



# UNIVERSITÀ DEL SALENTO

## Facoltà di Ingegneria

Corso di Laurea in Ingegneria dell'Informazione

---

Documentazione di Progetto

per il corso

*Internet of Things*

### Sistema di logistica avanzata



**Professore:**

*Luigi Patrono*

**Tutor:**

*Ing. Teodoro Montanaro*

**Studenti:**

*Mandorino Carmelo* [20103972]

*Settimo Simone* [20103623]

---

Dipartimento di Ingegneria dell'Innovazione

ANNO ACCADEMICO 2024/2025

# Indice

<b>1</b>	<b>Introduzione</b>	<b>1</b>
<b>2</b>	<b>Analisi dei requisiti</b>	<b>3</b>
2.1	Requisiti funzionali . . . . .	3
2.2	Requisiti non funzionali . . . . .	4
<b>3</b>	<b>Stato dell'arte</b>	<b>5</b>
3.1	Ricerca a Grande Vicinato Adattiva (ALNS) . . . . .	5
3.2	Deep Reinforcement Learning . . . . .	6
3.3	Google OR-Tools . . . . .	7
3.4	UPS ORION . . . . .	8
3.5	Piattaforme di ottimizzazione del routing in tempo reale . . . . .	9
3.6	Conclusioni . . . . .	10
<b>4</b>	<b>Tecnologie utilizzate</b>	<b>11</b>
4.1	GPS e protocollo MQTT . . . . .	11
4.2	MongoDB . . . . .	11
4.3	Autenticazione JWT . . . . .	12
4.4	React Native con Expo e AsyncStorage . . . . .	12
4.5	Java Spring Boot e API REST . . . . .	12
4.6	Python e OR-Tools . . . . .	13
4.7	Docker e AWS . . . . .	13
4.8	Firebase . . . . .	13
4.9	GraphHopper . . . . .	14
4.9.1	Tailscale . . . . .	14
<b>5</b>	<b>Soluzione proposta</b>	<b>15</b>
5.1	Architettura generale . . . . .	15
5.2	Funzionamento logico . . . . .	16
5.3	Attori principali . . . . .	17
5.4	Elenco dei Casi d'Uso . . . . .	17
5.5	Diagrammi di flusso principali . . . . .	22
<b>6</b>	<b>Validazione funzionale e/o prestazionale</b>	<b>25</b>
6.1	Interfaccia iniziale . . . . .	25
6.2	Caso d'uso reale: Assegnazione di una tratta da parte dell'amministratore . . . . .	27

6.3	Accettazione e completamento della tratta da parte del camionista . .	29
6.3.1	Segnalazione di un'anomalia . . . . .	30
6.4	Hardware e strumenti utilizzati . . . . .	32
6.5	Validazione funzionale . . . . .	32
6.6	Validazione prestazionale . . . . .	33
6.7	Conclusioni . . . . .	33
<b>7</b>	<b>Conclusioni e sviluppi futuri</b>	<b>34</b>
<b>8</b>	<b>Link utili</b>	<b>36</b>
	<b>Elenco delle figure</b>	<b>37</b>

# Capitolo 1

## Introduzione

Negli ultimi anni, la logistica avanzata ha assunto un ruolo sempre più centrale nella competitività delle aziende, soprattutto in settori dove la gestione dei trasporti è complessa e richiede l'ottimizzazione di molte variabili. La movimentazione di merci e mezzi di trasporto, infatti, non si limita più a pianificare semplicemente un percorso: occorre considerare vincoli temporali, capacità dei veicoli, disponibilità del personale e possibili anomalie lungo il tragitto. In questo contesto, il problema del *Pick-Up and Delivery* (PDP) rappresenta una delle sfide più frequenti e delicate. Esso consiste nel dover prelevare e consegnare merci in punti diversi, rispettando finestre temporali prestabilite (*time windows*) e limitazioni di capacità, minimizzando al contempo i costi operativi e il numero di veicoli utilizzati. Le soluzioni tradizionali a questo problema si basano spesso su metodi di pianificazione manuale o su software che trattano un sottoinsieme dei vincoli reali. Ciò comporta che, all'aumentare del numero di ordini e della complessità delle consegne, le inefficienze crescano in modo esponenziale. Alcuni sistemi commerciali di gestione delle flotte, ad esempio, riescono a calcolare percorsi ottimizzati in base alla distanza o al tempo, ma non integrano contemporaneamente vincoli di capacità, finestre temporali e riassegnazione dinamica in caso di imprevisti. Inoltre, in presenza di anomalie come guasti ai veicoli o deviazioni di percorso, molte soluzioni non dispongono di meccanismi automatici di ricalcolo e riorganizzazione della distribuzione. Il progetto sviluppato nasce con l'obiettivo di superare questi limiti, realizzando un sistema intelligente di logistica avanzata capace di integrare in un'unica piattaforma tutte le fasi del processo: dalla selezione degli ordini alla pianificazione automatica delle tratte, fino al monitoraggio in tempo reale e alla gestione delle anomalie. La nostra soluzione si rivolge a contesti aziendali in cui la pianificazione è critica, e prevede una struttura multiutente con due principali figure operative: l'**admin** e il **camionista**.

L'**admin** può selezionare, da un'interfaccia intuitiva, gli ordini da evadere in un determinato tempo. Gli ordini possono riguardare sia operazioni di carico che di scarico, e ciascuno di essi è caratterizzato da una finestra temporale di esecuzione e da una quantità di merce associata. Una volta selezionati gli ordini, il sistema applica un algoritmo di *Capacitated Vehicle Routing Problem with Time Windows* (CVRPTW) esteso al *Pick-Up and Delivery Problem* (PDP) e arricchito con il supporto alle *time windows*. L'algoritmo è stato adattato per rispondere ai vincoli specifici del nostro scenario, garantendo contemporaneamente: il rispetto delle fine-

stre temporali, della capacità di ciascun veicolo e dell'uso del minor numero possibile di camion.

Una volta generata la pianificazione, l'admin assegna le tratte calcolate ai camionisti disponibili. A questo punto il camionista riceve una notifica, tramite l'applicazione mobile, dell'ordine assegnato. Dopo aver accettato l'incarico, può visualizzare sulla mappa il percorso ottimale da seguire, suddiviso in tappe corrispondenti alle operazioni di carico e scarico. La navigazione è arricchita da un sistema di tracciamento GPS in tempo reale, che consente al backend di monitorare la posizione e di rilevare eventuali deviazioni dalla rotta prevista.

Uno degli aspetti più innovativi della soluzione è la gestione delle anomalie. In caso di deviazione significativa, guasto al mezzo o impossibilità a proseguire la consegna, il sistema è in grado di ricalcolare dinamicamente il percorso e, se necessario, riassegnare l'ordine a un altro veicolo disponibile, tenendo conto sia della vicinanza geografica sia della capacità residua. Questo approccio rende il sistema resiliente e adatto a scenari reali, in cui l'imprevisto è una variabile costante.

Dal punto di vista tecnologico, il progetto integra componenti eterogenee ma fortemente interconnesse:

- un algoritmo di ottimizzazione implementato in Python con **OR-Tools** e integrato con **GraphHopper** per il calcolo dei percorsi reali;
- un backend **Spring Boot** per la gestione dei dati, degli utenti e delle comunicazioni con i camionisti;
- un'app mobile sviluppata in **React Native** per l'interazione diretta con i conducenti, comprensiva di mappa interattiva, notifiche e tracciamento GPS.

In sintesi, la soluzione proposta affronta il problema precedentemente illustrato in maniera completa, gestendo in modo integrato pianificazione, monitoraggio e reazione agli imprevisti. L'obiettivo non è soltanto ottimizzare le rotte, ma fornire uno strumento capace di adattarsi in tempo reale alle condizioni operative, migliorando l'efficienza, riducendo i costi e aumentando l'affidabilità complessiva del servizio di trasporto.

# Capitolo 2

## Analisi dei requisiti

In questo capitolo vengono descritti i requisiti funzionali e non funzionali del sistema intelligente di logistica avanzata sviluppato. Tali requisiti sono stati definiti a partire dall'analisi delle necessità operative dell'azienda e dalle specifiche fornite nella fase di progettazione. L'obiettivo è garantire che il sistema soddisfi le esigenze reali di pianificazione, monitoraggio e gestione delle consegne, integrando funzionalità di ottimizzazione automatica e gestione delle anomalie.

### 2.1 Requisiti funzionali

I requisiti funzionali definiscono le azioni e i comportamenti che il sistema deve poter eseguire per raggiungere gli obiettivi prefissati.

- **Gestione ordini:** l'admin deve poter inserire, modificare e selezionare ordini di carico e scarico, specificando quantità, punti di ritiro/consegna e finestre temporali.
- **Ottimizzazione del percorso:** il sistema deve calcolare rotte ottimali rispettando vincoli di capacità, finestre temporali e minimizzando il numero di veicoli utilizzati.
- **Assegnazione tratte:** l'admin deve poter assegnare una tratta ottimizzata a un camionista disponibile.
- **Notifica ordini:** il camionista deve ricevere una notifica quando una nuova tratta gli viene assegnata. Una volta accettata la tratta, l'admin riceverà una notifica che il camionista ha accettato la tratta.
- **Accettazione consegna:** il camionista deve poter accettare l'ordine assegnato.
- **Visualizzazione percorso:** il camionista deve poter visualizzare la rotta da seguire su mappa, suddivisa in tappe (pick-up e delivery).
- **Tracciamento GPS:** il sistema deve acquisire e registrare la posizione in tempo reale del veicolo durante la consegna.

- **Gestione anomalie:** in caso di guasto, deviazione o impossibilità a completare il percorso, il sistema deve ricalcolare la rotta o riassegnare l'ordine a un altro veicolo/camionista disponibile o in transito.

## 2.2 Requisiti non funzionali

I requisiti non funzionali stabiliscono vincoli e caratteristiche qualitative che il sistema deve rispettare per garantire prestazioni, affidabilità e usabilità.

- **Prestazioni:** il calcolo delle rotte ottimizzate deve avvenire in un tempo accettabile (inferiore a pochi secondi per scenari di media complessità).
- **Scalabilità:** il sistema deve poter gestire un numero crescente di ordini e veicoli senza degradare le prestazioni in maniera significativa.
- **Affidabilità:** il sistema deve essere resistente a errori e garantire il recupero delle operazioni in caso di interruzioni.
- **Usabilità:** le interfacce per admin e camionista devono essere intuitive e facilmente utilizzabili anche da utenti non esperti.
- **Sicurezza:** tutte le comunicazioni tra i componenti (backend, app mobile, servizi esterni) devono avvenire tramite protocolli sicuri e autenticazione basata su token.
- **Manutenibilità:** il codice deve essere modulare e ben documentato, in modo da semplificare eventuali estensioni e aggiornamenti futuri.
- **Affidabilità del tracciamento GPS:** il sistema deve garantire un'accuratezza adeguata della posizione e continuità nella trasmissione dei dati.

