RouteDB: insert-booking     RouteDB-&gt;&gt;BookingDB: insert-booking-into cluster     BookingDB-&gt;&gt;Node Selector Service: msg send node for computation     Node Selector Service-&gt;&gt;ORS-Geo: get-cluster     ORS-Geo-&gt;&gt;ORS-App: rpc-get distance-matrix     ORS-App-&gt;&gt;Main Route Service: http-get distance-matrix     Main Route Service--&gt;&gt;ORS-App: distance-matrix     ORS-App-&gt;&gt;Main Route Service: msg-main-route     Main Route Service-&gt;&gt;Main Route Service: msg-route-ready     Main Route Service-&gt;&gt;Booking-Service: rpc-insert-route-info     Booking-Service-&gt;&gt;RouteDB: insert route-info     Main Route Service-&gt;&gt;Notify-Service: msg notify users</pre>																							

Nome	get-recommended-routes
Requisiti di riferimento	Raccomandazione di itinerari.
Descrizione	Get di tutte le recommended routes.
Input	
Output	<div> <b>Descrizione dell'output</b> </div> <p>             Lista contenente le informazioni di tutte le fermate di ogni percorso confermato che non è ancora partito, per ogni percorso sono presenti il suo id e la lista delle informazioni delle fermate che compongono il percorso. La lista delle informazioni delle fermate contiene, per ogni fermata, nome, latitudine, longitudine, un numero che indica la posizione della fermata all'interno del percorso (e.g., 2, seconda fermata del percorso), quante persone salgono e scendono dalla fermata e l'orario in cui si passa nella fermata.           </p>
Diagrammi di sequenza ed interazioni tra componenti del sistema	 <pre> sequenceDiagram     participant Gateway-API     participant Recommend-Service     participant DB-Service     participant RouteDB      Gateway-API-&gt;&gt;Gateway-API: get-recommended-routes     Gateway-API-&gt;&gt;Recommend-Service: rpc-get-recommended-routes     Recommend-Service-&gt;&gt;DB-Service: rpc-get-recommended-routes     DB-Service-&gt;&gt;RouteDB: get-recommended-routes     RouteDB--&gt;&gt;DB-Service: recommended-routes-info     DB-Service--&gt;&gt;Recommend-Service: recommended-routes-info     Recommend-Service--&gt;&gt;Gateway-API: parsed-recommended-routes-info           </pre>

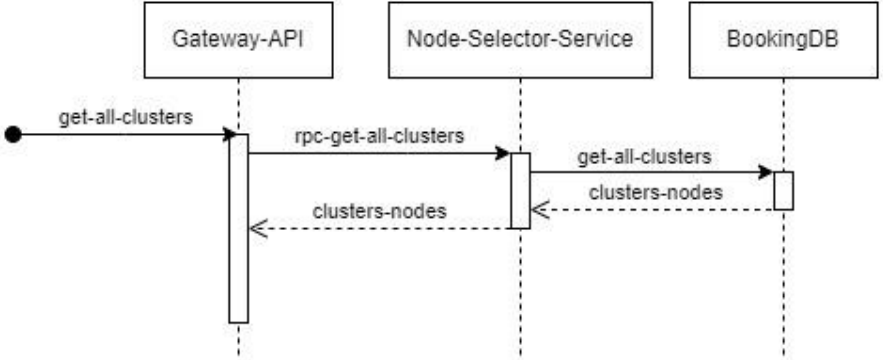
Nome	get-km-price-from-subroute							
Requisiti di riferimento	Calcolo costo sottopercorso.							
Descrizione	Get del costo e della distanza percorsa tra due fermate A e B appartenenti ad un recommended route.							
Input	<table><tr><th>Nome parametro</th><th>Descrizione del parametro</th></tr><tr><td>Bus stops (*)</td><td>Lista contenente tutte le coordinate delle fermate tra A e B (comprese).</td></tr><tr><td>Route id (*)</td><td>Id del recommended route.</td></tr></table>		Nome parametro	Descrizione del parametro	Bus stops (*)	Lista contenente tutte le coordinate delle fermate tra A e B (comprese).	Route id (*)	Id del recommended route.
	Nome parametro	Descrizione del parametro						
	Bus stops (*)	Lista contenente tutte le coordinate delle fermate tra A e B (comprese).						
	Route id (*)	Id del recommended route.						
(*) = obbligatorio								
Output	<table><tr><th>Descrizione dell'output</th></tr><tr><td>Una lista contenente il prezzo e la distanza percorsa per arrivare da A e B</td></tr></table>		Descrizione dell'output	Una lista contenente il prezzo e la distanza percorsa per arrivare da A e B				
	Descrizione dell'output							
Una lista contenente il prezzo e la distanza percorsa per arrivare da A e B								
Diagrammi di sequenza ed interazioni tra componenti del sistema	<pre>sequenceDiagram     participant User     participant Gateway-API     participant DB-Service     participant RouteDB      User-&gt;&gt;Gateway-API: get-km-price-from-subroute     activate Gateway-API     Gateway-API-&gt;&gt;DB-Service: rpc-get-total-km     activate DB-Service     DB-Service-&gt;&gt;RouteDB: get-route-distance     activate RouteDB     RouteDB--&gt;&gt;DB-Service: route-distance     deactivate RouteDB     DB-Service--&gt;&gt;Gateway-API: total-km     deactivate DB-Service     Gateway-API--&gt;&gt;User:      deactivate Gateway-API</pre>							

Nome	join-recommended-route																							
Requisiti di riferimento	Aggiunta a percorso raccomandato.																							
Descrizione	Aggiungere un itinerario proposto ad un recommended route, l’itinerario proposto che viene aggiunto deve avere le fermate di partenza e arrivo che coincidono con fermate già presenti nel percorso.																							
Input	<table><thead><tr><th>Nome parametro</th><th>Descrizione del parametro</th></tr></thead><tbody><tr><td>Route id (*)</td><td>Id del recommended route a cui va aggiunto l’itinerario proposto</td></tr><tr><td>Starting latitude (*)</td><td>Latitudine della fermata di partenza dell’itinerario proposto.</td></tr><tr><td>Starting longitude(*)</td><td>Longitudine della fermata di partenza dell’itinerario proposto.</td></tr><tr><td>Ending latitude (*)</td><td>Latitudine della fermata di arrivo dell’itinerario proposto.</td></tr><tr><td>Ending longitude (*)</td><td>Longitudine della fermata di arrivo dell’itinerario proposto.</td></tr><tr><td>Start datetime(*)</td><td>Data ed orario di partenza dell’itinerario proposto.</td></tr><tr><td>End datetime (*)</td><td>Data ed orario di arrivo dell’itinerario proposta.</td></tr><tr><td>Price (*)</td><td>Costo dell’itinerario proposto, va determinato utilizzando precedentemente get-km-price-from-subroute.</td></tr><tr><td>Distance (*)</td><td>Distanza percorsa dall’itinerario proposto get-km-price-from-subroute.</td></tr><tr><td>User (*)</td><td>Token o mail dell’utente che richiede l’itinerario proposto.</td></tr></tbody></table> <p>(*) = obbligatorio</p>		Nome parametro	Descrizione del parametro	Route id (*)	Id del recommended route a cui va aggiunto l’itinerario proposto	Starting latitude (*)	Latitudine della fermata di partenza dell’itinerario proposto.	Starting longitude(*)	Longitudine della fermata di partenza dell’itinerario proposto.	Ending latitude (*)	Latitudine della fermata di arrivo dell’itinerario proposto.	Ending longitude (*)	Longitudine della fermata di arrivo dell’itinerario proposto.	Start datetime(*)	Data ed orario di partenza dell’itinerario proposto.	End datetime (*)	Data ed orario di arrivo dell’itinerario proposta.	Price (*)	Costo dell’itinerario proposto, va determinato utilizzando precedentemente get-km-price-from-subroute.	Distance (*)	Distanza percorsa dall’itinerario proposto get-km-price-from-subroute.	User (*)	Token o mail dell’utente che richiede l’itinerario proposto.
Nome parametro	Descrizione del parametro																							
Route id (*)	Id del recommended route a cui va aggiunto l’itinerario proposto																							
Starting latitude (*)	Latitudine della fermata di partenza dell’itinerario proposto.																							
Starting longitude(*)	Longitudine della fermata di partenza dell’itinerario proposto.																							
Ending latitude (*)	Latitudine della fermata di arrivo dell’itinerario proposto.																							
Ending longitude (*)	Longitudine della fermata di arrivo dell’itinerario proposto.																							
Start datetime(*)	Data ed orario di partenza dell’itinerario proposto.																							
End datetime (*)	Data ed orario di arrivo dell’itinerario proposta.																							
Price (*)	Costo dell’itinerario proposto, va determinato utilizzando precedentemente get-km-price-from-subroute.																							
Distance (*)	Distanza percorsa dall’itinerario proposto get-km-price-from-subroute.																							
User (*)	Token o mail dell’utente che richiede l’itinerario proposto.																							
Output	<table><thead><tr><th>Descrizione dell’output</th></tr></thead><tbody><tr><td>Esito positivo o negativo dell’operazione.</td></tr></tbody></table>		Descrizione dell’output	Esito positivo o negativo dell’operazione.																				
Descrizione dell’output																								
Esito positivo o negativo dell’operazione.																								
Diagrammi di sequenza ed interazioni tra componenti del sistema	<pre>sequenceDiagram     participant Gateway-API     participant Recommend-Service     participant Payment-Service     participant DB-Service     participant RouteDB      Gateway-API-&gt;&gt;Recommend-Service: rpc-join-recommended-routes     activate Recommend-Service     Recommend-Service-&gt;&gt;Payment-Service: rpc-join-recommended-routes     activate Payment-Service     Payment-Service-&gt;&gt;DB-Service: rpc-join-recommended-routes     activate DB-Service     DB-Service-&gt;&gt;RouteDB: join-recommended-routes     activate RouteDB     RouteDB--&gt;&gt;DB-Service: outcome     deactivate RouteDB     DB-Service--&gt;&gt;Payment-Service: outcome     deactivate DB-Service     Payment-Service--&gt;&gt;Recommend-Service: outcome     deactivate Payment-Service     Recommend-Service--&gt;&gt;Gateway-API: outcome     deactivate Recommend-Service</pre>																							

Nome	get-random-cluster-from-it-id					
Requisiti di riferimento	Mostrare itinerari proposti compatibili.					
Descrizione	Get dal database a grafo del cluster di itinerari richiesti compatibili con quello specificato.					
Input	<table><tr><th>Nome parametro</th><th>Descrizione del parametro</th></tr><tr><td>Id itinerario richiesto (*)</td><td>Id dell’itinerario di cui si vuole fare la get del cluster di appartenenza.</td></tr></table> <p>(*) = obbligatorio</p>		Nome parametro	Descrizione del parametro	Id itinerario richiesto (*)	Id dell’itinerario di cui si vuole fare la get del cluster di appartenenza.
Nome parametro	Descrizione del parametro					
Id itinerario richiesto (*)	Id dell’itinerario di cui si vuole fare la get del cluster di appartenenza.					
Output	<table><tr><th>Descrizione dell’output</th></tr><tr><td>Lista contenente le informazioni di tutti gli itinerari richiesti e le corrispondenti relazioni di compatibilità presenti nel cluster degli itinerari richiesti compatibili con quello di cui si è specificato l’id.</td></tr></table>		Descrizione dell’output	Lista contenente le informazioni di tutti gli itinerari richiesti e le corrispondenti relazioni di compatibilità presenti nel cluster degli itinerari richiesti compatibili con quello di cui si è specificato l’id.		
Descrizione dell’output						
Lista contenente le informazioni di tutti gli itinerari richiesti e le corrispondenti relazioni di compatibilità presenti nel cluster degli itinerari richiesti compatibili con quello di cui si è specificato l’id.						
Diagrammi di sequenza ed interazioni tra componenti del sistema	<pre>sequenceDiagram     participant User     participant Gateway-API     participant Node-Selector-Service     participant BookingDB      User-&gt;&gt;Gateway-API: get-random-cluster-from-it-id     activate Gateway-API     Gateway-API-&gt;&gt;Node-Selector-Service: rpc-get-random-cluster-from-it-id     activate Node-Selector-Service     Node-Selector-Service-&gt;&gt;BookingDB: get-random-cluster-from-it-id     activate BookingDB     BookingDB--&gt;&gt;Node-Selector-Service: cluster-nodes     deactivate BookingDB     Node-Selector-Service--&gt;&gt;Gateway-API: cluster-nodes     deactivate Node-Selector-Service     Gateway-API--&gt;&gt;User:</pre>					

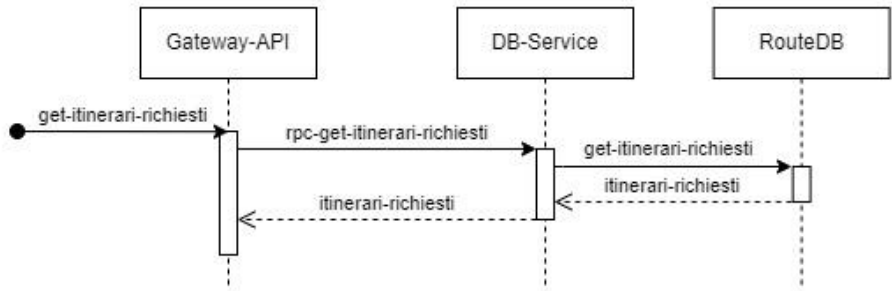


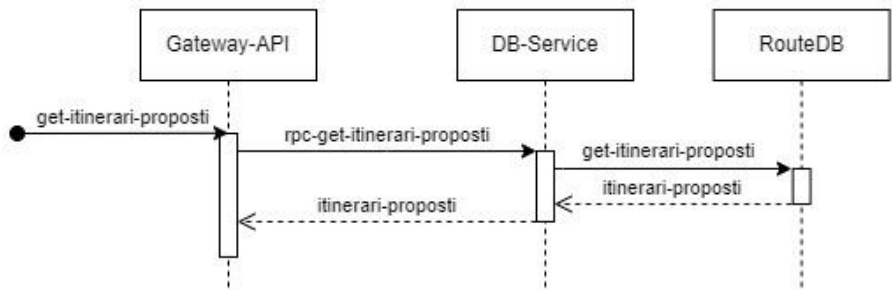
Nome	get-random-cluster
Requisiti di riferimento	Mostrare itinerari proposti compatibili casuali.
Descrizione	Get dal database a grafo di un cluster casuale di itinerari richiesti compatibili.
Input	
Output	<div> <b>Descrizione dell'output</b>            Lista contenente le informazioni di tutti gli itinerari richiesti e le corrispondenti relazioni di compatibilità presenti nel cluster scelto casualmente.         </div>
Diagrammi di sequenza ed interazioni tra componenti del sistema	<pre> sequenceDiagram     participant External     participant Gateway-API     participant Node-Selector-Service     participant BookingDB      External-&gt;&gt;Gateway-API: get-random-cluster     activate Gateway-API     Gateway-API-&gt;&gt;Node-Selector-Service: rpc-get-random-cluster     activate Node-Selector-Service     Node-Selector-Service-&gt;&gt;BookingDB: get-random-booking     activate BookingDB     BookingDB--&gt;&gt;Node-Selector-Service: booking     deactivate BookingDB     Node-Selector-Service--&gt;&gt;Gateway-API: booking     deactivate Node-Selector-Service     Gateway-API-&gt;&gt;Node-Selector-Service: rpc-get-random-cluster-from-it-id     activate Node-Selector-Service     Node-Selector-Service-&gt;&gt;BookingDB: get-random-cluster-from-it-id     activate BookingDB     BookingDB--&gt;&gt;Node-Selector-Service: cluster-nodes     deactivate BookingDB     Node-Selector-Service--&gt;&gt;Gateway-API: cluster-nodes     deactivate Node-Selector-Service     deactivate Gateway-API           </pre>

