University of Hertfordshire

Computer Science Masters Project

Assignment Title: FPR - Final Project Report
DeliveryGo - Delivery Driver Management System

Module Title:
Computer Science Masters Project

Module Code:
7COM1039

Submitted By:
Nazmul Hossain

Supervisor: Imran Khan 9
i.khan9@herts.ac.uk

Course: MSc Computer Science
SID: 23094184

Table of Contents

MSc Final Project Declaration

This report is submitted in partial fulfilment of the requirements for the degree of Master of Science in Computer Science at the University of Hertfordshire (UH).

It is my own work except where indicated in the report.

I did not use human participants in my MSc Project.

I hereby give permission for the report to be made available on the University of Hertfordshire website provided the source is acknowledged.

Project Title

DeliveryGo – Takeaway Delivery Driver Management System

Apple Store –
https://apps.apple.com/us/app/deliverygo-delivery-driver/id6751056501?platform=iphone

Website: www.deliverygo.co.uk

Introduction

In today's fast-paced food industry, takeaway restaurants rely heavily on delivery drivers to ensure timely service and customer satisfaction. While most restaurants already use till systems and electronic point-of-sale (ePOS) systems to manage their orders, a major gap still exists in managing their delivery operations. The management of drivers—covering work hours, shift allocation, delivery tracking, mileage calculation, wages, and performance reporting—is still done manually in many restaurants. Owners often rely on pen and paper to record delivery times, addresses, and costs, which not only consumes valuable time but also introduces errors and inconsistencies in calculations.

The manual process involves multiple repetitive steps each day. For instance, restaurant owners must first prepare paper templates to record start and end times for drivers, delivery postcodes, and order details. Every new order requires additional manual entries, from noting the delivery address to writing the start and end time of the journey. Drivers must also enter delivery addresses manually into navigation apps, often double-checking to avoid mistakes with similar-sounding street names. Contacting customers adds further delays, as drivers need to copy phone numbers from receipts into their phones. At the end of the day, all receipts are collected, and wages or petrol costs are calculated manually, often leading to inaccurate payments because costs are not based on actual mileage but on rough estimates. For example, a local delivery may be marked as £0.70 even if the distance exceeded one mile, or two deliveries to the same city may vary greatly in distance but are paid the same. This approach results in wasted time, miscalculations, and limited transparency in driver performance.

To address these challenges, the proposed solution is **DeliveryGo**, a software system designed specifically for takeaway restaurant owners to manage their delivery drivers more effectively. The system automates the entire process, from shift management to delivery tracking, reducing dependency on manual entries and eliminating calculation errors. Drivers can start their shifts by simply pressing a button in the app, and each delivery can be logged automatically by taking a photo of the receipt. The app extracts key information such as the delivery address, phone number, and verification code, and records it in the database. With one tap, drivers can access navigation, call customers directly, or update delivery status.

For restaurant owners, DeliveryGo provides a centralized dashboard that displays driver activity, live delivery status, wages, petrol costs, and performance metrics. Unlike the manual method, where calculations are inconsistent and time-consuming, the app ensures accuracy by basing wages and petrol fees on actual mileage and time worked. Reports can be generated daily, weekly, or monthly, offering a clear overview of delivery operations and driver efficiency. Features such as “authorize a driver to work,” right-to-work verification, performance monitoring, and detailed expense tracking further enhance accountability and compliance.

The significance of this system lies in its ability to simulate all tasks currently done manually, but in a faster, more reliable, and cost-effective way. By saving minutes on every delivery, the accumulated time savings over a day, week, or month become substantial. Additionally, accurate wage and mileage calculations build trust between drivers and restaurant owners, while real-time tracking ensures smoother communication and fewer delays. Ultimately, DeliveryGo transforms delivery management from a burdensome manual process into a streamlined digital workflow, enabling restaurants to focus on growing their business and serving their customers more effectively.

Problem Statement

Takeaway restaurants increasingly depend on multiple delivery drivers to meet growing customer demands. While they often employ till and ePOS systems to manage orders, they lack a dedicated platform to manage delivery operations effectively. As a result, critical tasks such as wage calculation, shift scheduling, delivery tracking, and performance monitoring are still handled manually through pen-and-paper records or basic spreadsheets. This outdated approach introduces inefficiencies, financial risks, and communication challenges that affect both business owners and drivers.

Wage Calculation: Weekly wages are calculated manually based on estimated hours, mileage, and deliveries, which is both time-consuming and prone to errors.

Shift Management: There is no automated process to record start and finish times or authorize driver shifts, increasing the risk of unauthorized work or missed shifts.

Delivery Analytics: Restaurants cannot track key performance indicators such as average delivery times, mileage costs, or overall driver efficiency.

Real-Time Monitoring: Owners lack tools to view live driver locations, delivery status, or optimized routes, leading to delays in customer updates and reduced accountability.

Expense Tracking: Petrol and delivery-related costs are logged manually, often leading to inaccurate expense reporting and profit calculation.

Payment Accuracy: Errors in per-mile calculations and wage distribution frequently cause disputes between drivers and owners.

Communication Gaps: Drivers must manually copy customer phone numbers, creating delays in resolving delivery issues.

Route Inefficiency: No tools exist to optimize routes or automatically identify addresses (e.g., through photo recognition), resulting in wasted time and higher fuel costs.

Lack of Transparency: Restaurant owners cannot generate detailed reports on driver performance, delivery statistics, or wage breakdowns, limiting visibility and accountability.

Existing Solutions

Currently, most takeaway restaurants use till systems and ePOS (Electronic Point of Sale) software to manage online and in-house orders, but these systems generally lack features for managing delivery drivers. Some restaurants use third-party delivery platforms like Uber Eats, Deliveroo, or Just Eat, which provide basic driver assignment and tracking features. However, these platforms primarily focus on their own network of drivers and do not cater to restaurants managing their own in-house delivery staff.

Other businesses attempt to fill the gap by using generic tools like Excel spreadsheets, Google Sheets, or manual notebooks to record driver hours, calculate wages, and track deliveries. A few small-scale workforce management apps exist, but they are often not tailored for the unique needs of takeaway restaurants, and most do not offer integration with existing ePOS systems.

Moreover, existing delivery tracking apps focus more on customer-side order tracking and lack features like shift scheduling, time tracking, wage calculation, performance monitoring, and real-time driver-to-owner communication.
The lack of a specialized, affordable, and integrated solution leaves takeaway restaurant owners struggling with inefficiency, inaccuracy, and unnecessary administrative burden.

Aims & Objectives

The aim of this project is to investigate the inefficiencies and challenges of manual delivery driver management in the takeaway restaurant sector, and to design, implement, and evaluate an automated solution — **DeliveryGo** — that addresses these pain points. The study seeks to demonstrate how automation can reduce administrative workload, improve wage and mileage accuracy, enhance operational efficiency, and provide transparency through integrated tracking, communication, and performance monitoring.

To achieve this aim, the following objectives were set:

Analyze operational challenges of manual delivery processes
Examine how traditional paper-based methods (e.g., handwritten logs, manual wage calculation, and fragmented communication) impact efficiency, accuracy, and fairness.

Review existing industry solutions and limitations
Critically assess current delivery management and dispatch tools, identifying gaps in their suitability for small and medium-sized takeaway businesses.

Define system requirements
Establish clear functional and non-functional requirements for an effective solution, focusing on automation, real-time navigation, communication, accurate wage calculation, and scalability.