# Capitolo 3

## Stato dell'arte

Il Pickup and Delivery Problem (PDP) con vincoli sul tempo e sulle capacità, rappresenta una delle varianti più complesse dei classici problemi di instradamento veicolare (VRP). In questo scenario ogni richiesta di trasporto è costituita da una coppia di punti, un ritiro (pickup) e una consegna (delivery), che devono essere pianificati nel rispetto di vincoli stringenti. Ciascun veicolo ha infatti una capacità massima di carico, ogni operazione deve avvenire entro una finestra temporale prestabilita e, soprattutto, il ritiro deve sempre precedere la consegna corrispondente. L'obiettivo principale è ridurre i costi complessivi di trasporto, ad esempio minimizzando la distanza percorsa o il tempo di viaggio, cercando al contempo di impiegare il minor numero possibile di veicoli. La difficoltà nasce dalla natura combinatoria del problema che include come sottocaso il noto Traveling Salesman Problem (TSP) ed è quindi classificato come NP-difficile. Questo significa che trovare soluzioni esatte per istanze di grandi dimensioni risulta proibitivo, rendendo necessarie tecniche avanzate di ottimizzazione.

Negli anni sono state proposte numerose soluzioni, sia in ambito accademico che industriale, che cercano di affrontare queste complessità. Nel presente capitolo verranno analizzate alcune delle più significative: da algoritmi metaeuristici e approcci basati sull'intelligenza artificiale, fino a software e piattaforme commerciali già utilizzate nel settore della logistica. Per ciascuna verranno descritte le caratteristiche principali e i limiti evidenziati, che costituiranno i punti di partenza per la soluzione sviluppata nel nostro progetto.

### 3.1 Ricerca a Grande Vicinato Adattiva (ALNS)

Tra le soluzioni più influenti in letteratura per il PDPTW si trova l'*Adaptive Large Neighborhood Search* (ALNS), proposta da Ropke e Pisinger (2006) [1]. Si tratta di una metaeuristica che estende l'approccio di *Large Neighborhood Search*, originariamente introdotto per il VRP con finestre temporali, e che combina fasi di rimozione e reinserimento delle richieste di trasporto per esplorare lo spazio delle soluzioni. La caratteristica "adattiva" consiste nel fatto che le diverse mosse disponibili vengono selezionate in base alle loro prestazioni: gli operatori che in passato hanno portato a miglioramenti significativi ricevono un peso maggiore e hanno quindi più probabilità di essere scelti nelle iterazioni successive. Questo meccanismo consente di bilanciare



la diversificazione (esplorando configurazioni molto diverse) e l'intensificazione (affinando le soluzioni già promettenti), rendendo l'ALNS robusto ed efficace su istanze con caratteristiche differenti.

Nel lavoro originale, gli autori hanno testato l'algoritmo su oltre 350 istanze di benchmark, arrivando a migliorare in più della metà dei casi le migliori soluzioni note all'epoca. L'ALNS è in grado di garantire la fattibilità delle soluzioni rispetto ai vincoli fondamentali del PDPTW: il carico dei veicoli non supera mai la capacità massima, gli orari di arrivo rispettano sempre le finestre temporali consentite (con eventuali tempi di attesa in caso di anticipo) e ogni operazione di pickup precede sempre la relativa consegna. L'algoritmo rifiuta automaticamente qualsiasi mossa che violerebbe queste condizioni, assicurando in ogni momento la validità del percorso.

Nonostante la sua efficacia, l'ALNS presenta alcuni limiti quando si passa da uno scenario teorico a un contesto logistico reale. Si tratta infatti di un algoritmo *offline*: richiede che tutti i dati siano disponibili a priori e produce un piano ottimizzato, ma non contempla l'aggiornamento continuo in caso di imprevisti, come ritardi, guasti o nuove richieste inserite all'ultimo minuto. Inoltre, come molte metaeuristiche, necessita di una calibrazione attenta dei parametri e diventa più oneroso in termini di calcolo con l'aumentare della dimensione del problema. L'ALNS di Ropke e Pisinger rappresenta uno dei riferimenti accademici per la risoluzione del PDPTW, garantendo soluzioni di alta qualità per istanze statiche. Tuttavia, per poterlo applicare in un sistema di logistica avanzata è necessario integrarlo in una piattaforma più ampia, capace di monitorare in tempo reale l'operatività e di ricalcolare i percorsi in risposta a eventi dinamici.

## 3.2 Deep Reinforcement Learning

Negli ultimi anni, accanto alle metaeuristiche tradizionali, si sono diffusi approcci basati su Intelligenza Artificiale e in particolare sul *Deep Reinforcement Learning* (DRL). Una proposta significativa in questo ambito è quella di Li et al. [2], che affrontano il PDPTW tramite un agente intelligente addestrato a costruire percorsi validi rispettando i vincoli di capacità, precedenza e finestre temporali. L'idea di fondo è far apprendere a una rete neurale, basata su architettura Transformer, la costruzione incrementale delle rotte: a ogni passo, l'agente decide quale nodo servire (pickup o delivery) ricevendo una ricompensa legata alla qualità della soluzione complessiva.

Per garantire il rispetto dei vincoli, il modello utilizza due accorgimenti chiave. In primo luogo, introduce un meccanismo di *attenzione eterogenea* che distingue tra nodi di ritiro e nodi di consegna, assicurando implicitamente il rispetto delle precedenze. In secondo luogo, adotta uno schema di *masking* delle azioni non valide, che impedisce ad esempio di selezionare un nodo di consegna prima del corrispondente pickup o di generare mosse che portino a superare la capacità del veicolo o a violare i time window. In questo modo la rete esplora soltanto sequenze di decisioni ammissibili, migliorando progressivamente la propria policy durante la fase di addestramento. Il punto di forza di questo approccio è la rapidità di inferenza: una volta addestrato, l'agente è in grado di generare soluzioni di buona qualità in

tempi estremamente ridotti, dimostrandosi competitivo con le migliori metaeuristiche disponibili in letteratura. I risultati sperimentali riportati dagli autori mostrano infatti che il metodo DRL supera spesso sia gli algoritmi euristici tradizionali sia altri modelli di deep learning applicati al PDPTW.

Tuttavia, l'adozione pratica di questi sistemi presenta ancora alcune criticità. L'addestramento richiede dataset di grandi dimensioni, risorse hardware costose e tempi di calcolo elevati. Inoltre, la rete neurale rimane una "black box" difficilmente interpretabile, un aspetto che può rappresentare un limite per l'uso industriale dove è importante garantire trasparenza e spiegabilità. Infine, l'integrazione di un modello DRL in scenari real-time non è immediata, poiché occorre prevedere meccanismi di aggiornamento o riaddestramento per adattarsi a condizioni operative mutevoli.

Quindi il DRL rappresenta uno stato dell'arte emergente molto promettente: offre la possibilità di combinare qualità delle soluzioni e tempi di risposta rapidi, aprendo prospettive interessanti per applicazioni dinamiche. Al tempo stesso, la sua maturità tecnologica non è ancora sufficiente per sostituire in toto gli algoritmi più consolidati, rendendolo ad oggi una soluzione complementare e di grande interesse per la ricerca.

### 3.3 Google OR-Tools

Tra le soluzioni già disponibili e ampiamente adottate sia in ambito accademico che industriale vi è Google OR-Tools, una libreria open-source sviluppata da Google e specializzata in problemi di ottimizzazione combinatoria [3]. Essa offre un potente solver di *Vehicle Routing Problem* capace di gestire varianti con capacità, finestre temporali e vincoli di pick-up e delivery.

L'utilizzo della libreria è reso agevole da API ad alto livello che consentono di modellare i principali vincoli logistici: la capacità dei veicoli, tramite la definizione di una dimensione "Capacity"; i vincoli temporali, grazie a variabili cumulative che tracciano l'orario di arrivo e garantiscono il rispetto delle finestre; e i vincoli di precedenza, mediante funzioni dedicate come *AddPickupAndDelivery*, che assicurano che ritiro e consegna siano effettuati dallo stesso veicolo nell'ordine corretto.

OR-Tools costruisce soluzioni iniziali con euristiche dedicate e le migliora tramite metaeuristiche come la ricerca locale guidata e la tabu search. In questo modo riesce a produrre soluzioni valide ed efficienti su problemi di piccola e media scala, pur mostrando maggiori difficoltà su istanze molto grandi, dove algoritmi specializzati ottengono spesso risultati migliori entro tempi ridotti. Nonostante ciò, la sua efficienza e facilità di integrazione hanno reso OR-Tools uno standard di fatto in numerosi progetti applicativi.

Va evidenziato, tuttavia, che OR-Tools nasce come componente di back-end e non offre funzionalità complete di sistema: non gestisce, ad esempio, interfacce utente, monitoraggio GPS o riassegnazioni automatiche in caso di imprevisti. In scenari reali viene tipicamente integrato in piattaforme più ampie che raccolgono i dati operativi (posizioni dei veicoli, traffico, nuove richieste) e richiamano periodicamente il solver per aggiornare i percorsi. Questo approccio consente di compensare la natura statica del risolutore, sfruttandone la rapidità per ottenere ricalcoli frequenti.

In sintesi, OR-Tools rappresenta una soluzione general-purpose estremamente utile

e versatile. I suoi principali limiti emergono su istanze di grandissima scala e nella mancanza di funzionalità avanzate di gestione real-time, ma proprio per questo può fungere da base solida da integrare in sistemi logistici più complessi. Nel nostro progetto lo utilizziamo come motore di ottimizzazione, arricchendolo con moduli per il tracking e la gestione dinamica delle consegne.

### 3.4 UPS ORION

Un esempio emblematico di sistema di logistica avanzata è ORION (On-Road Integrated Optimization and Navigation), sviluppato da UPS a partire dal 2003 con un investimento superiore ai 250 milioni di dollari [4]. Dal 2016 è pienamente operativo e ogni giorno calcola i percorsi ottimali per circa 55.000 autisti negli Stati Uniti, pianificando la sequenza di consegne e ritiri da effettuare.

