Rahguzar

Kaavish Report
presented to the academic faculty
by

Nabila Zahra	nz07162
Rabia Shahab	rs07528
Iqra Azfar	ia07614
Muhammad Youshay	my07103

In partial fulfillment of the requirements for
Bachelor of Science
Computer Science

Dhanani School of Science and Engineering

Habib University

Spring 2025

Rahguzar

This Kaavish project was supervised by:

Fatima Alvi Management Trainee Salesflo Pvt Ltd	Dr. Syeda Saleha Raza Faculty of Computer Science Habib University

Approved by the Faculty of Computer Science on _____.

Dedication

To our parents, whose unwavering love, endless sacrifices, and heartfelt prayers have been the foundation of everything we've accomplished. Your belief in us, even when we doubted ourselves, gave us the strength to move forward. This journey, and all that it has brought us, is a reflection of your dedication and support.

To our families and loved ones, thank you for your patience, encouragement, and constant presence. Your understanding made room for our long hours, tight deadlines, and moments of exhaustion.

To our teachers and mentors, we are deeply grateful for your guidance and the knowledge you've so generously shared. Your trust in our capabilities challenged us to think bigger and work harder.

To our teammates, thank you for the long nights, the shared goals, and the spirit of collaboration that brought this project to life. Rahguzar is a product of shared vision, hard work, and mutual respect.

Above all, we dedicate this work to Allah (SWT), the source of all knowledge and strength. Without His mercy and guidance, none of this would have been possible.

Acknowledgements

Alhamdulillah, all praise is due to Allah (SWT), whose endless mercy, guidance, and blessings gave us the strength and clarity to complete this project.

We are sincerely grateful to Dr. Syeda Saleha Raza, from the Faculty of Computer Science at Habib University, for her academic mentorship, thoughtful guidance, and unwavering support throughout the course of this project. Her insights and encouragement consistently pushed us to approach our work with both depth and discipline.

We also extend our heartfelt thanks to Ms. Fatima Alvi, Management Trainee at SalesFlo Pvt Ltd, for her practical guidance, domain expertise, and continuous availability. Her support helped us navigate the real-world complexities of the problem we set out to solve.

We are thankful to the Computer Science faculty at Habib University for equipping us with the knowledge and skills that formed the backbone of this project. Every course, discussion, and challenge over the past four years played a role in preparing us for this journey.

We deeply appreciate the collaboration with SalesFlo Pvt Ltd, our industry partner, for providing access to relevant datasets, business context, and timely feedback. Your involvement helped us ground our work in real operational needs.

Our gratitude also goes to the Kaavish Committee and the Office of Undergraduate Research for enabling a structured, well-supported project experience and for providing necessary resources and checkpoints along the way.

To our families and friends, thank you for your constant encouragement, patience, and understanding through all the late nights, deadlines, and moments of stress. Your support made all the difference.

Finally, we acknowledge one another. As teammates, we shared a vision, faced challenges, and built something we are proud of. Rahguzar is a result of that collective effort and commitment.

Abstract

Abstract goes here

Contents

List of Figures

List of Tables

1. Introduction

1.1 Problem Statement

In the dynamic and ever-evolving world of retail and distribution, planning Permanent Journey Plans (PJP) plays a vital role in organizing daily route schedules for field representatives, such as sales executives, to ensure consistent and efficient service to stores. PJPs involve determining which stores to visit on which days and assigning sales representatives while adhering to predefined constraints such as visit frequencies, even visit spacing, workload balancing, consistent assignments, holidays, and minimizing travel distances. However, this planning process is inherently complex. Traditional manual or semi-automated approaches are unable to handle the scale and intricacies of modern Fast-Moving Consumer Goods (FMCG) operations, leading to inefficient routes that increase travel times and operational costs. Misaligned schedules result in missed visits or inconsistent service, while poor resource utilization increases inefficiencies. These inefficiencies also contribute to increased fuel consumption and carbon emissions, further escalating costs and environmental impact. Moreover, managing thousands of stores across diverse regions poses significant scalability challenges.

To overcome these challenges, Rahguzar is proposed as an innovative web-based solution designed to automate and optimize PJP planning, enabling efficient route generation, effective workforce utilization, and reduced operational and environmental costs.

1.2 Proposed Solution

The proposed solution, Rahguzar, is a comprehensive web-based application developed to automate and optimize the planning of PJPs for distributors, managers, and field representatives, including sales executives, merchandisers, and order bookers,

in the FMCG sector. Rahguzar generates efficient, scalable schedules that adhere to critical constraints, such as store visit frequencies, even visit spacing, workload balancing, consistent assignments, holidays, and minimizing travel distances. These tailored schedules ensure that field representatives can execute their tasks effectively while aligning with the organization's operational goals.

The platform features a user-friendly, interactive map interface that allows managers to visualize, adjust, and schedule visits for daily, weekly, or monthly plans. Rahguzar determines the optimal number of field representatives required, improving workforce utilization and preventing inefficiencies such as overstaffing or underhiring.

By optimizing routes, Rahguzar minimizes travel times, operational costs, and fuel consumption, significantly reducing the carbon footprint and promoting sustainability. This automation reduces the need for manual intervention, saving valuable time for managers while ensuring that field representatives have clear, efficient routes tailored to their responsibilities. Rahguzar delivers a scalable, cost-effective, and environmentally sustainable solution for modern FMCG operations, enhancing productivity, consistency, and overall service quality.

1.3 Intended User

This section outlines the target users of Rahguzar, detailing the different types of users in the user base and their interactions with the platform.

- **Managers:** Managers are the primary users of the system, responsible for creating, adjusting, and scheduling optimized journey plans for field representatives. They will interact with the platform through a user-friendly map interface to design, visualize, and modify routes. The system allows managers to generate daily, weekly, or monthly plans, adapt to specific business constraints, and ensure operational efficiency.

- **Field Representatives:** While field representatives, such as sales executives and order bookers, do not directly interact with the system, they are key beneficiaries of its outputs. They rely on the optimized journey plans created by managers to execute their tasks efficiently in the field, ensuring consistent and effective service to stores.

1.4 Project Gantt chart and deliverables

Deliverables for Kaavish I:

- Requirements Specification Document
- System Architecture & Design Diagram
- Prototype for Frontend and Backend Components
- Fully Developed Route Optimization Algorithm
- Performance and Evaluation Report

Deliverables for Kaavish II:

- Comparison with existing Algorithms and improvements
- User-friendly web application with map interface
- Performance Metrics Dashboard
- Pilot Testing Framework and Results
- Final Project Documentation and Evaluation Report

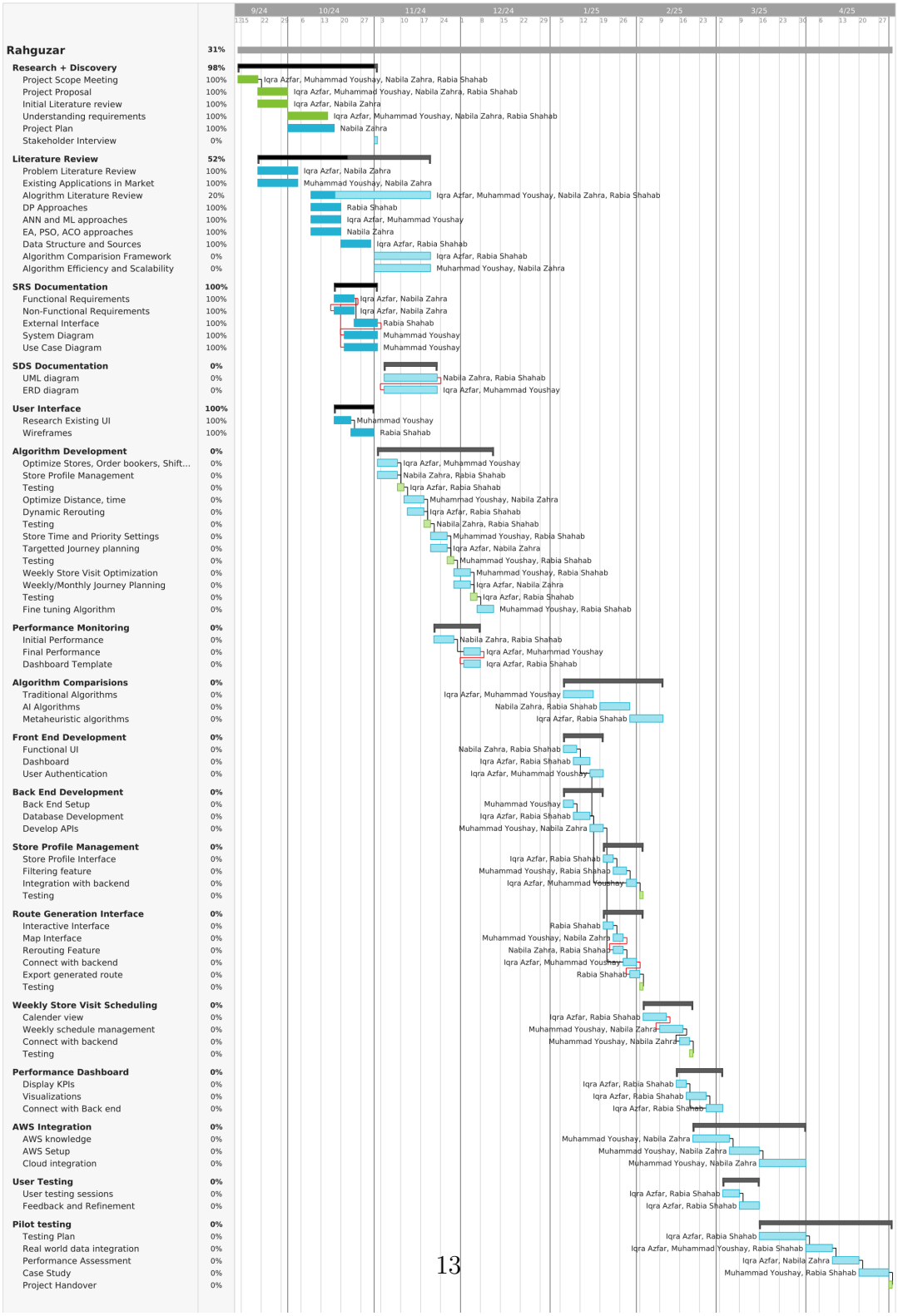

Figure 1.1: Project Gantt Chart

1.5 Key Challenges

Key challenges anticipated in our project will include:

- **Data Complexity and Integration:** Managing and integrating diverse datasets, including store profiles, geographic data, and workforce schedules, presents significant complexity, particularly in reconciling inconsistencies and ensuring data accuracy.
  Solution: Utilize specialized data processing libraries to streamline data handling and incorporate iterative testing phases to identify and address integration issues early in development.

- **Multi-Objective Optimization:** Balancing multiple optimization criteria, such as minimizing travel distances, reducing costs, meeting visit frequencies, and ensuring workload balancing, is inherently challenging due to competing priorities.
  Solution: Implement a weighted scoring system to prioritize objectives based on business needs and evaluate outputs from multiple algorithms to identify the most efficient and effective solutions.

- **API Costs and Performance:** Heavy reliance on mapping APIs, such as Google Maps, could lead to high costs and potentially slow performance when processing large-scale data.
  Solution: Optimize API usage through practices like batch processing and caching to reduce the number of API calls. Additionally, utilize trial phases of these services to test and implement them effectively without exceeding cost limits.

- **Adaptability to Business-Specific Constraints:** Organizations may have unique and variable constraints, such as holidays, visit frequency, or workload preferences, which require flexible accommodation in the route planning system.
  Solution: Design the system to allow customization of route parameters and enable managers to manually adjust routes through an intuitive interface, ensuring flexibility for diverse business requirements.

2. Literature Review

This section explores the techniques and methodologies used for optimizing Permanent Journey Plans (PJP), emphasizing their relevance to modern logistics and distribution systems. It highlights a range of optimization algorithms, including Dynamic Programming, Genetic Algorithms, Particle Swarm Optimization, Ant Colony Optimization, and Mixed-Integer Linear Programming, alongside the role of clustering methods like K-means. These approaches address critical challenges such as minimizing travel distances, adhering to time constraints, balancing workloads, and ensuring scalability in large-scale routing problems. The review establishes a foundation for understanding how these techniques contribute to efficient and effective PJP optimization.

2.0.1 Journey Planning for Industries

In today's highly competitive marketplace, the speed and efficiency of transportation can vastly influence the success and failure for industries like fast-moving consumer goods, retailers, logistics, and distribution centers [4]. For these sectors, logistics can account for over 30% of distribution costs and contribute 15-40% to the overall product price [5]. Therefore, businesses need to plan their journey routes with the utmost efficiency to enhance their operational productivity and reduce their costs. The journey plans generated, known as Permanent Journey Plans, or PJPs, have to meet meticulous criteria such as minimizing the distance traveled, making sure time-sensitive plans are met with punctuality, and maximizing the number of places that can be visited on a certain day [21]. Traditional approaches to journey planning often rely on manual processes or heuristic methods, making it a tedious process and unable to handle the complexities required by modern logistics systems, increasing the logistics cost by 10-30% [17, 2]. For developing countries such as Pakistan, the wholesale and retail trade sector contributes to 18% of the overall GDP, emphasizing on the need to further improve the routing efficiency to support the financial stability and growth of businesses in the retail sector [12].

Recent developments in algorithmic approaches, combined with advancements in operations research and computational techniques, have renovated how to assess journey planning problems, and how we can address them [3]. The Vehicle Routing Problem (VRP) is an NP-hard problem widely recognized in journey planning, serving as a benchmark for testing and improving optimization algorithms. These problems are applied in scenarios that require the optimization of Permanent Journey Plans, or PJPs, where the routes and schedules created have to be consistent over extended periods. The created PJPs must account for several constraints such as frequency of visits that ensure locations are serviced at appropriate intervals to meet the customer demands, time window constraints to align the schedules with store operating hours, and distributing workload evenly across the number of available sales force. Moreover, it is vital to balance the rigidity of fixed routes with the dynamic nature of real-world variables, such as changes in traffic, demand fluctuations, and operational constraints.

2.0.2 Optimization Techniques in Route Planning

Solving complex, NP-hard problems like Permanent Journey Plan (PJP) optimization requires advanced techniques to balance constraints and objectives efficiently [1]. Optimization methods such as Mixed-Integer Linear Programming (MILP), Dynamic Programming (DP), Genetic Algorithms (GA), Particle Swarm Optimization (PSO), and Ant Colony Optimization (ACO) have been widely applied to address diverse routing scenarios, including static planning and dynamic adjustments.

An important approach for improving computational efficiency in large-scale problems is clustering, which involves grouping locations based on proximity or other criteria. K-means clustering is one of the most commonly used techniques for this purpose [1], dividing locations into well-defined regions to simplify route optimization. By reducing the problem size, clustering allows optimization algorithms like MILP, GA, or PSO to operate independently within each cluster, making the overall process more computationally feasible.

MILP excels at providing exact solutions by modeling objectives and constraints, such as visit frequencies and workload balancing, as linear functions. Although MILP is computationally expensive for large-scale problems, integrating it with K-means clustering significantly reduces complexity [16]. DP, on the other hand, decomposes problems into smaller subproblems, solving each recursively and storing intermediate results. While useful for smaller PJP tasks, its computational demands make it more practical when combined with other methods for hybrid solutions [25].