Design the DeliveryGo framework
Create a conceptual model and system architecture that directly addresses the identified pain points, emphasizing features such as scheduling, tracking, reporting, and automated wage/mileage calculation.

Develop and test a working prototype
Build a prototype of the DeliveryGo system to simulate core functionalities, validating its effectiveness through comparative analysis with the manual process.

Evaluate outcomes against research questions
Compare the manual and automated approaches in terms of time savings, error reduction, wage accuracy, and efficiency, and critically discuss the extent to which the system meets the defined research question.

Investigation

Practical Investigation

The practical investigation was structured to evaluate the limitations of the traditional manual delivery management process and the improvements offered by the DeliveryGo application. This was achieved through a simulation that modelled both workflows using a controlled dataset of 30 customer orders. The simulation aimed to reflect realistic conditions commonly experienced by small takeaway restaurants in the UK, where resources are limited, and managers frequently rely on paper records, phone calls, and manual wage calculations.

The manual process was reconstructed first. This involved preparing daily paper templates, manually entering each order, cross-checking delivery addresses, and contacting customers by phone. Drivers were required to record fuel receipts and submit them to the restaurant owner at the end of each shift. Wages were then calculated using postcode categories rather than precise mileage, reflecting the legacy practices widely reported in restaurant operations (Anderson & Schwieterman, 2018). Time taken for each step was recorded to quantify inefficiencies, while qualitative notes captured error risks such as miscopied phone numbers or incorrect mileage estimates.

Subsequently, the DeliveryGo prototype was simulated with the same dataset. Orders were entered directly into the application, which automatically generated delivery records, tracked mileage via integrated GPS, and calculated wages based on actual distance travelled. This eliminated the need for paper logs, manual phone dialing, and retrospective reconciliation of receipts. Shift management was handled through digital check-in/check-out features, reducing administrative overhead. Metrics such as total preparation time, order entry time, and payment calculation time were collected, alongside observations on error reduction and communication efficiency.

Both scenarios were compared across time, cost, errors, and efficiency, providing a systematic framework for analysis. Time was measured in minutes required to complete routine tasks, cost in terms of driver compensation per delivery, errors in the likelihood of miscommunication or miscalculation, and efficiency as the ability to handle the delivery workload with minimal delays. This comparative design ensured that findings directly addressed the central research question of whether a dedicated digital system could significantly outperform the manual approach in delivery management.

This mixed quantitative–qualitative simulation approach is consistent with best practices in computer science investigation projects, where controlled modelling and comparative analysis are often used to demonstrate the practical implications of digital innovation (Oates, 2006). By replicating both workflows under equivalent conditions, the investigation produced robust and replicable findings suitable for academic evaluation.

Investigation Question

The primary research question for this project is:

"To what extent can a delivery driver management system improve operational efficiency, accuracy in wage and mileage-based calculations, and reduce time spent on administrative tasks compared to the manual methods used by takeaway restaurants?"

To enable a focused and measurable investigation, the research question is operationalized into the following sub-questions, each associated with an evaluation metric:

1. **How does the system improve the accuracy of wage calculations compared to manual methods?**

   Metric: Number of wage calculation errors before and after system implementation.

2. **How much time is saved on administrative tasks such as wage processing, shift tracking, and expense logging?**

   Metric: Average time spent per week on administrative work (manual vs automated).

3. **How does the system impact the ease and speed of locating delivery addresses for drivers?**

   Metric: Time taken to find an address manually vs using automatic address detection or route guidance.

This question was developed in response to consistent challenges reported in the literature on manual delivery management. Small restaurants frequently rely on paper notes, spreadsheets, or informal messaging systems to coordinate shifts and deliveries, which are time-intensive and error-prone (Harri, 2022). Manual wage computation based on postcode categories often results in under- or over-payment of drivers, while the absence of automated mileage tracking limits fairness and accountability (Anderson & Schwieterman, 2018). Furthermore, inefficiencies in locating delivery addresses and communicating with customers contribute to avoidable delays and reduced service quality (Samsara, 2023).

These sub-questions ensure that the investigation remains evidence-driven and replicable, enabling direct comparison between the manual simulation and the DeliveryGo application.

Rationale for the Investigation

The rationale for pursuing this investigation is twofold. First, there is an academic gap: while digital dispatching and automation have been widely studied in large-scale logistics operations (Track-POD, 2023; Onetime 360, 2023), little research has examined their application in small, resource-constrained food businesses. Second, there is a practical gap: many UKS takeaway restaurants still rely on fragmented, outdated methods that cannot scale, lack compliance features, and increase the risk of wage disputes or miscommunication (Brown et al., 2019).

By assessing whether an all-in-one, purpose-built application such as DeliveryGo can streamline wage calculation, optimize route management, and simplify driver administration, this investigation aims to determine whether automation can generate meaningful time and cost savings while improving fairness and accuracy. Addressing this question will therefore contribute to both academic understanding and practical solutions for efficiency in the takeaway industry.

Investigation Level

The investigation for this project will be carried out through a **practical and test-based approach**, focusing on quantitative evaluation using controlled scenarios. No primary data will be collected from human participants, and therefore, ethics approval is not required.

The system will be evaluated by simulating real-world delivery operations, such as wage calculation, delivery tracking, mileage logging, and address detection. Performance will be measured by comparing the automated system's output against results generated from traditional manual processes. Key evaluation criteria will include time saved, accuracy of calculations, and delivery coordination efficiency.

This approach ensures that the investigation remains focused on measurable outcomes, while also reflecting realistic use cases without involving live user testing.

Literature Review (Secondary Research)

Background Research

Delivery driver management in small takeaway restaurants remains a largely manual and fragmented process. The literature highlights several recurring challenges, particularly in scheduling, delivery tracking, wage calculation, and communication within delivery teams. This section reviews existing research and identifies gaps that the proposed **DeliveryGo** system aims to address.

Manual Workforce Management Challenges in Takeaway Delivery

Small restaurants often depend on pen-and-paper records or basic spreadsheets for tracking driver shifts, hours worked, and deliveries completed (Anderson & Schwieterman, 2018). This manual approach is not only inefficient but also prone to wage calculation errors and poor shift coordination (Brown et al., 2019). Managers also struggle with verifying attendance and managing workloads, especially during peak times (Miller & Chen, 2020). These methods lack the scalability and accuracy needed in today's fast-paced delivery environment.

DeliveryGo responds to this by automating time tracking, start/finish times, and integrating hourly rates and per-mile calculations for accurate wage processing.

Automation and Digital Solutions for Driver Management

Automated systems have been proven to improve scheduling accuracy, minimize payroll errors, and optimize resource allocation in small businesses (Willcocks et al., 2015). Robotic Process Automation (RPA) is increasingly being used for tasks such as wage computation, compliance checks, and shift planning (Lacity & Willcocks, 2018). However, the adoption rate among takeaway restaurants remains low due to cost, complexity, and lack of tailored solutions (Smith & Lee, 2021).

DeliveryGo addresses these barriers by offering an affordable, tailored solution with built-in automation for tasks like driver authorization, right-to-work checks, and due payment tracking.

Real-Time Tracking and Delivery Optimization

GPS tracking technology improves delivery operations by enabling real-time route optimization and delivery status updates (Kumar & Zhao, 2020). Live driver location visibility has been linked to increased customer satisfaction and driver accountability (Park & Lee, 2019). Yet, many restaurants face integration issues between delivery tracking tools and their internal systems, leading to data silos and operational delays (Forbes Technology Council, 2023).