Il problema affrontato da ORION è essenzialmente un grande VRP con finestre temporali, in cui occorre rispettare sia i vincoli di capacità dei veicoli sia gli slot promessi ai clienti. Oltre a minimizzare la distanza totale percorsa, il sistema tiene conto anche della stabilità operativa: i percorsi vengono mantenuti simili di giorno in giorno per non disorientare gli autisti e ridurre modifiche inutili.

Il cuore algoritmico di ORION si basa su metaeuristiche appositamente sviluppate, arricchite nel tempo per gestire vincoli pratici complessi, come la riduzione delle svolte a sinistra, il rispetto di normative di sicurezza stradale, o la gestione di punti di consegna difficilmente accessibili. La capacità viene gestita in due fasi: prima l'assegnazione dei pacchi ai veicoli (dispatch), poi l'ottimizzazione interna di ciascun giro, tenendo conto del limite di fermate giornaliere. Grazie a queste strategie, UPS ha riportato risparmi enormi: già nel 2016 il sistema aveva ridotto di milioni i chilometri percorsi, con un beneficio economico superiore ai 300 milioni di dollari annui e un impatto ambientale positivo dovuto alla diminuzione delle emissioni di CO<sub>2</sub>.

Un aspetto distintivo di ORION è l'integrazione con dispositivi GPS e tablet di bordo (DIAD), che permettono di raccogliere dati in tempo reale su percorsi, orari e ritardi. Questo consente un monitoraggio continuo da parte della centrale operativa. Nelle versioni più recenti (Dynamic ORION), il sistema è in grado di ricalcolare i percorsi anche durante la giornata per reagire a eventi imprevisti, pur privilegiando la stabilità e limitando i cambiamenti non strettamente necessari.

ORION rappresenta un modello di riferimento per l'applicazione delle tecniche di Operations Research su larga scala. Tuttavia, si tratta di una soluzione altamente personalizzata per il network UPS, difficile da replicare altrove sia per i costi che per le specificità organizzative. Inoltre, il sistema rimane in parte statico: il ricalcolo avviene con frequenza limitata e l'adattamento in tempo reale è supervisionato dai dispatcher.

Nel nostro progetto ci ispiriamo ad alcuni principi chiave di ORION, come l'integrazione tra ottimizzazione e monitoraggio real-time e l'attenzione alla robustezza operativa, con l'obiettivo di riproporre in scala ridotta un sistema simile, adatto a un contesto prototipale e universitario.

### 3.5 Piattaforme di ottimizzazione del routing in tempo reale

Accanto ai grandi sistemi proprietari come ORION, il mercato offre oggi numerose piattaforme commerciali di *route optimization* per la logistica last-mile, fornite come soluzioni chiavi in mano da provider tecnologici e software house. Questi sistemi, in genere basati su architetture cloud, integrano l'ottimizzazione dei percorsi con il tracciamento GPS in tempo reale e dashboard centralizzate per la gestione delle consegne. Esempi rappresentativi includono *Onfleet* [5], *Routific*, *Shipsy*, *FarEye* e *Upper Route Planner*, oltre a moduli di ottimizzazione inclusi in suite enterprise più ampie (ad es. Oracle OTM o SAP TM).

Un tratto distintivo di tali piattaforme è la visibilità end-to-end: consentono di monitorare costantemente la posizione dei veicoli, stimare in tempo reale gli orari di arrivo (ETA) e notificare deviazioni o ritardi. L'ottimizzazione non si basa soltanto su dati statici (distanze, capacità, time windows), ma integra anche informazioni dinamiche come traffico, condizioni meteo, disponibilità effettiva dei driver o indicatori di performance. In pratica, il software ricalcola e aggiusta i giri sul campo, garantendo il rispetto delle fasce orarie promesse nonostante perturbazioni operative. Le funzionalità algoritmiche sono generalmente basate su metaeuristiche scalabili (simulated annealing, tabu search, algoritmi genetici) eseguite su infrastrutture cloud. Le piattaforme più avanzate supportano pienamente il paradigma *pickup-delivery con time windows*, vincolando la sequenza di ritiro e consegna, sebbene in alcuni casi vengano semplificate le operazioni distinguendo veicoli dedicati a consegne e veicoli dedicati ai ritiri. Oltre all'aspetto algoritmico, grande attenzione è rivolta all'usabilità: i planner possono intervenire manualmente, aggiungere vincoli o modificare le rotte proposte tramite interfacce grafiche intuitive. Un ulteriore valore aggiunto è la gestione delle anomalie. Attraverso app sugli smartphone dei driver o dispositivi telematici di bordo, le piattaforme ricevono aggiornamenti continui sullo stato delle consegne (completata, fallita, rifiutata, ecc.) e possono reagire di conseguenza. Ad esempio, se un autista resta bloccato nel traffico, il sistema può proporre un rerouting automatico; se un ordine viene cancellato, le consegne in sospeso vengono ridistribuite tra i mezzi disponibili. Inoltre, molte soluzioni inviano notifiche automatiche ai clienti finali, riducendo i tentativi di consegna falliti e migliorando l'esperienza complessiva.

Dal punto di vista critico, queste piattaforme rappresentano lo stato dell'arte commerciale nella gestione distribuita delle consegne, ma rimangono soluzioni generiche e chiuse. In scenari particolari di PDPTW con vincoli non standard (es. compatibilità veicolo-merce, regole su ZTL urbane, vincoli multi-pickup complessi), possono richiedere costose personalizzazioni o non supportare nativamente certe casistiche. Inoltre, essendo servizi SaaS, spesso impongono che i processi aziendali si adattino al software, più che il contrario.

Nel nostro progetto intendiamo prendere ispirazione da tali piattaforme, ma con un approccio più flessibile e personalizzabile: costruendo internamente sia il motore di ottimizzazione sia il sistema di monitoraggio real-time, possiamo modellare vincoli specifici e integrare fonti IoT di dati, replicando su scala ridotta la filosofia di una vera *control tower* last-mile.

## 3.6 Conclusioni

Dall'analisi dello stato dell'arte emergono chiaramente due pilastri fondamentali per affrontare in modo efficace il problema del *Pickup and Delivery con Time Windows* (PDPTW). Da un lato, algoritmi di ottimizzazione avanzati come l'ALNS, le librerie open-source (es. OR-Tools) o i più recenti approcci basati su apprendimento automatico, capaci di produrre soluzioni di elevata qualità per istanze complesse. Dall'altro lato, sistemi di monitoraggio e adattamento in tempo reale, come ORION o le piattaforme last-mile, che consentono di mantenere la robustezza operativa in presenza di imprevisti quali ritardi, guasti o nuove richieste. Gli strumenti attuali eccellono ciascuno in una dimensione specifica: alcuni garantiscono ottimi risultati di ottimizzazione ma sono pensati in ottica statica, altri offrono ricche funzionalità di controllo e visibilità sul campo ma senza algoritmi di routing all'avanguardia. La vera sfida consiste dunque nell'integrazione di questi due mondi, per costruire un sistema unico che unisca qualità algoritmica e adattività operativa. La soluzione proposta nel nostro progetto si pone proprio in questa direzione: sviluppare una piattaforma intelligente che combini un motore di ottimizzazione allo stato dell'arte per il PDPTW con moduli di tracking e ricalcolo dinamico basati su IoT e tecniche di AI. L'obiettivo è colmare i limiti evidenziati nelle soluzioni esistenti e proporre un sistema capace non solo di generare piani efficienti, ma anche di aggiornarli in tempo reale in risposta a condizioni variabili del mondo reale. In questo modo, ci proponiamo di ottenere un prototipo innovativo che rappresenti un passo avanti rispetto alle soluzioni tradizionali: un sistema che coniughi efficienza dei percorsi e resilienza operativa, rispondendo alle esigenze concrete della logistica avanzata.

# Capitolo 4

## Tecnologie utilizzate

In questo capitolo vengono illustrate le tecnologie che hanno avuto un ruolo centrale nello sviluppo del progetto. L'attenzione è posta su come ciascuna tecnologia sia stata concretamente impiegata nella realizzazione della soluzione: dall'ottimizzazione delle rotte al tracciamento dei veicoli, fino alla gestione dei dati e all'interazione con gli utenti finali.

### 4.1 GPS e protocollo MQTT

Il sistema prevede il tracciamento costante dei veicoli in transito. La posizione geografica viene acquisita tramite **GPS**, sfruttando i moduli di localizzazione presenti sugli smartphone. Questa informazione viene poi inviata periodicamente al backend attraverso il protocollo **MQTT** (Message Queuing Telemetry Transport).

La scelta di MQTT non è casuale: rispetto ad alternative come HTTP o WebSocket, garantisce una trasmissione leggera ed efficiente anche in condizioni di rete mobile instabile. In particolare:

- i camionisti pubblicano la loro posizione su un *topic* dedicato;
- i microservizi di backend si sottoscrivono a tali topic e aggiornano lo stato delle consegne in tempo reale;
- in caso di deviazioni o anomalie, il sistema può attivare procedure di ricalcolo immediato.

In questo modo si realizza una comunicazione asincrona e affidabile, adatta a scenari distribuiti e dinamici come quello della logistica.

### 4.2 MongoDB

Per la persistenza dei dati è stato scelto **MongoDB**, un database NoSQL orientato ai documenti. Il suo modello flessibile consente di rappresentare entità eterogenee, come nodi geografici, ordini, rotte e veicoli, senza vincoli rigidi di schema, favorendo l'evoluzione del sistema. Ad esempio:

- un documento **Order** può includere riferimenti a pickup e delivery nodes, stato corrente e vincoli temporali;
- un documento **Vehicle** descrive capacità, disponibilità e stato operativo;
- un documento **Route** conserva l'elenco ordinato dei nodi da visitare, arricchito con informazioni sugli orari stimati.

Questa struttura si adatta bene alla natura variabile delle informazioni logistiche, riducendo la necessità di migrazioni complesse.

### 4.3 Autenticazione JWT

La sicurezza dell'accesso alle API è garantita tramite **JSON Web Token (JWT)**. Ogni utente autenticato (admin o camionista) riceve un token firmato digitalmente che viene allegato alle successive richieste HTTP. Questo approccio presenta diversi vantaggi:

- garantisce un meccanismo stateless, evitando di mantenere sessioni sul server;
- permette di gestire ruoli e permessi in modo granulare (ad esempio distinguendo tra le operazioni disponibili per admin e camionisti);
- semplifica l'integrazione con i microservizi, che possono verificare la validità del token in maniera indipendente.