Metaheuristic algorithms like GA, PSO, and ACO provide robust alternatives for

handling the multi-objective and dynamic nature of PJP optimization. GA iteratively refines solutions through evolutionary strategies like selection and crossover, effectively balancing travel distances and visit frequencies [15]. PSO, inspired by swarm behavior, optimizes routes by simulating collective decision-making and works particularly well when combined with clustering methods like K-means to refine regional solutions [11]. ACO, modeled on the foraging behavior of ants, is adept at exploring combinatorial solution spaces, avoiding local optima, and identifying efficient routes [20].

Clustering and optimization algorithms work hand in hand for large-scale PJP problems. Clustering simplifies the problem by dividing it into smaller regions, while optimization techniques like GA, PSO, and MILP refine routes within each cluster. This integrated approach ensures computational efficiency, scalability, and practical applicability, making it well-suited for addressing PJP-specific challenges such as visit frequencies, even workload distribution, and time-sensitive constraints.

While researching for ways to solve the problem of creating optimal PJPs, we have come accross several approaches that solve a similar problem while optimizing other objectives that are discussed in detail below.

2.0.3 Dynamic Programming (DP) to Solve VRPTW

Dynamic Programming (DP) is a widely-used optimization technique capable of solving complex routing problems with constraints, making it particularly suitable for Permanent Journey Plan (PJP) optimization. Its ability to decompose a problem into smaller, manageable subproblems while ensuring global optimality makes it a valuable tool for addressing challenges such as visit frequencies, even visit spacing, time windows, and workload balancing.

A study [14] applied a DP algorithm to solve the Vehicle Routing Problem with Time Windows (VRPTW). This framework incrementally builds routes by sequentially adding stops while verifying feasibility under constraints. In a PJP context, this approach can be extended to align store visits with predefined schedules and ensure adherence to operating hours. The study reported a reduction in routes by 18% and travel distances by over 5%, highlighting the efficiency gains possible when dynamic adjustments are incorporated into route planning. For PJPs, such gains translate into increased store coverage and reduced operational costs.

Another study by Desaulniers et al. (2002) [6] focused on the use of DP to optimize fleet routing while accounting for time window constraints. Their approach incorporated a state-space relaxation method, reducing computational overhead while maintaining solution quality. For PJP, this method can ensure that sales represen-

tatives follow efficient schedules without exceeding working hour limits or violating store-specific time windows.

Furthermore, a study by Feillet et al. (2004) [7] introduced a label-setting algorithm based on DP to solve VRP with soft time windows. This approach allowed flexibility in time window adherence, a feature particularly useful for PJP scenarios where strict adherence to schedules may not always be possible. By penalizing minor deviations rather than rejecting infeasible solutions outright, this method ensures that PJPs remain robust under dynamic operational conditions.

A hybrid DP approach by Kilby et al. (2002) [13] combined DP with heuristic techniques to address large-scale routing problems. The hybrid method used DP to solve subproblems within clusters, which were formed using heuristic clustering methods. This is particularly relevant for PJPs, where clustering stores into smaller groups based on proximity or workload can significantly reduce the computational complexity of route planning.

2.0.4 Mixed-Integer Linear Programming (MILP) for PJP

Mixed-Integer Linear Programming (MILP) is a mathematical optimization technique widely used in route planning and scheduling problems. It is particularly relevant for optimizing PJPs, where the focus is on visiting stores to generate demand while adhering to constraints such as visit frequencies, time windows, workload balancing, and minimizing travel distances.

MILP allows for the formulation of optimization problems as linear objective functions with linear constraints. For PJP scenarios, the objective might involve minimizing the total distance traveled or the time taken, while constraints ensure adherence to store-specific visit frequencies, consistent assignments for sales representatives, and balanced workloads. Unlike heuristic methods, MILP offers mathematically optimal solutions, making it a valuable tool for small to medium-scale PJP problems [16].

A study by Sierksma and Tijssen (1998) highlights the use of MILP in sales territory alignment and routing problems, which are closely related to PJP optimization. Their approach ensures balanced workload distribution among sales representatives and minimizes travel costs [23].

For larger problems, MILP is often combined with clustering techniques to break down the problem into smaller subproblems. In such cases, clustering algorithms segment stores into manageable groups based on geographical proximity or other criteria, and MILP is applied within each cluster to generate optimal routes [18]. This hybrid approach improves computational feasibility while maintaining solution

quality.

Despite its computational intensity, MILP serves as an excellent benchmark for evaluating heuristic or metaheuristic approaches, such as Genetic Algorithms (GA) and Particle Swarm Optimization (PSO). For example, a study by Schneider et al. (2014) demonstrated that MILP models could effectively optimize vehicle routing problems with time windows, which is directly applicable to PJP scenarios involving time-sensitive store visits [22].

2.0.5 Metaheuristic Approaches for VRPTW

Metaheuristic algorithms like Genetic Algorithms (GA) and Ant Colony Optimization (ACO), widely applied to solve the Vehicle Routing Problem with Time Windows (VRPTW), are also effective for optimizing Permanent Journey Plans (PJP) due to their ability to handle multi-objective and constraint-driven problems. A range of single-objective and multi-objective approaches has been explored to address the Vehicle Routing Problem with Time Windows (VRPTW). Brock University conducted a study [19] comparing single-objective and multi-objective methods, emphasizing a non-biased approach without weighted sum scoring to avoid compromising between vehicle usage and travel distance. The genetic algorithm (GA) used demonstrated optimal or near-optimal results for clustered data but was less consistent for uniformly distributed customer locations. The multi-objective approach faced challenges with Pareto dominance, leading to suboptimal compromises and limited generalizability to complex real-world scenarios.

A novel Multi-Objective Evolutionary Algorithm (MOEA) incorporating Jaccard's coefficient was proposed to improve population diversity and Pareto-based approximations [9]. This method outperformed single-objective algorithms like EA with deterministic or randomized selection, achieving 2% to 5% better outcomes. However, its scalability was limited when applied to datasets with higher travel distances, posing challenges for large or complex datasets due to increased computation time.

A bi-objective GA combined with goal programming was designed to minimize vehicle use and travel costs, leveraging Pareto ranking and fitness evaluation with Pareto ranks [10]. This method showed positive results for clustered data, correlating vehicle count with travel cost. Yet, it struggled with unclustered datasets and mixed scenarios, which led to higher costs and limited effectiveness for larger, dynamically constrained problems.

The Hybrid Ant Colony Optimization (HACO) approach [26] aimed to solve the multi-objective vehicle routing problem with flexible time windows (MOVRPFlexTW)

by balancing customer satisfaction and cost. HACO used a hybrid strategy with multiple mutation operators and incorporated Pareto optimality, achieving good convergence speed and accuracy in Solomon's benchmark problems. While HACO demonstrated robustness and adaptability to higher-dimensional data, finding optimal parameters was sensitive, potentially impacting efficiency and scalability for larger datasets.

These GA-based strategies each have strengths and limitations. The Brock University study highlighted trade-offs and biases when using Pareto dominance in multi-objective approaches. MOEA's incorporation of Jaccard's coefficient improved diversity but faced scalability issues with higher travel distances. The bi-objective GA showed potential with clustered data but lacked stability in non-clustered scenarios. HACO, in contrast, outperformed traditional GAs and EAs, showing potential for real-world application and better scalability. Collectively, these findings highlight the need for further refinement and hybrid solutions to enhance generalizability and computational efficiency for real-world VRPTW problems.

2.0.6 Hybrid Clustering Algorithms

Another method to approach the VRP problem is to combine optimization techniques with clustering algorithms to solve complex VRP problems under real-life constraints. The clustering algorithms will be covered in the sections below.

Various clustering and optimization algorithms have been developed to address the complexities of Vehicle Routing Problems (VRPs). A three-phase algorithm approach from paper [28] that uses clustering, Mixed-Integer Linear Programming (MILP), and a Traveling Salesman Problem (TSP) approach has proven effective, reducing transport costs by 10%-20% in real-world cases. This method clusters customers into manageable subproblems, optimizes routes using MILP, and fine-tunes them with a TSP approach. However, its performance depends on initial clustering and struggles with large-scale problems.

Another approach [27] uses the Discrete Firefly Algorithm (DFA) for clustering and a modified 2-opt algorithm for route optimization. DFA outperformed k-Means and manual methods, achieving up to a 15% cost reduction and faster computation. Its main drawback is the dependence on initial clustering quality and computational intensity.

A hybrid algorithm [24] for VRPs with Time Windows (VRPTW) begins with K-means clustering and refines it using OPTICS. Routes are generated using a Nearest Neighbor heuristic and optimized with 2-opt, showing a 5.06% distance reduction for smaller instances and 11.97% for larger ones. Its limitations include assumptions of

static conditions and a need for parameter tuning.

The KACO algorithm for the Dynamic Location Routing Problem (DLRP) discussed in paper [8] integrates K-means clustering with Ant Colony Optimization (ACO), using immigrant schemes to adapt routes dynamically. It outperformed traditional ACO, showing better tour lengths and adaptability under dynamic conditions. However, challenges with clustering quality and high computational demands remain.

These approaches demonstrate that combining clustering with optimization methods effectively handles VRPs, offering cost and efficiency benefits as compared to only adapting optimization algorithms to solve VRP problems. Moreover, they showcase better adaptability and scalability to real-world datasets as opposed to algorithms that solely relied on optimization to solve VRP that could not perform well with large and real-world datasets.

Drawing from the extensive range of methodologies explored, it is evident that optimization algorithms and clustering techniques play a crucial role in addressing complex routing and scheduling challenges across various domains. Genetic Algorithms (GA) have demonstrated their utility in multi-objective optimization, balancing competing factors such as distance minimization and timely service delivery through evolutionary processes. Similarly, Particle Swarm Optimization (PSO), inspired by the collective behavior of swarms, offers computational efficiency and adaptability in determining optimal routes and clustering solutions. Ant Colony Optimization (ACO), with its biologically inspired probabilistic approach, has proven adept at solving combinatorial problems like the Vehicle Routing Problem (VRP) by navigating through solution spaces to find optimal paths while evading local optima. Dynamic Programming (DP), although computationally intensive, provides exact solutions to problems by systematically breaking them into smaller subproblems.

The integration of these algorithms with clustering techniques, such as those used for grouping locations based on proximity and service requirements, enhances the efficiency of solving the vehicle routing and scheduling problems. Techniques like hierarchical clustering, k-means clustering, and density-based spatial clustering play pivotal roles in defining clusters that optimize routing paths, reduce travel times, and ensure that each route adheres to time windows and service constraints. These clustering approaches, combined with the optimization algorithms, help in forming practical, real-world solutions for complex logistical challenges. Through this literature review, it is clear that continued research and innovation in these areas are vital for developing more robust and scalable solutions to complex optimization problems, ultimately enhancing the efficiency and effectiveness of route optimization and planning in various industries.

3. Software Requirement Specification (SRS)

This Software Requirements Specification (SRS) outlines the software and system requirements for Rahguzar, a route optimization project aimed at improving efficiency in retail and distribution. It details functional requirements for system capabilities and non-functional requirements like performance and scalability. The document also includes system block diagrams, use cases, and external interfaces to clarify the system's design and interactions.

3.1 Functional Requirements

The functional requirements for this system focus on creating an efficient journey-planning solution tailored for order bookers and sales representatives. This system is designed to optimize routes based on key operational parameters and provide dynamic rerouting, while also supporting targeted store visits and frequency schedules. Additional modules include a user-friendly web platform, pilot testing and validation, and performance monitoring to ensure enhanced efficiency and measurable sales impact.The following are the functional requirements for each module and their respective functions.

Module 1: Route Optimization Algorithm

Function 1: Route Generation

- Implement an algorithm that creates journey plans optimized for order bookers and sales representatives.
- The optimization should consider multiple factors:

Sub Function 1: Order Booker Shift Times
Align route planning with the working hours of each order booker.

Sub Function 2: Number of Shops to Visit
Adjust routes to maximize the number of shops visited within constraints.

Sub Function 3: Number of Order Bookers Available
Allocate routes based on the number of available staff.

Sub Function 4: Total Distance Traveled
Minimize the total distance covered during journeys.

Sub Function 5: Total Travel Time
Reduce the time taken to complete the planned routes.

Sub Function 6: Store Service Times
Account for different service time requirements at each store.

- Include a feature to assign priority to specific parameters to adapt the optimization to business needs.

Function 2: Dynamic Rerouting

- It allows users to interactively adjust key parameters of the route optimization algorithm, such as the number of order bookers and the number of stores per booker, directly on the display screen.
- It simply enables real-time rerouting, with the algorithm instantly updating the allocation of stores and route paths to reflect user preferences, providing flexibility and adaptability in journey planning.

Function 3 : Store Visit Frequency:

- Design the algorithm to include visit frequency for each store (e.g., once, twice, or three times a week).
- Ensure that the routes align with required visit frequencies without any overlap or gaps.

Function 4: Journey Planning for Different Timeframes:

- Support the generation of routes by allowing monthly PJP creation with customizable weekly patterns (like odd/even weeks) and enable downloads for extended periods, such as a year. based on the target timeframe.

- Ensure that stores are scheduled accurately according to their frequency needs while minimizing travel and maximizing coverage.

Function 5: Store Profiling:

- Maintain detailed profiles for each store, including:

  Sub Function 1: Geographical hierarchy (region, zone, territory, town).

  Sub Function 2: Sales channel type (e.g., wholesale or retail).

- Enable users to filter stores by parameters like region, sales channel, or visit frequency. Allow custom filters linked to all stores by uploading relevant data (e.g., location, sales volume), enabling dynamic application across stores. This approach enhances flexibility and control in route optimization.

Function 6: Targeted Journey Plans:

- Enable the system to generate routes targeting specific store types (e.g., only wholesale stores in a particular region).

Module 2: Web Application Platform

Function 1: User Interface for Route Management:

- Create a user-friendly web interface for managers to access and manage journey plans.

Function 2: Integrate Map Interface:

- Designing, viewing, and adjusting routes.
- Displaying essential shop details (e.g., store profiles, geographical location).
- Defining and viewing boundaries like regions, territories, and areas.

Function 3: Manual Override and Adjustment:

- Provide a manual override feature that allows managers to create or adjust routes manually, even if automated suggestions exist.
- Include input options for master data (store profiles, visit frequencies, geographical details) directly within the platform.

Function 4: Downloadable Reports:

- Allow users to export optimized or manually adjusted routes in a downloadable format for offline analysis or distribution.

Module 3: Pilot Testing and Validation

Function 1: Test Plan Creation:

- Develop a comprehensive pilot testing framework to validate the system's effectiveness.

Function 2: Real-World Data Integration:

- Gather real-world data during pilot testing, ensuring data is accurate and reliable.

Function 3: Performance Assessment:

- Collect and analyze data (travel time, number of shops visited, sales impact, etc.) to assess system performance.
- Implement an adjustment process based on pilot feedback to enhance the algorithm.