DeliveryGo's real-time tracking, address recognition from photos, route optimization, and delivery status features help bridge this integration gap.

Identified Research Gap

While ePOS systems and tracking apps exist, they often fail to provide a **comprehensive solution** that includes **wage automation, driver performance analytics, expense logging (petrol, daily delivery costs), customer contact integration, and legal compliance features**. These are critical for small businesses looking to reduce administrative load and improve efficiency.

DeliveryGo is designed to close this gap by offering an all-in-one system tailored specifically for takeaway delivery operations—combining scheduling, tracking, wage management, driver authorization, and performance monitoring under one platform.

Responsible Technology- Ethical/Legal/Professional and Social Issues

The system is designed to increase transparency and accountability in driver management while protecting the rights and dignity of all users. Ethical issues addressed include:

Privacy and Consent: DeliveryGo ensures that drivers' personal information and location data are only collected and used with informed consent. Location tracking is only active during active shifts and is transparently communicated to users.

Fair Treatment: Wage calculations are automated based on pre-defined hourly rates and mileage, reducing the risk of human error or bias. The system helps enforce fair and transparent payment practices.

Right to Work Verification: While automating the verification of legal work eligibility, the system avoids discriminatory practices by treating all users equally, relying solely on documentation validation.

Ethical Considerations

A key ethical concern is the **responsible collection and use of personal data**, especially real-time location and wage information. All data collection must be transparent, and consent based. Drivers will be informed of what data is being collected, why it is needed, and how it will be used. Location tracking, for example, will only be active during scheduled delivery hours and clearly indicated to the user.

Data minimization will be applied to collect only the information necessary for system functionality, reducing potential risks of misuse. Additionally, **automated wage calculation** introduces ethical responsibility to ensure transparency and accuracy, avoiding disputes or potential underpayment.

The system must also be designed to avoid any **discriminatory or biased practices**, such as improper handling of driver verification or right-to-work checks. Ethical design involves treating all users equally regardless of nationality, background, or language, ensuring fair access and use.

Legal Considerations

The project adheres to relevant data protection and employment laws, particularly:

General Data Protection Regulation (GDPR): All personal data collected (e.g., names, addresses, shift logs) is processed in accordance with GDPR principles—lawfulness, transparency, purpose limitation, data minimization, and security. Users have the right to access, correct, or delete their data.

Employment Law Compliance: Features such as right-to-work checks and wage transparency support legal compliance with UK labor regulations.

Data Storage & Security: Secure storage practices, including encryption, and HTTPS communication, are implemented to protect sensitive information from unauthorized access.

Professional Standards

The system is developed following professional guidelines set by the **British Computer Society (BCS)** and **ACM Code of Ethics**, ensuring:

Integrity and Competence: The project avoids overpromising technical capabilities and maintains accuracy in all representations.

Accountability: Errors or system limitations are logged, documented, and communicated. Updates and bug fixes follow a structured CI/CD process to ensure responsible release management.

Maintainability and Documentation: Full system documentation is maintained to ensure knowledge transfer, user training, and future auditing.

Social Impact

DeliveryGo has the potential to positively impact small takeaway businesses by reducing operational stress and improving financial accuracy and transparency. Social considerations include:

Empowering Small Businesses: The platform supports small restaurants often overlooked by large delivery platforms, allowing them to better manage their workforce.

Worker Satisfaction: By automating wage tracking and minimizing disputes, the system promotes better working relationships and fairness for drivers.

Digital Inclusion: A simple and intuitive mobile interface ensures the system is accessible to users with limited technical skills or experience.

Timeline & Budget

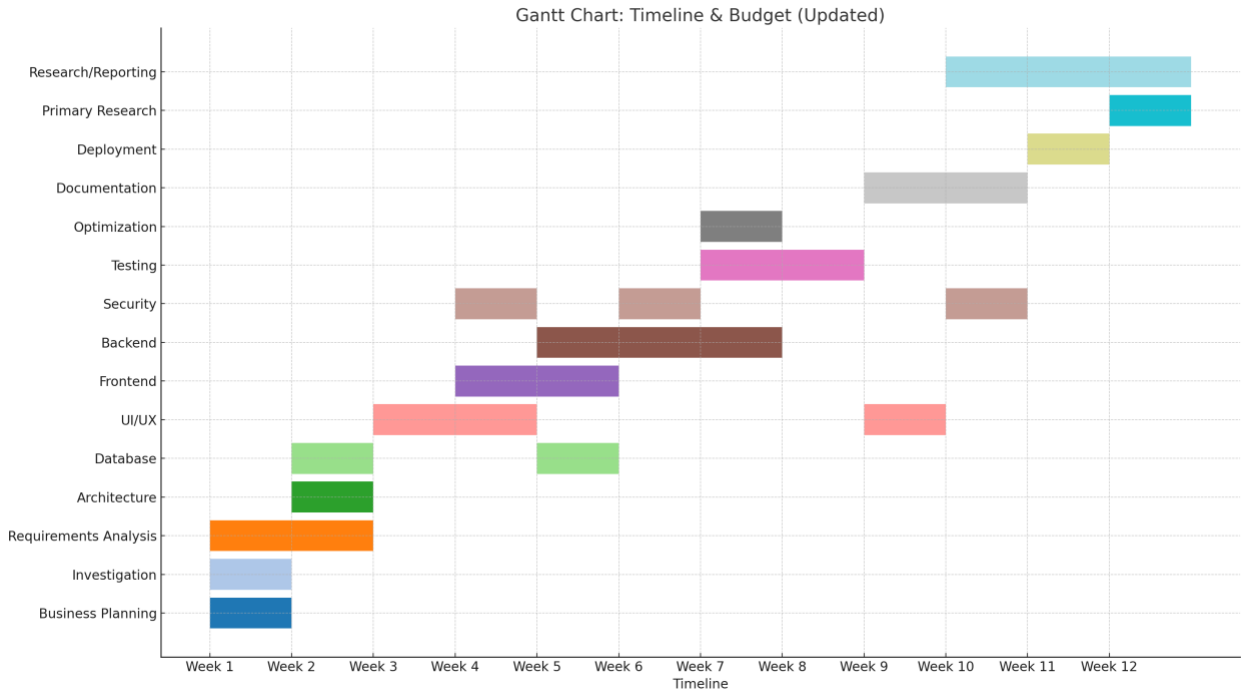


Figure 1 - Work Timeline Chart

Week	Category	Activities / Deliverables
Week 1	🧠 Business Planning	Define product vision, mission, goals, target audience, and market analysis
	📋 Requirements Analysis	Draft product backlog, gather user stories, prioritize features
	⚖️ Legal & Compliance Analysis	Explore license options (open source), review GDPR, and draft terms of service
Week 2	📋 Requirements Analysis	Refine backlog, define acceptance criteria
	⚙️ Software Architecture Design	Finalize tech stack, system modules, data flow diagrams
	🧩 Database Design	ER diagrams, relationships, schema design for PostgreSQL/Supabase
Week 3	🎨 UI/UX & Branding Design	Wireframes, color palette, font selection, initial screens via Figma