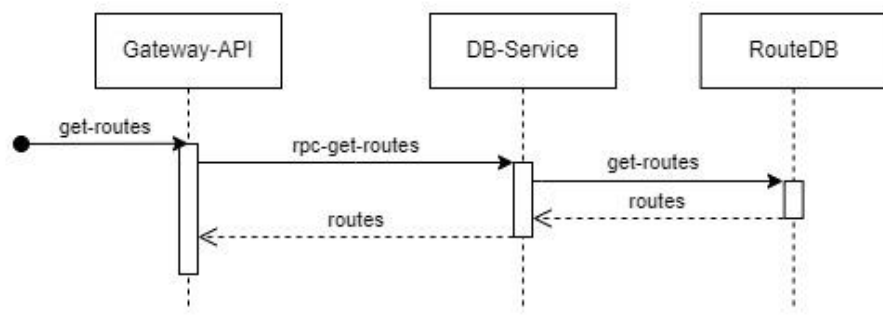
Nome	get-all-clusters
Requisiti di riferimento	Mostrare tutti gli itinerari proposti compatibili.
Descrizione	Get di tutti i cluster di itinerari compatibili presenti su database a grafo.
Input	
Output	<div> <b>Descrizione dell'output</b>            Lista contenente le informazioni di tutti gli itinerari richiesti e le corrispondenti relazioni di compatibilità presenti in tutti i cluster.         </div>
Diagrammi di sequenza ed interazioni tra componenti del sistema	 <pre> sequenceDiagram     participant Start     participant Gateway-API     participant Node-Selector-Service     participant BookingDB      Start-&gt;&gt;Gateway-API: get-all-clusters     activate Gateway-API     Gateway-API-&gt;&gt;Node-Selector-Service: rpc-get-all-clusters     activate Node-Selector-Service     Node-Selector-Service-&gt;&gt;BookingDB: get-all-clusters     activate BookingDB     BookingDB--&gt;&gt;Node-Selector-Service: clusters-nodes     deactivate BookingDB     Node-Selector-Service--&gt;&gt;Gateway-API: clusters-nodes     deactivate Node-Selector-Service     deactivate Gateway-API           </pre>

Nome	make-route-raw											
Requisiti di riferimento	Creazione di un percorso.											
Descrizione	Esecuzione dell’algoritmo di creazione del percorso, usando una matrice delle distanze personalizzata											
Input	<table><tr><th>Nome parametro</th><th>Descrizione del parametro</th></tr><tr><td>Matrice delle distanze(*)</td><td>È una matrice delle varie distanze temporali tra i nodi</td></tr><tr><td>Hash delle precedenze(*) )</td><td>È un hash che associa ad ogni nodo i nodi che devono per forza succedergli nella rotta finale</td></tr><tr><td>Limiti dei nodi(*)</td><td>Un dizionario che associa ad ogni nodo dei limiti temporali per quando ci si può passare</td></tr><tr><td>Itinerari(*)</td><td>Dizionario che associa ad ogni itinerario il suo nodo di partenza e il suo nodo di arrivo</td></tr></table> <p>(*) = obbligatorio</p>		Nome parametro	Descrizione del parametro	Matrice delle distanze(*)	È una matrice delle varie distanze temporali tra i nodi	Hash delle precedenze(*) )	È un hash che associa ad ogni nodo i nodi che devono per forza succedergli nella rotta finale	Limiti dei nodi(*)	Un dizionario che associa ad ogni nodo dei limiti temporali per quando ci si può passare	Itinerari(*)	Dizionario che associa ad ogni itinerario il suo nodo di partenza e il suo nodo di arrivo
Nome parametro	Descrizione del parametro											
Matrice delle distanze(*)	È una matrice delle varie distanze temporali tra i nodi											
Hash delle precedenze(*) )	È un hash che associa ad ogni nodo i nodi che devono per forza succedergli nella rotta finale											
Limiti dei nodi(*)	Un dizionario che associa ad ogni nodo dei limiti temporali per quando ci si può passare											
Itinerari(*)	Dizionario che associa ad ogni itinerario il suo nodo di partenza e il suo nodo di arrivo											
Output	<table><tr><th>Descrizione dell’output</th></tr><tr><td>Ritorna la lista dei nodi ordinati che rappresentano il percorso, quanto tempo si impiega a percorrerlo, il numero di job con Deviazione inaccettabile, Deviazione inaccettabile <u>media</u>, Deviazione inaccettabile massima, Deviazione accettabile media, Deviazione accettabile massima</td></tr></table>		Descrizione dell’output	Ritorna la lista dei nodi ordinati che rappresentano il percorso, quanto tempo si impiega a percorrerlo, il numero di job con Deviazione inaccettabile, Deviazione inaccettabile <u>media</u> , Deviazione inaccettabile massima, Deviazione accettabile media, Deviazione accettabile massima								
Descrizione dell’output												
Ritorna la lista dei nodi ordinati che rappresentano il percorso, quanto tempo si impiega a percorrerlo, il numero di job con Deviazione inaccettabile, Deviazione inaccettabile <u>media</u> , Deviazione inaccettabile massima, Deviazione accettabile media, Deviazione accettabile massima												
Diagrammi di sequenza ed interazioni tra componenti del sistema	<pre>sequenceDiagram     participant Gateway-API     participant MakeRouteService     Gateway-API-&gt;&gt;MakeRouteService: make-route-raw     MakeRouteService--&gt;&gt;Gateway-API: route</pre>											

Nome	make-route												
Requisiti di riferimento	Creazione di un percorso.												
Descrizione	Esecuzione dell’algoritmo a partire dai nodi in coordinate, la matrice viene calcolata tramite OpenRouteService												
Input	<table><tr><th>Nome parametro</th><th>Descrizione del parametro</th></tr><tr><td>Coordinate dei nodi</td><td>Lista delle coordinate di tutti i nodi</td></tr><tr><td>Hash delle precedenze(*)</td><td>È un hash che associa ad ogni nodo i nodi che devono per forza succedergli nella rotta finale</td></tr><tr><td>Limiti dei nodi(*)</td><td>Un dizionario che associa ad ogni nodo dei limiti temporali per quando ci si può passare</td></tr><tr><td>Itinerari(*)</td><td>Dizionario che associa ad ogni itinerario il suo nodo di partenza e il suo nodo di arrivo</td></tr></table> <p>(*) = obbligatorio</p>			Nome parametro	Descrizione del parametro	Coordinate dei nodi	Lista delle coordinate di tutti i nodi	Hash delle precedenze(*)	È un hash che associa ad ogni nodo i nodi che devono per forza succedergli nella rotta finale	Limiti dei nodi(*)	Un dizionario che associa ad ogni nodo dei limiti temporali per quando ci si può passare	Itinerari(*)	Dizionario che associa ad ogni itinerario il suo nodo di partenza e il suo nodo di arrivo
Nome parametro	Descrizione del parametro												
Coordinate dei nodi	Lista delle coordinate di tutti i nodi												
Hash delle precedenze(*)	È un hash che associa ad ogni nodo i nodi che devono per forza succedergli nella rotta finale												
Limiti dei nodi(*)	Un dizionario che associa ad ogni nodo dei limiti temporali per quando ci si può passare												
Itinerari(*)	Dizionario che associa ad ogni itinerario il suo nodo di partenza e il suo nodo di arrivo												
Output	<table><tr><th>Descrizione dell’output</th></tr><tr><td>Ritorna la lista dei nodi ordinati che rappresentano il percorso, quanto tempo si impiega a percorrerlo, il numero di job con Deviazione inaccettabile, Deviazione inaccettabile media, Deviazione inaccettabile massima, Deviazione accettabile media, Deviazione accettabile massima</td></tr></table>			Descrizione dell’output	Ritorna la lista dei nodi ordinati che rappresentano il percorso, quanto tempo si impiega a percorrerlo, il numero di job con Deviazione inaccettabile, Deviazione inaccettabile media, Deviazione inaccettabile massima, Deviazione accettabile media, Deviazione accettabile massima								
Descrizione dell’output													
Ritorna la lista dei nodi ordinati che rappresentano il percorso, quanto tempo si impiega a percorrerlo, il numero di job con Deviazione inaccettabile, Deviazione inaccettabile media, Deviazione inaccettabile massima, Deviazione accettabile media, Deviazione accettabile massima													
Diagrammi di sequenza ed interazioni tra componenti del sistema	<pre>sequenceDiagram     participant Gateway-API     participant Node-Selector-Service     participant ORS-Dao     participant ORS-App     participant Make-Route-Service      Gateway-API-&gt;&gt;Gateway-API: make-route     activate Gateway-API     Gateway-API-&gt;&gt;Node-Selector-Service: rpc-make-route     deactivate Gateway-API     activate Node-Selector-Service     Node-Selector-Service-&gt;&gt;ORS-Dao: rpc-get-distance-matrix     deactivate Node-Selector-Service     activate ORS-Dao     ORS-Dao-&gt;&gt;ORS-App: http-get-distance-matrix     deactivate ORS-Dao     activate ORS-App     ORS-App--&gt;&gt;ORS-Dao: distance-matrix     deactivate ORS-App     ORS-Dao--&gt;&gt;Node-Selector-Service: distance-matrix     deactivate ORS-Dao     Node-Selector-Service-&gt;&gt;Make-Route-Service: rpc-make-route     deactivate Node-Selector-Service     activate Make-Route-Service     Make-Route-Service--&gt;&gt;Node-Selector-Service: route     deactivate Make-Route-Service     Node-Selector-Service--&gt;&gt;Gateway-API: route     deactivate Node-Selector-Service     deactivate Gateway-API</pre>												

Nome	get-itinerari-richiesti
Requisiti di riferimento	Mostrare itinerari richiesti
Descrizione	Get dal database relazionale di tutti gli itinerari richiesti presenti.
Input	
Output	<div> <b>Descrizione dell'output</b>            Lista contenente le informazioni di tutti gli itinerari richiesti.         </div>
Diagrammi di sequenza ed interazioni tra componenti del sistema	 <pre> sequenceDiagram     participant Start     participant Gateway-API     participant DB-Service     participant RouteDB      Start-&gt;&gt;Gateway-API: get-itinerari-richiesti     activate Gateway-API     Gateway-API-&gt;&gt;DB-Service: rpc-get-itinerari-richiesti     activate DB-Service     DB-Service-&gt;&gt;RouteDB: get-itinerari-richiesti     activate RouteDB     RouteDB--&gt;&gt;DB-Service: itinerari-richiesti     deactivate RouteDB     DB-Service--&gt;&gt;Gateway-API: itinerari-richiesti     deactivate DB-Service     Gateway-API--&gt;&gt;Start:      deactivate Gateway-API           </pre>

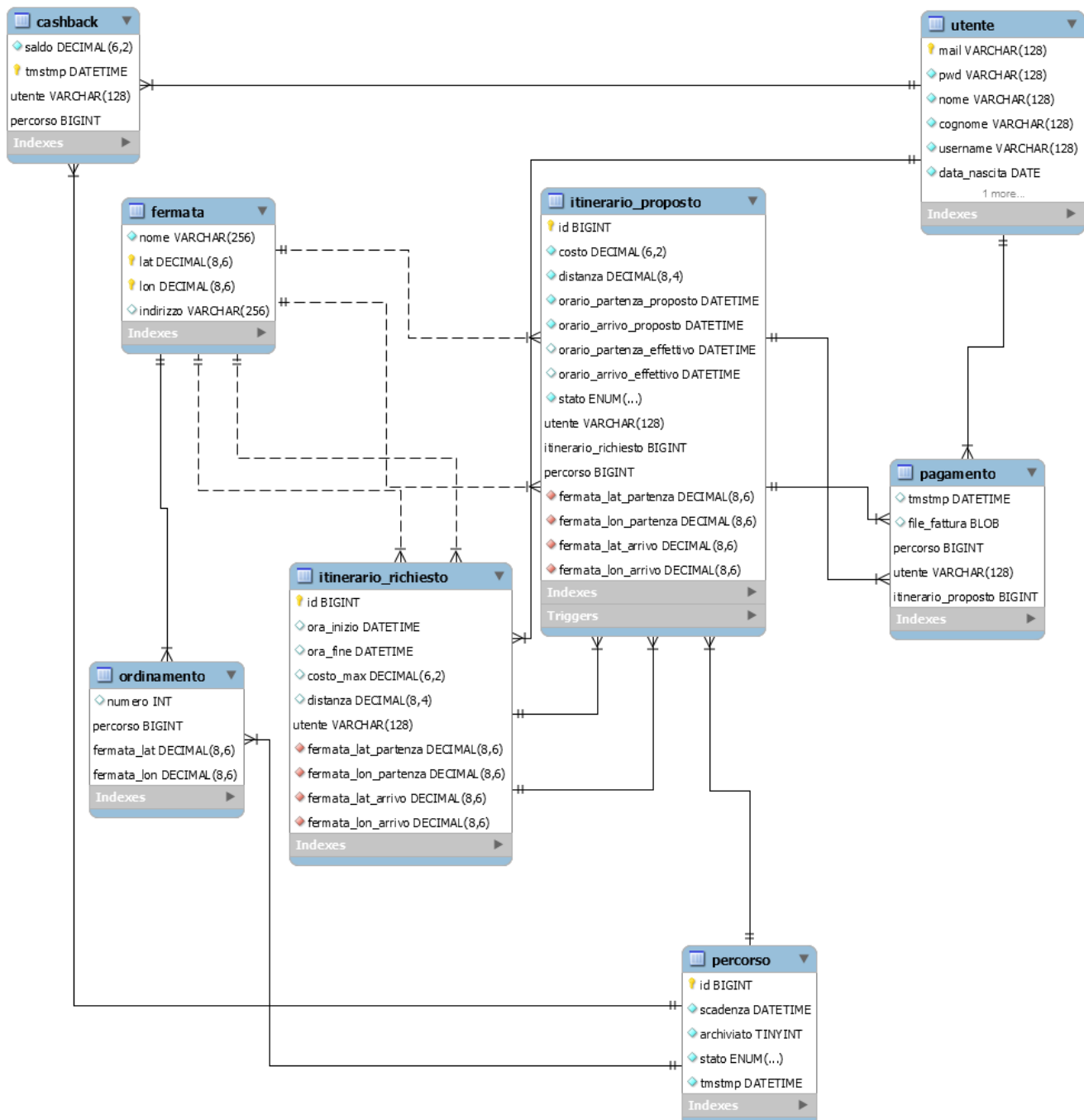
Nome	get-itinerari-proposti
Requisiti di riferimento	Mostrare itinerari proposti
Descrizione	Get dal database relazionale di tutti gli itinerari proposti presenti.
Input	
Output	<div>Descrizione dell'output</div> <div>Lista contenente le informazioni di tutti gli itinerari proposti.</div>
Diagrammi di sequenza ed interazioni tra componenti del sistema	 <pre> sequenceDiagram     participant Start as ( )     participant Gateway-API     participant DB-Service     participant RouteDB      Start-&gt;&gt;Gateway-API: get-itinerari-proposti     activate Gateway-API     Gateway-API-&gt;&gt;DB-Service: rpc-get-itinerari-proposti     activate DB-Service     DB-Service-&gt;&gt;RouteDB: get-itinerari-proposti     activate RouteDB     RouteDB--&gt;&gt;DB-Service: itinerari-proposti     deactivate RouteDB     DB-Service--&gt;&gt;Gateway-API: itinerari-proposti     deactivate DB-Service     Gateway-API--&gt;&gt;Start:      deactivate Gateway-API           </pre>