Function 4: Case Study Development:

- Compile a case study summarizing the outcomes, findings, and system performance.

Module 4: Performance Monitoring and Reporting

Function 1: Key Performance Indicators (KPIs):

- Track and display metrics like:

  Sub Function 1: Average travel time per route.

  Sub Function 2: Number of shops visited per shift.

  Sub Function 3: Number of order bookers required per distribution.

Sub Function 4: Sales uplift due to route optimization.

Sub Function 5: Algorithm response time for route recalculations.

Function 2: North Star Metric Tracking:

- Focus on monitoring the primary metric: "Increase in average daily sales per sales representative."

Function 3: Metrics Dashboard:

- Develop a dashboard to visualize KPIs, providing insights into efficiency improvements, time savings, and sales increases.

3.2 Non-functional Requirements

The Non-functional requirements define the quality and performance attributes of this system, focusing on scalability, usability, and reliability. They ensure optimal performance under varying conditions while maintaining data security and adaptability for user needs. Additionally, following requirements also facilitate maintainability for future enhancements, ensuring the system remains effective over time.

1. **Scalability, Efficiency, and Memory Optimization**
    - The system is designed to efficiently handle large datasets and scale up to 2000 stores effectively as the amount of data increases, ensuring optimal performance and quick response times within 2 minutes for route generation.
    - It should also have a low memory footprint, ensuring that it can scale effectively without degrading system performance or requiring excessive computational resources.
2. **Usability**
    - A user-friendly web application interface is required, allowing users to manage, optimize, and adjust journey plans easily.
    - The map interface must provide intuitive visualization and interactivity for planning and real-time adjustments.

3. **Reliability and Performance**

    - The AI-driven route optimization algorithm should be robust, quickly generating optimal routes based on parameter changes, such as order booker shifts or the number of stores per booker.
    - The system should maintain a minimum uptime of 95%, allowing for up to 36 hours of downtime per month for a smooth user experience.

4. **Security**

    - Data security protocols should protect sensitive information such as store profiles and geographical hierarchies, especially when integrating third-party APIs like Google Maps.
    - User authentication mechanisms should be implemented to ensure that only authorized personnel can access sensitive data and functionalities within the system.

5. **Adaptability and Flexibility**

    - Manual override functionality should allow users to adjust routes when needed, ensuring flexibility in route planning.
    - The system should support long-term planning options, including daily, weekly, or monthly route generation and optimization.

6. **Maintainability**

    - The architecture should allow for future enhancements, such as integrating additional parameters or algorithms for optimization, without significant rework.

3.3 External Interfaces

3.3.1 User Interfaces

The following wireframes have been created for our system to give an overview of the features it will provide and what an overview of the interface will look like.

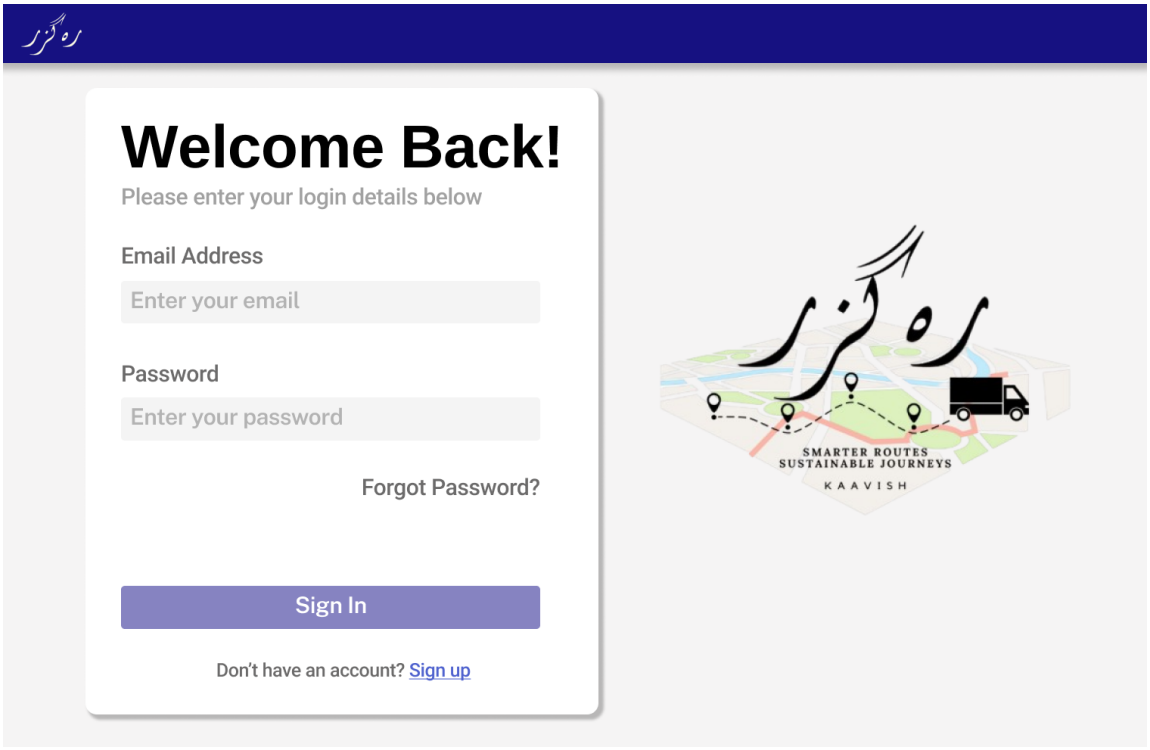

Figure 3.1: Sign In Screen

The sign-in screen is shown in Figure 3.1, letting the user enter their email and password. They also have the option to access the sign up screen using the hyperlink in the image called "Sign up".

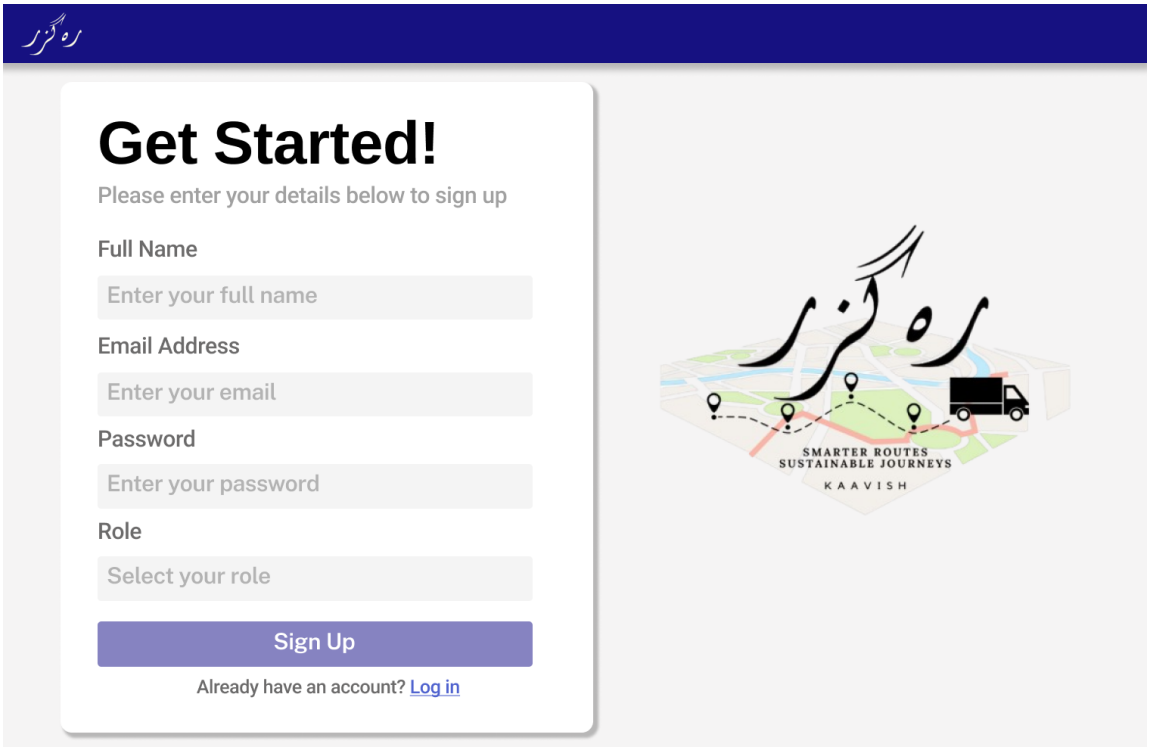

Figure 3.2: Sign Up Screen

The sign-up screen is shown in Figure 3.2, letting the user enter their full name, email and password to create an account.

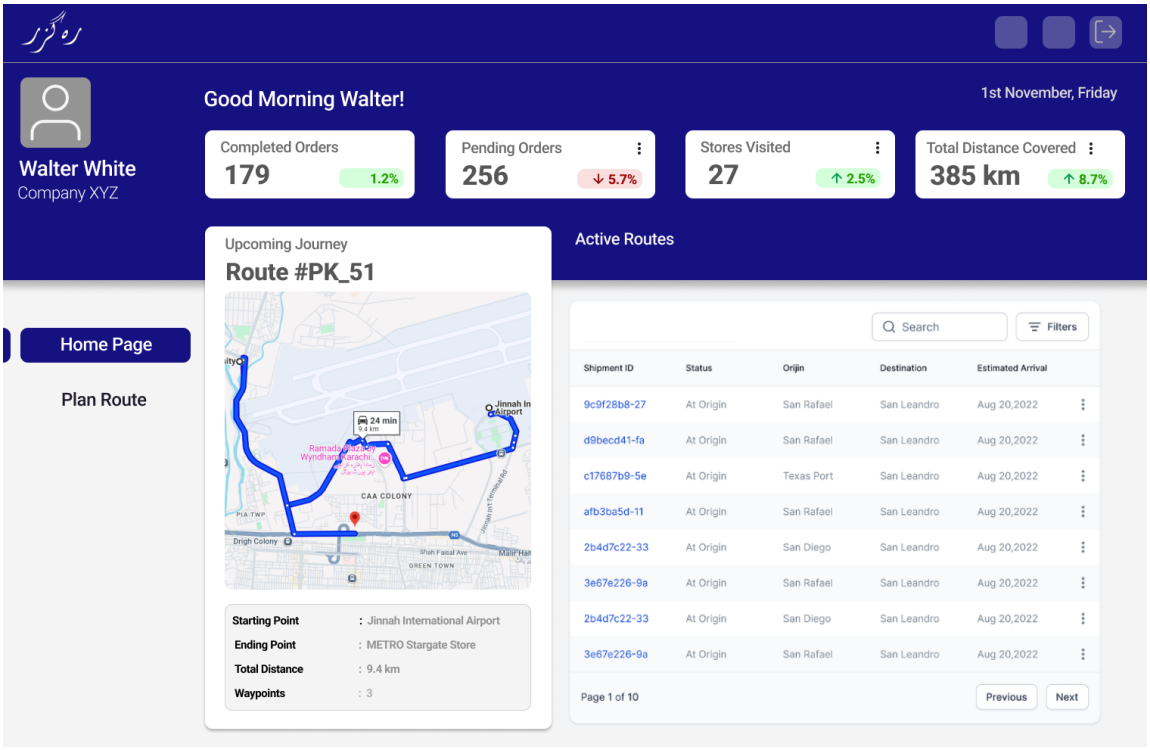

Figure 3.3: Home Page

The "Home Page" screen is shown in Figure 3.3, letting the user see the statistics of the day such as pending and completed orders. The user can view the upcoming recent journey and its related information such as starting and ending points, the total distance of the journey. The user will also be seeing the list of active routes and journeys that are/ will be taking place throughout the day.

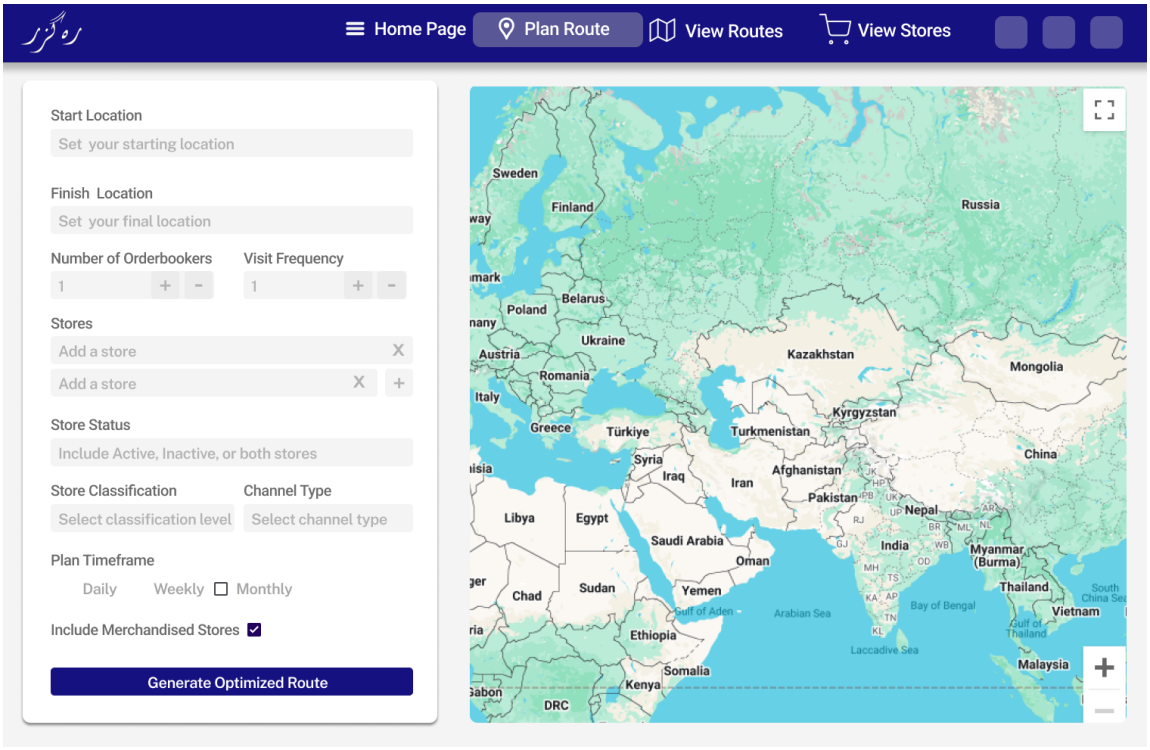

Figure 3.4: Plan Route Page

The “Plan Route” screen is shown in Figure 3.4. In this screen, the manager will be able to create routes by setting the starting point, adding the stores to stop at, the number of orderbookres that will be involved in the journey, apply filters to stores by their classification code and choosing if they want to include only merchandised stores in the plan. Moreover, the manager can make these routes for a single day, week, or a month.