	 Software Diagrams	Class diagrams, architectural structure, sequence diagrams
Week 4	 Finalize UI	Mobile responsiveness, accessibility checks, feedback review
	 Mobile App Dev – Sprint 1	Implement Reach Native dashboard layout, login/auth UI, basic routing
	 Security Planning	Threat model, access roles (admin/user), data protection approach
Week 5	 Backend Dev – Sprint 1	Setup Node.js and Supabase Edge Function, auth APIs, SMS Auth, workspace/user models, connect to database
	 Frontend Dev – Sprint 2	Feature planning board (Kanban-style), navigation components
	 Database Implementation	Create tables, indexes, test data
Week 6	 Backend Dev – Sprint 2	Feature CRUD, roadmap logic, checklist endpoints
	 Security Implementation	Middleware, input validation, cookie/session handling
Week 7	 Performance Optimization	Lazy loading, dynamic imports, API caching
	 Backend – Sprint 3	Notifications, status updates, markdown rendering
	 Testing – Sprint 1	Unit testing backend APIs, test frontend flows
Week 8	 Testing – Sprint 2	Apple Store TestFlight Testing, Google Console Testing
	 Deployment Prep	Apple Store and Google Play Store
Week 9	 Documentation Draft	API docs, system overview, tech stack rationale, feature matrix
	 Final UI Review	Polish design, mobile testing, brand assets export
Week 10	 Final Documentation	README, user guide, license, and contribution guides

	Primary research Security Audit	User experiments, data collection Permissions, password rules, access logging, vulnerability scan
Week 11	Final Deployment Data analysis Final Agile Review	Launch app, bind domain, SSL setup, database Statistical comparison new vs old system Address supervisor feedback, patch issues, version lock
Week 12	User Testing + Feedback Report & Viva Preparation	Internal/external user testing, analytics integration Finalize report, write conclusion, make presentation slides, submit GitHub repo

Figure 2 - Work Timeline Details Table

Resources Required

To successfully complete this project, the following technical and non-technical resources will be essential:

Category	Resources/Tools
Google Play Store	Google Developer Account (£79 Paid)
Apple Store	Apple Developer Account (£25 Paid)
Distance Matrix API	Google Distance Matrix API
Character Recognition - OCR	Google Vision API, ChatGPT API
SMS Authentication	Twilio SMS Auth and OTP System
Development Tools	**VS Code**, Node.js/Next.js, PostgreSQL, ORM
Frontend Framework	**Next.js**, React Native, Tailwind CSS, Framer Motion
UI/Design	**Figma** (for wireframes and mockups)
Database	**PostgreSQL Supabase**
Version Control	Git + GitHub (private repository)
Deployment	**Vercel** (Frontend), React Native, **Supabase Edge Functions**
API Documentation	Swagger, Postman
Security Tools	HTTPS, CORS configurations
Testing Tools	Vitest(unit testing), Playwright (end-to-end testing)
Android & IOS Platform	React Native for Cross Platform Development
DevOps Tools	Expo, ESA Build, Docker, GitHub Actions (CI/CD), DotENV,

Figure 3 - Resources Required Table

Project Planning and Development- Project Methods

This section outlines the systematic approach undertaken in the planning, design, development, and evaluation of the DeliveryGo.

Project Management (Agile Methodology)

The project was managed using an Agile methodology, which was chosen because of its flexibility, iterative structure, and suitability for software development projects where requirements evolve over time. Unlike traditional waterfall approaches that rely on rigid sequential phases, Agile emphasizes incremental progress, adaptability, and stakeholder feedback (Beck et al., 2001). This methodology aligned well with the DeliveryGo project, where both technical implementation and evaluative simulations required continuous refinement.