Nome	get-routes
Requisiti di riferimento	Mostrare percorsi creati.
Descrizione	Get dal database relazionale di tutti i percorsi presenti.
Input	
Output	<div>Descrizione dell'output</div> <div>Lista contenente le informazioni di tutti i percorsi.</div>
Diagrammi di sequenza ed interazioni tra componenti del sistema	 <pre> sequenceDiagram     participant Start     participant Gateway-API     participant DB-Service     participant RouteDB      Start-&gt;&gt;Gateway-API: get-routes     activate Gateway-API     Gateway-API-&gt;&gt;DB-Service: rpc-get-routes     activate DB-Service     DB-Service-&gt;&gt;RouteDB: get-routes     activate RouteDB     RouteDB--&gt;&gt;DB-Service: routes     deactivate RouteDB     DB-Service--&gt;&gt;Gateway-API: routes     deactivate DB-Service     Gateway-API--&gt;&gt;Start:      deactivate Gateway-API           </pre>

## Progettazione di dettaglio del sistema

### Dizionario e Modello dei Dati

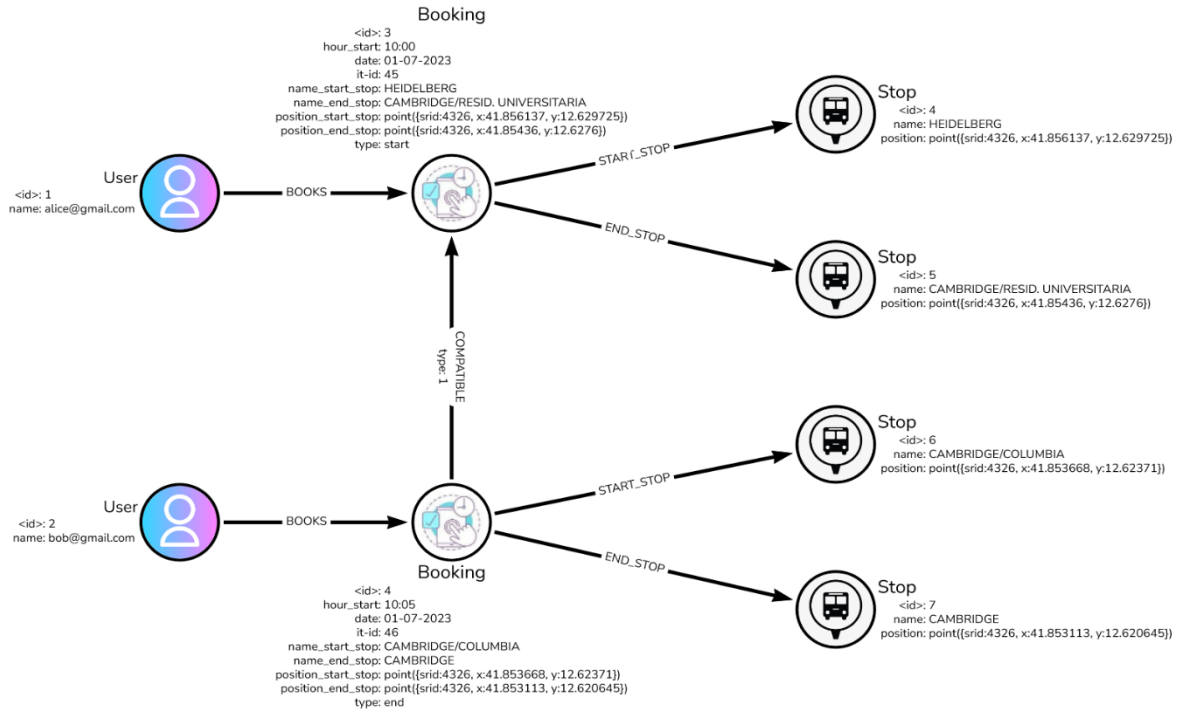
### Database Relazionale





Entità	Descrizione	Attributi	Identificatori
Percorso	Percorso generato dal sistema	Id, Scadenza, Archiviato, Stato, Timestamp	Id
Fermata	Fermate disponibili per la preotazione	Nome, Lat, Lon, Indirizzo	Lat, Lon
Itinerario richiesto	Itinerario richiesto dall'utente in fase di prenotazione	Id, ora_inizio, ora_fine, costo_max, distanza, utente, fermata_lat_partenza, fermata_lon_partenza, fermata_lat_arrivo, fermata_lon_arrivo	Id
Itinerario proposto	Itinerario proposto all'utente dopo la creazione del percorso dal sistema	Id, costo, distanza, orario_partenza_proposto, orario_arrivo_proposto, orario_partenza_effettivo, orario_arrivo_effettivo, stato, utente, itinerario_richiesto, fermata_partenza, fermata_arrivo, percorso	Id, percorso
Ordinamento	L'ordine in cui vengono percorse le fermate in un certo percorso	Numero, percorso, fermata	Percorso, fermata_lat, fermata_lon
Pagamento	Storico dei pagamenti di ogni utente quando decide di accettare un itinerario proposto	Tmstmp, file_fattura, percorso, utente, itinerario_proposto	Utente, itinerario_proposto, Percorso
Cashback	Storico del cashback che gli utenti ricevono quando un altro utente si aggiunge al loro percorso quando è già stato confermato	Saldo, tmstmp, utente	Utente, tmstmp
Utente	Utenti del sistema	Mail, pwd, nome, cognome, username, data_nascita	mail

## DB dei Bookings di Neo4J



Nodo	Descrizione	Attributi	Identificatori
Utente	Ogni nodo rappresenta l'utente che ha inserito un booking nel sistema.	<id>, name	<id>
Booking	Ogni nodo rappresenta l'itinerario che l'utente desidera effettuare, da una fermata ad un'altra in un determinato orario e giornata.	<id>, hour_start, hour_end, date, name_start_stop, name_end_stop, position_start_stop, position_end_stop, type	<id>
Stop	Ogni nodo rappresenta l'effettiva fermata geografica.	<id>, name, position	<id>

Relazioni	Descrizione	Attributi	Identificatori
BOOKS	Utente-Booking Relaziona gli utenti con i loro effettivi booking.	<id>	<id>
START_STOP	Booking-Stop Associa ad ogni booking la fermata di partenza.	<id>	<id>
END_STOP	Booking-Stop Associa ad ogni booking la fermata di arrivo.	<id>	<id>
COMPATIBLE	Booking-Booking La relazione collega quali sono le fermate che sia per motivi spaziali che temporali possono essere effettuate da un'unica corsa.	<id>, type	<id>

## Le motivazioni delle scelte tecnologiche

### Docker

Abbiamo scelto di utilizzare Docker come framework per l'esecuzione dei container della nostra applicazione a microservizi perché permette di creare immagini standardizzate che possono essere eseguite in qualsiasi ambiente che lo supporti e perché offre una gestione semplice dell'orchestrazione dei container sia localmente con Docker compose che in sviluppi futuri con Kubernetes.

### Python

Come linguaggio di programmazione abbiamo scelto Python per semplificare lo sviluppo, dato che è un linguaggio più ad alto livello e quindi questo ci ha permesso di concentrarci maggiormente sulle funzionalità piuttosto che su aspetti di basso livello. Inoltre, Python offre una libreria RPC semplice da utilizzare.

Abbiamo inoltre implementato il pattern Circuit Breaker in modo tale che se un microservizio dovesse fallire evitiamo un effetto a cascata, dovuto ad una richiesta RPC, che potrebbe portare all'interruzione dell'intero sistema

### MySQL

Per mantenere le informazioni di persistenza e lo storico, abbiamo optato per un database relazionale che modella meglio questo tipo di scenari dove c'è una buona complessità di relazioni tra le entità.

### Neo4j

Abbiamo valutato l'utilizzo del database a grafo Neo4j come soluzione naturale e ottimale per rappresentare l'insieme di *prenotazioni* destinate a essere selezionate come candidate per l'algoritmo di creazione del percorso. Neo4j offre strumenti avanzati per il calcolo rapido delle distanze spaziali e temporali tra le diverse fermate, rendendole facilmente *compatibili* per la creazione di un percorso con le necessarie relazioni. Grazie a questa funzionalità, possiamo facilmente raggruppare le *prenotazioni* più adatte per formare un percorso coerente.

### RabbitMQ

Per le code di messaggi abbiamo utilizzato RabbitMQ, implementando delle code persistenti che utilizzano un pattern di tipo push per il delivery dei messaggi che nel contesto della nostra applicazione risulta essere più adatto. Pertanto, per la comunicazione tra alcuni microservizi abbiamo usato code di messaggi per introdurre disaccoppiamento temporale e di sincronia.

### Docker compose

Per l'orchestrazione di molteplici container, abbiamo scelto Docker Compose perché permette un deployment più semplice su un singolo nodo durante la fase di sviluppo e supporta comunque la replicazione dei container per la scalabilità orizzontale. Come sviluppo futuro, prevediamo di integrare Kubernetes per poter orchestrare i microservizi su un'infrastruttura composta da più nodi distribuiti.

## OpenRouteService

Abbiamo utilizzato questo servizio open-source esterno per calcolare le matrici di distanza temporale e spaziale che poi servono per calcolare le compatibilità e per l'esecuzione dell'algoritmo euristico. Inoltre, permette di creare il percorso stradale tra la serie di fermate ordinate del percorso creato dall'algoritmo da poter inserire su mappa.

## I componenti (HW/SW) di cui è composto il servizio

Di seguito i microservizi che compongono l'applicazione:

### BookingService

Microservizio che si occupa di inserire in RouteDB e BookingDB le richieste degli utenti e di proporre ad essi i risultati dell'esecuzione dell'algoritmo per poi gestire l'accettazione o il rifiuto delle rotte proposte.

### NodeSelectorService

Microservizio che si occupa di prelevare dei cluster di *prenotazioni compatibili* dal database a grafo BookingDB e di reperire da OpenRouteService tutte le informazioni necessarie da passare all'algoritmo euristico e fornirgliel.

### MakeRouteService

Microservizio che si occupa di generare la *rotta* migliore possibile basandosi sulle informazioni richieste degli utenti, cercando comunque un compromesso accettabile da proporre agli utenti.

### RecommendService

Microservizio che si occupa di recuperare dal database i *recommended routes* ai quali gli utenti possono aggiungersi e gestire la capienza massima degli autobus nei vari tratti di percorrenza dei *recommended routes*.

### DBService

Microservizio che si occupa di gestire le query al database relazionale RouteDB.

### LoginService

Microservizio che si occupa di gestire il login e la registrazione degli utenti.

### PaymentService

Microservizio che si occupa di gestire i pagamenti e il cashback.

### NotifyService

Microservizio che si occupa di notificare gli utenti quando viene generato un *percorso* ed i relativi *itinerari proposti*.

### OrsDao

Microservizio che gestisce l'interazione con il container di OpenRouteService per le chiamate alle sue API.

## OrsApp

Il container che offre le API di OpenRouteService, abbiamo deciso di utilizzare la versione containerizzata perché non c'erano limiti sul numero e tipo di richieste.

## GatewayAPI

Microservizio che si occupa di esportare verso l'esterno le API REST.

## Frontend

Container che espone l'interfaccia utente dell'applicazione.

## BookingDB

Database relazionale SQL.

## RouteDB

Database a grafo Neo4j.

## Connettori dei microservizi

### RPC

Per connettere i microservizi che necessitano di una comunicazione sincrona abbiamo utilizzato la libreria "xmlrpc" inclusa di default in Python per semplicità.

### RabbitMQ

Per connettere i microservizi che necessitano un disaccoppiamento maggiore per necessità di scalabilità orizzontale abbiamo inserito delle code di messaggi RabbitMQ gestite utilizzando la libreria Pika di Python.

### HTTP

Per connettere il Frontend con il GatewayAPI e per OpenRouteService abbiamo utilizzato il protocollo HTTP richiamando le API REST esportate.

## Algoritmo Euristico per Calcolare il Percorso

Per calcolare il percorso a partire dai nodi abbiamo deciso di utilizzare un algoritmo euristico e non un risolutore di problemi di Programmazione Matematica perchè nel contesto del nostro servizio non abbiamo necessariamente bisogno di una soluzione ottima, ma è sufficiente anche una buona soluzione che sia vicina all'ottimo. Per questo motivo abbiamo optato per l'euristica dando vantaggio alle prestazioni, pensando a situazioni in cui il servizio sia soggetto ad un elevato carico di richieste. Inoltre, il problema che risolviamo è una variante del Traveling Salesman Problem (TSP), che è di tipo NP hard, e pertanto non esiste una soluzione esatta ottenibile in modo efficiente. Inoltre, i risolutori di Programmazione Matematica generalmente non sono open-source e richiedono un format particolare per i dati, che avrebbe richiesto un tempo non trascurabile per la conversione dell'input.