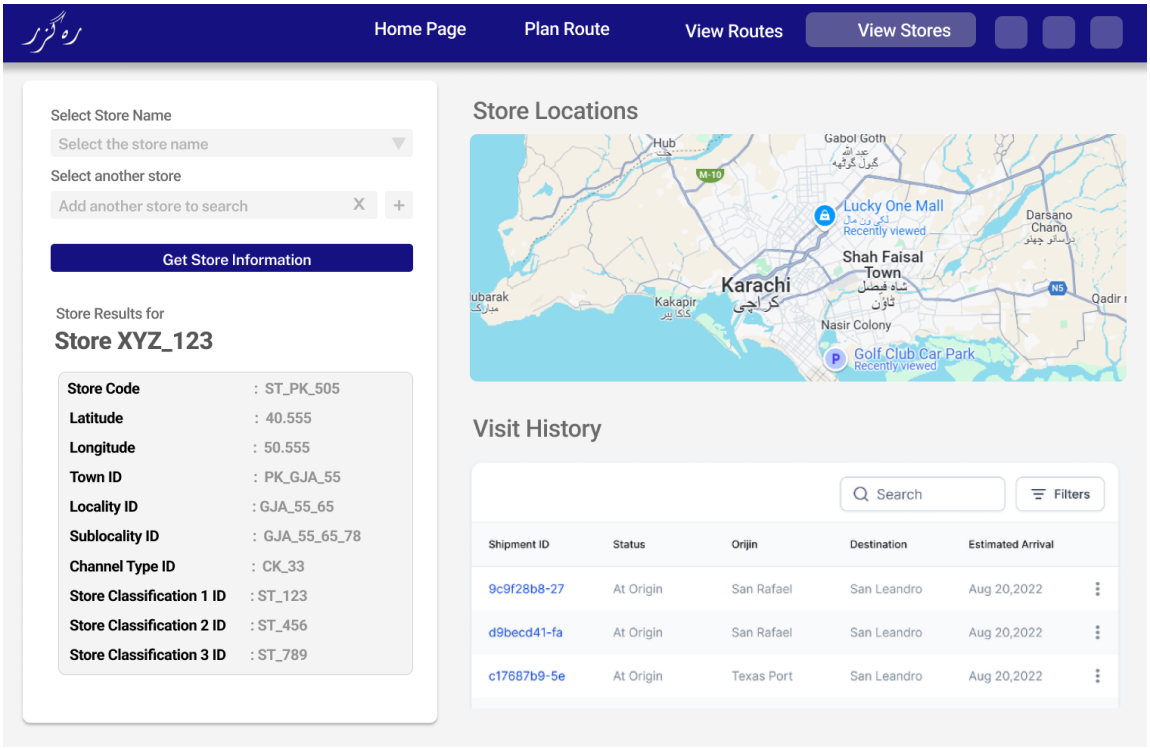

Figure 3.5: View Stores Page

The "View Stores" screen is shown in Figure 3.5. In this screen, the manager will be able to search a certain store and view its information such as its coordinates, its geographical hierarchy. They will be able to see the stores location on the map along with past orderbooker visits data. The manager has the option to view more than 1 store's information at a time.

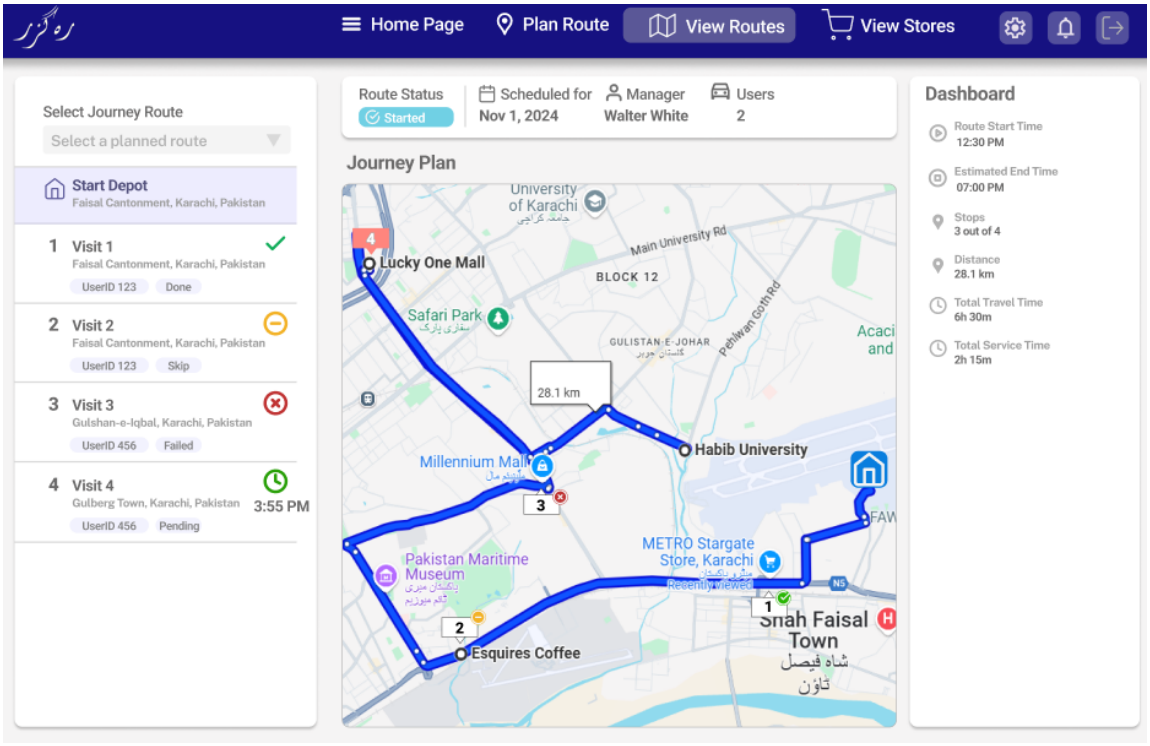

Figure 3.6: View Routes Page

The "View Routes" screen is shown in Figure 3.6. In this screen, the manager will be able to select a certain route and view its information such as the sets of locations it is comprised of, the status of the route. If a certain route has started the manager can see the status of each store location (whether it has been visited or not, the estimated time of visiting). Moreover, the manager can access the dashboard that displays route information such as the starting adn estimated end time, the total distance of the route, the distance covered by it and the number of stops involved.

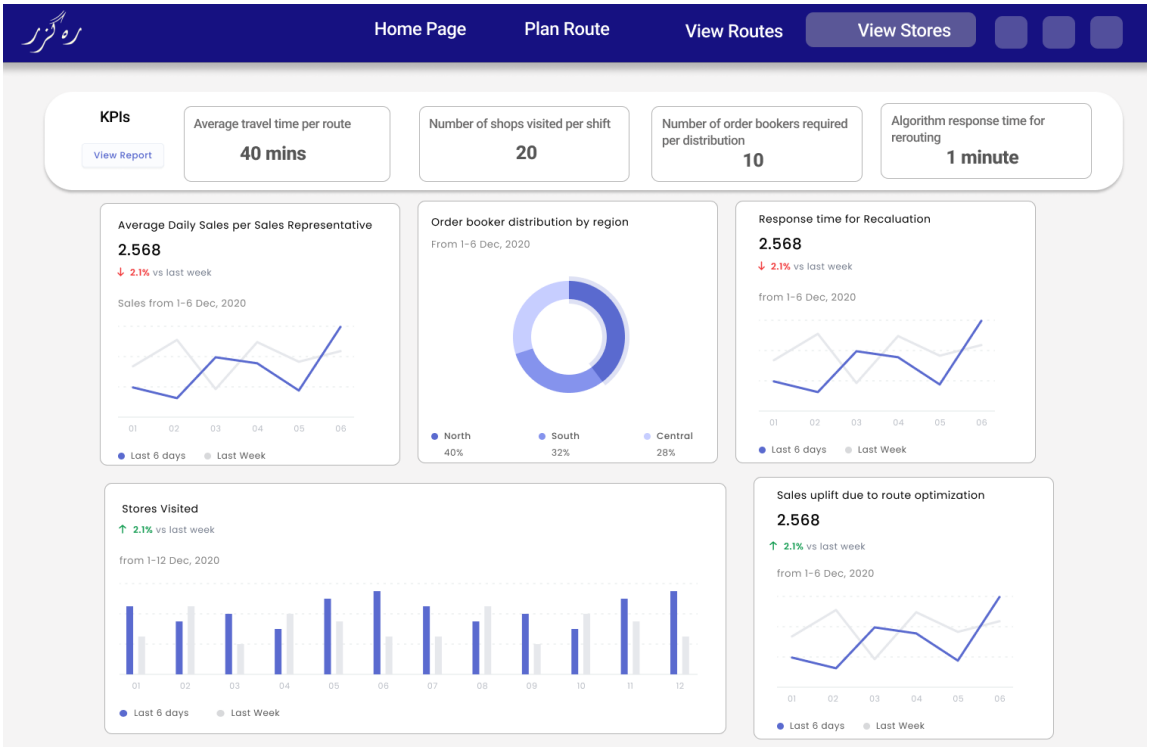

Figure 3.7: Dashboard Page

The "Dashboard" screen is shown in Figure 3.7. In this screen, the manager will be able to view certain KPIs that indicate the overall performance of the routes such as Average Daily Sales per Sales Representative, Average travel time per route, and etc.

3.3.2 Application Program Interface (API)

The API layer is the backbone of our system, enabling seamless communication between the frontend, backend, and external services. Designed to support efficient data exchange and operational scalability, this API structure ensures the system remains responsive and adaptable to business needs.

REST API Endpoints

The system is built with RESTful API endpoints that manage data transfer between components. Key endpoints include:

- **Route Management Endpoints:** These endpoints facilitate route creation, configuration, and optimization. Managers can define routes, set parameters such as shift times, service times, and visit frequency, and retrieve optimized route plans tailored to specific business needs.
- **Visit Management Endpoints:** Used to track and manage visits by field representatives. These endpoints enable updates on visit status, start and end times, store visit durations, and compliance with predefined visit schedules.
- **Store Profile Endpoints:** This set of endpoints allows access to and modification of store data. Store profiles include geographic location, classification, operational hours, and specific requirements, supporting managers in executing their tasks efficiently.

Authentication and Authorization

To secure API access, the Rahguzar system employs JWT (JSON Web Tokens) for user authentication, ensuring that only authorized users can access or modify data.

- **JWT Authentication:** When a user logs in, a JWT is issued and must be included with all subsequent requests. This token-based approach enhances security by verifying user identity on each API call.

External Services Integration

To support essential functionalities like mapping, the Rahguzar system integrates with external services:

- **Google Maps API:** This API provides map visualization and geolocation services, enabling managers and field staff to view real-time route details. It is also used for calculating distances, travel times, and generating optimized paths between stops on a route.

Route Optimization Engine

The Route Optimization Engine is a core component of Rahguzar, leveraging advanced algorithms to generate efficient routes.

- **Python + OR-Tools Integration:** Built on Python with OR-Tools, this engine generates optimized routes by balancing factors such as total travel time, number of stops, and time spent at each store. This feature is essential

for ensuring that routes are aligned with operational goals while maximizing efficiency.

- **Geospatial Services (Geopy):** Geopy is used for geocoding services and calculating geospatial data, helping to accurately map store locations and account for distances between various stops. This service plays a key role in generating precise, reliable routes for field operations.

Data Synchronization

The system employs MongoDB as the primary data store, providing robust data storage and synchronization capabilities.

- **Database Sync with MongoDB:** All route data, store profiles, and visit records are stored in MongoDB, ensuring persistent storage of essential information. MongoDB's flexibility supports the diverse data needs of the Rahguzar system, from route optimization to visit management.

- **Data Update Mechanisms:** The system includes mechanisms to synchronize data across different modules and handle real-time updates, especially for managers and field staff working offline. This ensures data integrity and consistency, even in dynamic environments. To ensure scalability and high availability, MongoDB and related services are hosted on a cloud platform such as AWS, enabling the system to handle larger datasets and user traffic efficiently as the application scales.

Data Monitoring

To enable effective monitoring, the API supports data updates and performance tracking:

- **Performance Analytics:** API endpoints allow managers to monitor route efficiency through Key Performance Indicators (KPIs), such as average travel time, visit completion rates, and operational metrics. This feature facilitates continuous improvement of route plans and decision-making based on real data, backed by scalable cloud infrastructure for robust data processing and analytics.

3.3.3 Hardware/Communication Interfaces

Our System relies on specific hardware and communication protocols to ensure seamless operation and efficient data management.

Client-Side Hardware Requirements

Manager Workstations: Managers access the system through web browsers on desktops or laptops for route planning and monitoring. Recommended specifications include:

- **Processor:** Dual-core or higher.
- **RAM:** 4GB minimum.
- **Internet Connection:** Stable broadband with at least 5 Mbps for real-time data processing.
- **Browser Compatibility:** Supports modern web browsers like Chrome, Firefox, and Edge.

Server-Side Infrastructure

Cloud-Based Servers: The backend system, hosted on a cloud platform such as AWS, provides scalability, availability, and fault tolerance. Specifications include:

- **Processor:** Multi-core virtual CPUs (e.g., AWS EC2).
- **Memory:** 4GB or more, scalable as required.
- **Storage:** SSD-based storage for rapid access to route, store, and visit data.
- **Network Configuration:** Configured for secure HTTPS communication to maintain data integrity and privacy.

Communication Interfaces

- **Internet Connection:** All interactions between the frontend and backend rely on a stable internet connection for data synchronization and API requests.

- **Data Communication Protocols:**
  - **HTTPS:** Secure communication between frontend, backend, and external services like Google Maps API.
  - **API Calls:** RESTful API requests enable data transfer for route management, visit updates, and user authentication.

External Services Communication

Google Maps API: Used for map visualization, geolocation, and distance calculation, requiring internet connectivity for location-based data.

3.4 Use Cases

- **User Authentication and Access Control**
  - **Description:** Ensure secure access to the system for both managers and field staff.
  - **Actions:**
    * As a manager, I want to log in securely to access my dashboard and manage routes.
    * As the system, I need to authenticate users to ensure only authorized managers and field staff access the application.
- **Route Creation, Configuration, and Optimization**
  - **Description:** Enable the creation of optimized routes based on configurable parameters.
  - **Actions:**
    * As a manager, I want to create a new optimized route for my field team, configuring parameters like shift timings and visit frequency to align with operational requirements.
    * As the system, I need to generate optimized routes using parameters such as shift times, visit frequencies, and priorities, ensuring efficient route suggestions for managers.
- **Dynamic Rerouting and Manual Overrides**

  - **Description:** Allow for real-time adjustments to routes, either dynamically by the system or manually by the manager.
  - **Actions:**
    * As a manager, I want to make manual adjustments to routes to adapt to urgent needs or unexpected changes.
    * As a manager, I want the system to dynamically reroute based on parameter changes after the route is generated.
    * As the system, I need to recalculate routes dynamically if the manager makes changes to parameters, ensuring updated routes without disrupting existing operations.

- **Route Sharing and Export**
  - **Description:** Facilitate the sharing of finalized routes with field staff.
  - **Actions:**
    * As a manager, I want to export and share finalized routes with field staff so they have clear guidance on their daily tasks and destinations.
    * As the system, I need to support data export so that managers can download route plans and reports for offline access if needed.

- **Plan Generation (Weekly/Monthly)**
  - **Description:** Generate long-term journey plans based on store visit requirements.
  - **Actions:**
    * As a manager, I want the ability to generate weekly or monthly journey plans, providing my team with a consistent schedule for store visits.
    * As the system, I need to automatically generate journey plans for weekly and monthly schedules based on specified visit frequencies.