### 4.4 React Native con Expo e AsyncStorage

L'app mobile per i camionisti è stata realizzata in **React Native**, sfruttando **Expo** per accelerare lo sviluppo e i test. Attraverso l'applicazione, il conducente:

- riceve notifiche push con l'assegnazione di una nuova consegna;
- può accettare l'ordine;
- visualizza il percorso ottimizzato su mappa, aggiornato in tempo reale.

Per la persistenza locale è stato utilizzato **AsyncStorage**, che consente di memorizzare dati anche tra diverse sessioni (ad esempio lo stato di un ordine accettato o l'ultima rotta caricata), migliorando l'affidabilità dell'app anche in caso di interruzioni di rete.

### 4.5 Java Spring Boot e API REST

Il backend gestionale è stato sviluppato con **Spring Boot**, un framework consolidato per la creazione di microservizi. Attraverso le **API REST** esposte, è possibile:

- per l'amministratore: gestire ordini, clienti e veicoli, e lanciare l'ottimizzazione delle rotte, per poi assegnarle;

- per i camionisti: ricevere gli incarichi, aggiornare lo stato della consegna e inviare la posizione in tempo reale.

Spring Boot è stato scelto per la sua modularità, l'integrazione nativa con componenti di sicurezza (JWT, OAuth) e la facilità di deployment in ambienti containerizzati.

## 4.6 Python e OR-Tools

La componente di ottimizzazione delle rotte è stata sviluppata in **Python**, sfruttando la libreria **OR-Tools** di Google. Grazie a OR-Tools è stato possibile modellare varianti complesse del **Capacitated Vehicle Routing Problem con Time Windows e Pickup & Delivery**, includendo:

- vincoli di capacità dei veicoli;
- finestre temporali di servizio;
- precedenze tra pickup e delivery.

Il solver combina euristiche costruttive e metaeuristiche di miglioramento per generare soluzioni di alta qualità in tempi ridotti, rendendo praticabile l'uso in scenari quasi real-time.

## 4.7 Docker e AWS

Tutti i microservizi sono stati containerizzati con **Docker**, garantendo portabilità e riproducibilità dell'ambiente di esecuzione. Il deployment è avvenuto su **Amazon Web Services (AWS)**, sfruttando l'infrastruttura cloud EC2 per garantire:

- scalabilità (possibilità di aumentare le risorse in base al carico);
- affidabilità (replica dei container su più istanze);
- integrazione con servizi gestiti per il database e la messaggistica.

## 4.8 Firebase

Per le notifiche push è stato utilizzato **Firebase Cloud Messaging**. Questo servizio ha consentito di instaurare un canale di comunicazione immediata tra backend e app mobile: ogni volta che un ordine viene assegnato o modificato, il conducente riceve un avviso in tempo reale, migliorando l'efficienza operativa e riducendo i tempi di reazione. Inoltre anche l'admin riceve delle notifiche, come nel caso in cui il camionista segnala un'anomalia o nel caso in cui il camionista accetta una tratta, in modo tale da permettere all'admin di essere aggiornato sulla situazione generale.



## 4.9 GraphHopper

Per la generazione dei percorsi stradali è stato utilizzato **GraphHopper**, un motore open source per il calcolo di itinerari basato su dati **OpenStreetMap**. GraphHopper consente di trasformare una sequenza di nodi ottimizzati dal solver (pickup, consegne, deposito) in rotte reali percorribili su strada, fornendo:

- distanze e tempi di percorrenza più aderenti al contesto reale;
- indicazioni turn-by-turn e segmentazione del percorso in tappe intermedie;

Questa componente è risultata fondamentale per colmare il divario tra il modello astratto di ottimizzazione (basato su nodi e distanze euclidee) e la viabilità effettiva, permettendo di restituire all'utente finale una mappa navigabile e un percorso realistico.

### 4.9.1 Tailscale

Per la gestione delle connessioni tra i dispositivi è stata utilizzata la tecnologia **Tailscale**, una VPN di nuova generazione che sfrutta il protocollo WireGuard per garantire connessioni sicure e veloci. In particolare, Tailscale è stato fondamentale per collegare la macchina EC2 di AWS con il computer locale utilizzato durante lo sviluppo. Grazie a questa connessione diretta, è stato possibile accedere al server **GraphHopper** per il calcolo delle tratte senza dover eseguire l'elaborazione pesante direttamente sulla macchina locale, evitando così sovraccarichi e migliorando l'efficienza del sistema. Tailscale ha quindi permesso di creare una rete privata semplice e sicura, garantendo al contempo stabilità nelle comunicazioni tra i diversi componenti del progetto.

# Capitolo 5

## Soluzione proposta

In questo capitolo viene descritto nel dettaglio il progetto realizzato. Dopo aver introdotto la logica generale che governa il sistema, vengono presentati i principali moduli software e il loro funzionamento, con particolare attenzione agli aspetti implementativi e tecnologici.

### 5.1 Architettura generale

La soluzione segue un'architettura a microservizi, in cui le principali componenti sono:

- **DeliveryService** (Java/Spring Boot): gestisce ordini, nodi, veicoli e amministrazione;
- **PositionService**: gestisce principalmente ciò che riguarda l'utente (login, registrazione, modifica dati...)
- **VehicleRoutingService** (Python/OR-Tools): elabora i dati ricevuti e calcola i percorsi ottimali rispettando capacità, time windows e vincoli di pickup/delivery;
- **App camionista** (React Native): interfaccia mobile per i driver, che ricevono, accettano e seguono le tratte assegnate; raccoglie e gestisce le posizioni GPS dei camionisti tramite MQTT;
- **Modulo notifiche** (Firebase Cloud Messaging): invia notifiche push in tempo reale.

La comunicazione tra i servizi avviene tramite API REST e messaggi MQTT, mentre i dati persistono su un database MongoDB. L'infrastruttura è stata containerizzata con Docker e distribuita su AWS.

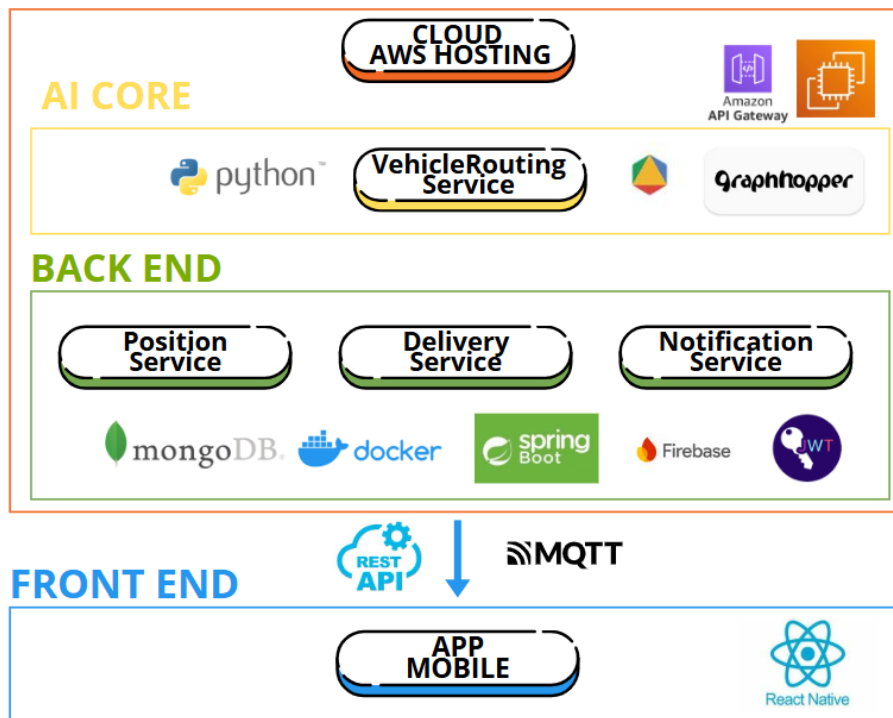


Figura 5.1: Architettura generale applicazione

## 5.2 Funzionamento logico

Il flusso logico principale è riassumibile in questi passi:

1. L'**admin** inserisce nodi (pickup/delivery), veicoli e nuovi ordini tramite il **DeliveryService**.
2. L'ottimizzatore (**VehicleRoutingService**) riceve i dati e calcola le rotte ottimali con **OR-Tools**.
3. Le tratte vengono salvate su **MongoDB** e inviate ai camionisti selezionati.
4. Il camionista riceve una **notifica push** sull'app mobile e può accettare l'ordine.
5. Durante il viaggio, la posizione GPS del camionista viene inviata periodicamente tramite **MQTT** (è stata inoltre introdotta una versione simulata, in cui premendo sulla mappa, è possibile spostarsi per poter testare rimanendo sul posto).
6. In caso di anomalie (guasto, deviazione, nuovo ordine urgente), il sistema può attivare un **ricalcolo** delle rotte.

## 5.3 Attori principali

- **Camionista:** utilizza l'app mobile per ricevere ordini, accettare le tratte assegnate, visualizzare i percorsi ottimizzati e inviare la posizione GPS in tempo reale.
- **Amministratore:** accede al portale gestionale per creare clienti, ordini e veicoli, avviare l'ottimizzazione delle rotte, assegnare i camionisti e monitorare l'avanzamento delle consegne.
- **Sistema:** include i microservizi backend (Spring Boot, MongoDB, MQTT, Firebase) e il motore di ottimizzazione (Python + OR-Tools).

## 5.4 Elenco dei Casi d'Uso

### UC1 – Login

**Attori principali:** Camionista, Admin

**Precondizioni:** L'utente è registrato e possiede credenziali valide.

**Postcondizioni:** L'utente accede al sistema con token JWT.

**Descrizione:**

1. L'utente inserisce email e password.
2. Il sistema autentica l'utente e genera un token JWT.
3. Il token viene salvato e usato per le successive richieste.
4. L'utente accede rispettivamente all'app mobile (camionista) o alla dashboard (admin).

**Estensioni:**

- 1 Le credenziali inserite non sono valide.

### UC2 – Registrazione

**Attori principali:** Camionista

**Precondizioni:** Il camionista non è ancora registrato.

**Postcondizioni:** Viene creato un nuovo camionista con stato iniziale *AVAILABLE*.

**Descrizione:**

1. Il camionista inserisce i dati nel portale.
2. Il sistema registra il nuovo utente e lo salva nel database.
3. Se tutto è valido, viene creato un nuovo camionista nel database e viene mandata un'email di benvenuto.