Il problema che tenta di risolvere l'algoritmo, come detto precedentemente, è una variante del TSP, in italiano noto come il problema del commesso viaggiatore. È una variante perché ci sono dei vincoli leggermente diversi ed alcuni in più:

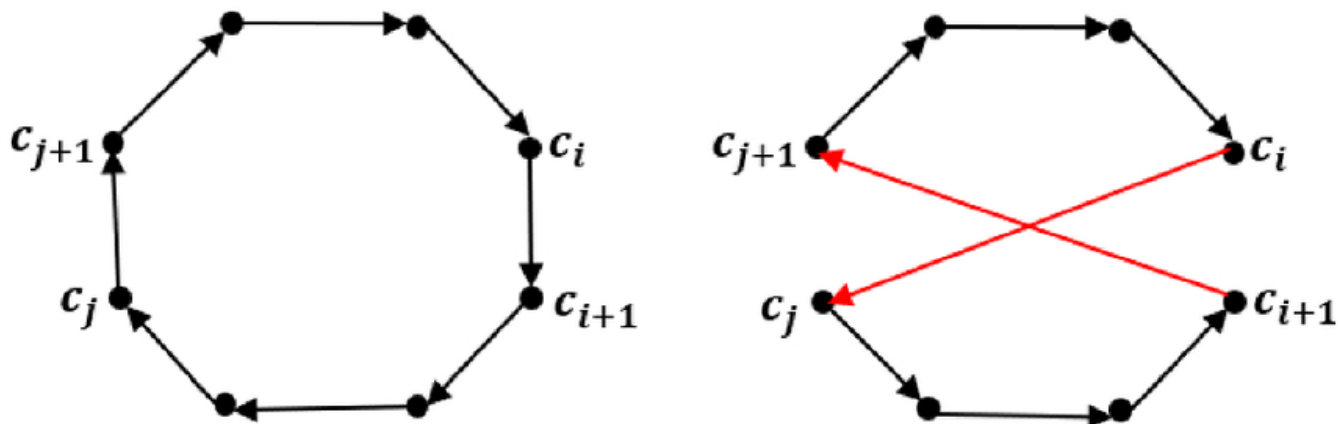
- Nel nostro caso il percorso non ha nodi di inizio e fine prefissati, e quindi potenzialmente il percorso può partire e finire da un qualsiasi nodo preso in considerazione.
- Ci sono dei vincoli di precedenza tra alcuni nodi (l'arrivo richiesto da un utente non può essere prima della partenza).
- Ci sono delle finestre temporali da rispettare o comunque tali per cui la cui variazione da esse va minimizzata.
- Gli autobus non possono fermarsi ad aspettare in una fermata per evitare di intasare il traffico; quindi, la soluzione non può prevedere la sosta momentanea nei nodi del percorso per tentare di soddisfare meglio i vincoli temporali.

L'algoritmo che abbiamo sviluppato è una variante dell'algoritmo 2-OPT (2-OPT, s.d.) usato per risolvere problemi di TSP con soluzioni sub-ottimali.

L'algoritmo 2-OPT è un algoritmo euristico utilizzato per risolvere problemi di ottimizzazione combinatoria, in particolare il TSP. Esso cerca di migliorare una soluzione esistente del TSP attraverso una serie di operazioni di scambio che rimuovono le intersezioni nel percorso.

L'algoritmo 2-OPT funziona nel seguente modo:

1. Inizializzazione: Si parte da una soluzione iniziale del TSP.
2. Scambio: Si selezionano due archi non consecutivi nel percorso e si scambiano i loro estremi, creando una nuova soluzione.
3. Valutazione: Si valuta la nuova soluzione e si confronta con la soluzione precedente. Se la nuova soluzione è migliore, si accetta, altrimenti si scarta.
4. Ripetizione: Si ripete il processo di scambio e valutazione fino a quando non si trova una soluzione molto vicina all'ottimo globale.

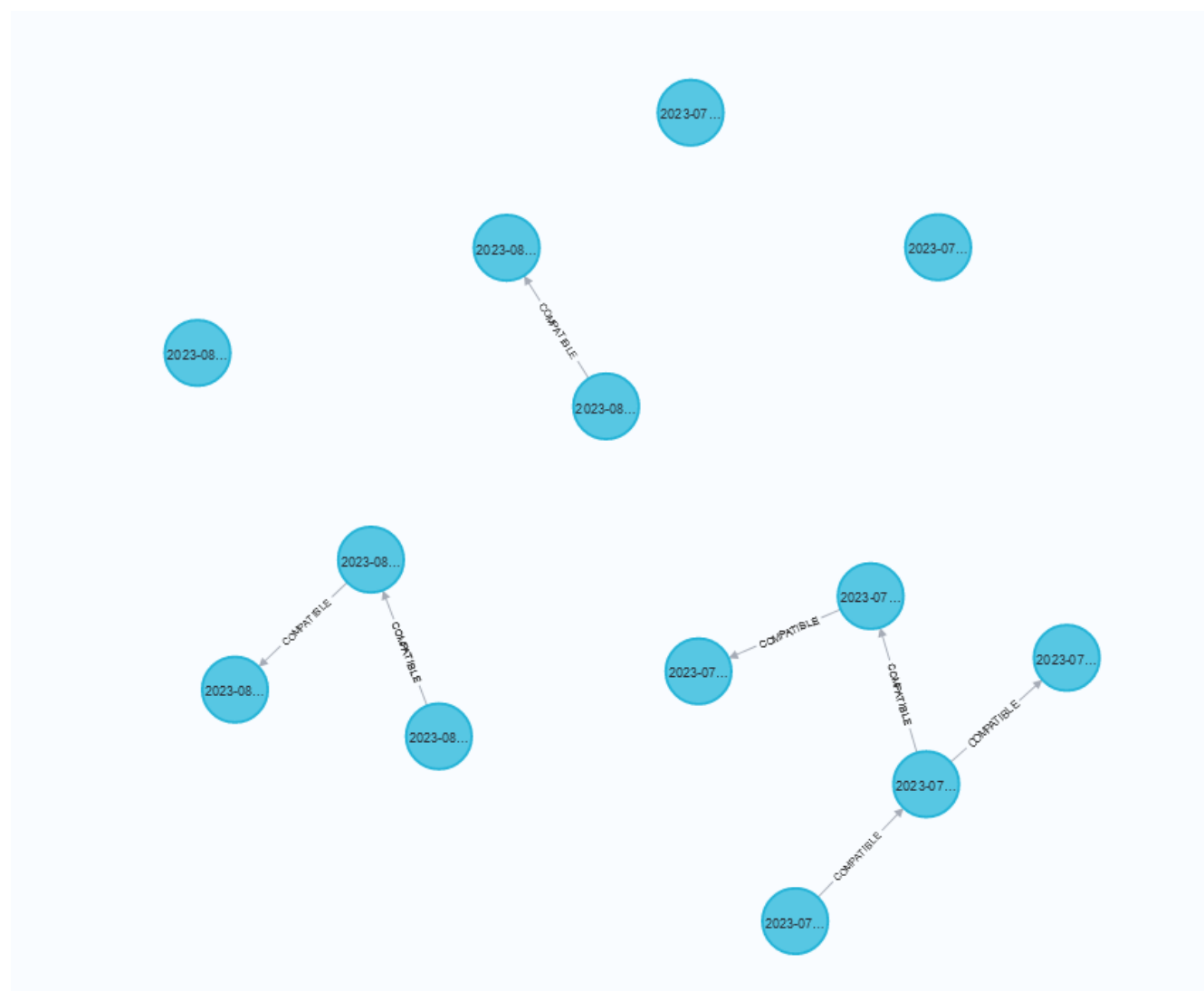


Nella nostra variante ci sono delle differenze nelle fasi:

1. Inizializzazione: Per generare una soluzione ammissibile da cui partire usiamo un algoritmo di ordinamento topologico Kahn, 1962)
2. Scambio: Si selezionano due archi non consecutivi nel percorso e si scambiano i loro estremi, creando una nuova soluzione. Si vanno a scambiare solo gli archi che non creano problemi nella precedenza dei nodi. Al contrario del 2-OPT classico si scambiano anche il primo nodo e l'ultimo.
3. Valutazione: In questa fase per valutare se la nuova soluzione è migliore usiamo un altro algoritmo che si occupa di compattare la soluzione in base agli orari richiesti dagli utenti per cercare di minimizzare vari parametri, quali il tempo di viaggio, il numero di utenti in ritardo, il ritardo medio e il ritardo massimo.
4. Ripetizione: Si ripete il processo di scambio e valutazione fino a quando non si trova una soluzione molto vicina all'ottimo globale.

## Clustering delle Prenotazioni

Il clustering delle prenotazioni ha il ruolo di preparare i nodi che insieme saranno più adatti a generare un buon percorso una volta passati all'algoritmo. I nodi formeranno dei cluster attraverso degli archi di collegamento all'interno del database a grafo. Questi archi rappresentano la *compatibilità* tra due nodi che verrà definita attraverso la presenza contemporanea di due tipi di *compatibilità*: *spaziale e temporale*.





## Compatibilità spaziale

Due nodi Booking sono *compatibili spazialmente* se viene rispettata una delle tre condizioni di “vicinanza” tra le fermate. Due fermate sono “vicine” se la funzione “point.distance(a.position,b.position)” restituisce un risultato inferiore di 2,5 Km. La funzione “point.distance(a.position,b.position)” è presente nelle librerie di Neo4J e serve a calcolare la distanza geospaziale tra due punti utilizzando la formula di Haversine, dove le position sono le coordinate delle fermate in formato WGS-84 CRS (2D).

Le condizioni di vicinanza sono:

1. Se il nodo 1 ha la fermata di arrivo “vicina” con quella di partenza del nodo 2, allora verrà tracciato un collegamento dal nodo 1 al nodo 2.
2. Se il nodo 1 ha la fermata di partenza “vicina” con quella di arrivo del nodo 2, allora verrà tracciato un collegamento dal nodo 2 al nodo 1 (in pratica stessa casistica di prima ma con direzione opposta).
3. Se il nodo 1 e il nodo 2 hanno le partenze “vicine”, allora verrà tracciata una freccia da uno dei due nodi verso l'altro.

## Compatibilità temporale

Una condizione preliminare affinché due nodi Booking siano *compatibili temporalmente* è che i nodi afferiscano allo stesso giorno.

$i, j$  = Nodi Booking del grafo

$D_{Si,Ej}$  = distanza temporale dalla fermata di partenza del nodo  $i$  alla fermata di arrivo del nodo  $j$

$Si$  = orario di partenza del nodo  $i$

$Ei$  = orario di arrivo del nodo  $i$

$Off1(D_{Si,Ej}) = \max(5, 0.15 \cdot D_{Si,Ej})$

$Off2(D_{Si,Ej}) = \max(10, 0.3 \cdot D_{Si,Ej})$

Ci sono tre casi saranno dipendenti dalla compatibilità spaziale:

1. Se hanno una compatibilità spaziale di tipo 1 e  $Ei < Sj$ :  $i$  e  $j$  saranno compatibili temporalmente se:  

$$D_{Ei,Sj} - Off1(D_{Ei,Sj}) < |Ei - Sj| < D_{Ei,Sj} + Off2(D_{Ei,Sj})$$
2. Se hanno una compatibilità spaziale di tipo 2 e  $Si > Ej$ :  $i$  e  $j$  saranno compatibili temporalmente se:  

$$D_{Si,Ej} - Off1(D_{Si,Ej}) < |Si - Ej| < D_{Si,Ej} + Off2(D_{Si,Ej})$$
3. Se hanno una compatibilità spaziale di tipo 3:  $i$  e  $j$  saranno compatibili temporalmente se:  

$$D_{Si,Sj} - Off1(D_{Si,Sj}) < |Si - Sj| < D_{Si,Sj} + Off2(D_{Si,Sj})$$

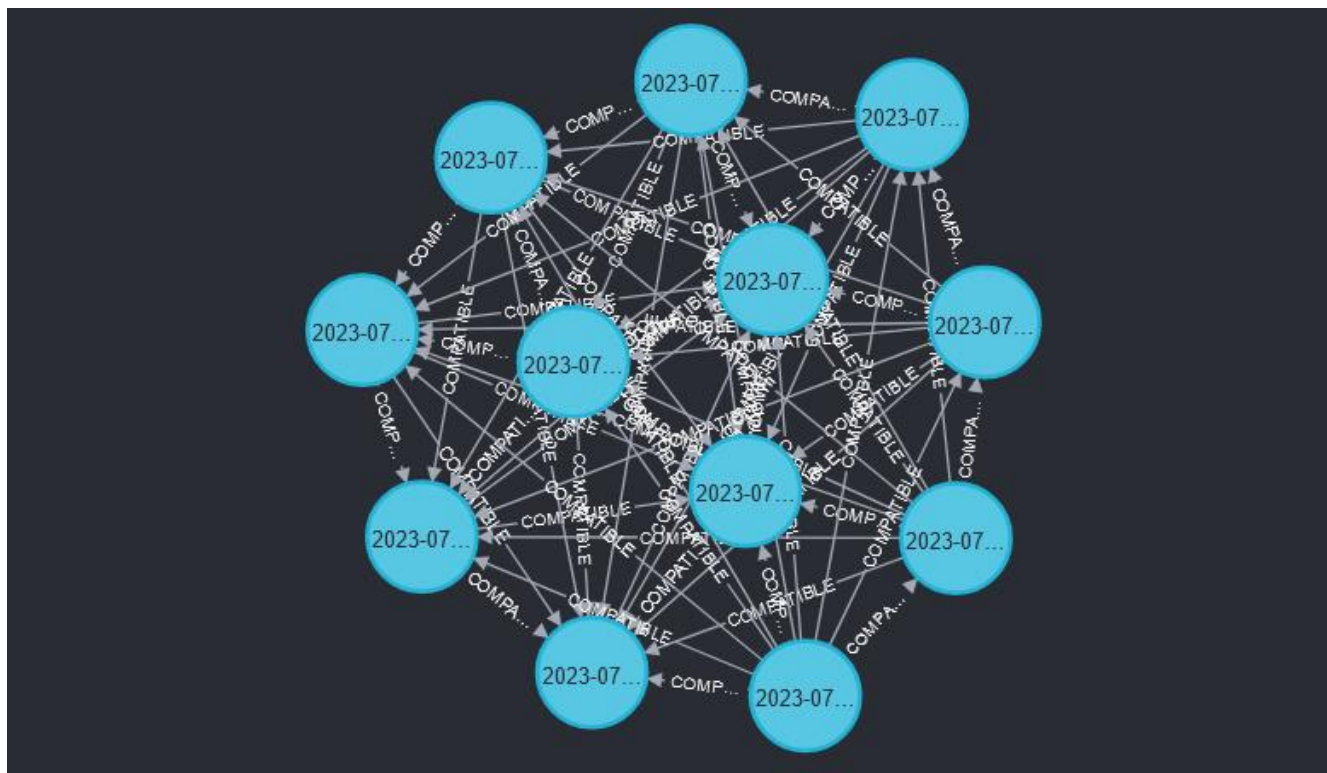
## Costo del Biglietto

Il costo del biglietto viene calcolato in base a vari fattori e ipotesi:

- Costo fisso di partenza di 0,30€ per i costi di gestione
- 8 lt/100km
- Prezzo del carburante 1,85€
- 35% supplemento dovuto al fatto che il percorso viene confermato se almeno il 75% delle persone lo accetta
- 50% di ulteriore supplemento dovuto al profitto e ad altri costi di gestione

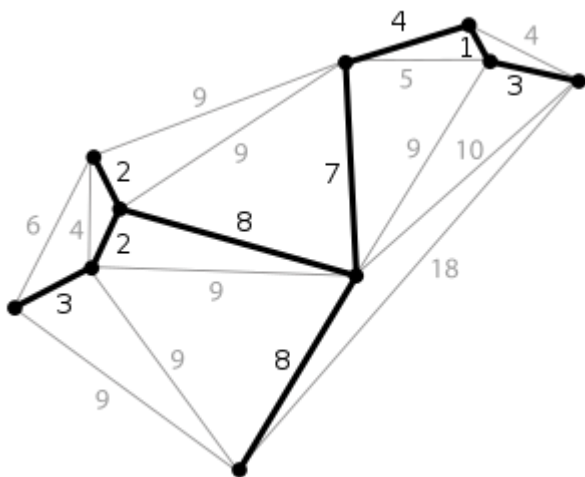
## Dettaglio sul prelevamento da un Cluster

Per ottenere un insieme di prenotazioni compatibili da Neo4j, che possa essere utilizzato come input per il calcolo del percorso, è necessario considerare la gestione di un grafo di prenotazioni come il seguente:



Supponiamo di dover prelevare un sottogruppo di nodi, avendo un numero limitato di posti disponibili sull'autobus. La visita in profondità, specialmente in situazioni di grafo completo, può risultare inefficiente, poiché per garantire di prelevare un certo numero di nodi è necessario fare altrettanti hop a partire da un nodo iniziale e questo, già per un numero di hop superiori a sette, fa sì che una visita in profondità richieda tempi non accettabili.