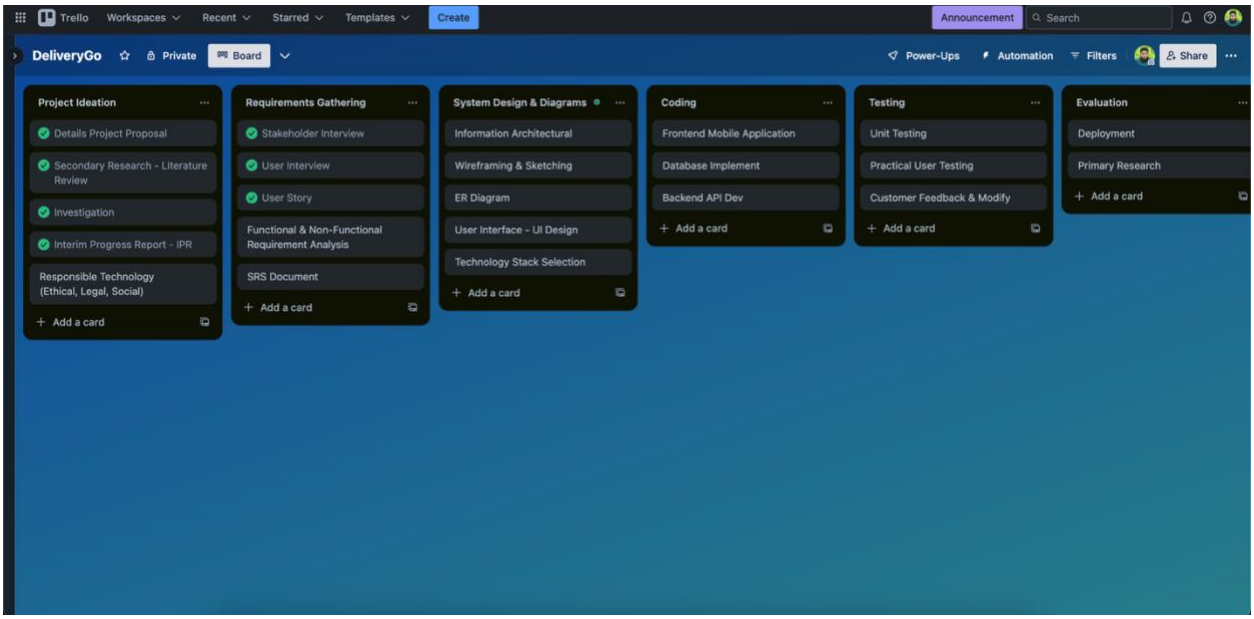


Figure 4 - Middle of the Project - Trello Agile Dashboard

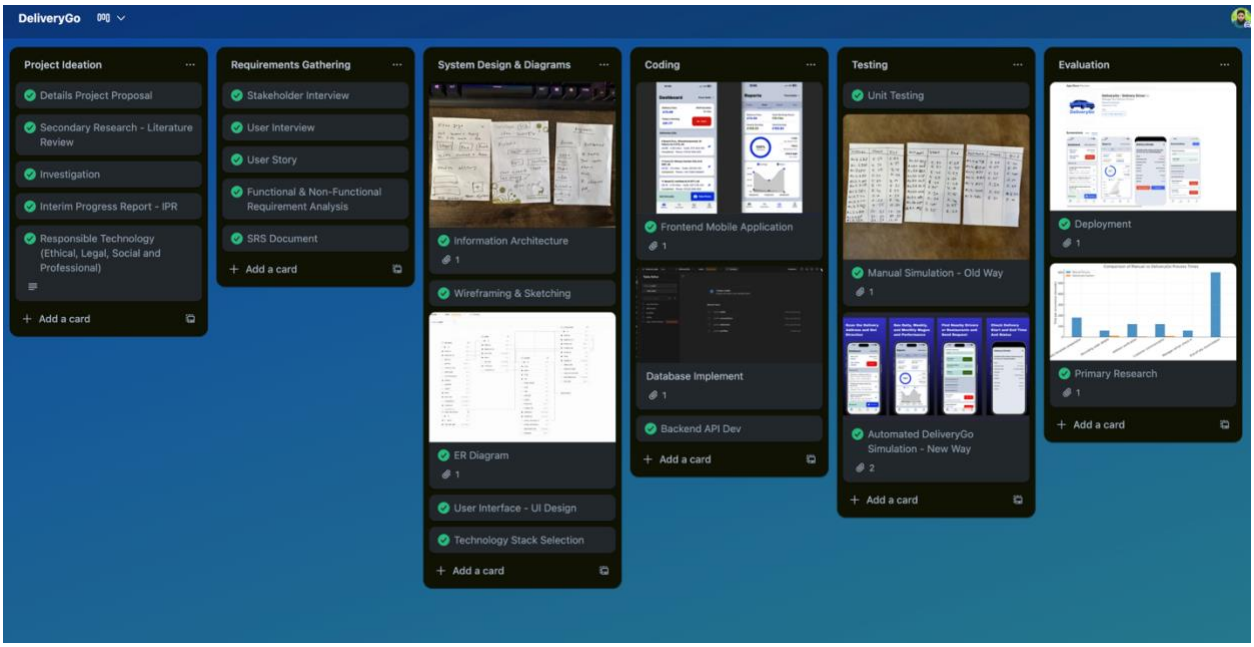


Figure 5 - End of the project - Trello Agile Dashboard

Tools & Technologies

The development and evaluation of the DeliveryGo application required a combination of modern development frameworks, cloud-based services, and open-source libraries. The selection of these tools was guided by criteria of scalability, accessibility, and alignment with the project's objective of building a lightweight yet robust delivery driver management system for small takeaway businesses.

Visual Studio Code (VS Code):
The primary integrated development environment (IDE) was Visual Studio Code, chosen for its extensive plugin ecosystem, debugging capabilities, and seamless integration with modern JavaScript and TypeScript workflows.

React Native (Expo Framework):
The mobile application was developed using React Native, a cross-platform framework that enables the creation of native iOS and Android applications from a single codebase. The Expo framework was used to streamline the development process, providing integrated build tools, testing environments, and simplified deployment pipelines. React Native and Expo were chosen for their ability to accelerate prototyping and ensure broad accessibility without requiring multiple native development teams (Facebook Open Source, 2025; Expo, 2025).

Git & GitHub:
Version control was managed using Git, with GitHub serving as the central repository. In addition to collaborative code management, GitHub was leveraged for issue tracking, pull requests, and continuous integration/continuous deployment (CI/CD) pipelines through GitHub Actions, ensuring disciplined and auditable development cycles.

Supabase (with PostgreSQL Backend):
Supabase was utilized as the backend platform, offering real-time database functionality, authentication, and API generation. It is built on top of PostgreSQL, an established relational database management system known for its robustness, scalability, and strong support for transactional workloads. Supabase's integration with PostgreSQL allowed DeliveryGo to manage user profiles, shifts, deliveries, and connection data efficiently, while its serverless infrastructure reduced the need for extensive backend configuration (Supabase, 2025; PostgreSQL Global Development Group, 2025).

Zustand (State Management):
For client-side state management, Zustand was implemented as a lightweight and efficient solution to handle user sessions, delivery records, and other shared application states. Zustand's minimalistic API reduced complexity compared to alternatives such as Redux, making it a suitable choice for rapid development and scalability (Zustand, 2025).

Google Maps Platform (GPS & Navigation Services):
The Google Maps API was integrated to provide real-time mapping, navigation, and mileage tracking. This enabled precise calculation of delivery distances and times, supporting the project's objective of improving the accuracy of wage and mileage-based calculations. By leveraging geolocation services, DeliveryGo was able to replace error-prone manual estimations with objective, verifiable data (Google Developers, 2025).

Google OCR (Optical Character Recognition):
Google Vision OCR was employed for the extraction of address details from receipts and other textual inputs, reducing manual data entry errors. The integration of OCR technology streamlined the process of managing delivery orders, aligning with the aim of minimizing administrative burden (Google Vision OCR, 2025).

Deployment Platforms (Vercel):
For frontend web deployment and scalability, DeliveryGo utilized modern cloud-hosting solutions such as Vercel. These platforms provided continuous integration pipelines, automated builds, and global content delivery networks (CDNs), ensuring that the application could be reliably hosted, updated, and scaled in line with demand (Vercel, 2025; Netlify, 2025).

PostgreSQL:
Serving as the relational database, PostgreSQL provided reliability and transactional integrity for managing structured data such as delivery logs, driver records, payments, and connections.

Next.js (React Framework):
Next.js was selected for building performant and search engine–optimized web interfaces, offering both server-side rendering (SSR) and static site generation (SSG). These capabilities ensured faster load times and improved accessibility for restaurant-facing dashboards.

React.js: React formed the foundation of the component-based architecture, enabling the development of reusable UI elements and efficient state management. Its widespread adoption and strong community support reduced technical risk and facilitated faster problem resolution.

Tailwind CSS: For styling, Tailwind CSS provided a utility-first approach that accelerated responsive design and guaranteed accessibility-compliant interfaces. Its modularity reduced development overhead compared to traditional CSS frameworks.

React Native: Enables development of native Android and iOS applications from a shared codebase. Offers smooth integration with mobile-specific APIs like GPS, camera (for address via photo), push notifications, and telephony (call customer button).

Expo / React Native CLI: Used for building, testing, and deploying mobile applications.

Node.js: Where server-side logic was required, Node.js provided a performant, event-driven runtime. Its compatibility with Supabase APIs and JavaScript-based stack supported end-to-end consistency across the system.

Figma: The application's user interface and user experience were prototyped using Figma. Collaborative features enabled iterative refinement and stakeholder feedback prior to implementation.

Drawsql.app: Database architecture and system entity-relationship (ER) diagrams were designed using DrawSQL, ensuring clarity in schema design and alignment with relational data modelling principles.

Requirements Gathering

Requirements gathering is a critical phase in software engineering, forming the foundation upon which system design and implementation are based. As the DeliveryGo project could not involve real-world participants due to ethical and practical constraints, the requirements were elicited through scenario-based simulations and persona-driven assumptions. This method is recognized as a valid approach for exploratory projects where direct user access is not feasible (Sommerville, 2016).

Simulated Scenario: To approximate realistic requirements, a representative scenario was constructed:

User Story 1 – Automatic Address Recognition

As a user, I want to get the address automatically, so I don't have to type it manually.

Purpose:
This story reflects the need for faster and more accurate address entry, especially when delivery instructions are received via messaging apps or images. This feature will utilize image recognition or location metadata to extract addresses directly from photos, reducing manual entry errors and saving time.

User Story 2 – Automated Wage and Expense Calculation

As a business owner, I want to calculate wages and delivery expenses automatically, so I don't have to calculate it manually with pen and paper.

Purpose:
This story addresses the administrative burden faced by business owners. By automating wage calculations based on hourly rates, delivery counts, mileage, and petrol expenses, DeliveryGo eliminates errors and saves significant time. This also ensures transparency and consistency in payments.

Functional Requirements

The functional requirements outline the core capabilities that the **DeliveryGo** system must offer to support takeaway restaurant operations and address user needs. Each function is derived from user stories, and system objectives.