4. Il camionista può ora effettuare l'accesso con le sue credenziali

**Estensioni:**

1. L'email inserita è già registrata.
2. Dati mancanti o non validi.

## UC3 – Recupero Password

**Attori principali:** Camionista

**Precondizioni:** Il camionista ha un account registrato ma ha dimenticato la password.

**Postcondizioni:** Il camionista ha impostato una nuova password ed è in grado di accedere al sistema.

**Descrizione:**

1. Il camionista seleziona l'opzione "Hai dimenticato la password?" dalla schermata di login.
2. Inserisce l'indirizzo email associato all'account.
3. Il sistema invia un codice da inserire per poter cambiare la password tramite indirizzo email.
4. Il camionista inserisce il codice e la nuova password.
5. Il sistema aggiorna la password nel database e notifica l'avvenuto cambio.

**Estensioni:**

1. L'email inserita non è associata a nessun account.
2. Il token non è valido.

## UC4 - Modifica Profilo

**Attori principali:** Camionista

**Precondizioni:** Il camionista ha già loggato con il suo account.

**Postcondizioni:** Il camionista ha modificato i dati interessati.

**Descrizione:**

1. Il camionista, dopo aver selezionato l'opzione "Modifica profilo", modifica i dati personali e preme "Salva modifiche".
2. Il sistema aggiorna il profilo del camionista.

**Estensioni:**

1. Se campi vuoti o errati, restituisce errore.

## UC5 - Modifica Password

**Attori principali:** Camionista

**Precondizioni:** Il camionista ha già loggato con il suo account.

**Postcondizioni:** Il camionista ha modificato la password.

**Descrizione:**

1. Il camionista, dopo aver selezionato l'opzione "Cambia password", inserisce email, vecchia password e nuova password.
2. Il sistema aggiorna la password dell'utente dell'utente.

**Estensioni:**

1. L'email o la password non corrispondono.

## UC6 - Elimina account

**Attori principali:** Camionista

**Precondizioni:** Il camionista ha già loggato con il suo account.

**Postcondizioni:** Il camionista ha eliminato il suo account

**Descrizione:**

1. Il camionista seleziona l'opzione "Elimina account".
2. Il sistema chiede se l'utente è sicuro di eliminare l'account, ed in caso affermativo lo elimina.

## UC7 – Creazione di un ordine

**Attori principali:** Admin

**Precondizioni:** Esistono nodi (punti di ritiro e consegna) registrati.

**Postcondizioni:** Un nuovo ordine è salvato a sistema.

**Descrizione:**

1. L'admin compila i campi dell'ordine: nodo di pickup, nodo di delivery, quantità e finestre temporali.
2. Il sistema valida i dati e registra l'ordine.

## UC8 – Ottimizzazione delle rotte

**Attori principali:** Admin, Sistema

**Precondizioni:** Sono presenti ordini non assegnati, veicoli disponibili e camionisti disponibili.

**Postcondizioni:** Il sistema produce un set di rotte ottimizzate.

**Descrizione:**

1. L'admin avvia l'ottimizzazione dal portale.
2. Il backend invia a Python-OR Tools la lista di ordini, veicoli e nodi.
3. OR-Tools calcola i percorsi e restituisce una soluzione con vincoli rispettati.
4. Le rotte vengono salvate nel database.

**Estensioni:**

1. Non ci sono camionisti disponibili.
2. Non ci sono veicoli disponibili

## UC9 – Assegnazione di una rotta

**Attori principali:** Admin, Camionista

**Precondizioni:** Esiste una rotta ottimizzata, e dei camionisti disponibili.

**Postcondizioni:** La rotta viene assegnata a un camionista disponibile.

**Descrizione:**

1. L'admin seleziona uno o più camionisti tra indisponibili.
2. Il sistema aggiorna il veicolo e collega l'utente alla rotta.
3. Al camionista viene inviata una notifica push con i dettagli della tratta.

## UC10 – Accettazione della tratta

**Attori principali:** Camionista

**Precondizioni:** Una rotta è stata assegnata al camionista.

**Postcondizioni:** Lo stato dell'ordine passa a *IN PROGRESS*.

**Descrizione:**

1. Il camionista apre l'app e visualizza la rotta.
2. Conferma l'accettazione.
3. Il sistema aggiorna lo stato della rotta e del veicolo a *IN TRANSIT*.

## UC11 – Tracking della posizione

**Attori principali:** Camionista, Sistema

**Precondizioni:** La tratta è in corso.

**Postcondizioni:** Le posizioni vengono registrate in tempo reale.

**Descrizione:**

1. L'app invia la posizione GPS del camionista via protocollo MQTT.
2. Il sistema aggiorna il database e mette a disposizione i dati di tracking.

## UC12 – Gestione del cambio di rotta

**Attori principali:** Camionista, Sistema

**Precondizioni:** La rotta è in corso.

**Postcondizioni:** Viene effettuato un ricalcolo.

**Descrizione:**

1. Si verifica un imprevisto (deviazione significativa).
2. L'app o il sistema rilevano la deviazione.
3. Il sistema genera una nuova rotta.

## UC13 – Completamento della tratta

**Attori principali:** Camionista, Sistema

**Precondizioni:** Tutti i nodi della rotta sono stati serviti.

**Postcondizioni:** Ordini e rotte vengono aggiornati come *COMPLETED*, camionista e veicoli tornano *AVAILABLE*.

**Descrizione:**

1. Il camionista segnala la conclusione della tratta.
2. Il backend aggiorna lo stato degli ordini e del veicolo.
3. L'admin può visualizzare lo storico della rotta completata.

## UC14 – Gestione della anomalia

**Attori principali:** Camionista

**Precondizioni:** La rotta è in corso.

**Postcondizioni:** Viene assegnata la rotta ad un altro camionista.

**Descrizione:**

1. Si verifica un imprevisto (il camionista non può procedere più con la sua rotta a causa di un guasto per esempio) e lo segnala con apposito pulsante.
2. Il sistema riceve la segnalazione ed effettua il ricalcolo ottimizzato degli ordini "attivi" del veicolo guasto
3. Il sistema assegna la tratta da continuare ad un camionista disponibile o in transito.



## 5.5 Diagrammi di flusso principali

### Come avviene la registrazione del camionista?

Il processo di registrazione inizia con l'interazione dell'utente che, attraverso l'applicazione mobile, richiede l'accesso alla schermata di registrazione. L'app restituisce il modulo in cui l'utente inserisce i dati richiesti (nome, cognome, email, password, numero di telefono e genere, se desiderato).

Dopo una prima validazione locale, l'app invia i dati al 'PositionService', che verifica l'univocità dell'email e la correttezza dei campi ricevuti. Se la registrazione è valida, il sistema crea un nuovo utente nel database e invia una risposta positiva all'applicazione, che reindirizza l'utente alla schermata di login.

Inoltre, viene generato un messaggio di notifica automatico per l'invio dell'email di conferma all'indirizzo indicato in fase di registrazione.

Questa email rappresenta una conferma non interattiva dell'avvenuta creazione dell'account.

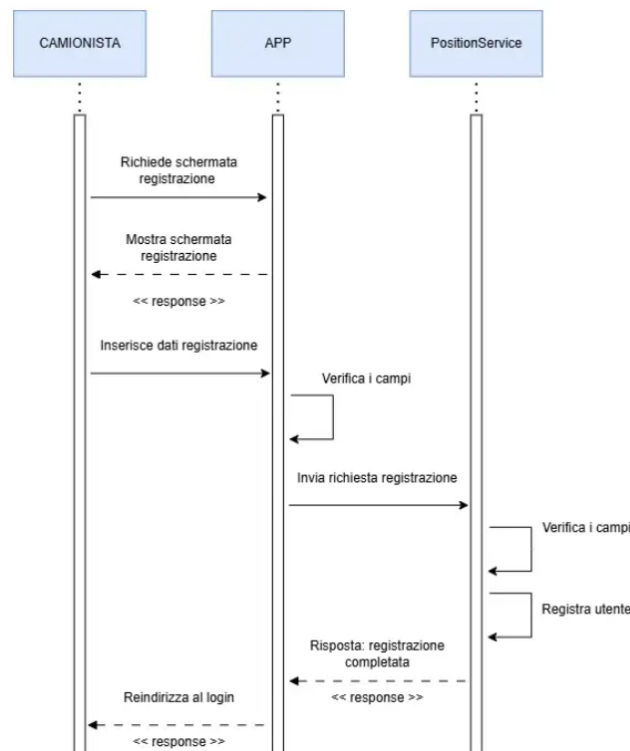


Figura 5.2: Diagramma di flusso della registrazione

## Come viene creata ed assegnata una tratta?

L'Admin seleziona gli ordini e clicca il pulsante per ottimizzare la rotta. Il `DeliveryService` prepara i dati (nodi, distanze, capacità, time windows) e invoca il motore Python con `OR-Tools`. La soluzione restituita contiene la rotta migliore calcolata (sequenze ordinate di fermate per veicolo) che rispettano precedenza e capacità. L'admin seleziona uno o più camionisti tra quelli con lo stato *AVAILABLE*. Alla conferma, la rotta viene *assegnata* a un camionista e l'app riceve una notifica push (Firebase) con i dettagli.