Per ovviare a questo problema, è stata scelta la funzione “apoc.path.spanningTree” della libreria “apoc” di Neo4j. Questa funzione permette di creare un albero di copertura (spanning tree) a partire dal grafo e da un nodo di partenza.



In questo modo, percorrendo i path dello spanning tree, si possono prelevare i nodi necessari per riempire l'autobus con un insieme di prenotazioni compatibili, ottimizzando il processo di selezione delle prenotazioni.

## Codifica RESTful delle interfacce dei servizi

Nome	get-token											
Descrizione	Generazione di un token per poter utilizzare le altre operazioni dell'API											
Resource URI	api/get_token											
Tipo di richiesta http	GET											
Content type della richiesta												
Content type della risposta	application/json											
Request body												
Path parameters												
Query parameters												
Response body	<table><tr><th>Codice risposta HTTP</th><th>Tipo/struttura</th><th>Descrizione</th></tr><tr><td>200 OK</td><td>json</td><td>{ "token": token-string }</td></tr><tr><td>400 ERROR</td><td>json</td><td>{"status": "ERROR"}</td></tr></table>			Codice risposta HTTP	Tipo/struttura	Descrizione	200 OK	json	{ "token": token-string }	400 ERROR	json	{"status": "ERROR"}
Codice risposta HTTP	Tipo/struttura	Descrizione										
200 OK	json	{ "token": token-string }										
400 ERROR	json	{"status": "ERROR"}										

Nome	get-bus-stops		
Descrizione	Get della lista di tutte le fermate gestite dal servizio.		
Resource URI	api/get_bus_stops		
Tipo di richiesta http	GET		
Content type della richiesta			
Content type della risposta	application/json		
Request body			
Path parameters			
Query parameters			
Response body			
	Codice risposta HTTP	Tipo/struttura	Descrizione
	200 OK	json array	[{ "lang": longitudine-float "lat": latitudine-float, "name": nome fermata-string }, ...]
	400 ERROR	json	{"status":"error"}

Nome	get-bus-stops-in-rect											
Descrizione	Get della lista di tutte le fermate le cui coordinate si trovano all'interno di un rettangolo di coordinate.											
Resource URI	api/get_bus_stops											
Tipo di richiesta http	GET											
Content type della richiesta												
Content type della risposta	application/json											
Request body												
Path parameters												
Query parameters												
Response body	<table><tr><th>Codice risposta HTTP</th><th>Tipo/struttura</th><th>Descrizione</th></tr><tr><td>200 OK</td><td></td><td>[{ "lang": longitudine-float, "lat": latitudine-float "name": nome fermata-string }, ...]</td></tr><tr><td>400 ERROR</td><td>json</td><td>{"status":"error"}</td></tr></table>			Codice risposta HTTP	Tipo/struttura	Descrizione	200 OK		[{ "lang": longitudine-float, "lat": latitudine-float "name": nome fermata-string }, ...]	400 ERROR	json	{"status":"error"}
Codice risposta HTTP	Tipo/struttura	Descrizione										
200 OK		[{ "lang": longitudine-float, "lat": latitudine-float "name": nome fermata-string }, ...]										
400 ERROR	json	{"status":"error"}										

Nome	route-from-map		
Descrizione	Richiesta di un itinerario richiesto, per avviare la procedura di prenotazione.		
Resource URI	/api/route_from_map		
Tipo di richiesta http	GET		
Content type della richiesta			
Content type della risposta	application/json		
Request body			
Path parameters			
	Parametro	Tipo	Descrizione
Query parameters			
	Parametro	Tipo	Descrizione
	user	String	Token o mail dell’utente.
	starting_point	String	Nome della fermata di partenza.
	start_lat	Float	Latitudine della fermata di partenza.
	start_lng	Float	Longitudine della fermata di partenza.
	ending_point	String	Nome della fermata di arrivo.
	end_lat	Float	Latitudine della fermata di arrivo.
	end_lng	Float	Longitudine della fermata di arrivo.
	start-finish	String	Stringa che indica se la data e l’orario inseriti sono quelli in cui si vuole partire o arrivare, può assumere due valori “start” o “finish”.

	date	String YYYY-MM-DD	Data di partenza o arrivo.
	time	String HH:MM:SS	Orario di partenza o arrivo.
Response body			
	<b>Codice risposta HTTP</b>	<b>Tipo/struttura</b>	<b>Descrizione</b>
	200 OK	json	{ "it_req_id": id_it_richiesto bigint }
	400 ERROR	json	{"status": "error"}



Nome	get-recommended-routes		
Descrizione	Get di tutti i recommended routes.		
Resource URI	/api/get_recommended_routes		
Tipo di richiesta http	GET		
Content type della richiesta			
Content type della risposta	application/json		
Request body			
Path parameters			
Query parameters			
Response body			
	<b>Codice risposta HTTP</b>	<b>Tipo/struttura</b>	<b>Descrizione</b>
	200 OK	json array	<pre>[{   "route": route id,   "segments": [{     "end": [latitudine fermata arrivo, longitudine fermata arrivo],     "seat_available": #posti,     "start": [latitudine fermata partenza, longitudine fermata partenza]   }, ...],   "steps": [{     "climb_up": #persone che salgono,     "climb_down": #persone che scendono,     "name": nome fermata,</pre>

			"position": [latitudine fermata, longitudine fermata], "step": numero dello step, "timestamp": orario in cui si passa per la fermata }, ...] }, ...]
	400 ERROR	json	{"status":"error"}

Nome	get-km-price-from-subroute											
Descrizione	Get del costo e della distanza percorsa tra due fermate A e B appartenenti ad un recommended route.											
Resource URI	/api/ get_km_price_from_subroute											
Tipo di richiesta http	POST											
Content type della richiesta												
Content type della risposta	application/json											
Request body	<table><tr><th>Parametro</th><th>Tipo/struttura</th><th>Descrizione</th></tr><tr><td>bus_stops</td><td>Array of arrays</td><td>[[latitudine fermata A, longitudine fermata A], ..., [latitudine fermata B, longitudine fermata B],]</td></tr><tr><td>route</td><td>Bigint</td><td>Id del recommended route.</td></tr></table>			Parametro	Tipo/struttura	Descrizione	bus_stops	Array of arrays	[[latitudine fermata A, longitudine fermata A], ..., [latitudine fermata B, longitudine fermata B],]	route	Bigint	Id del recommended route.
Parametro	Tipo/struttura	Descrizione										
bus_stops	Array of arrays	[[latitudine fermata A, longitudine fermata A], ..., [latitudine fermata B, longitudine fermata B],]										
route	Bigint	Id del recommended route.										
Path parameters												
Query parameters												
Response body	<table><tr><th>Codice risposta HTTP</th><th>Tipo/struttura</th><th>Descrizione</th></tr><tr><td>200 OK</td><td>json</td><td>{ "price": costo per arrivare da A a B, "distance": distanza percorsa per arrivare da }</td></tr><tr><td>400 ERROR</td><td>json</td><td>{"status":"error"}</td></tr></table>			Codice risposta HTTP	Tipo/struttura	Descrizione	200 OK	json	{ "price": costo per arrivare da A a B, "distance": distanza percorsa per arrivare da }	400 ERROR	json	{"status":"error"}
Codice risposta HTTP	Tipo/struttura	Descrizione										
200 OK	json	{ "price": costo per arrivare da A a B, "distance": distanza percorsa per arrivare da }										
400 ERROR	json	{"status":"error"}										

Nome	join-recommended-route																																
Descrizione	Aggiungere un itinerario proposto ad un recommended route, l’itinerario proposto che viene aggiunto deve avere le fermate di partenza e arrivo che coincidono con fermate già presenti nel percorso.																																
Resource URI	/api/join_recommended_route																																
Tipo di richiesta http	GET																																
Content type della richiesta																																	
Content type della risposta	application/json																																
Request body																																	
Path parameters																																	
Query parameters	<table><tr><th>Parametro</th><th>Tipo</th><th>Descrizione</th></tr><tr><td>route_id</td><td>Bigint</td><td>Id del recommended route</td></tr><tr><td>start_lat</td><td>Float</td><td>Latitudine della fermata di partenza</td></tr><tr><td>start_lng</td><td>Float</td><td>Longitudine della fermata di partenza</td></tr><tr><td>end_lat</td><td>Float</td><td>Latitudine della fermata di arrivo</td></tr><tr><td>end_lng</td><td>Float</td><td>Longitudine della fermata di arrivo</td></tr><tr><td>start_date</td><td>String YYYY-MM-DD HH:MM:SS</td><td>Data di partenza</td></tr><tr><td>end_date</td><td>String YYYY-MM-DD HH:MM:SS</td><td>Data di arrivo</td></tr><tr><td>price</td><td>Float</td><td>Costo dell’itinerario proposto</td></tr><tr><td>distance</td><td>Float</td><td>Distanza percorsa dall’itinerario proposto</td></tr></table>			Parametro	Tipo	Descrizione	route_id	Bigint	Id del recommended route	start_lat	Float	Latitudine della fermata di partenza	start_lng	Float	Longitudine della fermata di partenza	end_lat	Float	Latitudine della fermata di arrivo	end_lng	Float	Longitudine della fermata di arrivo	start_date	String YYYY-MM-DD HH:MM:SS	Data di partenza	end_date	String YYYY-MM-DD HH:MM:SS	Data di arrivo	price	Float	Costo dell’itinerario proposto	distance	Float	Distanza percorsa dall’itinerario proposto
Parametro	Tipo	Descrizione																															
route_id	Bigint	Id del recommended route																															
start_lat	Float	Latitudine della fermata di partenza																															
start_lng	Float	Longitudine della fermata di partenza																															
end_lat	Float	Latitudine della fermata di arrivo																															
end_lng	Float	Longitudine della fermata di arrivo																															
start_date	String YYYY-MM-DD HH:MM:SS	Data di partenza																															
end_date	String YYYY-MM-DD HH:MM:SS	Data di arrivo																															
price	Float	Costo dell’itinerario proposto																															
distance	Float	Distanza percorsa dall’itinerario proposto																															

	user	String	Token o mail dell'utente
Response body			
	Codice risposta HTTP	Tipo/struttura	Descrizione
	200 OK		{"status": "Ok"}
	400 ERROR	json	{"status": "error"}

Nome	get-random-cluster-from-it-id											
Descrizione	Get dal database a grafo del cluster di itinerari richiesti compatibili con quello specificato.											
Resource URI	/api/get-random-cluster-from-it-id											
Tipo di richiesta http	GET											
Content type della richiesta												
Content type della risposta	application/json											
Request body												
Path parameters												
Query parameters	<table><tr><th>Parametro</th><th>Tipo</th><th>Descrizione</th></tr><tr><td>it_id</td><td>Bigint</td><td>Id dell’itinerario richiesto di cui si vuole vedere il cluster di appartenenza.</td></tr><tr><td>limit</td><td>Int</td><td>Dimensione massima del cluster ottenuto.</td></tr></table>			Parametro	Tipo	Descrizione	it_id	Bigint	Id dell’itinerario richiesto di cui si vuole vedere il cluster di appartenenza.	limit	Int	Dimensione massima del cluster ottenuto.
Parametro	Tipo	Descrizione										
it_id	Bigint	Id dell’itinerario richiesto di cui si vuole vedere il cluster di appartenenza.										
limit	Int	Dimensione massima del cluster ottenuto.										
Response body	<table><tr><th>Codice risposta HTTP</th><th>Tipo/struttura</th><th>Descrizione</th></tr><tr><td>200 OK</td><td>json array</td><td>[{ "id": id nodo neo4j, "date": data di partenza o arrivo dell’itinerario richiesto, "name_start_stop": nome della fermata di partenza, "name_end_stop":nome della fermata di arrivo, "position_start_stop":lista delle</td></tr></table>			Codice risposta HTTP	Tipo/struttura	Descrizione	200 OK	json array	[{ "id": id nodo neo4j, "date": data di partenza o arrivo dell’itinerario richiesto, "name_start_stop": nome della fermata di partenza, "name_end_stop":nome della fermata di arrivo, "position_start_stop":lista delle			
Codice risposta HTTP	Tipo/struttura	Descrizione										
200 OK	json array	[{ "id": id nodo neo4j, "date": data di partenza o arrivo dell’itinerario richiesto, "name_start_stop": nome della fermata di partenza, "name_end_stop":nome della fermata di arrivo, "position_start_stop":lista delle										

			coordinate della fermata di partenza, " position end_stop ": lista delle coordinate della fermata di arrivo, "type": tipo dell'itinerario richiesto (start o finish), "user":token o mail dell'utente, "it_id": id dell'itinerario richiesto, "hour": orario di partenza o arrivo dell'itinerario richiesto }, ...]
	400 ERROR	json	{"status":"ERROR"}

Nome	get-random-cluster								
Descrizione	Get dal database a grafo di un cluster casuale di itinerari richiesti compatibili.								
Resource URI	/api/get_random_cluster								
Tipo di richiesta http	GET								
Content type della richiesta									
Content type della risposta	application/json								
Request body									
Path parameters									
Query parameters									
Response body	<table><tr><th>Codice risposta HTTP</th><th>Tipo/struttura</th><th>Descrizione</th></tr><tr><td>200 OK</td><td>json array</td><td>[{ "id": id nodo neo4j, "date": data di partenza o arrivo dell'itinerario richiesto, "name_start_stop": nome della fermata di partenza, "name_end_stop":nome della fermata di arrivo, "position start_stop":lista delle coordinate della fermata di partenza, " position end_stop ": lista delle coordinate della fermata di arrivo, "type": tipo dell'itinerario</td></tr></table>			Codice risposta HTTP	Tipo/struttura	Descrizione	200 OK	json array	[{ "id": id nodo neo4j, "date": data di partenza o arrivo dell'itinerario richiesto, "name_start_stop": nome della fermata di partenza, "name_end_stop":nome della fermata di arrivo, "position start_stop":lista delle coordinate della fermata di partenza, " position end_stop ": lista delle coordinate della fermata di arrivo, "type": tipo dell'itinerario
Codice risposta HTTP	Tipo/struttura	Descrizione							
200 OK	json array	[{ "id": id nodo neo4j, "date": data di partenza o arrivo dell'itinerario richiesto, "name_start_stop": nome della fermata di partenza, "name_end_stop":nome della fermata di arrivo, "position start_stop":lista delle coordinate della fermata di partenza, " position end_stop ": lista delle coordinate della fermata di arrivo, "type": tipo dell'itinerario							



			richiesto (start o finish), "user":token o mail dell'utente, "it_id": id dell'itinerario richiesto, "hour": orario di partenza o arrivo dell'itinerario richiesto }, ...]
	400 ERROR	json	{"status":"ERROR"}