- **Store Profile Integration and Viewing**
  - **Description:** Provide access to detailed store profiles to aid in route planning.
  - **Actions:**

    - As a manager, I want to view detailed store profiles, including location and sales channel, to prioritize visits and plan routes effectively.
    - As the system, I need to integrate and maintain comprehensive store profiles, making it easier for managers to access relevant data during route planning.

- **Interactive Map Visualization**
  - **Description:** Enable map-based visualization of routes and store locations.
  - **Actions:**
    - As a manager, I want to see routes on an interactive map, allowing me to visualize store locations and make adjustments as needed.
    - As the system, I need to display routes and navigation data on an interactive map interface for both managers and field staff.

- **Performance Tracking and KPI Monitoring**
  - **Description:** Track and analyse performance metrics to evaluate route effectiveness.
  - **Actions:**
    - As a manager, I want to track KPIs like travel time and visit completion rates to assess route performance and make improvements.
    - As the system, I need to automate KPI tracking (e.g., average travel time, visit completion) to provide ongoing insights without requiring manual input from managers.

- **System Data Synchronization**
  - **Description:** Sync store and route data to ensure accurate and up-to-date information.
  - **Actions:**
    - As a manager, I want the system to view store information and route statuses, giving me reliable data for decision-making.
    - As the system, I need to synchronize with external databases to maintain current and accurate store and route data for all users.

- **Store Visit Recording and Analysis**

- **Description:** Record visit details and enable analysis of visit data for operational insights.
- **Actions:**
  - As a manager, I want each store visit's details recorded, allowing me to analyse time spent and identify areas for improvement.
  - As the system, I need to log each store visit's start time, end time, and duration, providing accurate records for analysis and performance tracking.

- **Pilot Testing and Validation**
  - **Description:** Conduct pilot tests to validate system effectiveness in real-world scenarios.
  - **Actions:**
    - As a manager, I want to test the system in a pilot environment to evaluate its performance and impact before full deployment.
    - As the system, I need to support pilot testing by recording metrics during live tests, allowing managers to assess effectiveness and identify areas for improvement.

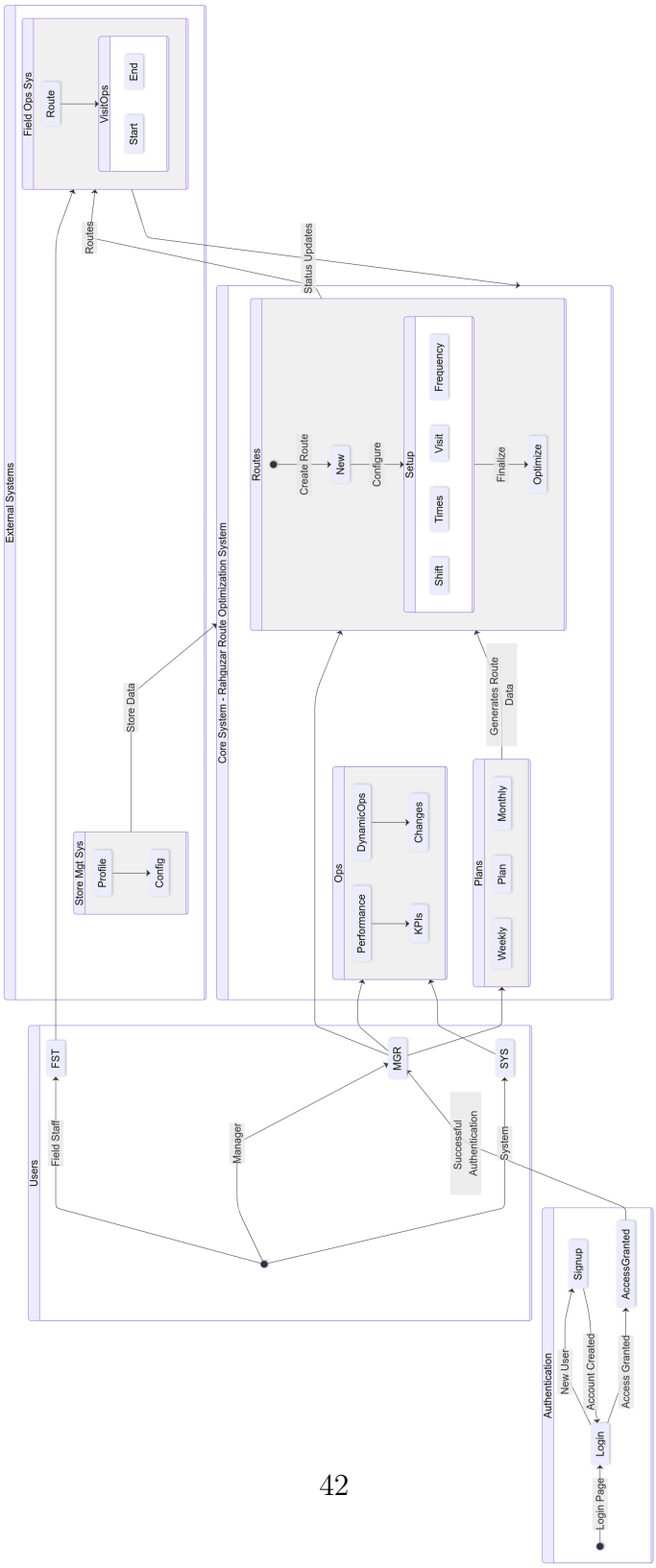

Figure 3.8: Rahguzar's High-Level Use Case Diagram for the Entire System
(Click here to view a high-quality version of this image)

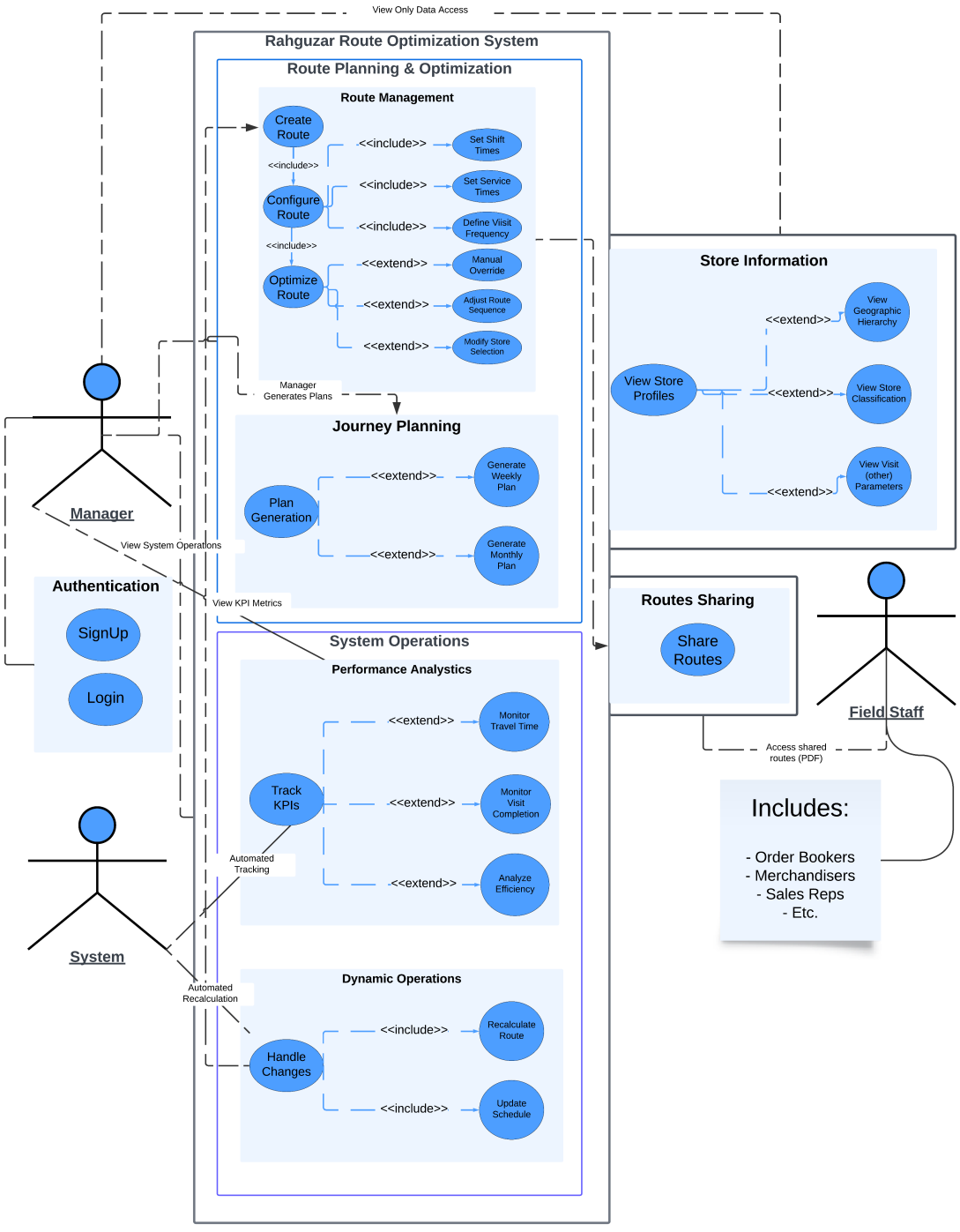

Figure 3.9: Rahguzar's Core System Use Case Diagram within the Main System

3.5 Datasets

This section outlines the primary datasets that will support Rahguzar's intelligent route planning and scheduling functionalities. These datasets provide crucial information on stores, visits, journey plans, and operational factors essential for optimized route generation. As per the NDA with SalesFlo, we are not permitted to share or disclose the data provided to us. Further data and details are awaited from SalesFlo, and this section reflects the information we currently have.

Overview of Required Datasets

1. **Store Universe**
   This dataset provides foundational information on each store, including location, operational status, and distributor associations. Key fields include:

   - **storecode:** Unique identifier for each store.
   - **storestatus:** Operational status (active or inactive).
   - **latitude and longitude:** Geographic coordinates.
   - **distributorcode:** Code for the distributor linked to the store.
   - **pjpcode:** Code for the Permanent Journey Plan (PJP) associated with the store.

   This dataset is essential for determining which stores to include in routes based on location and operational status.

2. **Store Hierarchy**
   This dataset offers classification and hierarchical details, allowing route customization based on store attributes such as region or sales channel. Key fields include:

   - **storecode:** Identifier for each store.
   - **areatype, townid, localityid:** Geographical identifiers.
   - **channeltypeid and channelid:** Sales channel identifiers (e.g., retail, wholesale).

- **storeclassificationIDs:** These classification IDs will be used to further distinguish between certain store categories.

This data enables Rahguzar to organize routes by region and sales channel, optimizing for specific geographic and operational criteria.

3. **Visits**
   The Visits dataset records historical visit information for each store, including time spent, visit order, and outcomes. Key fields include:

   - **visitid:** Unique identifier for each visit.
   - **pjpcode:** Journey plan code associated with the visit.
   - **visitdate, visitstarttime, visitendtime:** Time data for analyzing visit durations.
   - **visitspenttimeinseconds:** Total time spent at the store during the visit.
   - **orderstatus:** Status of any order placed during the visit.
   - **visiststatus:** Status determining if a visit has been completed or not.
   - **syncdown, syncup, syncdowndatetime:** These fields track the synchronization status and timing of visit records: syncdown shows if data was downloaded to the device, syncup if it was uploaded to the server, and syncdowndatetime logs the last download timestamp.

   This dataset supports route optimization by providing insights into visit frequency and duration, refining future scheduling.

Additional Datasets Needed

To enhance Rahguzar's functionality, the following additional datasets may be required:

1. **Order Booker Shifts Dataset**
   This dataset contains information on the availability and shifts of order bookers. Key fields include:

   - **bookerid:** Unique identifier for each order booker.
   - **shiftstarttime and shiftendtime:** Start and end times of each shift.
   - **availabilitystatus:** Indicates whether the booker is available or unavailable.

This dataset is essential for scheduling routes within the working hours of each order booker, optimizing route generation according to workforce availability.

2. **Geographic Data**
   Geographic and road network data provide detailed map and routing information, potentially sourced via the Google Maps API or GIS data services. Key components include:

   - **roadnetwork:** Map of roads, including distances and travel times.
   - **trafficpatterns:** Historical traffic data for congestion analysis.
   - **geofencingdata:** Predefined boundaries for specific zones, regions, or territories.

   This dataset enables precise route calculations, factoring in real-time or historical traffic conditions to minimize travel time.

3. **Store Activity Requirements**
   This dataset details the specific activities and time requirements for each store visit, allowing Rahguzar to optimize for different tasks performed at each store. Key fields include:

   - **storecode:** Identifier for each store.
   - **activitytype:** Type of activity required (e.g., stock check, merchandising).
   - **estimatedtime:** Average time required to complete each activity.

   This dataset supports the system's ability to tailor visit durations based on store requirements, leading to more accurate time estimates for each route.

Data Utilization in Rahguzar

Together, these datasets will support Rahguzar's core functionalities, including:

- **Efficient Route Planning:** Leveraging store locations, operational statuses, and road networks for optimal routing.
- **Dynamic Scheduling:** Using order booker shifts and activity requirements to generate flexible schedules aligned with store needs and workforce availability.
- **Traffic and Geographic Optimization:** Utilizing geographic data to plan routes that avoid congestion and optimize for specific zones or territories.

- **Geographical Filtering:** Geographical and channel-based categorization from the Store Hierarchy dataset will help managers plan routes tailored to specific regions and sales priorities.
- **Route Efficiency Analysis:** The Visits dataset will provide insights into average visit durations and travel times, which will inform the route optimization algorithm and support KPI tracking for system performance.

Further details and data will be incorporated as they become available from SalesFlo.