1. **User Authentication and Authorization**
    a. Allow secure login and access based on user roles (e.g., admin, driver).
    b. Verify driver eligibility (right-to-work check).
2. **Shift & Time Tracking**
    a. Enable drivers to check in and check out.
    b. Automatically record start and finish times.
3. **Weekly Wages Calculation**
    a. Calculate wages based on:
        i. Total hours worked
        ii. Price per mile
        iii. Number of deliveries
4. **Mileage-Based Payment Calculation**
    a. Record mileage per delivery and calculate additional payments based on preset rates.
5. **Delivery Management**
    a. Assign delivery orders to drivers.
    b. Update and track delivery status (e.g., ongoing, delivered).
    c. Record and display delivery time averages.
6. **Route Optimization**
    a. Provide route suggestions for multiple deliveries to minimize travel time and cost.
7. **Automatic Address Detection**
    a. Extract address from images or shared content using OCR or metadata.
8. **Driver Tracking**
    a. Show live driver location on a map for monitoring and reporting.
9. **Customer Communication**
    a. Provide a “Call Customer” button within the app for instant communication.
10. **Expense & Cost Tracking**
    a. Record daily delivery logs.
    b. Log petrol expenses manually or from fuel cards.
    c. Calculate and summarize daily and weekly expenses.
11. **Driver Performance Monitoring**
    a. Track and report on:
        i. Number of deliveries
        ii. On-time rate
        iii. Missed or delayed deliveries
12. **Report Generation**
    a. Generate detailed reports on:

i. Wages
ii. Delivery summaries
iii. Driver performance
iv. Outstanding payments

13. Due Payment Management

a. Track unpaid wages or reimbursements and mark them as settled when paid.

Non-Functional Requirements

These define how the system should behave, focusing on quality, performance, and operational standards.

Performance
The system must handle up to 100 concurrent users without noticeable latency. Location tracking updates should occur in near real-time (< 5 seconds delay).

Scalability
The system must support multiple restaurants, each with multiple drivers and deliveries per day.

Security
All personal and wage-related data must be stored securely (e.g., hashed passwords, encrypted wage records).
Compliance with **GDPR** and **right-to-work** legal requirements is mandatory.

Usability
Mobile and web interfaces must be intuitive and user-friendly, designed with accessibility in mind. Onboarding time for new users should be less than 30 minutes.

Reliability
99.9% uptime for backend services hosted on Azure. System must support offline data caching and sync when internet is restored.

Maintainability
Codebase should follow modular design and be documented for easy maintenance.
CI/CD pipelines must enable smooth deployment and updates.

Portability
Mobile application must run seamlessly on both **Android** and **iOS** using **React Native**.

Business Needs

DeliveryGo is built to fulfill specific business goals and solve real-world operational issues faced by takeaway restaurants managing in-house delivery staff.

Operational Efficiency
Eliminate manual tracking of hours, routes, and wages through automation. Reduce administrative workload and improve accuracy in wage and cost calculations.

Cost Control
Track petrol usage, delivery mileage, and staff wages to monitor profitability per shift or per driver. Identify delivery inefficiencies via route optimization and performance analytics.

Legal Compliance
Simplify and centralize driver authorization, including right-to-work verification and status management. Minimize risks associated with undocumented employment practices.

Transparency & Accountability
Provide real-time insights into driver activities, earnings, and delivery timelines. Increase driver accountability and reduce wage disputes through accurate logs and reports.

Customer Satisfaction
Enhance delivery reliability through tracking and communication features. Reduce delays and improve order accuracy with route optimization and live driver tracking.

Information Architecture

The information architecture of DeliveryGo defines how data is structured, organized, and accessed across the system. It establishes clear data flows between the frontend, React Native mobile app and Next.js web dashboard), backend services (Supabase with Edge Functions and Node.js APIs), and the PostgreSQL database. Core entities such as users, deliveries, shifts, expenses, and payments were modelled using an Entity-Relationship Diagram (ERD), ensuring logical data representation, referential integrity, and consistency. The system adopts a multi-tier, layered architecture to separate concerns and improve scalability, maintainability, and security.

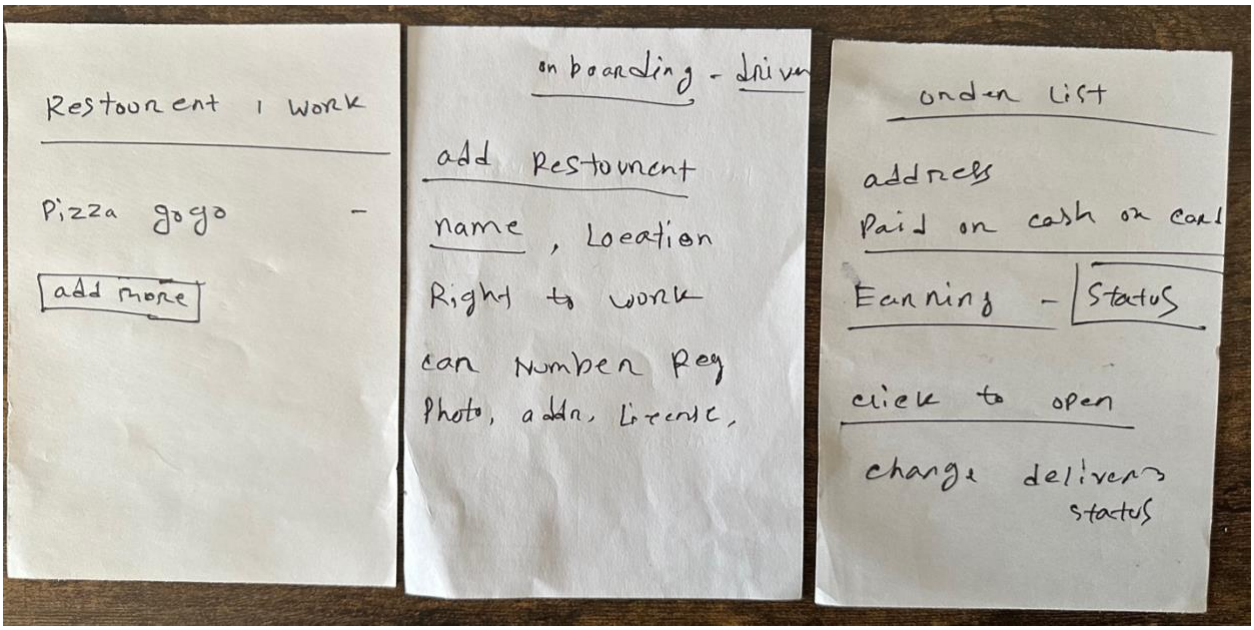


Figure 6 - Information Architecture and Sketching

At the highest level, DeliveryGo operates within a client–server model. The mobile application, built using React Native with Expo, represents the primary interface for delivery drivers, enabling them to log shifts, manage deliveries, and track mileage in real time. In parallel, the web dashboard developed with Next.js provides restaurant owners with the tools required to assign deliveries, monitor driver performance, and generate payment reports. These two presentation layers communicate with the backend through secure APIs and real-time data subscriptions.

The backend layer was implemented using Supabase, which integrates authentication, database access, and real-time functionality. Supabase Edge Functions and lightweight Node.js microservices were employed for operations that required additional server-side computation, such as payment calculation or secure delivery assignments. By leveraging this serverless and modular design, the system avoided the overhead of complex infrastructure management while maintaining flexibility for future scaling.