Nome	get-all-clusters		
Descrizione	Get di tutti i cluster di itinerari compatibili presenti su database a grafo.		
Resource URI	/get_all_clusters		
Tipo di richiesta http	GET		
Content type della richiesta			
Content type della risposta	application/json		
Request body			
Path parameters			
Query parameters			
Response body			
	<b>Codice risposta HTTP</b>	<b>Tipo/struttura</b>	<b>Descrizione</b>
	200 OK	json	<pre>{   "clusters": [     {       "id": id nodo neo4j,       "date": data di partenza o arrivo dell'itinerario richiesto,       "name_start_stop": nome della fermata di partenza,       "name_end_stop": nome della fermata di arrivo,       "position_start_stop": lista delle coordinate della fermata di partenza,</pre>

			" position end_stop ": lista delle coordinate della fermata di arrivo, "type": tipo dell'itinerario richiesto (start o finish), "user":token o mail dell'utente, "it_id": id dell'itinerario richiesto, "hour": orario di partenza o arrivo dell'itinerario richiesto }, ...] , ...] }
	400 ERROR	json	{"status":"ERROR"}

Nome	make-route-raw		
Descrizione	Esecuzione dell'algoritmo di creazione del percorso		
Resource URI	/api/make-route-raw		
Tipo di richiesta http	POST		
Content type della richiesta	application/json		
Content type della risposta	application/json		
Request body			
	Parametro	Tipo/struttura	Descrizione
	Matrice delle distanze	json	"dist_matrix": [[0.0, 4.07], [4.27, 0.0]]
	Hash delle precedenze	json	{"prec_hash": { "0": [nodo con vincolo di precedenza, nodo con vincolo di precedenza , nodo con vincolo di precedenza , ecc..], "1": [] }}
	Limiti dei nodi	json	{"node_limit": { "0": [limite inferiore, limite superiore], }, ... }
Itinerari	json	"user_routes": [{"user": mail utente "it_id": id dell'itinerario richiesto,	

			<pre>"date": data dell'itinerario, "nodes": [nodo di partenza, nodo di arrivo]}, , ... ] }</pre>						
Path parameters									
Query parameters									
Response body	<table><tr><th>Codice risposta HTTP</th><th>Tipo/struttura</th><th>Descrizione</th></tr><tr><td>200 OK</td><td>json</td><td><pre>{"steps": [   {     "id": "0",     "date": Data di partenza,     "time": Orario di partenza,     "location": [       Latitudine,       Longitudine     ]   },   {     "id": "1",     "date": "2023-07-28",     "time": "12:04:00",     "location": [       12.630395,       41.859081     ]   } ],</pre></td></tr></table>			Codice risposta HTTP	Tipo/struttura	Descrizione	200 OK	json	<pre>{"steps": [   {     "id": "0",     "date": Data di partenza,     "time": Orario di partenza,     "location": [       Latitudine,       Longitudine     ]   },   {     "id": "1",     "date": "2023-07-28",     "time": "12:04:00",     "location": [       12.630395,       41.859081     ]   } ],</pre>
Codice risposta HTTP	Tipo/struttura	Descrizione							
200 OK	json	<pre>{"steps": [   {     "id": "0",     "date": Data di partenza,     "time": Orario di partenza,     "location": [       Latitudine,       Longitudine     ]   },   {     "id": "1",     "date": "2023-07-28",     "time": "12:04:00",     "location": [       12.630395,       41.859081     ]   } ],</pre>							

			<div> "travel_time": tempo di viaggio,  "n_tardy": numero di job in ritardo,  "mean_unacceptable_deviance": deviazione inaccetttabile media,  "users_travel_time": {  "7": tempo tragitto,  "5": tempo tragitto,  "6": tempo tragitto  },  "user_routes": [{  "user": mail dell'utente,  "it_id": id dell'itinerario proposto,  "date": data,  "nodes": [  "0",  "1"  ] },  ... ] } </div>
	400 ERROR	json	{ "status": "ERROR" }

Nome	make-route														
Descrizione	Esecuzione dell'algoritmo a partire dai nodi in coordinate														
Resource URI	/api/make-route														
Tipo di richiesta http	POST														
Content type della richiesta	application/json														
Content type della risposta	application/json														
Request body	<table><tr><th>Parametro</th><th>Tipo/struttura</th><th>Descrizione</th></tr><tr><td>Coordinate dei nodi</td><td>json</td><td><pre>{   "points_location": {     "0": [latitudine,           longitudine],     "1": [latitudine, longitudine],     ...   } }</pre></td></tr><tr><td>Hash delle precedenze</td><td>json</td><td><pre>{ "prec_hash": {   "0": [nodo con vincolo di precedenza, nodo con vincolo di precedenza , nodo con vincolo di precedenza , ecc..],   "1": [] }} </pre></td></tr><tr><td>Limiti dei nodi</td><td>json</td><td><pre>{ "node_limit": {   "0": [limite inferiore, limite superiore], }, ... }</pre></td></tr></table>			Parametro	Tipo/struttura	Descrizione	Coordinate dei nodi	json	<pre>{   "points_location": {     "0": [latitudine,           longitudine],     "1": [latitudine, longitudine],     ...   } }</pre>	Hash delle precedenze	json	<pre>{ "prec_hash": {   "0": [nodo con vincolo di precedenza, nodo con vincolo di precedenza , nodo con vincolo di precedenza , ecc..],   "1": [] }} </pre>	Limiti dei nodi	json	<pre>{ "node_limit": {   "0": [limite inferiore, limite superiore], }, ... }</pre>
Parametro	Tipo/struttura	Descrizione													
Coordinate dei nodi	json	<pre>{   "points_location": {     "0": [latitudine,           longitudine],     "1": [latitudine, longitudine],     ...   } }</pre>													
Hash delle precedenze	json	<pre>{ "prec_hash": {   "0": [nodo con vincolo di precedenza, nodo con vincolo di precedenza , nodo con vincolo di precedenza , ecc..],   "1": [] }} </pre>													
Limiti dei nodi	json	<pre>{ "node_limit": {   "0": [limite inferiore, limite superiore], }, ... }</pre>													

			}						
	Itinerari	json	"user_routes": [{"user": mail utente "it_id": id dell'itinerario richiesto, "date": data dell'itinerario, "nodes": [nodo di partenza, nodo di arrivo]},  , ... ] }						
Path parameters									
Query parameters									
Response body	<table><tr><th>Codice risposta HTTP</th><th>Tipo/struttura</th><th>Descrizione</th></tr><tr><td>200 OK</td><td>json</td><td>{"steps": [     {         "id": "0",         "date": Data di partenza,         "time": Orario di partenza,         "location": [             Latitudine,             Longitudine         ]     },     {         "id": "1",         "date": "2023-07-28",         "time": "12:04:00",</td></tr></table>			Codice risposta HTTP	Tipo/struttura	Descrizione	200 OK	json	{"steps": [ { "id": "0", "date": Data di partenza, "time": Orario di partenza, "location": [ Latitudine, Longitudine ] }, { "id": "1", "date": "2023-07-28", "time": "12:04:00",
Codice risposta HTTP	Tipo/struttura	Descrizione							
200 OK	json	{"steps": [ { "id": "0", "date": Data di partenza, "time": Orario di partenza, "location": [ Latitudine, Longitudine ] }, { "id": "1", "date": "2023-07-28", "time": "12:04:00",							



			<pre>       "location": [         12.630395,         41.859081       ]     }   ],   "travel_time": tempo di viaggio,   "n_tardy": numero di job in ritardo,  "mean_unacceptable_deviance" : deviazione inaccetttabile media,   "users_travel_time": {     "7": tempo tragitto,     "5": tempo tragitto,     "6": tempo tragitto   },   "user_routes": [{     "user": mail dell'utente,     "it_id": id dell'itinerario proposto,     "date": data,     "nodes": [       "0",       "1"     ]   }],   ...}] </pre>
	400 ERROR	json	{"status": "ERROR"}

Nome	get-itinerari-richiesti								
Descrizione	Get dal database relazionale di tutti gli itinerari richiesti presenti.								
Resource URI	/api/get_itinerari_richiesti								
Tipo di richiesta http	GET								
Content type della richiesta									
Content type della risposta	application/json								
Request body									
Path parameters									
Query parameters									
Response body	<table><tr><th>Codice risposta HTTP</th><th>Tipo/struttura</th><th>Descrizione</th></tr><tr><td>200 OK</td><td>json array</td><td>[{ "it_id": id dell'itinerario richiesto, "datetime": orario e data di partenza o arrivo dell'itinerario richiesto, "utente": token o mail dell'utente, "starting_point": nome della fermata di partenza, "start_lat": latitudine della fermata di partenza, "start_lon": longitudine della fermata di partenza, "ending_point": nome della fermata di arrivo,</td></tr></table>			Codice risposta HTTP	Tipo/struttura	Descrizione	200 OK	json array	[{ "it_id": id dell'itinerario richiesto, "datetime": orario e data di partenza o arrivo dell'itinerario richiesto, "utente": token o mail dell'utente, "starting_point": nome della fermata di partenza, "start_lat": latitudine della fermata di partenza, "start_lon": longitudine della fermata di partenza, "ending_point": nome della fermata di arrivo,
Codice risposta HTTP	Tipo/struttura	Descrizione							
200 OK	json array	[{ "it_id": id dell'itinerario richiesto, "datetime": orario e data di partenza o arrivo dell'itinerario richiesto, "utente": token o mail dell'utente, "starting_point": nome della fermata di partenza, "start_lat": latitudine della fermata di partenza, "start_lon": longitudine della fermata di partenza, "ending_point": nome della fermata di arrivo,							

			"end_lat": latitudine della fermata di arrivo, "end_lon": longitudine della fermata di arrivo } , ...]
	400 ERROR	json	{"status": "ERROR"}

Nome	get-itinerari-proposti								
Descrizione	Get dal database relazionale di tutti gli itinerari proposti presenti.								
Resource URI	/api/get_itinerari_proposti								
Tipo di richiesta http	GET								
Content type della richiesta									
Content type della risposta	application/json								
Request body									
Path parameters									
Query parameters									
Response body	<table><tr><th>Codice risposta HTTP</th><th>Tipo/struttura</th><th>Descrizione</th></tr><tr><td>200 OK</td><td>json array</td><td>[{ "it_id": id dell'itinerario proposto, "start_datetime": orario e data di partenza dell'itinerario proposto, "end_datetime": orario e data di arrivo dell'itinerario proposto, "cost": costo in € dell'itinerario proposto, "distance": distanza percorsa dall'itinerario proposto, "state": enum che indica lo stato dell'itinerario proposto, può essere pending, confirmed o rejected, "it_req_id": id dell'itinerario richiesto</td></tr></table>			Codice risposta HTTP	Tipo/struttura	Descrizione	200 OK	json array	[{ "it_id": id dell'itinerario proposto, "start_datetime": orario e data di partenza dell'itinerario proposto, "end_datetime": orario e data di arrivo dell'itinerario proposto, "cost": costo in € dell'itinerario proposto, "distance": distanza percorsa dall'itinerario proposto, "state": enum che indica lo stato dell'itinerario proposto, può essere pending, confirmed o rejected, "it_req_id": id dell'itinerario richiesto
Codice risposta HTTP	Tipo/struttura	Descrizione							
200 OK	json array	[{ "it_id": id dell'itinerario proposto, "start_datetime": orario e data di partenza dell'itinerario proposto, "end_datetime": orario e data di arrivo dell'itinerario proposto, "cost": costo in € dell'itinerario proposto, "distance": distanza percorsa dall'itinerario proposto, "state": enum che indica lo stato dell'itinerario proposto, può essere pending, confirmed o rejected, "it_req_id": id dell'itinerario richiesto							

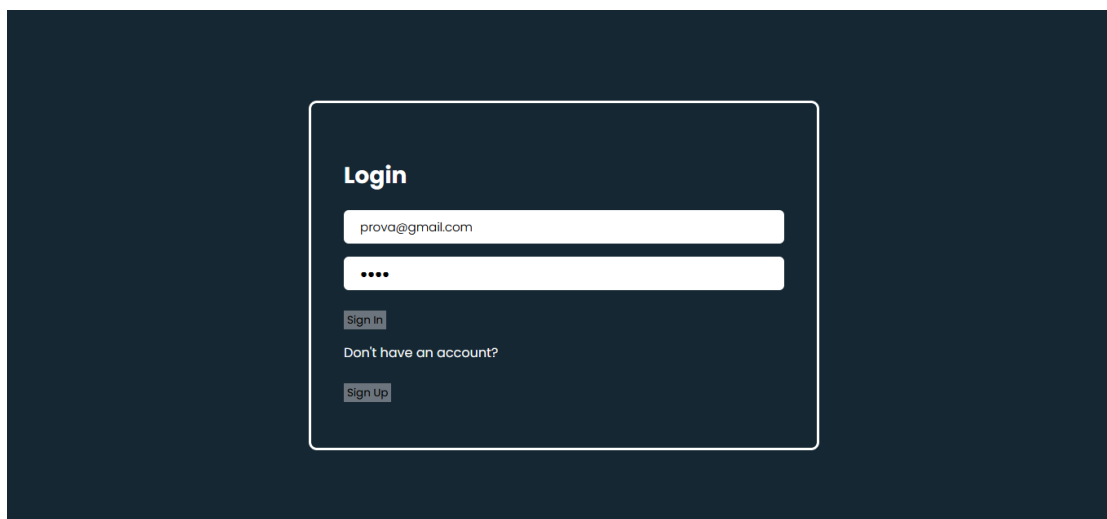
			associato all'itinerario proposto "route_id": id del percorso associato all'itinerario proposto, "utente": token o mail dell'utente, "starting_point": nome della fermata di partenza, "start_lat": latitudine della fermata di partenza, "start_lon": longitudine della fermata di partenza, "ending_point": nome della fermata di arrivo, "end_lat": latitudine della fermata di arrivo, "end_lon": longitudine della fermata di arrivo } , ...]
	400 ERROR	json	{"status":"ERROR"}

Nome	get-routes											
Descrizione	Get dal database relazionale di tutti i percorsi presenti.											
Resource URI	/api/get_routes											
Tipo di richiesta http	GET											
Content type della richiesta												
Content type della risposta	application/json											
Request body												
Path parameters												
Query parameters												
Response body	<table><tr><th>Codice risposta HTTP</th><th>Tipo/struttura</th><th>Descrizione</th></tr><tr><td>200 OK</td><td>json array</td><td>[{ "route_id": id del percorso, "expiration": data e orario di scadenza del percorso, "archiviato": flag che indica se il percorso è scaduto o meno, "stato": enum che indica lo stato del percorso, può essere pending, confirmed o rejected } , ...]</td></tr><tr><td>400 ERROR</td><td>json</td><td>{"status":"ERROR"}</td></tr></table>			Codice risposta HTTP	Tipo/struttura	Descrizione	200 OK	json array	[{ "route_id": id del percorso, "expiration": data e orario di scadenza del percorso, "archiviato": flag che indica se il percorso è scaduto o meno, "stato": enum che indica lo stato del percorso, può essere pending, confirmed o rejected } , ...]	400 ERROR	json	{"status":"ERROR"}
Codice risposta HTTP	Tipo/struttura	Descrizione										
200 OK	json array	[{ "route_id": id del percorso, "expiration": data e orario di scadenza del percorso, "archiviato": flag che indica se il percorso è scaduto o meno, "stato": enum che indica lo stato del percorso, può essere pending, confirmed o rejected } , ...]										
400 ERROR	json	{"status":"ERROR"}										