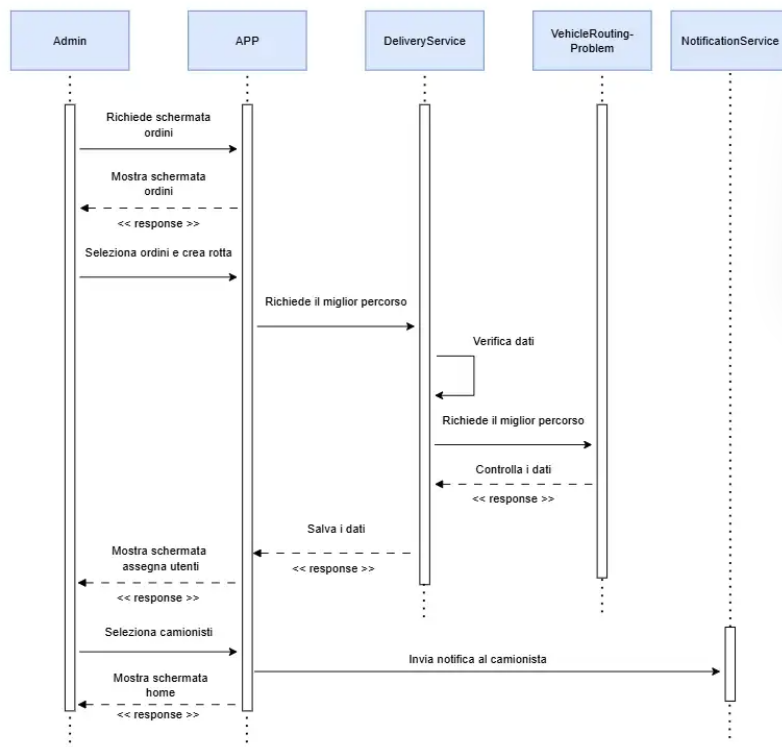


Figura 5.3: Diagramma di flusso di creazione e assegnazione tratta

## Ricalcolo tratta per deviazione di rotta

Quando il camionista esce dal percorso previsto, l'app continua a campionare la posizione GPS e scambia i dati tramite MQTT; sulla base di questi aggiornamenti l'app verifica lo scostamento rispetto alla rotta salvata e, se la deviazione supera la soglia, invia al DeliveryService una richiesta di ricalcolo. Il servizio costruisce la nuova richiesta a GraphHopper a partire dalla posizione corrente e dalle tappe ancora da completare, ottiene un itinerario aggiornato e risponde all'app con il nuovo tracciato. L'app sostituisce la rotta precedente con quella ricalcolata e mostra immediatamente al camionista il percorso aggiornato, permettendo di proseguire senza interruzioni verso le consegne residue.

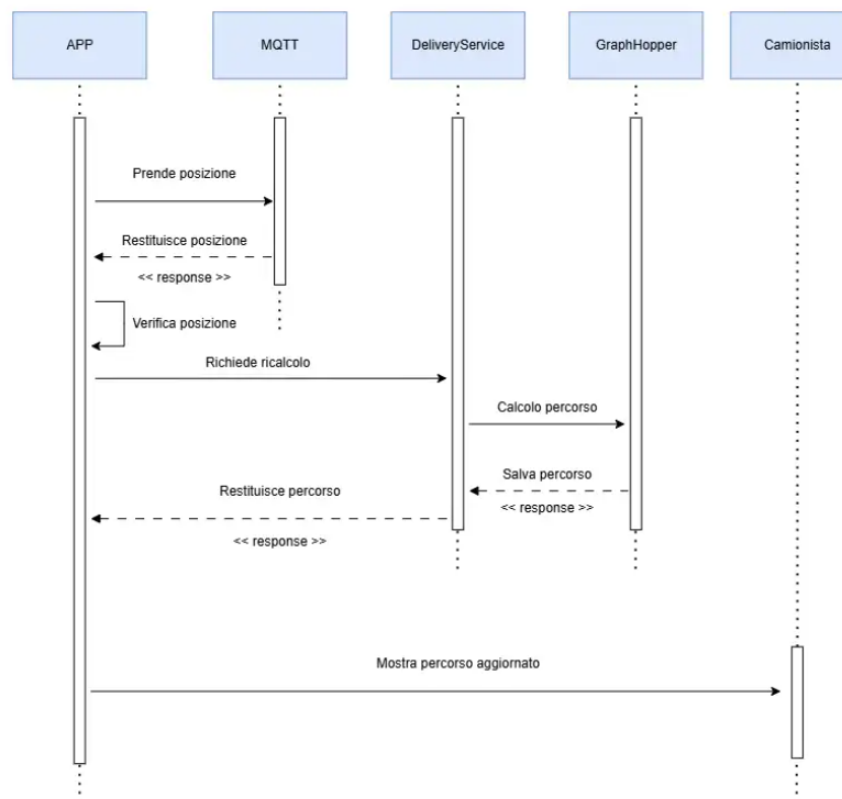


Figura 5.4: Diagramma di flusso del ricalcolo tratta per deviazione di rotta

# Capitolo 6

## Validazione funzionale e/o prestazionale

In questo capitolo vengono presentate le attività di validazione funzionale e prestazionale condotte sull'applicazione sviluppata. L'obiettivo è verificare che il sistema soddisfi i requisiti progettuali sia in termini di corretto funzionamento delle funzionalità implementate, sia rispetto alle prestazioni attese in scenari realistici.

### 6.1 Interfaccia iniziale

All'avvio dell'applicazione sia l'amministratore che il camionista hanno la possibilità di accedere tramite le proprie credenziali. Come si vede nella Figura 6.1, l'interfaccia di login è semplice e permette l'autenticazione immediata da parte di entrambi i ruoli.

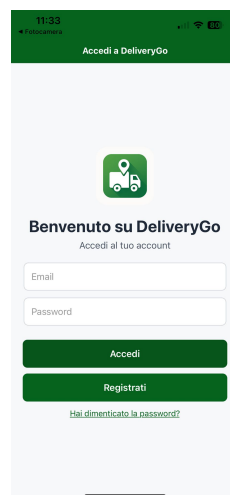


Figura 6.1: Interfaccia login

Nel caso in cui l'utente non disponga ancora di un account, può procedere con la registrazione inserendo i propri dati personali, come mostrato in Figura 6.2. Il sistema verifica la validità delle informazioni e, in caso di esito positivo, abilita l'accesso al profilo ed invia un'email di conferma.

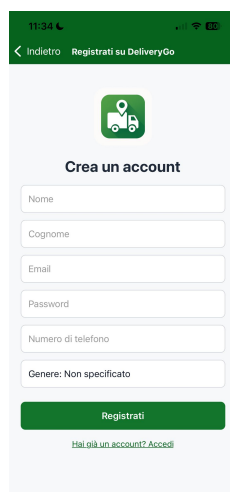


Figura 6.2: Interfaccia registrazione

È stata inoltre implementata la funzionalità di recupero delle credenziali: se un utente dimentica la password, può richiedere un reset tramite l'apposito link. Viene inviata automaticamente una email con un token di sicurezza che consente di impostare una nuova password, come si osserva in Figura 6.3.

In questo modo si garantisce un processo di autenticazione sicuro e completo, che copre le esigenze di amministratori e camionisti, mantenendo allo stesso tempo una buona esperienza utente.

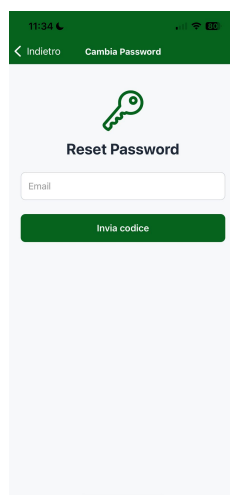


Figura 6.3: Interfaccia per il reset della password

## 6.2 Caso d'uso reale: Assegnazione di una tratta da parte dell'amministratore

Una delle funzionalità centrali dell'applicazione riguarda la possibilità per l'amministratore di assegnare una tratta ottimizzata a un camionista disponibile. Come si vede nelle schermate riportate in Figura 6.4, il flusso si articola in più fasi. Dopo aver effettuato l'accesso, l'admin può visualizzare i veicoli registrati nel sistema, insieme al loro stato corrente. Una volta creati gli ordini e lanciato il processo di ottimizzazione, il sistema restituisce una o più rotte calcolate tramite OR-Tools. Queste rotte rimangono in attesa fino a quando non vengono assegnate a un conducente. L'amministratore seleziona quindi una rotta e accede alla schermata di dettaglio. Qui sono visibili la targa del veicolo assegnato, l'elenco ordinato delle tappe da raggiungere (punti di pickup e consegna) e le relative informazioni logistiche (indirizzi e orari stimati). A questo punto viene mostrata la lista dei camionisti registrati nel sistema: vengono mostrati unicamente quelli disponibili (*AVAILABLE*). Una volta selezionato il camionista desiderato, l'admin conferma l'operazione con il pulsante *Assegna ordine*.

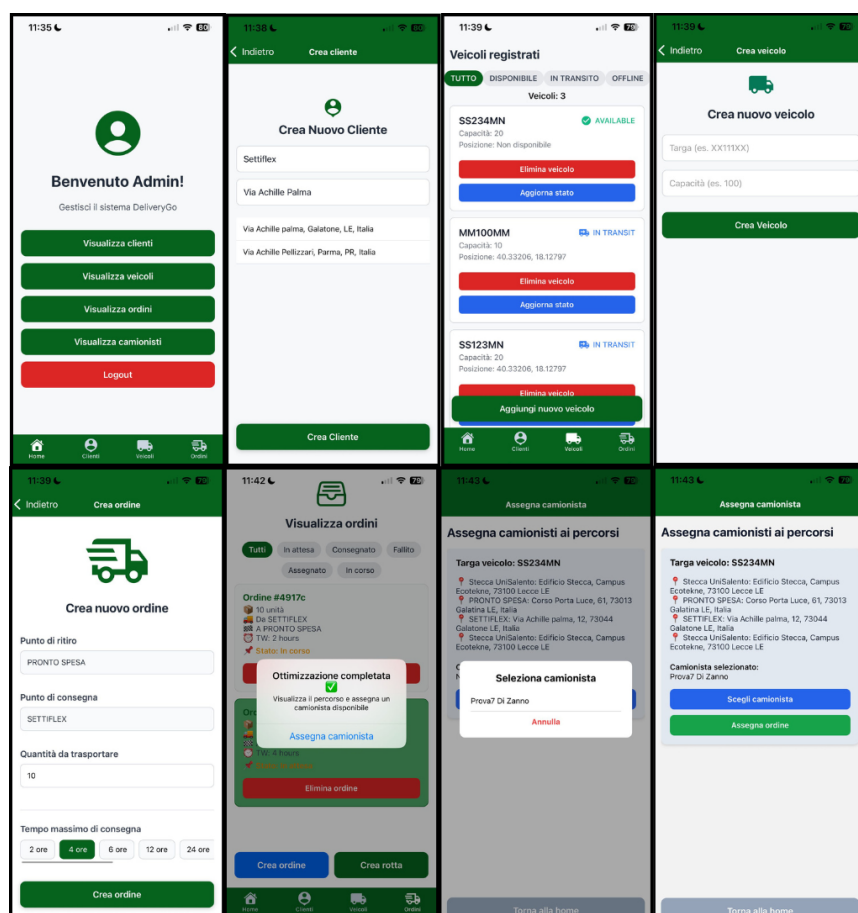


Figura 6.4: Sequenza di schermate per l'assegnazione di una tratta da parte dell'amministratore

Il sistema aggiorna lo stato degli ordini in *ASSIGNED*, registra l'associazione tra veicolo e conducente e invia automaticamente una notifica push all'app mobile del camionista, come si vede in Figura 6.5 tramite **Firestore Cloud Messaging**. In questo modo il conducente riceve immediatamente i dettagli della consegna da effettuare e può avviare l'esecuzione del percorso.