3.6 System Diagram

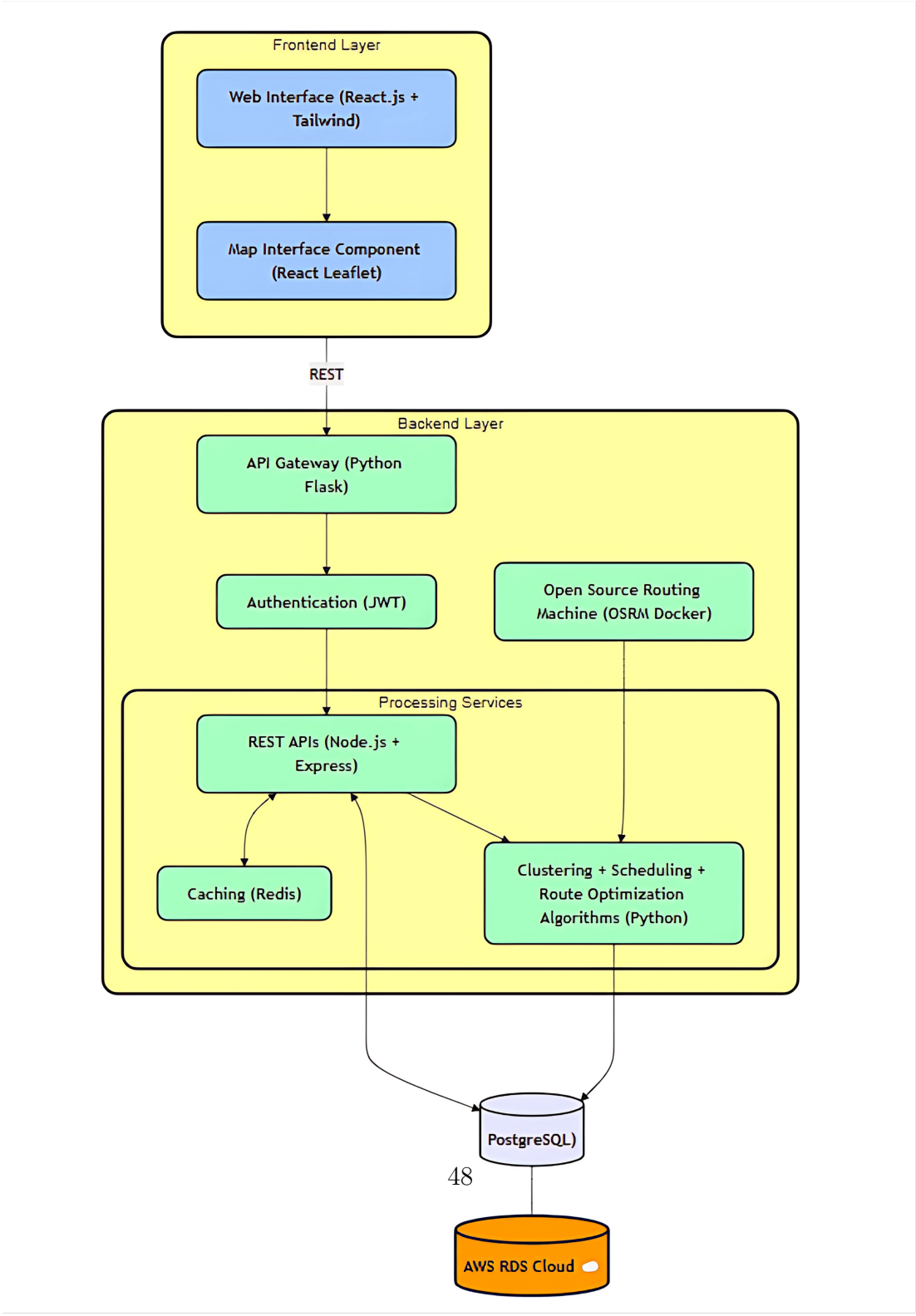

Figure 3.10: Rahguzar's System Architecture Diagram

4. Software Design Specification (SDS)

This chapter provides important artifacts related to design of our project. It includes the Software and Data design of our project.

4.1 Software Design

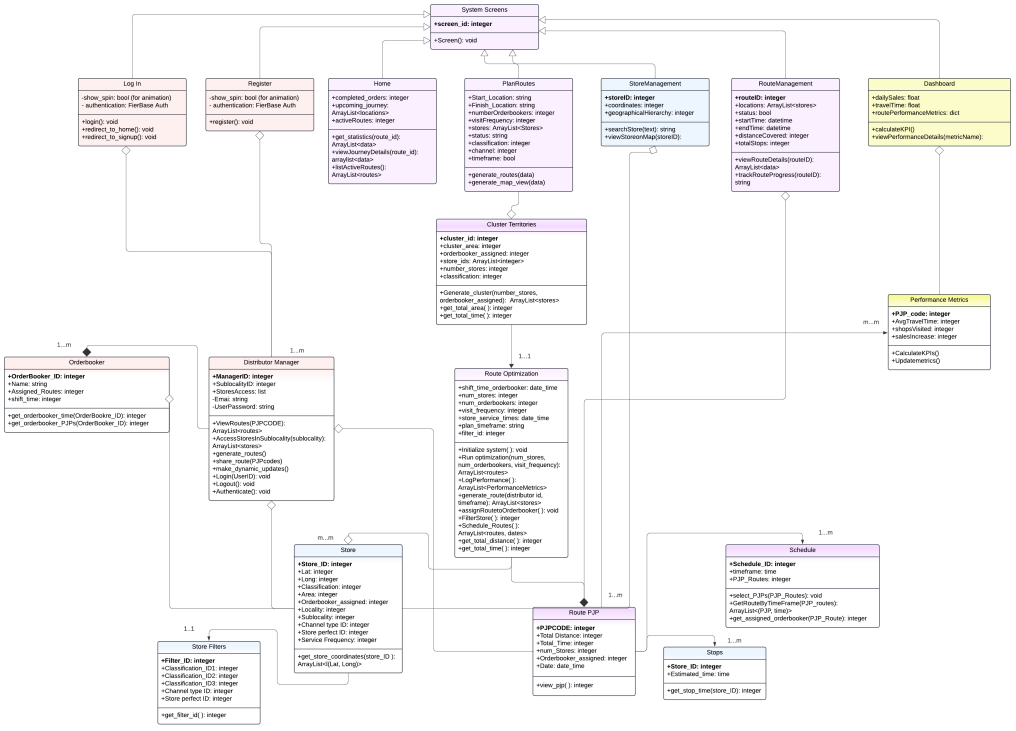

Figure 4.1: UML Class Diagram

As shown in Figure 4.3, the UML diagram starts off by showing the screens that are connected to the system. The Register screen creates the object for the distributor manager, giving them access to the system and screens mentioned such as home, plan routes, route management, and etc. Next, the user has the option to Plan Routes from that screen where they give certain parameters mentioned in the attributes. This creates a cluster object for different areas containing different stores. Next, the optimal PJP route will be determined in each cluster that creates the object for clusters. After this, the stores in each cluster will be connected to form a route, creating a routes class that is the optimal PJP plan. These routes can also be viewed on a schedule view to visually analyze how the upcoming PJPs look like. Moreover, a separate stores table is connected to the child table of stores filters that provides the further classification of each store id. Lastly, the dashboard screen gets the KPIs from the Performance Metrics class to display how the performance of each PJP is and to visualize it in graphs and display the metrics.

4.2 Data Design

This section presents the structure of our database that caters to persistent data storage in our project. The structure is shown as a normalized data model for relational databases. It clearly shows entities, attributes, relationships with their cardinalities, and primary and foreign keys. We have used Postgresql ERD to build our data model.

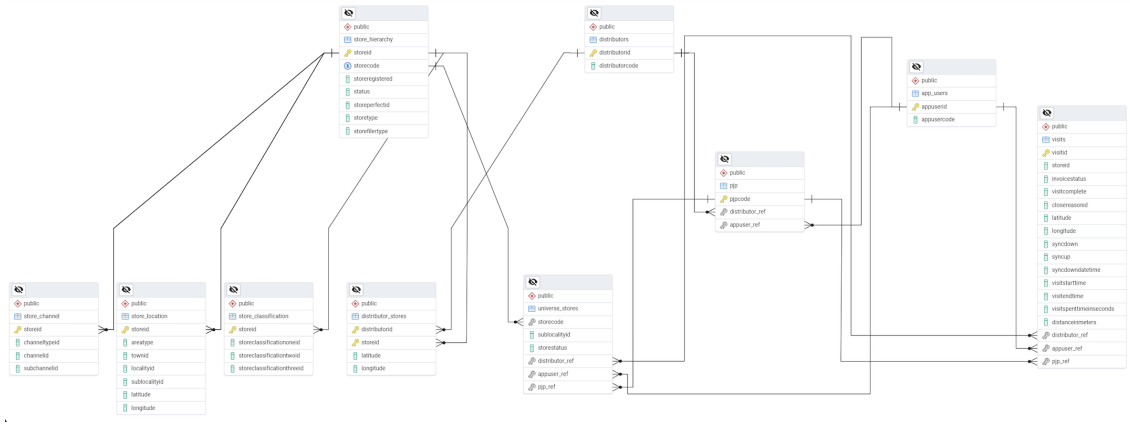

Figure 4.2: ERD Diagram

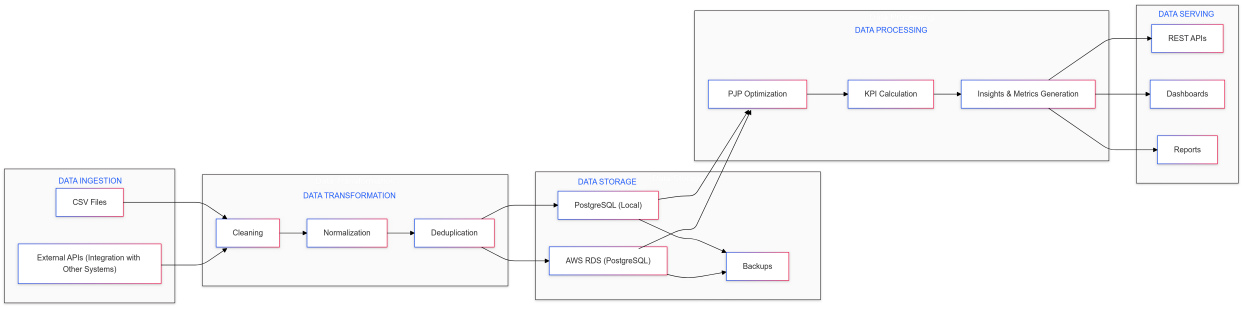

Figure 4.3: End-to-end data pipeline showcasing ingestion, transformation, storage, processing, and serving

4.3 Technical Details

As shown above, our project includes an ERD created based on the cleaned data. Additionally, following are the technical aspects of the algorithm (Evolutionary Algorithm) utilized so far.
The primary objectives of our algorithm are:

- Maximizing the number of shops visited within the constraints.
- Minimizing the total distance traveled to improve route efficiency.
- Minimizing overall time spent, which includes travel and time spent at each store.

The constraints of our algorithm are:

- **Number of order bookers**: Determines the workload distribution and affects route planning.
- **Shift times**: Incorporates break times, holidays, and working hours.
- **Duration at each store**: This is a fixed value for each visit and must be accounted for in total route time.
- **Store priority**: A priority metric for stores, which could be calculated based on sales data.

These objectives and constraints guide the design and functionality of our route optimization algorithm. The algorithm dynamically adapts to these parameters to give optimal route plans.

5. Methodology

The methodology for Rahguzar was designed as a structured, research-driven response to the operational inefficiencies in manual Permanent Journey Plan (PJP) scheduling. The project aimed to generate optimized PJPs that adhered to real-world constraints in the FMCG distribution landscape of Pakistan. Given the scale and complexity of the problem, we adopted a modular and algorithmically hybrid approach, structured into three main phases: Clustering, Scheduling, and Route Optimization.

Our methodology combined custom algorithm development with empirical evaluation on real-world datasets. This allowed us to balance theoretical rigor with practical usability, ensuring that the final solution could be deployed under real operational conditions.

5.1 Problem Formulation

The PJP problem is modeled as a weekly, constraint-aware variation of the VRPTW. Given a set of stores with known coordinates, visit frequencies, service durations, and territory assignments, the objective is to generate an optimized journey plan for each order booker such that:

- Store visits satisfy required weekly frequencies
- Visits are spaced evenly across working days
- Total route time (travel + service) per day remains within shift limits (typically 8 hours)
- Order booker workload is balanced across the team
- Routes are geographically compact and operationally realistic

Due to the NP-hard nature of the problem, exact optimization methods are impractical at city scale. Instead, we use a hybrid, heuristic-based approach structured into three phases.

5.2 Three-Phase Algorithmic Architecture

5.2.1 Clustering

The clustering phase assigns stores to order bookers through a hybrid clustering pipeline that balances geographic proximity with workload equity. It includes:

- **Graph-Based Clustering:** A minimum spanning tree (MST) is constructed from store locations to group nearby stores into initial clusters.
- **K-Means Refinement:** Each graph-based cluster is refined by minimizing intra-cluster variance. The centroid C_i of each cluster is updated as:

$$C_i = \frac{1}{N_i} \sum_{x \in S_i} x$$

- **Outlier Reassignment:** Stores that lie significantly far from their assigned cluster centroid are identified using:

$$D_{s,c} > \mu + \sigma \cdot k$$

  and reassigned to the nearest neighboring cluster.

- **Workload Balancing:** Store workloads are redistributed by minimizing the squared workload error:

$$SSE = \sum_{c} (W_c - \bar{W})^2$$

5.2.2 Scheduling

After clustering, each order booker's list of stores is scheduled across the working week using a custom *Evolutionary Algorithm (EA)*. The EA process includes:

- **Population Initialization:** Initial schedules are generated randomly and based on geographic heuristics.

- **Fitness Evaluation:** Each schedule is evaluated using:

$$F = T_{\text{total}} + P_{\text{mismatch}} + P_{\text{imbalance}} + P_{\text{geo}}$$

- **Constraints:** Daily workload must satisfy:

$$T_{\text{day}} = \sum_{i=1}^{n} T_{\text{travel},i} + \sum_{i=1}^{n} T_{\text{service},i} \leq 480$$

- **Crossover and Mutation:** Tournament selection, crossover, and mutation operations are used to refine schedules across generations.

5.2.3 Route Optimization

For each day's assigned store list, visits are sequenced using *Ant Colony Optimization (ACO)*. The probability of visiting store j from i is:

$$P_{ij} = \frac{(\tau_{ij})^{\alpha}(\eta_{ij})^{\beta}}{\sum_{k \in N_i}(\tau_{ik})^{\alpha}(\eta_{ik})^{\beta}}$$

Pheromone trails are updated as:

$$\tau_{ij} = (1 - \rho) \cdot \tau_{ij} + \sum_{\text{ants}} \Delta\tau_{ij} \quad \text{where} \quad \Delta\tau_{ij} = \frac{Q}{L_k}$$

Parameters used:

- $\alpha = 1$, $\beta = 3$ (pheromone and distance influence)
- $\rho = 0.1$ (evaporation rate), $Q = 1$ (pheromone deposit factor)
- L_k: route length of ant k

Distances and travel times are computed using the OSRM backend for real-world accuracy.

5.3 Data Preprocessing Strategy

The input datasets were provided by SalesFlo in Excel format. The preprocessing pipeline involved:

- Parsing and cleaning using Python
- Merging datasets by unique store identifiers
- Migrating structured data into a PostgreSQL database
- Filtering stores by operational zones for the pilot phase

This ensured consistency and readiness for integration into the optimization pipeline.

5.4 Assumptions

To maintain focus and computational tractability, the following assumptions were made:

- Static average travel times are used; live traffic is not modeled
- Each route begins and ends at a fixed distributor
- Store service durations are known and fixed
- Planning is performed on a weekly horizon excluding Sundays

5.5 Algorithmic Complexity

Rahguzar's modular pipeline breaks the PJP problem into three heuristically optimized stages: clustering, scheduling, and routing. While the overall problem is NP-hard, each component is designed to ensure scalability through local heuristics and bounded iterations.

- **Clustering:** Graph-based clustering is performed using pairwise haversine distances with complexity $O(n^2)$, followed by K-Means refinement ($O(nki)$) and outlier reassignment ($O(n)$). This phase is executed once and scales well for realistic values of n (number of stores) and k (number of order bookers).