## Il sistema realizzato

### L'utente vuole richiedere un *itinerario*.

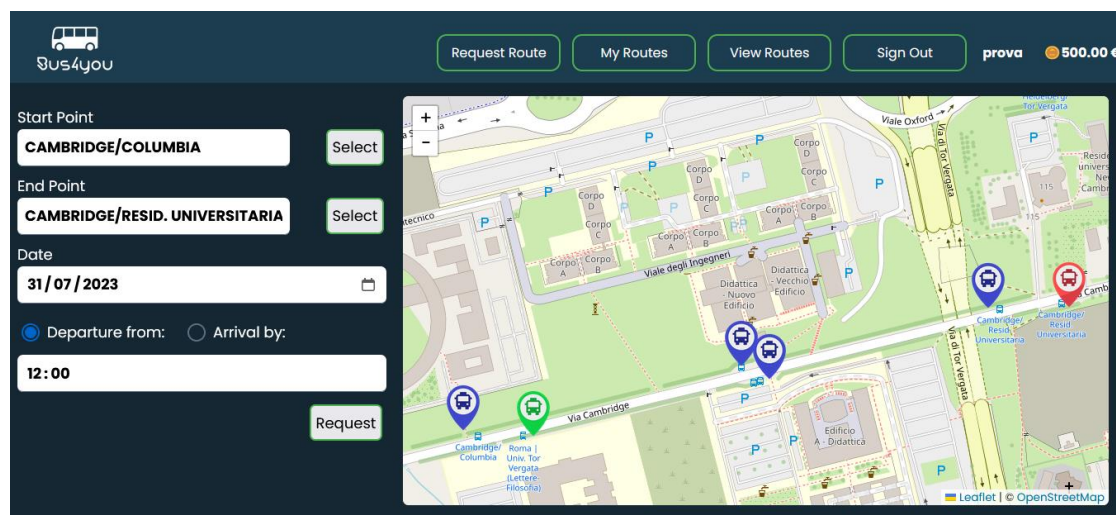
L'utente effettua l'accesso alla propria area personale utilizzando le proprie credenziali, nel caso non le disponga può effettuare il sign up (registrazione).



A login form on a dark blue background. The form is titled "Login" and contains two input fields: one for email (pre-filled with "prova@gmail.com") and one for password (masked with dots). Below the fields are two buttons: "Sign In" and "Sign Up". A link "Don't have an account?" is positioned between the two buttons.

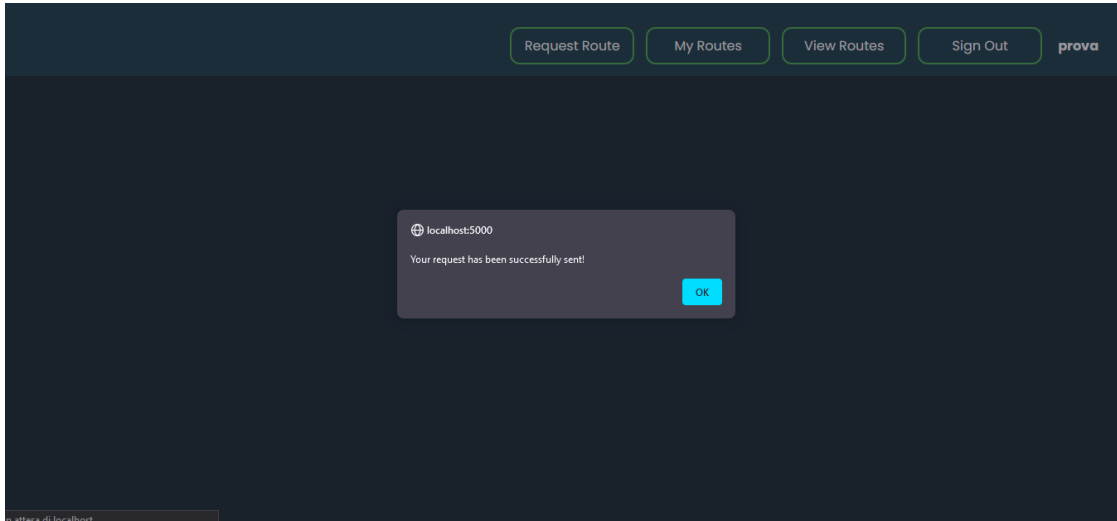
L'utente può fare la request di un itinerario nella sezione Request Route.

Per compilare la sua richiesta dovrà inserire la fermata di partenza e di arrivo (Start Point, End Point) cliccando sulle fermate presenti sulla mappa di destra, inserire una data non precedente a quella odierna e infine specifica l'orario indicando se è un orario di partenza o di arrivo.



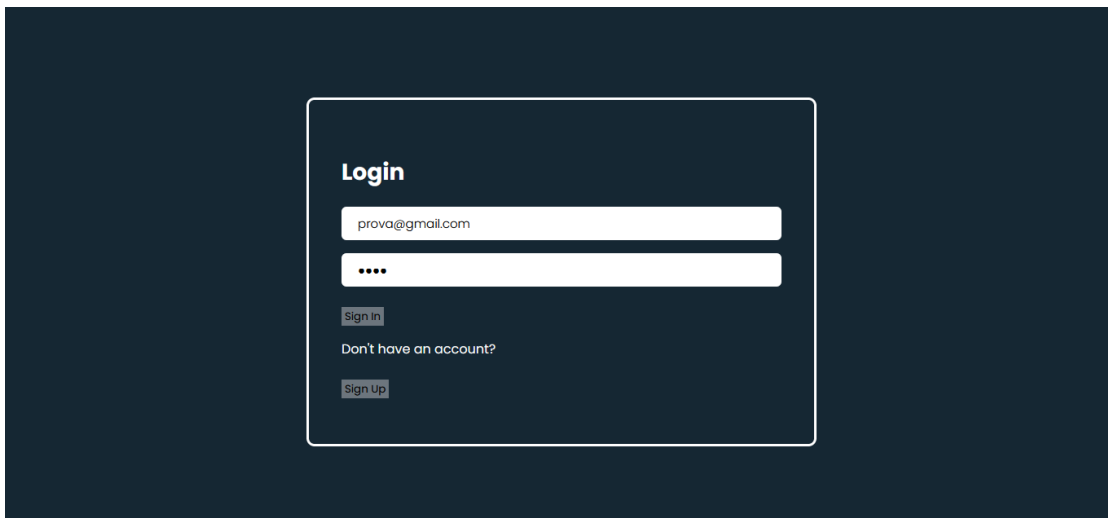
A web interface for requesting a route. On the left, there are form fields for "Start Point" (CAMBRIDGE/COLUMBIA), "End Point" (CAMBRIDGE/RESID. UNIVERSITARIA), "Date" (31/07/2023), and "Time" (12:00). There are radio buttons for "Departure from:" and "Arrival by:". A "Request" button is at the bottom. On the right, there is a map showing a campus with various buildings and bus stops. The map includes labels like "Viale degli Ingegneri", "Viale Oxford", and "Via Cambridge". The interface also features a top navigation bar with buttons: "Request Route", "My Routes", "View Routes", "Sign Out", and a balance indicator "prova 500.00 €".

Se la richiesta è andata a buon fine si avrà la seguente schermata:



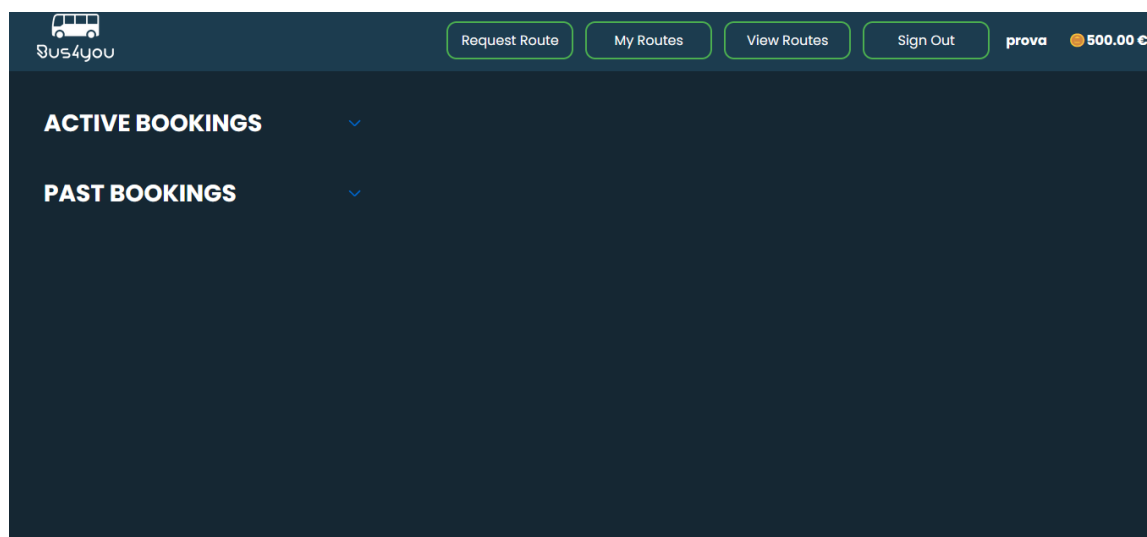
## 2) L'utente vuole vedere i suoi *itinerari proposti*.

L'utente effettua l'accesso alla propria area personale utilizzando le proprie credenziali, nel caso non ne disponga può effettuare il sign up (registrazione).

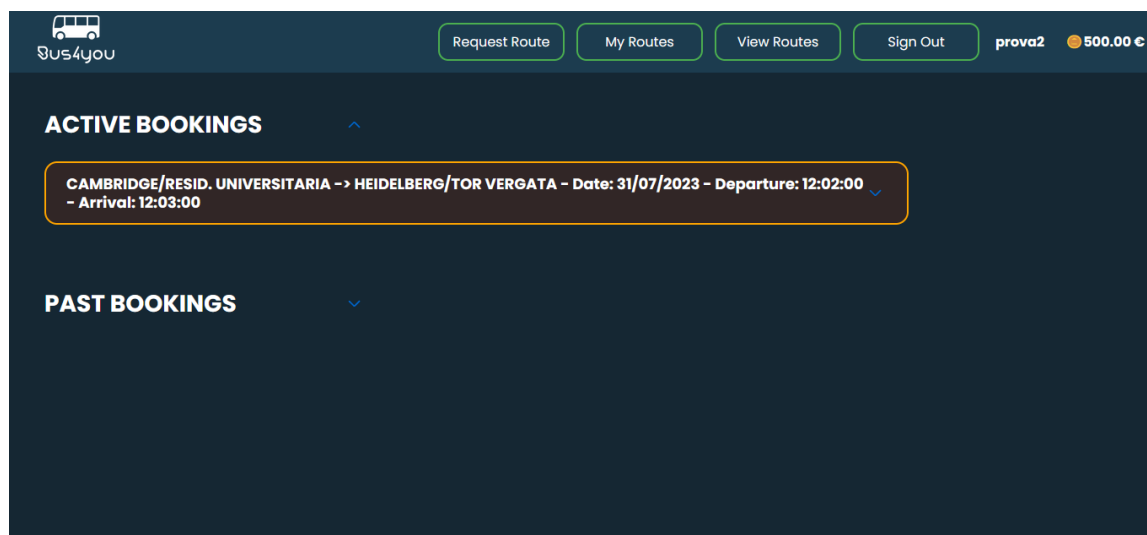





Nella sezione My Routes è possibile visualizzare sia le richieste attive che quelle passate.




Cliccando su Active Booking è possibile vedere le prenotazioni al momento attive.



È inoltre possibile cliccare sulla relativa prenotazione per vederne i dettagli.


**Sus4you**

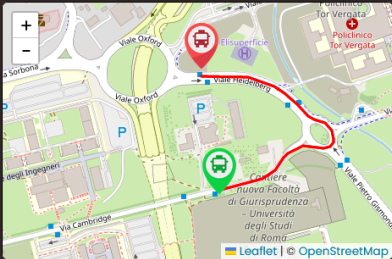
[Request Route](#)
[My Routes](#)
[View Routes](#)
[Sign Out](#)
**prova2**  **500.00 €**

### ACTIVE BOOKINGS

**CAMBRIDGE/RESID. UNIVERSITARIA -> HEIDELBERG/TOR VERGATA - Date: 31/07/2023 - Departure: 12:02:00 - Arrival: 12:03:00**


Date: 31/07/2023  
 Departure Time: 12:02:00  
 Arrival Time: 12:03:00  
 Departure Stop: CAMBRIDGE/RESID. UNIVERSITARIA  
 Arrival Stop: HEIDELBERG/TOR VERGATA  
 Price: 0.47  
 Itinerary Status: pending  
 Route Status: pending  
 Route Deadline: 21/07/2023 00:59:40


[Confirm Book](#)
[Reject Book](#)



L'utente vuole confermare o rifiutare un suo *itinerario proposto* dal servizio.

Nella sezione My Routes è possibile visualizzare i dettagli della prenotazione richiesta al servizio.