System Design

The system design phase established how each component of DeliveryGo works together to meet project requirements. A layered design was implemented comprising:

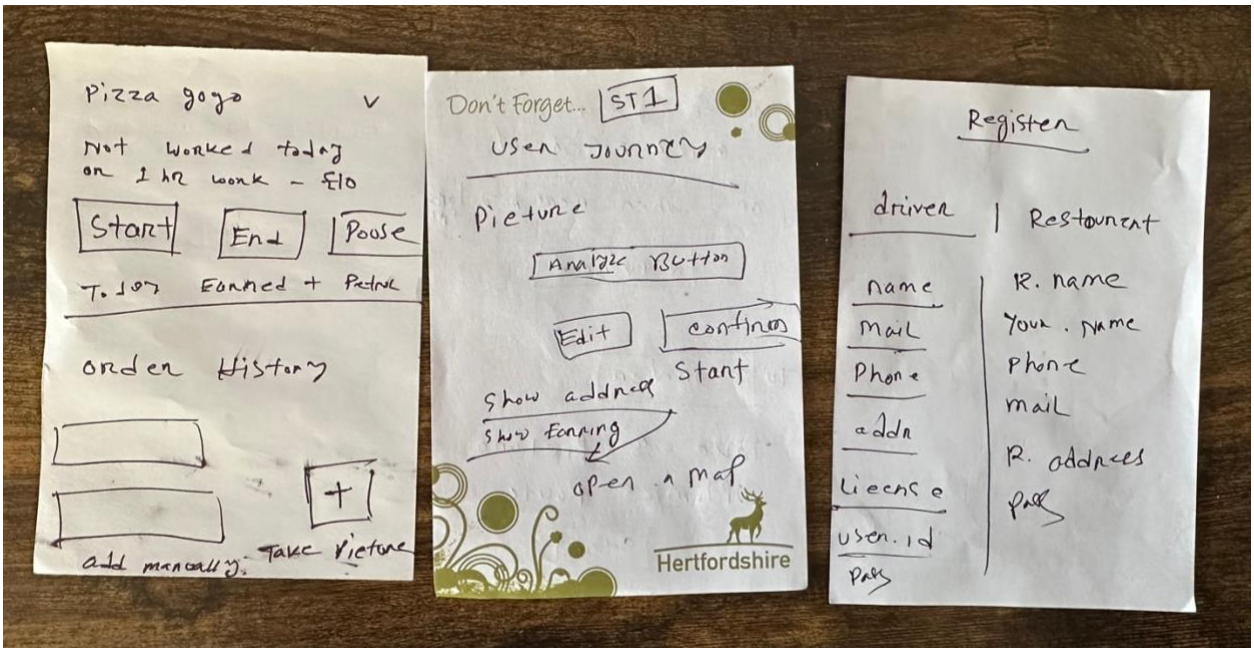


Figure 7 - System Design Sketching

Frontend Layer: React Native mobile app (for drivers) and Next.js web dashboard (for restaurants).

Backend Layer: Supabase for authentication, real-time data subscriptions, and PostgreSQL hosting; Supabase Edge Functions and Node.js microservices for server-side logic.

Database Layer: PostgreSQL database accessed through Supabase ORM, providing secure and efficient queries.

External APIs: Google Maps for navigation and mileage tracking, Tesseract OCR for address recognition, Twilio for SMS-based authentication.

ER Diagram

The DeliveryGo data model centers on a single profiles entity that represents both actors in the system—drivers and restaurants—distinguished by a role attribute. Each profile stores core identity and contact details, plus remuneration defaults (hourly_rate, mileage_rate). Profiles are created in lock-step with platform identities via a foreign key to auth.users, ensuring account provenance and enabling ON DELETE CASCADE user lifecycle management. A convenience pointer, active_connection_id, records the currently selected relationship context for a user.

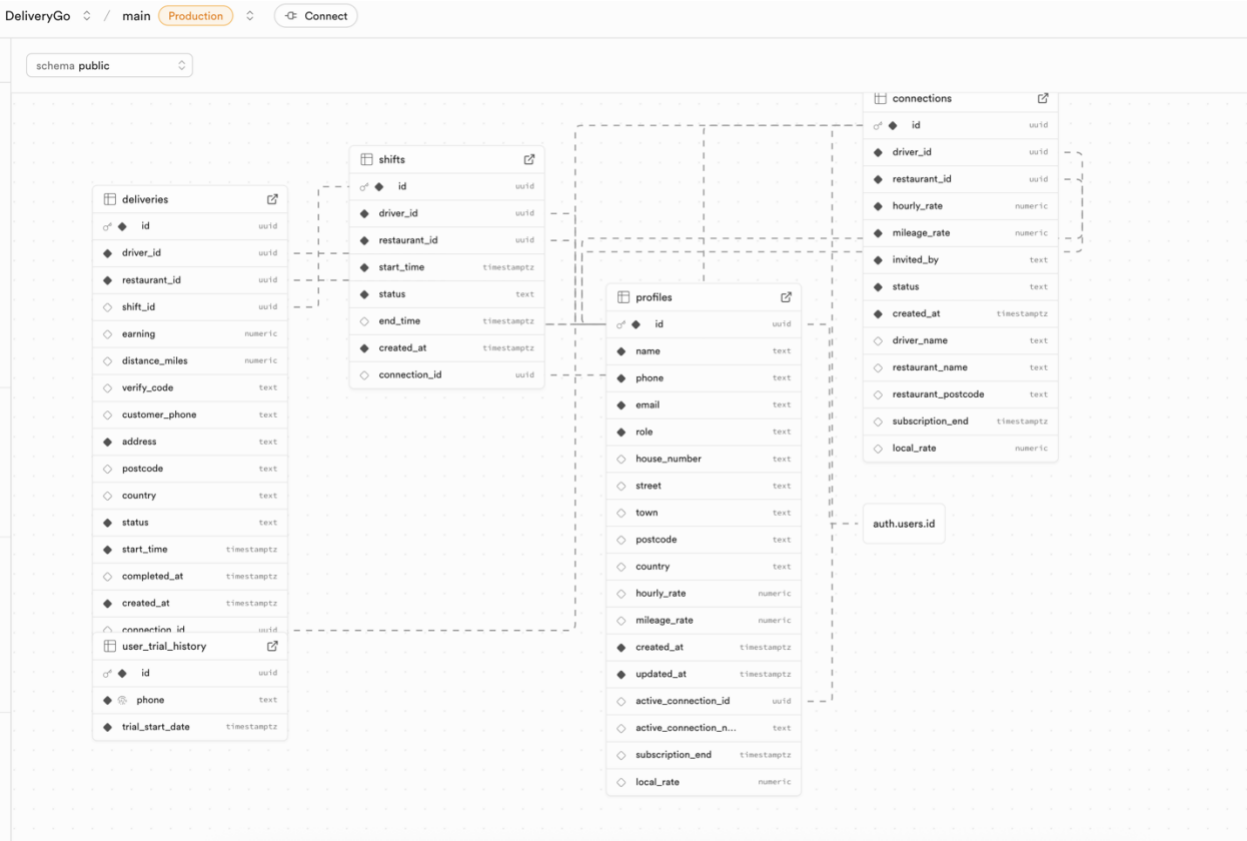


Figure 8 - ER Diagram

Operational relationships between drivers and restaurants are captured in **connections**, a **linking entity** that models a **many-to-many** association between profiles. Each row binds exactly one driver to one restaurant and carries **contract-specific terms** (hourly_rate, mileage_rate), invitation provenance (invited_by), and workflow state (status). The UNIQUE(driver_id, restaurant_id) constraint prevents duplicate ties, establishing a canonical contract per pair.