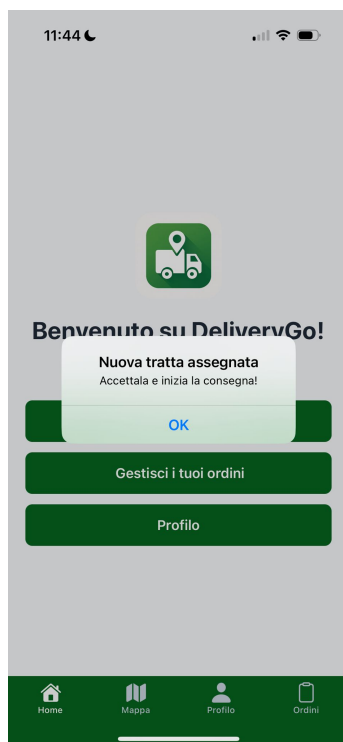


Figura 6.5: Interfaccia del camionista a cui è stata assegnata la consegna

### 6.3 Accettazione e completamento della tratta da parte del camionista

Dopo il login, il camionista accede alla schermata principale dell'app e può consultare gli *ordini assegnati*. Come si vede in Figura 6.6, l'utente visualizza il riepilogo della rotta (targa del veicolo e sequenza delle fermate) e può procedere con il pulsante *Accetta tratta*. All'accettazione, lo stato degli ordini passa a *IN\_PROGRESS* e viene mostrata la mappa con il percorso attivo. L'interfaccia guida il conducente tappa per tappa: al raggiungimento della prossimità della fermata corrente compare un avviso contestuale (es. *Carico/Consegna: nome tappa*) e viene reso disponibile il comando *Avanza alla prossima tappa*, che consente di proseguire lungo la sequenza pianificata.

La posizione viene inviata periodicamente al backend tramite MQTT, consentendo l'aggiornamento in tempo reale della rotta e l'eventuale ricalcolo in caso di deviazioni. Al completamento dell'ultima tappa, l'ordine viene marcato come *DELIVERED* e lo storico viene aggiornato, rendendo immediatamente visibile l'esito sia all'utente sia al pannello admin.

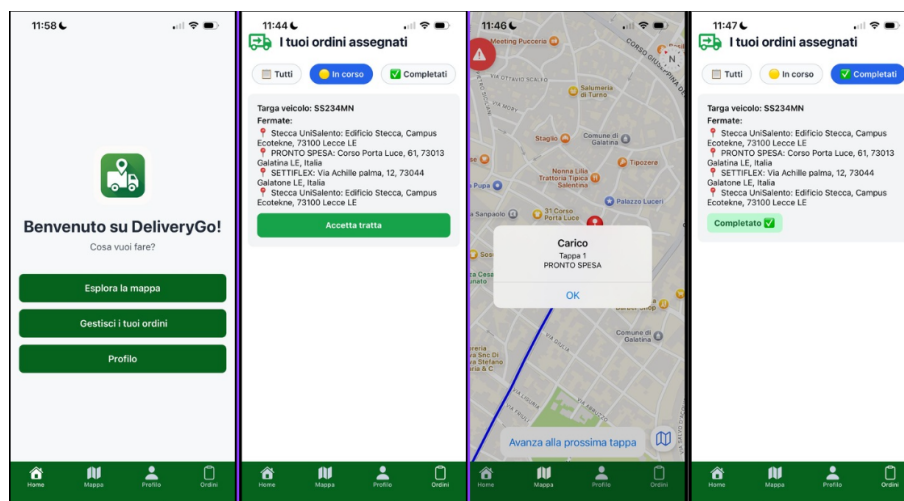


Figura 6.6: Flusso lato camionista: accettazione della tratta, navigazione a tappe e completamento. Per chiarezza, nello screenshot è mostrato solo un *singolo passo* della consegna.



Non potendo effettuare spostamenti fisici durante la fase di test, oltre la posizione presa dal GPS del nostro dispositivo, è stata introdotta una modalità di posizione simulata, come si vede in Figura 6.7: toccando lo schermo della mappa è possibile spostare virtualmente il veicolo, avanzare lungo la tratta e completare le consegne.

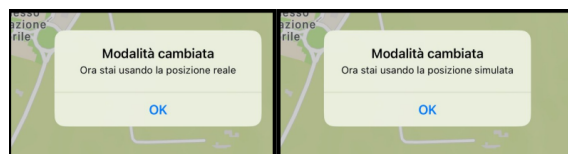


Figura 6.7: Switch delle modalità

### 6.3.1 Segnalazione di un'anomalia

In caso di imprevisti durante il viaggio, come ad esempio un guasto del veicolo o un altro tipo di anomalia operativa, il camionista ha la possibilità di segnalare l'incidente premendo il pulsante di allerta, identificato da un triangolo rosso, presente nell'interfaccia dell'applicazione. Questo pulsante è visibile in tutte le schermate relative alla gestione del percorso e alla navigazione del camionista. Un esempio di tale interfaccia è mostrato nella Figura 6.8.



Figura 6.8: Interfaccia di segnalazione anomalia

Una volta che il camionista conferma la segnalazione dell'anomalia, il sistema interrompe il processo di consegna corrente e procede con la riassegnazione dell'ordine in modo automatico. L'algoritmo di riassegnazione tiene conto di vari fattori operativi, tra cui i vincoli di capacità del veicolo e le finestre temporali per ogni

ordine, per garantire che la nuova assegnazione sia ottimale. La riassegnazione può avvenire in due modalità principali:

- **Ordini già *PICKED-UP***: se il veicolo guasto aveva già caricato merce, il sistema seleziona uno o più veicoli in transito o disponibili, valutando la loro vicinanza al punto di guasto e la capacità residua. In questo caso, gli ordini vengono trasferiti al veicolo più adatto, in modo da minimizzare i ritardi e garantire la continuità delle operazioni. Il sistema assicura che i nuovi veicoli assegnati siano in grado di completare il percorso rispettando i vincoli di capacità e finestra temporale.
- **Ordini *PENDING***: se il guasto si verifica prima che il camion abbia caricato la merce, gli ordini vengono immediatamente riassegnati a veicoli disponibili nel deposito. In questo caso, l'algoritmo sceglie un veicolo disponibile che soddisfi i requisiti di capacità e finestre temporali, garantendo che il nuovo camion parta rapidamente per il ritiro delle merci e continui il percorso senza intoppi.

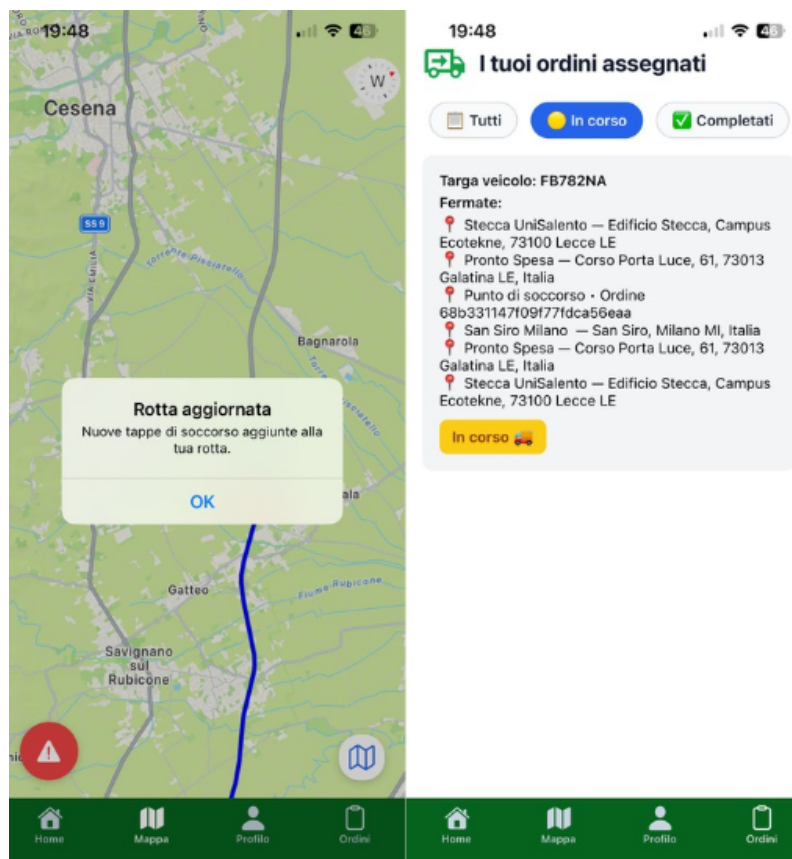


Figura 6.9: Interfaccia dell'utente che ha ricevuto la tratta dovuta all'anomalia

Questo meccanismo permette di gestire in modo dinamico le anomalie, riducendo al minimo i ritardi nelle consegne.

## 6.4 Hardware e strumenti utilizzati

La validazione è stata condotta utilizzando i seguenti dispositivi e strumenti:

- **Smartphone Iphone** (con GPS attivo) utilizzato dal camionista per l'app React Native;
- **Computer** per eseguire i microservizi Java Spring Boot e il motore Python di ottimizzazione durante i test in locale;
- **Server cloud AWS** per il deployment containerizzato con Docker;
- **Broker MQTT Mosquitto** per la gestione della comunicazione in tempo reale;
- **MongoDB Atlas** per la persistenza dei dati.

## 6.5 Validazione funzionale

Dal punto di vista funzionale, l'applicazione è stata sottoposta a una serie di test mirati a verificare che tutte le funzionalità principali fossero correttamente implementate e integrate tra loro. In particolare, sono state verificate le seguenti caratteristiche:

- **Calcolo e visualizzazione delle rotte ottimizzate:** a seguito della selezione di un insieme di ordini da parte dell'amministratore, il sistema ha generato soluzioni di routing rispettando i vincoli di capacità e le finestre temporali. Le rotte sono state correttamente mostrate sia nel pannello admin sia sull'app mobile, con una rappresentazione grafica chiara e coerente.
- **Gestione delle notifiche push:** i camionisti hanno ricevuto notifiche immediate tramite Firebase al momento dell'assegnazione di una nuova tratta. Questo ha permesso di ridurre i tempi di latenza tra la decisione dell'admin e la ricezione delle informazioni da parte del conducente.
- **Aggiornamento in tempo reale della posizione:** durante la simulazione di una consegna, la posizione del veicolo è stata costantemente inviata tramite MQTT e aggiornata nella mappa visualizzata dal backend e dall'app mobile. I marker dei veicoli hanno seguito correttamente gli spostamenti, con refresh costanti e senza perdita di pacchetti significativa.
- **Ricalcolo della rotta in caso di deviazione:** quando il veicolo ha abbandonato il percorso previsto, il sistema ha rilevato la deviazione e ha richiesto a GraphHopper il calcolo di una nuova rotta dal punto corrente verso le destinazioni rimanenti. La nuova tratta è stata restituita in tempi brevi e visualizzata correttamente sul dispositivo del camionista.