**Sus4you**


[Request Route](#)
[My Routes](#)
[View Routes](#)
[Sign Out](#)
**prova2**  **500.00 €**

### ACTIVE BOOKINGS

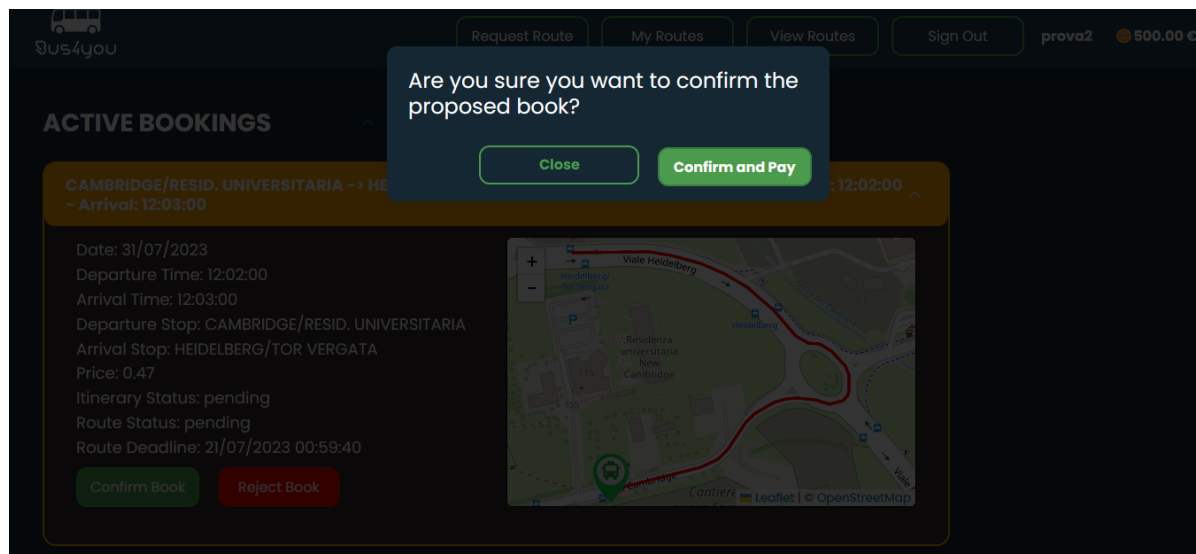
**CAMBRIDGE/RESID. UNIVERSITARIA -> HEIDELBERG/TOR VERGATA - Date: 31/07/2023 - Departure: 12:02:00 - Arrival: 12:03:00**

Date: 31/07/2023  
 Departure Time: 12:02:00  
 Arrival Time: 12:03:00  
 Departure Stop: CAMBRIDGE/RESID. UNIVERSITARIA  
 Arrival Stop: HEIDELBERG/TOR VERGATA  
 Price: 0.47  
 Itinerary Status: pending  
 Route Status: pending  
 Route Deadline: 21/07/2023 00:59:40

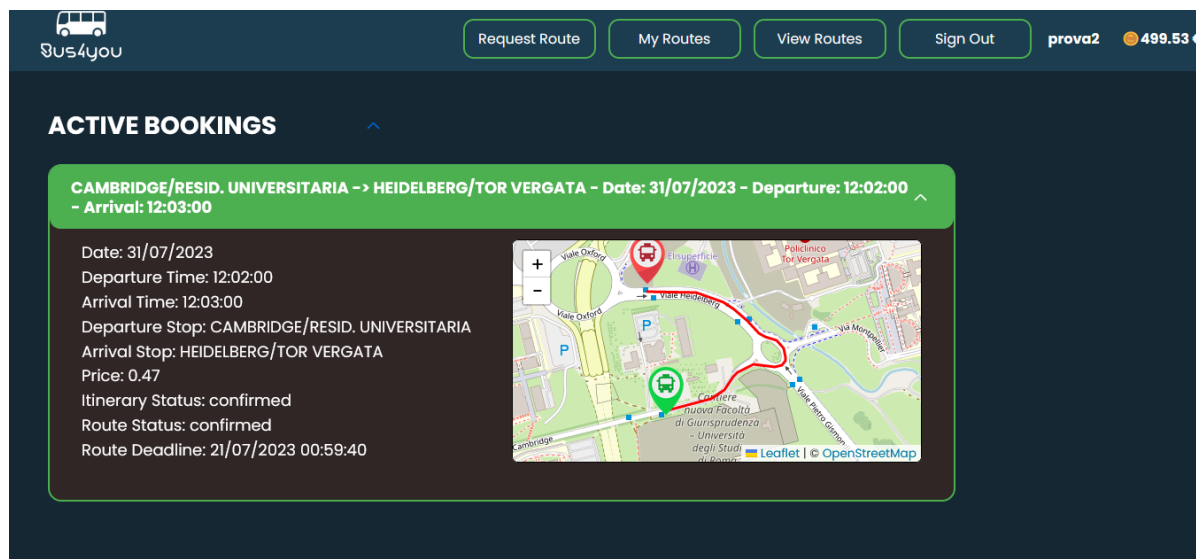
[Confirm Book](#)
[Reject Book](#)



È possibile confermare l'itinerario proposto dal servizio, nel caso si confermi allora il servizio mostrerà una schermata di conferma:

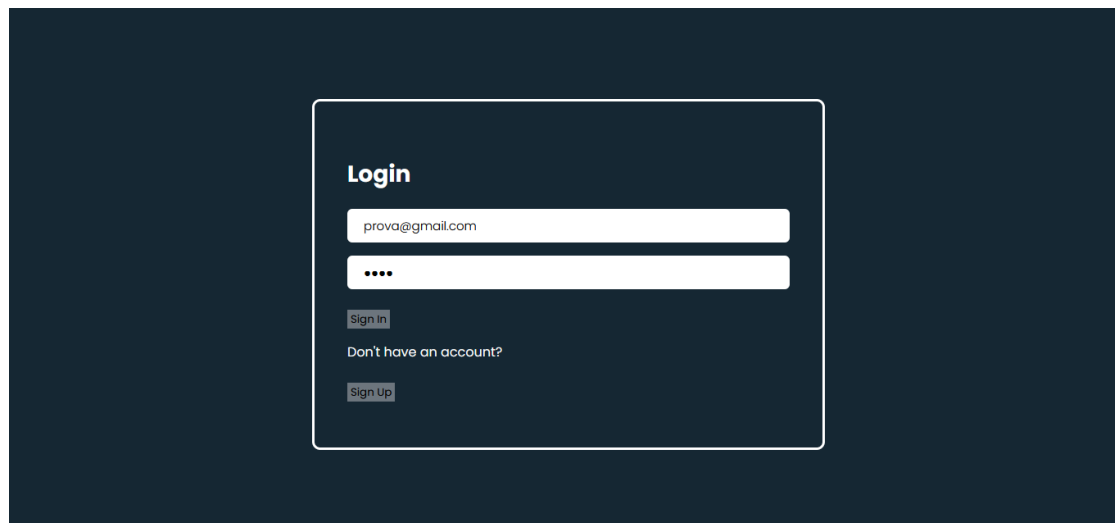


Se un numero sufficiente di utenti ha confermato la tratta che il mezzo di trasporto effettuerà allora lo stato della richiesta diventerà confirmed e quindi la richiesta è andata a buon fine.



## L'utente vuole aggiungersi ad una recommended route già esistente.

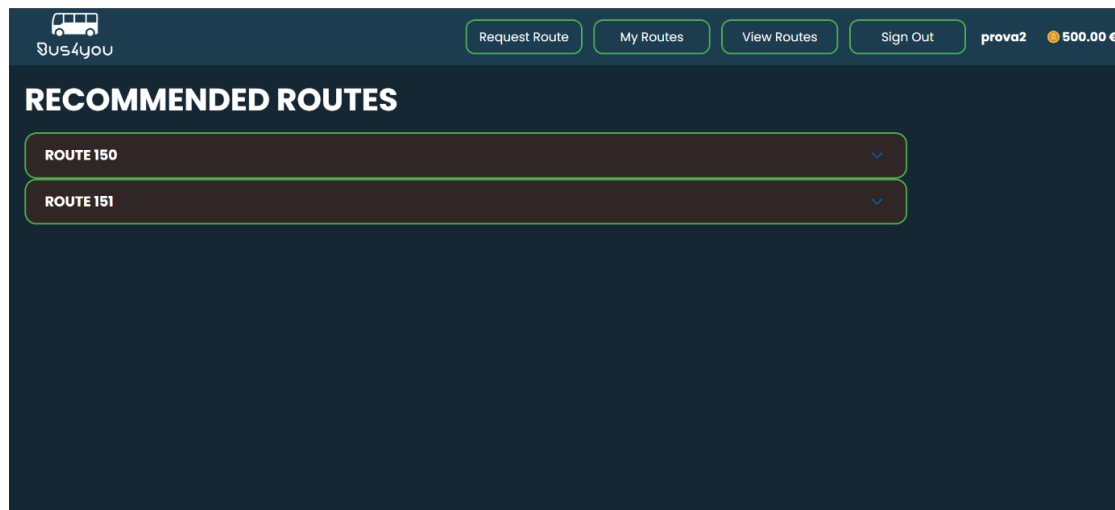
L'utente effettua l'accesso alla propria area personale utilizzando le proprie credenziali, nel caso non le disponga può effettuare il sign up (registrazione).



The screenshot shows a login interface on a dark blue background. A white-bordered box contains the following elements:

- Login** header
- Email input field containing "prova@gmail.com"
- Password input field with four dots
- [Sign In](#) button
- [Don't have an account?](#) link
- [Sign Up](#) button

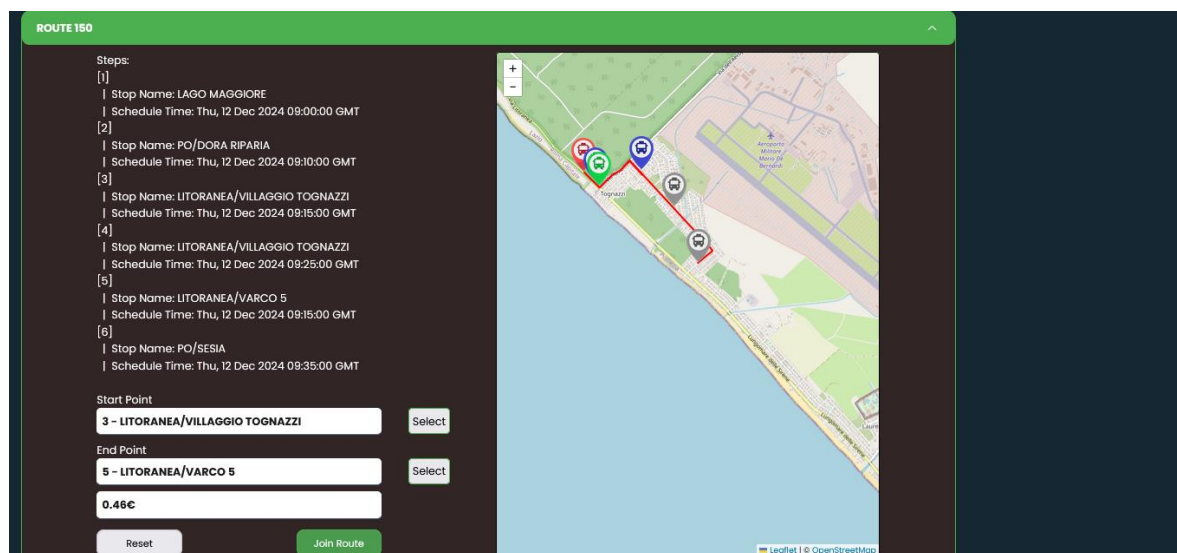
Nella sezione view Routes l'utente può visualizzare le tratte che sono già state confermate.



The screenshot shows the "view Routes" section of the application. At the top, there is a navigation bar with a bus icon and the text "Bus4you". To the right of the icon are buttons: "Request Route", "My Routes", "View Routes" (which is active), and "Sign Out". Further right, the user's name "prova2" and a balance of "500.00 €" are displayed. Below the navigation bar, the section is titled "RECOMMENDED ROUTES". It contains two items, each in a dark blue box with a green border and a green chevron icon on the right:

- ROUTE 150
- ROUTE 151

Cliccando su una delle rotte è possibile vedere le informazioni della tratta e di tutte le fermate che effettuerà, l'utente potrà cliccare sulla fermata di partenza e di arrivo per unirsi a tale tratta, una volta fatto ciò verrà anche mostrato il prezzo del biglietto ridotto per quella combinazione di fermate.



**ROUTE 150**

Steps:

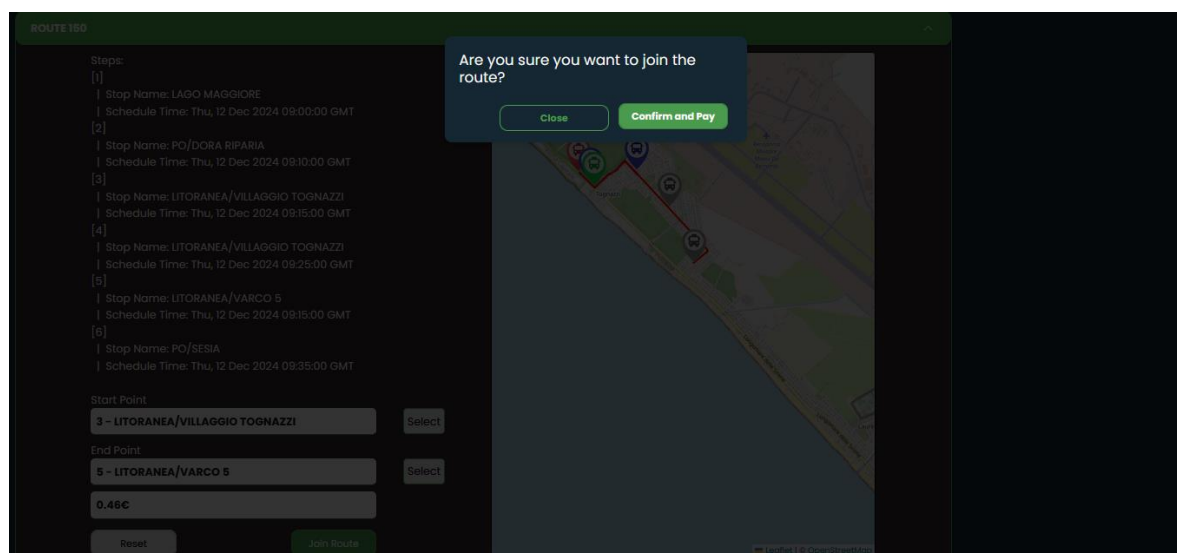
- [1] Stop Name: LAGO MAGGIORE  
Schedule Time: Thu, 12 Dec 2024 09:00:00 GMT
- [2] Stop Name: PO/DORA RIPARIA  
Schedule Time: Thu, 12 Dec 2024 09:10:00 GMT
- [3] Stop Name: LITORANEA/VILLAGGIO TOGNAZZI  
Schedule Time: Thu, 12 Dec 2024 09:15:00 GMT
- [4] Stop Name: LITORANEA/VILLAGGIO TOGNAZZI  
Schedule Time: Thu, 12 Dec 2024 09:25:00 GMT
- [5] Stop Name: LITORANEA/VARCO 5  
Schedule Time: Thu, 12 Dec 2024 09:15:00 GMT
- [6] Stop Name: PO/SESIA  
Schedule Time: Thu, 12 Dec 2024 09:35:00 GMT

Start Point

End Point

**0.46€**

Verrà mostrato un messaggio di conferma cliccando su join route.



**ROUTE 150**

Steps:

- [1] Stop Name: LAGO MAGGIORE  
Schedule Time: Thu, 12 Dec 2024 09:00:00 GMT
- [2] Stop Name: PO/DORA RIPARIA  
Schedule Time: Thu, 12 Dec 2024 09:10:00 GMT
- [3] Stop Name: LITORANEA/VILLAGGIO TOGNAZZI  
Schedule Time: Thu, 12 Dec 2024 09:15:00 GMT
- [4] Stop Name: LITORANEA/VILLAGGIO TOGNAZZI  
Schedule Time: Thu, 12 Dec 2024 09:25:00 GMT
- [5] Stop Name: LITORANEA/VARCO 5  
Schedule Time: Thu, 12 Dec 2024 09:15:00 GMT
- [6] Stop Name: PO/SESIA  
Schedule Time: Thu, 12 Dec 2024 09:35:00 GMT

Start Point

End Point

**0.46€**

Are you sure you want to join the route?

## Bibliografia

2-OPT. (s.d.). Tratto da Wikipedia: <https://en.wikipedia.org/wiki/2-opt>

Kahn, A. B. (1962). Topological sorting of large networks. *Communications of the ACM*, vol. 5, n. 11, 558–562.  
doi:<https://dl.acm.org/doi/10.1145/368996.369025>