- **Scheduling (EA):** The Evolutionary Algorithm runs for G_{EA} generations (typically 80–100) with a population size P (typically 40–50). Fitness evaluation includes total route time, visit frequency matching, and workload balancing. Complexity per generation is $O(P \cdot n)$, giving:

$$O(P \cdot n \cdot G_{EA})$$

- **Routing (ACO):** For each daily route of r stores (where r is typically 10–25), Ant Colony Optimization runs m ants (usually 6–7) for G_{ACO} generations (15–20 iterations). The per-route complexity is:

$$O(m \cdot r^2 \cdot G_{ACO})$$

  Over all n stores (divided across clusters and days), the total routing complexity is approximated as:

$$O(n \cdot m \cdot r \cdot G_{ACO})$$

Total System Complexity:

Combining the three stages, the total time complexity of Rahguzar's algorithm is:

$$O(n^2 + nk + P \cdot n \cdot G_{EA} + n \cdot m \cdot r \cdot G_{ACO})$$

This formulation reflects both theoretical bounds and empirical performance. In practice, early stopping, parallelism, and per-cluster decomposition significantly reduce runtime. Rahguzar consistently generated optimized plans for hundreds of stores in under 5 minutes on a standard multi-core machine.

6. Development

The development of Rahguzar was guided by the goal of building a robust, scalable, and user-friendly platform capable of automating Permanent Journey Plan (PJP) scheduling in FMCG field operations. The project was developed in phases, each aligning with different milestones from backend API construction to frontend deployment, database integration, and cloud migration.

6.1 Backend Development and API Architecture

The backend of Rahguzar was developed using Flask, a lightweight Python-based microframework. Flask was chosen for its flexibility and simplicity, allowing us to quickly develop RESTful APIs that catered to the core functionalities of route generation, user management, and data synchronization. The API layer served as the communication bridge between the frontend and the backend logic, as well as with third-party services like Google Maps.

The route optimization engine was integrated into this backend stack using custom Python scripts and OR-Tools for solving routing problems. This engine consumed input parameters—such as store locations, visit frequencies, shift timings, and the number of available order bookers—and returned optimized PJPs. These responses were served through API endpoints that exposed functionalities for plan generation, route export, real-time updates, and manual overrides.

Throughout development, emphasis was placed on designing clean, modular API endpoints. Authentication and authorization mechanisms were implemented using JSON Web Tokens (JWT), ensuring secure access control for different types of users, particularly field managers.

6.2 Frontend Development with React

The frontend was implemented using React, which offered a highly responsive and component-driven interface for interacting with Rahguzar's planning and monitoring features. React's ability to efficiently manage state and update the DOM in real-time was crucial for building features like dynamic map rendering, drag-and-drop store reordering, and dashboard KPI tracking.

The user interface included several key screens such as the sign-in and registration pages, a dashboard with route statistics, a map-based route planner, and modules for viewing stores and journeys. Special attention was given to providing managers with a seamless experience for planning, editing, and visualizing routes, supported by an interactive map powered by the Google Maps API.

Frontend and backend integration was handled via asynchronous API calls using Axios. These calls ensured real-time feedback during route planning sessions, allowing users to view updated PJPs instantly after adjusting route parameters or filters.

6.3 Database Design and Integration

Initially, PostgreSQL was selected as the relational database for managing store profiles, visit logs, user data, and system configurations. The schema was normalized and designed to accommodate the system's growing need for relational queries, especially during the optimization phase where data consistency and referential integrity were crucial.

The data design followed an entity-relationship model, with core tables representing stores, visits, routes, and users. Each route was linked to a cluster of stores and assigned to specific order bookers based on availability and priority. Visit records captured timestamps, sync states, and performance metrics for later analysis.

During development, we encountered limitations with local hosting, especially as the dataset size and user load increased. This led to the decision to migrate the database to a cloud-hosted environment.

6.4 Migration to AWS RDS

To ensure scalability, reliability, and availability, the PostgreSQL database was migrated to Amazon Web Services Relational Database Service (AWS RDS). AWS RDS

provided automatic backups, failover capabilities, and monitoring tools that were critical as the project moved into pilot testing and real-world deployment phases.

The migration involved exporting local database snapshots and restoring them to a provisioned PostgreSQL RDS instance. We reconfigured the Flask backend to connect to the cloud-hosted instance, updated environment variables for security, and used role-based permissions for managing database access. After thorough testing, this transition significantly improved system performance, particularly in handling concurrent API requests and large data queries during route generation and visualization.

AWS RDS also allowed for better integration with other cloud services, laying the foundation for scaling Rahguzar beyond the initial pilot regions.

6.5 Integration with External Services

Mapping functionality was powered by the Google Maps API, which was integrated into both the backend and frontend layers. On the backend, it was used to compute distance matrices between store locations for the route optimization engine. On the frontend, it enabled real-time map rendering, location-based filtering, and drawing of optimized route paths.

Additionally, Geopy was used for geospatial computations, such as distance calculations and reverse geocoding, complementing the Google Maps integration. These services allowed for a smooth and data-accurate user experience when planning and viewing routes.

6.6 Testing and Deployment

Throughout development, both unit and integration tests were written to ensure API stability and frontend-backend communication integrity. Postman and pytest were used for backend testing, while Cypress and manual QA testing supported frontend validation.

The final deployment used an EC2-based architecture for the backend Flask server, React served via an S3 bucket and CloudFront, and PostgreSQL managed through AWS RDS. This cloud-based setup ensured high uptime, fast response times, and readiness for real-world deployment at scale.

7. Experiments and Results

7.1 Algorithmic Phases and Experiment Results

This section presents the methodology and results from a series of experiments conducted to determine the optimal combinations of algorithms for different phases of the optimization process.

7.1.1 Phase 1: Hybrid Clustering Approach

The first aim of the algorithm is to divide the list of stores to be serviced into a number of clusters which is determined by the number of orderbookers. The objective is to group close stores in the same cluster, so that the orderbookers can service them as the scheduler will decide. To achieve this, clustering algorithms were tested on their own adn in a hybrid approach.

Clustering Algorithms Tested

The following clustering algorithms were evaluated for grouping stores among orderbookers based on geographical proximity and workload balancing:

- **KMeans Clustering**
  - Groups stores based on latitude and longitude using KMeans.
  - Minimizes the sum of squared distances to cluster centroids.
  - Post-clustering, stores are reallocated from overloaded to underloaded clusters to balance workload.
- **Hierarchical Clustering**
  - Uses Ward's linkage to merge clusters based on minimum variance increase.

  - Clusters formed by cutting the dendrogram at desired number of orderbookers.
  - Produces compact and cohesive geographic clusters.

- **Gaussian Mixture Models**
  - Models clusters as Gaussian distributions with flexible shapes and orientations.
  - Assigns stores to clusters based on probability of belonging to a Gaussian component.
  - Covariance type set to "full" to allow non-spherical clusters.

- **Hybrid Approach 1 (Graph-Based Clustering + KMeans)**
  - First forms clusters using graph-based connectivity within a distance threshold.
  - Then refines cluster centers using KMeans to match desired number of clusters.
  - Ensures geographic cohesion and adjusts for cluster count.

- **Hybrid Approach 2 (Graph-Based Clustering + KMeans + Geographical Constraints)**
  - Builds on Hybrid 1 by detecting and reassigning outlier stores.
  - Outliers identified by distance from cluster centroid beyond a defined threshold.
  - Reassigned to nearest cluster to improve cohesion and balance.

Moreover, to ensure that optimal clusters were being made, cluster balancing was employed to redistribute stores between clusters to ensure workloads—based on service effort and travel time—are roughly equal. Overloaded clusters transfer nearby stores to underloaded ones, and this process repeats until workloads fall within a set tolerance or a maximum number of iterations is reached, promoting fair and efficient distribution.

7.1.2 Phase 1: Experiments Results

The silhouette score measures how well points are grouped in a clustering task. It checks if points are close to others in their own cluster, indicating a good score, or if they are far from points in other clusters (also good score). The score ranges from -1 to 1, where 1 means perfect clusters, 0 means overlap, and negative scores indicate that points might be in the wrong cluster.

Distributor ID	KMeans	Gaussian Mixture Models	Hierarchical	Hybrid Approach-1	Hybrid Approach-2
1	0.3537	0.2835	0.3537	0.5044	0.5044
6	0.1839	0.2130	0.1839	0.1839	0.1684
7	0.2737	0.3376	0.3892	0.3980	0.4437

Table 7.1: Silhouette Score Comparison for Different Clustering Algorithms

From the results in Table 7.1, it can be observed that the Hybrid Approach-2 outperformed all other clustering algorithms, achieving the highest silhouette score of 0.5044 for Distributor 1 and 0.4437 for Distributor 7. This indicates that the clusters formed using this approach were more cohesive and well-separated compared to those formed by the other algorithms. Moreover, it finds a good balance between workload distribution and geographic proximity. For example, for distributor ID 7, there's a little variance in the clusters being formed however, it has the highest silhouette score as geographical proximity has been set as the priority.

7.1.3 Phase 2: Scheduling with Evolutionary Algorithm (EA)

Once clusters have been formed in Phase 1, the next critical step is to generate a feasible schedule that defines which shops should be visited on each day of the planning period (either a single day or a custom range, as selected by the user). The primary objective during scheduling is to balance the workload across both the available days and all assigned orderbookers. This ensures that no orderbooker is disproportionately burdened and that their daily tasks are distributed as evenly as possible.

To identify the most effective scheduling strategy, a series of experiments were conducted using various algorithms. Each algorithm was evaluated based on its ability to generate balanced, constraint-compliant schedules. At the core of this evaluation process is the fitness function.

The fitness function plays a vital role in determining how practical and efficient a schedule is. It computes a cost value for each candidate schedule—lower values indicate better solutions. Hard constraints are applied first: for example, if the total

route time on any day exceeds a predefined daily limit (e.g., 480 minutes), a substantial penalty is applied to immediately discourage infeasible schedules. It also checks for visit mismatches—stores not visited the required number of times—penalizing any deviations.

Additionally, soft constraints are incorporated to further refine the schedule. These include a day-balancing penalty when daily route times fall outside the ideal range (e.g., 360–420 minutes), and a geographical penalty when neighboring stores requiring only one visit are assigned to different days. These constraints collectively guide the algorithm toward generating schedules that are not only valid but also optimized for balance, practicality, and real-world efficiency. The following scheduling algorithms were used:

- Simulated Annealing
- Ant Colony Optimization (ACO)
- Mixed Integer Linear Programming (MILP)
- Evolutionary Algorithm (EA)

Through a series of experiments, the best combination for the scheduling phase was determined, which was Evolutionary Algorithm (EA) for scheduling. Algorithms like ACO showed that the in serach for the optimal schedule, it can often get trapped in a local optima. For MILP and simulated annealing, generating a schedule can be computationally expensive and time consuming. On the other hand, EA showed optmial results with genetic diversity maintained that avoids premature convergence, hence EA is the best choice for scheduling.

7.1.4 Phase 3: Route Optimization with Ant Colony Optimization (ACO)

After a schedule has been created, the next step is to optimize the routes for each orderbooker. This involves determining the most efficient path for each orderbooker to follow, ensuring that they can visit all assigned stores in the shortest possible time while adhering to any constraints (e.g., time windows, vehicle capacity). The goal is to minimize travel distance and time while maximizing efficiency. This phase is crucial for ensuring that the orderbookers can complete their routes within the allocated time and resources, ultimately leading to improved service levels and reduced operational costs. Moreover, it helps determine the order of visiting the stores and

the best route to take, considering factors such as road networks and other logistical constraints. For route optimization, several algorithms were tested to identify the most effective one. The route optimization algorithms evaluated were:

- Particle Swarm Optimization (PSO)
- Ant Colony Optimization (ACO)
- Google Optimization Tools (OR-Tools)
- Evolutionary Algorithm (EA)
- Dynamic Programming

It was found that the Ant Colony Optimization (ACO) route optimizer performed the best across all tested route optimization algorithms. While POS lead to explorative results, it would often get trapped in local optimas and not perform up to the optimal mark. Using Google OR-Tools and Dynamic Programming showed that performance would degrade due to the very large datasets and complex constraints involved. Whereas ACO's pheromone mechanisms lef to effective exploration of the solution space, allowing high-quality route optimization solutions in a reasonable time frame.

7.1.5 Phase 2 and 3: Experiments Overview

A series of experiments were conducted across different distributors with varying numbers of stores and orderbookers. The following distributors were involved in the experiments:

Distributor ID	**Number of Stores**	**Number of Orderbookers**
1	395	2
6	426	4
7	580	5
131	42	1

Table 7.2: Distributors and their Assigned Stores and Orderbookers

The purpose of the experiments was to determine the best combinations of scheduling and route optimization algorithms for each distributor. The experiments were designed to evaluate the performance of different algorithmic combinations in

terms of distance and travel time minimization. This was done by analyzing the total distance and the total traveltime minimized across all orderbookers (results show average of all distributors), and the stores serviced distribution per orderbooker for each day.

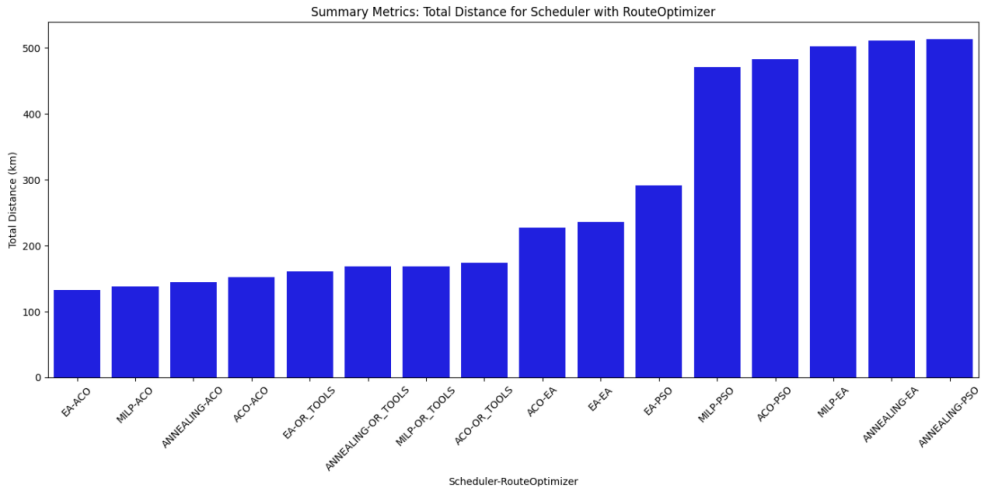

Figure 7.1: Total distance for all distributors compared over different combinations.