Questi test hanno confermato che l'applicazione soddisfa pienamente i requisiti funzionali, consentendo un'interazione fluida e affidabile tra amministratore e camionisti.

## 6.6 Validazione prestazionale

Dal punto di vista prestazionale, sono stati condotti test volti a misurare la reattività e l'efficienza del sistema in condizioni realistiche di utilizzo. Gli aspetti principali analizzati sono stati i seguenti:

- **Tempi di ottimizzazione:** utilizzando OR-Tools, la generazione di soluzioni è avvenuta mediamente in pochi secondi. Ciò dimostra la capacità del sistema di operare in scenari quasi real-time, senza compromettere la qualità della soluzione.
- **Latenza di aggiornamento:** l'invio della posizione tramite MQTT e la successiva propagazione verso il backend e l'app mobile hanno garantito una latenza media inferiore al secondo. Questo valore risulta adeguato per un sistema di tracciamento logistico, in cui l'aggiornamento deve essere percepito come immediato dall'utente finale.
- **Scalabilità:** grazie alla containerizzazione con Docker e al deployment su AWS, il sistema ha mostrato la capacità di scalare orizzontalmente eseguendo più istanze dei microservizi in parallelo. I test hanno evidenziato come l'architettura a microservizi permetta di distribuire il carico tra diversi nodi, garantendo la continuità del servizio anche in presenza di un numero crescente di utenti e consegne.
- **Resilienza:** in caso di interruzioni temporanee della connessione di rete, l'app mobile ha mantenuto lo stato degli ordini grazie a *AsyncStorage*, ripristinando la sincronizzazione non appena la connessione è tornata disponibile. Questo comportamento ha migliorato l'affidabilità complessiva del sistema.

Nel complesso, i risultati ottenuti confermano che l'applicazione è in grado di garantire prestazioni adeguate a scenari di utilizzo realistici, offrendo tempi di risposta contenuti, aggiornamenti costanti e capacità di scalare in funzione del carico.

## 6.7 Conclusioni

La validazione ha dimostrato che l'applicazione è in grado di soddisfare i requisiti sia dal punto di vista funzionale sia prestazionale. L'integrazione tra GPS, MQTT, OR-Tools e GraphHopper consente di gestire in maniera efficace scenari complessi di logistica, garantendo al contempo tempi di risposta adeguati e una buona esperienza d'uso per amministratori e camionisti.

# Capitolo 7

## Conclusioni e sviluppi futuri

Il lavoro presentato ha avuto come obiettivo la realizzazione di un sistema intelligente per la gestione e l'ottimizzazione delle consegne nel contesto della logistica avanzata. L'architettura sviluppata ha integrato diverse tecnologie — tra cui **GPS**, **MQTT**, **GraphHopper**, **OR-Tools**, **MongoDB**, **Spring Boot** e **React Native** — al fine di coprire l'intero ciclo operativo: dalla creazione e ottimizzazione delle rotte, fino al tracciamento in tempo reale dei veicoli e alla gestione delle anomalie.

Gli obiettivi prefissati sono stati pienamente raggiunti:

- l'**admin** è in grado di creare ordini, clienti e veicoli, avviare l'ottimizzazione delle tratte e assegnarle ai camionisti;
- i **camionisti** ricevono notifiche push, accettano le consegne e seguono il percorso ottimizzato tramite l'app mobile;
- il sistema consente il monitoraggio in tempo reale e un ricalcolo dinamico in caso di deviazioni o anomalie.

## Sviluppi futuri

Nonostante i risultati conseguiti, il sistema presenta ampi margini di miglioramento. Possibili sviluppi futuri includono:

- **Gestione dettagliata degli ordini:** attualmente il sistema considera la quantità come semplice unità (pallet/pedane). In futuro si potrebbe estendere il modello per monitorare i singoli prodotti o insiemi di prodotti, anche tramite tecnologie IoT (es. RFID, sensori di peso, tag NFC).
- **Ottimizzazione in base alla tipologia dei carichi:** integrare l'algoritmo con vincoli specifici legati alla natura delle merci (alimentari, fragili, pericolose, refrigerate), prevedendo l'assegnazione dei veicoli più adatti alle diverse tipologie.
- **Uso di dati real-time:** includere informazioni dinamiche come traffico, incidenti, condizioni meteo o lavori stradali, per adattare le rotte in tempo reale, migliorando precisione ed efficienza del calcolo.

- **Navigazione avanzata:** arricchire l'app con indicazioni turn-by-turn direttamente sul percorso calcolato, sfruttando le istruzioni dettagliate restituite da GraphHopper, in modo simile a Google Maps.
- **Gestione avanzata delle anomalie:** attualmente la riassegnazione viene gestita in modo semplificato (veicoli AVAILABLE o vicini IN TRANSIT). In futuro si potrebbe implementare un ricalcolo completo e dinamico di tutti gli ordini attivi, sfruttando algoritmi Python più sofisticati e combinando l'elaborazione con il backend Java.
- **Machine Learning e predizione dei tempi di consegna:** utilizzare modelli predittivi per stimare i tempi di arrivo (ETA) sulla base di dati storici, condizioni del traffico e performance passate, così da aumentare l'affidabilità delle stime.
- **Business Intelligence e reportistica:** sviluppare dashboard avanzate per fornire all'azienda indicatori chiave (KPI) come puntualità, utilizzo della flotta, costi operativi e percentuale di anomalie gestite.
- **Scalabilità e multi-azienda:** estendere il sistema per supportare più aziende o scenari multi-deposito, garantendo prestazioni elevate anche su larga scala.
- **Piattaforma di monitoraggio per le aziende:** si potrebbe creare una piattaforma dedicata al monitoraggio in tempo reale per le aziende. Questa interfaccia permetterebbe agli utenti aziendali di avere una visione completa dello stato degli ordini e delle consegne, con funzionalità per monitorare le performance in tempo reale, identificare eventuali ritardi o problemi e avere accesso a report dettagliati sulle operazioni di logistica. Questa interfaccia dedicata renderebbe il sistema più fruibile per le aziende, permettendo loro di ottimizzare i processi e prendere decisioni più informate in tempo reale.

In conclusione, il progetto ha dimostrato la fattibilità di un approccio basato su microservizi e algoritmi di ottimizzazione, ponendo solide basi per estensioni future. L'integrazione di tecniche IoT avanzate, dati real-time e modelli di Intelligenza Artificiale potrà trasformare questo prototipo in una piattaforma completa, capace di rispondere a scenari reali di logistica avanzata con efficienza, affidabilità e adattabilità.

# Capitolo 8

## Link utili

Qui sono riportati i link ai repository GitHub sviluppati per il progetto.

- **GitHub Page:** <https://unisalento-idalab-iotcourse-2024-2025.github.io/wot-project-presentation-MandorinoSettimo/>
- **PositionService:** <https://github.com/UniSalento-IDALab-IoTCourse-2024-2025/wot-project-PositionService-MandorinoSettimo.git>
- **DeliveryService:** <https://github.com/UniSalento-IDALab-IoTCourse-2024-2025/wot-project-DeliveryService-MandorinoSettimo.git>
- **NotificationService:** <https://github.com/UniSalento-IDALab-IoTCourse-2024-2025/wot-project-NotificationService-MandorinoSettimo.git>
- **VehicleRouting:** <https://github.com/UniSalento-IDALab-IoTCourse-2024-2025/wot-project-VehicleRoutingProblem-MandorinoSettimo.git>
- **Frontend:** <https://github.com/UniSalento-IDALab-IoTCourse-2024-2025/wot-project-Frontend-MandorinoSettimo.git>
- **Presentation:** <https://github.com/UniSalento-IDALab-IoTCourse-2024-2025/wot-project-presentation-MandorinoSettimo.git>
- **Video:** <https://drive.google.com/file/d/1wd0aGBeAj9x9s1P9QRUQh0hPAP3QtTT3/preview>

# Elenco delle figure

5.1	Architettura generale applicazione . . . . .	16
5.2	Diagramma di flusso della registrazione . . . . .	22
5.3	Diagramma di flusso di creazione e assegnazione tratta . . . . .	23
5.4	Diagramma di flusso del ricalcolo tratta per deviazione di rotta . . . . .	24
6.1	Interfaccia login . . . . .	25
6.2	Interfaccia registrazione . . . . .	26
6.3	Interfaccia per il reset della password . . . . .	26
6.4	Sequenza di schermate per l'assegnazione di una tratta da parte dell'amministratore . . . . .	27
6.5	Interfaccia del camionista a cui è stata assegnata la consegna . . . . .	28
6.6	Flusso lato camionista: accettazione della tratta, navigazione a tappe e completamento. Per chiarezza, nello screenshot è mostrato solo un <i>singolo passo</i> della consegna. . . . .	29
6.7	Switch delle modalità . . . . .	30
6.8	Interfaccia di segnalazione anomalia . . . . .	30
6.9	Interfaccia dell'utente che ha ricevuto la tratta dovuta all'anomalia . . . . .	31



# Bibliografia

- [1] S. Ropke and D. Pisinger, “An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows,” *Transportation Science*, vol. 40, no. 4, pp. 455–472, 2006.
- [2] R. Li, W. Song, Z. Cao, J. Zhang, and P. S. Tan, “Learning to solve pickup and delivery problems via heterogeneous attention,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 8, pp. 7660–7668, 2021.
- [3] Google, “Or-tools: Google’s open source optimization suite.” <https://developers.google.com/optimization>, 2024. Accessed: 2025-08-17.
- [4] J. Levis and U. O. R. Team, “Orion: On-road integrated optimization and navigation,” *Interfaces*, vol. 46, no. 1, pp. 4–21, 2016. Winner of the Franz Edelman Award.
- [5] O. Inc., “Onfleet: Last-mile delivery software.” <https://onfleet.com/>, 2024. Accessed: 2025-08-17.