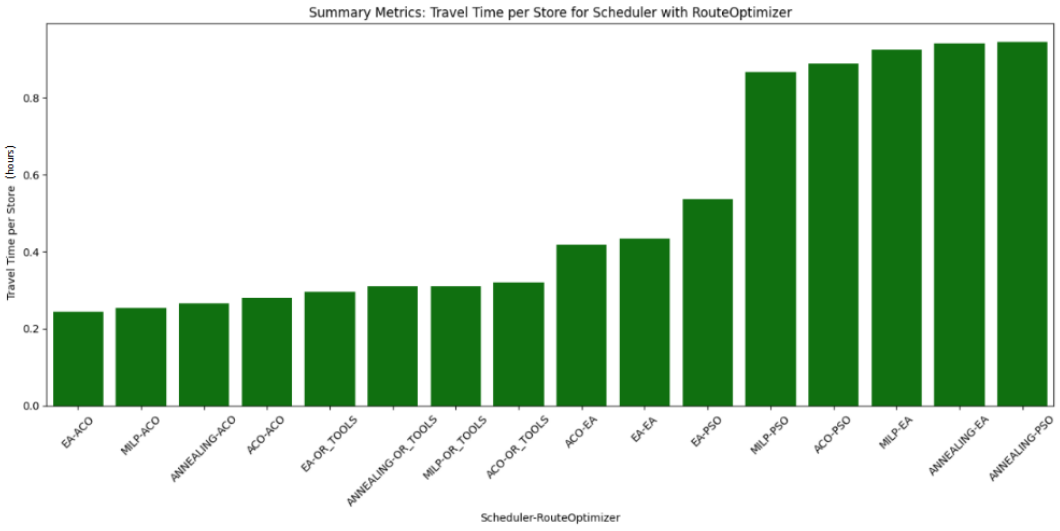

Figure 7.2: Total time for all distributors compared over different combinations.

The results in Figure 7.1 and Figure 7.2 show that the best scheduler-route optimizer combination was EA-ACO, minimizing the distance travelled and time taken to travel to as less as 150 km and 0.2 hours, as compared to almost 500 kms and 1 hour for the worst combination. It can be observed that ACO is the top 4 best route optimizers, whereas the least optimal performance was showed EA nad PSO route optimizers, and MILP and simulated annealing schedulers.

7.1.6 Phase 2 and 3: Experiment Results

The experiments were conducted by combining different scheduling algorithms with different route optimization algorithms. The findings of each graph is summarized below.

The best performing scheduler-route optimizer combinations for each distributor are as follows:

Distributor	Best Distance Minimized Scheduler-Route Optimizer	Best Travel Time Minimized Scheduler-Route Optimizer
Distributor 7	EA-ACO	EA-ACO
Distributor 6	EA-ACO	EA-ACO
Distributor 1	EA-ACO	EA-ACO
Distributor 131	ACO-EA	ACO-EA

Table 7.3: Best performing Scheduler-Route Optimizer combinations by distributor

7.1.7 Conclusion

Based on the results from the experiments, the combination of Evolutionary Algorithm (EA) for scheduling and Ant Colony Optimization (ACO) for route optimization proved to be the best-performing solution, as evidenced by its consistent performance across all distributors, particularly in minimizing both distance and travel time.

The ACO route optimizer, in particular, outperformed all other route optimization algorithms across the experiments. For distributor 131, however, the ACO-EA combination was identified as the best solution. This highlights the importance of considering both the scheduling and route optimization algorithms in tandem when evaluating their overall performance in real-world scenarios.

8. Conclusion and Future Work

8.1 Conclusion

In this project, we presented Rahguzar, a comprehensive solution for optimizing Permanent Journey Plan (PJP) scheduling in the Fast-Moving Consumer Goods (FMCG) industry. Our system integrates customer clustering, schedule generation via Evolutionary Algorithms, and route optimization using Ant Colony Optimization (ACO), delivering a robust end-to-end pipeline for sales team planning and management.

Through the integration of real-world datasets and a user-friendly web-based interface, Rahguzar enables dynamic and efficient route planning, helping sales managers minimize travel costs while maximizing coverage and visit compliance. Performance evaluations demonstrate the system's ability to produce realistic, optimized schedules with significantly improved route efficiency compared to manually designed plans.

The modular architecture allows easy adaptation to various geographical scales, constraints, and business-specific requirements. Moreover, our backend pipeline has been successfully deployed and integrated with a React-based frontend, creating a seamless experience for end users.

8.2 Future Work

In future iterations of Rahguzar, several enhancements can be explored to improve its effectiveness and integration with real-world FMCG workflows. One key direction is the incorporation of real-time traffic data and dynamic scheduling capabilities, enabling the system to adapt routes on the fly in response to disruptions such as delays or cancellations. Integration with Salesflo's existing mobile platform—which is already widely used by field agents—can also be explored to enable seamless access

to route plans, check-ins, and live updates. This would allow on-ground activities to be synchronized with the backend optimization engine, enhancing both usability and data flow. Another promising direction is the use of historical and real-time sales data to drive scheduling decisions. By aligning visit frequency and prioritization with customer-specific sales performance, seasonal trends, and product movement, Rahguzar can help optimize not only travel efficiency but also commercial outcomes. Together, these enhancements can transform Rahguzar into a fully adaptive, data-driven system for sales force automation.

9. Reflections

9.1 Individual Reflections

9.2 Team Reflection

Working together on Rahguzar was a deeply collaborative experience that taught us the value of shared vision, consistent communication, and mutual accountability. While we were united by a shared goal, the process of getting there involved navigating a variety of challenges that tested our coordination, communication, and adaptability. One of the most significant challenges we encountered was working at different paces. Each team member had their own academic and personal commitments, and synchronizing our work schedules proved difficult at times. There were also disagreements on how to approach certain parts of the problem, particularly in algorithm design, interface behavior, and system architecture. At one point, we struggled to align our approaches to problem-solving, particularly in how to structure the optimization logic, divide backend responsibilities, and sequence integration with the frontend. Each of us had different interpretations of the problem and varying ideas on how best to implement certain features. These disagreements sometimes led to delays and overlapping efforts. However, they also turned out to be important learning opportunities. What helped us move forward was our ability to step back, have honest discussions, and actively listen to one another. The guidance and support we received from our supervisor and mentors during these phases were especially valuable. Their input helped us untangle complex decisions, identify a shared direction, and turn moments of friction into productive design conversations. Despite the differences in working styles and technical opinions, we learned to trust each other's strengths and make space for every perspective. As the project progressed, we grew more coordinated and developed a rhythm that allowed us to work more effectively together. Ultimately,Rahguzar was not just a technical achievement, but a product of collective resilience, trust, and shared commitment to solving a real-world problem

together.

9.3 Process Reflection

Looking back at the process, one of the things that worked particularly well was our decision to follow an iterative and research-driven development approach. Early in the project, we focused on understanding the problem deeply through industry interviews, literature review, and small-scale experiments. This helped us avoid jumping prematurely into implementation and instead guided our design with clarity and purpose. Another strength was our consistent engagement with real-world data and constraints. By validating our design choices against actual operational needs provided by SalesFlo, we ensured that our solution stayed grounded and practical.

However, we also faced challenges that impacted our efficiency. A major bottleneck occurred when all of us were working on different parts of the codebase simultaneously. While this parallel effort helped accelerate feature development, it made integration complex and time-consuming. Merging work from multiple branches while ensuring compatibility and system stability required multiple debugging sessions and technical compromises. In hindsight, a more modular and version-controlled integration plan from the start could have reduced friction.

We also underestimated the time needed for certain tasks, particularly data integration and testing. Delays in data availability affected our ability to validate early outputs, and unexpected inconsistencies in the dataset required additional preprocessing.

Despite these challenges, the process evolved and improved over time. We became more deliberate in dividing tasks, more consistent in communication, and more disciplined in tracking progress. This experience highlighted the importance of not just technical knowledge, but also process design and team coordination in building real-world systems.

9.4 Plan vs Achievment

Appendix A. More Math

Here, we describe the background math for the techniques used in the text.

Appendix B. Data

Here is a dump of our 2TB data set. Enjoy!

Appendix C. Code

Here is our code.

```
print('Hello World!')
print('Computing true random number.')
print('Capturing interstellar radiation.')
print('This will take time!')
import random
import time
time.sleep(3600*random.randint(1,10))
print(4)
```

Our code can be found at `https://github.com/habib-university/Kaavish-Template`.

References

[1] Mohammad Khurshed Alam and Mohd Herwan Sulaiman. "Optimal loading analysis with penalty factors for generators using brute force method". In: *Lecture Notes in Electrical Engineering* (2022), pp. 37–46. DOI: 10.1007/978-981-16-8690-0_4.

[2] Alexis. *How route optimization can save time and money*. Dec. 2023. URL: http://gofleet.com/how-route-optimization-can-save-time-and-money/.

[3] Dalia Essa Almaatani. "New Computational Approaches for the Transportation Models". Accessed: 2024-12-06. MA thesis. Laurentian University of Sudbury, Sept. 2014. URL: https://laurentian.scholaris.ca/items/3c4e7436-4b97-4664-9289-957f8690fddf.

[4] Karina Corona-Gutiérrez, Samuel Nucamendi-Guillén, and Eduardo Lalla-Ruiz. "Vehicle routing with cumulative objectives: A state of the art and analysis". In: *Computers & Industrial Engineering* 169 (2022), p. 108054. ISSN: 0360-8352. DOI: 10.1016/j.cie.2022.108054. URL: https://www.sciencedirect.com/science/article/pii/S0360835222001243.

[5] Xin Cui. "Logistics cost control research of FMCG industry". In: *Advanced Materials Research* 468-471 (Feb. 2012), pp. 1866–1869. DOI: 10.4028/www.scientific.net/amr.468-471.1866.

[6] Guy Desaulniers, Jacques Desrosiers, and Marius Solomon. "Accelerating strategies for column generation methods". In: *Transportation Science* 36.2 (2002), pp. 213–226. DOI: 10.1287/trsc.36.2.213.5720.

[7] Dominique Feillet et al. "An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems". In: *Networks* 44.3 (2004), pp. 216–229. DOI: 10.1002/net.20034.

[8] Shangce Gao et al. “Ant colony optimization with clustering for solving the dynamic location routing problem”. In: *Applied Mathematics and Computation* 285 (July 2016). Available online 16 April 2016, pp. 149–173. DOI: 10.1016/j.amc.2016.04.021. URL: https://doi.org/10.1016/j.amc.2016.04.021.

[9] Abel Garcia-Najera and John A. Bullinaria. “An improved multi-objective evolutionary algorithm for the vehicle routing problem with time windows”. In: *Computers & Operations Research* 38.1 (Jan. 2011), pp. 287–300. DOI: 10.1016/j.cor.2010.05.004.

[10] Keivan Ghoseiri and Seyed Farid Ghannadpour. “Multi-objective vehicle routing problem with time windows using goal programming and genetic algorithm”. In: *Applied Soft Computing* 10.4 (Sept. 2010), pp. 1096–1107. DOI: 10.1016/j.asoc.2010.04.001.

[11] Martin Hughes, Marc Goerigk, and Trivikram Dokka. “Particle swarm metaheuristics for robust optimisation with implementation uncertainty”. In: *Computers & Operations Research* 122 (Oct. 2020), p. 104998. DOI: 10.1016/j.cor.2020.104998.

[12] Umair Javed. *The Undocumented Wholesale-Retail Sector*. Insights for Change. Accessed: 2024-12-06. Consortium for Development Policy Research, Oct. 2023. URL: https://cdpr.org.pk/wp-content/uploads/2023/10/The-Undocumented-Wholesale-Retail-Sector.pdf.

[13] Philip Kilby, Patrick Prosser, and Paul Shaw. “Dynamic VRP using local search heuristics”. In: *Operational Research Society* 53.2 (2002), pp. 235–244. DOI: 10.1057/palgrave.jors.2601298.

[14] A. Kok et al. “Dynamic programming algorithm for the vehicle routing problem with time windows and EC social legislation”. In: *Journal of Chemical Physics - J CHEM PHYS* (Jan. 2009).

[15] Abdullah Konak, David W. Coit, and Alice E. Smith. “Multi-objective optimization using Genetic Algorithms: A tutorial”. In: *Reliability Engineering & System Safety* 91.9 (Sept. 2006), pp. 992–1007. DOI: 10.1016/j.ress.2005.11.018.

[16] Gilbert Laporte. “Vehicle Routing Problems with Time Windows”. In: *Transportation Science* 39.1 (2017), pp. 119–129. DOI: 10.1287/trsc.1050.0127.

[17] Raman Maini and Rajeev Goel. “Vehicle routing problem and its solution methodologies: A survey”. In: *International Journal of Logistics Systems and Management* 28.4 (2017), pp. 419–432. DOI: 10.1504/ijlsm.2017.10008188.

[18] Gábor Nagy and Said Salhi. "Heuristic Algorithms for Single and Multiple Depot Vehicle Routing Problems with Pickups and Deliveries". In: *European Journal of Operational Research* 177.2 (2007), pp. 1620–1641. DOI: 10.1016/j.ejor.2005.10.030.

[19] Beatrice Ombuki, Brian J. Ross, and Franklin Hanshar. "Multi-Objective Genetic Algorithms for Vehicle Routing Problem with Time Windows". In: *Applied Intelligence* 24.1 (Feb. 2006), pp. 17–30. DOI: 10.1007/s10489-006-6926-z. URL: https://link.springer.com/article/10.1007/s10489-006-6926-z.

[20] Wan Othman et al. "Solving Vehicle Routing Problem using Ant Colony Optimisation (ACO) Algorithm". In: *International Journal of Research and Engineering* 5 (Nov. 2018), pp. 500–507. DOI: 10.21276/ijre.2018.5.9.2.

[21] Sandhya and Rajeev Goel. "Multi objective vehicle routing problem: A survey". In: *Asian Journal of Computer Science and Technology* 7.3 (Nov. 2018), pp. 1–6. DOI: 10.51983/ajcst-2018.7.3.1903.

[22] Matthias Schneider, Alexander Stenger, and Dominik Goeke. "The Electric Vehicle-Routing Problem with Time Windows and Recharging Stations". In: *Transportation Science* 48.4 (2014), pp. 500–520. DOI: 10.1287/trsc.2013.0490.

[23] Gerard Sierksma and Gerard A. Tijssen. "Sales Territory Alignment Using Mathematical Programming". In: *European Journal of Operational Research* 105.3 (1998), pp. 567–581. DOI: 10.1016/S0377-2217(96)00497-0.

[24] Andrés Villalba and Elsa Cristina Gonzalez la Rotta. "Clustering and heuristics algorithm for the vehicle routing problem with time windows". In: *International Journal of Industrial Engineering Computations* 13 (2022), pp. 165–184. DOI: 10.5267/j.ijiec.2021.12.002.

[25] Feidiao Yang et al. "Boosting Dynamic Programming with Neural Networks for Solving NP-hard Problems". In: PMLR, 2018, pp. 726–739.

[26] Huizhen Zhang et al. "A hybrid ant colony optimization algorithm for a multi-objective vehicle routing problem with flexible time windows". In: *Information Sciences* 490 (July 2019), pp. 166–190. DOI: 10.1016/j.ins.2019.03.070.

[27] Emir Zunic et al. "A Cluster-Based Approach to Solve Rich Vehicle Routing Problems". In: Mar. 2021, pp. 103–123. ISBN: 978-3-030-71845-9. DOI: 10.1007/978-3-030-71846-6_6.

[28] Emir Zunic et al. “Cluster-based approach for successful solving real-world vehicle routing problems”. In: *Volume 21* (2020), pp. 619–626. DOI: 10.15439/2020F184.