Work sessions are recorded in **shifts**, each tied to a **driver** and a **restaurant**, and optionally to a specific connection_id to lock the session to the applicable contract terms. Temporal fields (start_time, end_time) and a finite state machine (status ∈ {active, ended}) support accurate wage computation and auditability. **Cascading deletes** from profiles guarantee orphan-free cleanup of historical shift data when an actor is removed.

Task-level execution is represented by **deliveries**. Each delivery references the responsible **driver** and **restaurant**, and may link to the **shift** during which it occurred and to the governing **connection**. Attributes capture operational and financial facts, including address, postcode, distance_miles, earning, verification and contact data, timestamps (start_time, completed_at), and lifecycle (status ∈ {ongoing, completed}). This structure enables reconstruction of productivity, punctuality, and cost metrics per actor, per shift, or per contract.

A supporting **user_trial_history** table records one trial entitlement per unique phone number, enforced by a UNIQUE(phone) constraint. This is intentionally **decoupled** from profiles to allow pre-registration trials and to protect pricing logic from account churn.

Across the schema, **referential integrity** is enforced via foreign keys with appropriate **ON DELETE** actions (CASCADE for ownership, SET NULL for optional associations). Timestamps default to now() to guarantee temporal traceability. In the ER diagram, cardinalities are: **profiles 1..—*connections*—..1 profiles**; **profiles 1..*—shifts**; **profiles 1..*—deliveries**; **shifts 1..*—deliveries (optional)**. This normalised model minimises redundancy while preserving the contractual and temporal context required for fair, auditable wage and mileage calculations.

React Native Application

The DeliveryGo mobile application was developed using React Native, chosen for its ability to deliver high-performance, cross-platform applications from a single codebase. This approach significantly reduced development overhead while ensuring consistent functionality across both iOS and Android platforms (Akter et al., 2021). The framework's modular component architecture and strong community support made it well suited for rapid prototyping and production deployment in alignment with agile development practices (Majchrzak et al., 2022).

The driver application implemented all core workflows identified during requirements gathering. These included secure authentication via Supabase's phone-based login system, shift management (start, active tracking, and end), delivery creation and status updates, and automated wage calculation based on mileage and hourly rates. Google Maps integration supported mileage tracking and navigation, while real-time synchronization with Supabase ensured that driver updates were reflected immediately on the restaurant dashboard. Such near-instant updates are critical in logistics applications, where delays in data propagation can reduce operational efficiency (Zhang et al., 2020).

The development process was streamlined by adopting the Expo ecosystem, which provided rapid testing, device API integration, push notification services, and over-the-air (OTA) updates. Expo EAS Build facilitated the generation of signed binaries for submission to the Apple App Store and Google Play Console, enabling a smooth release pipeline and reducing deployment friction (Expo, 2023). Application state was managed using Zustand, a lightweight state management library designed for predictable data handling, ensuring that session, delivery, and shift information were accessible globally across screens without redundant API calls.

Overall, the React Native application provided drivers with a lightweight, responsive, and user-friendly interface that was capable of supporting real-world delivery management tasks. By combining React Native with Supabase, Expo, and third-party APIs, the system balanced performance, maintainability, and scalability, while also conforming to academic principles of sound system design and applied research.

Database Implementation

The database layer of DeliveryGo was implemented using PostgreSQL, deployed through Supabase, to provide a reliable and scalable foundation for data persistence. PostgreSQL was selected for its proven stability, advanced relational features, and strong support for transactional workloads, all of which are critical in maintaining accuracy across deliveries, shifts, and payment calculations (Stonebraker & Kemnitz, 1991). Supabase extended this functionality by offering real-time subscriptions, authentication, and RESTful endpoints automatically generated from the schema, thereby reducing backend overhead while preserving flexibility (Supabase, 2023).

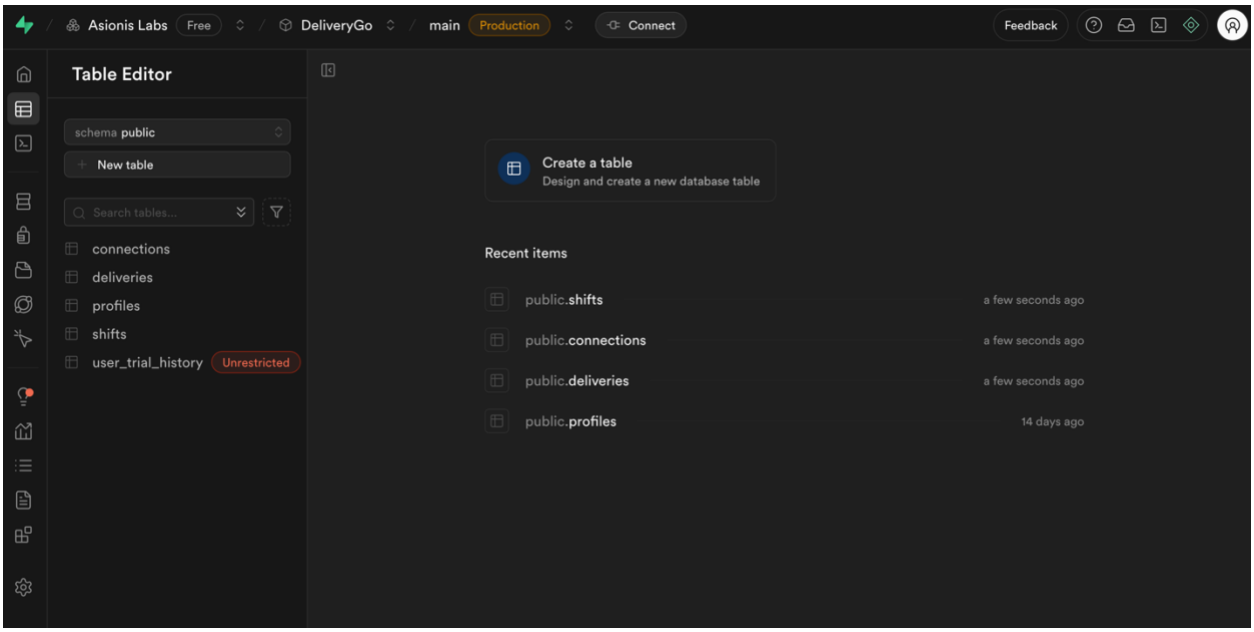


Figure 9 - Database Implements

The schema was derived from the Entity-Relationship Diagram (ERD), which modelled the key entities of the system: profiles, connections, shifts, and deliveries. The profiles table represented both drivers and restaurants, with roles distinguished through a controlled attribute. A connections table established contractual relationships between restaurants and drivers, enabling flexible management of hourly and mileage rates. Shifts recorded working sessions for drivers, while deliveries captured granular operational data such as addresses, distance travelled, earnings, and timestamps. Referential integrity was enforced through foreign key constraints, with cascading delete policies to ensure data consistency and avoid orphaned records.

Supabase's real-time features enabled event-driven synchronization across clients. For example, when a driver completed a delivery, the associated update in the deliveries table was instantly propagated to the restaurant dashboard, ensuring operational transparency. In addition, Supabase Row-Level Security (RLS) policies were applied to enforce role-based access control, ensuring that sensitive data such as driver contact details or payment rates could only be accessed by authorized users, thus aligning with GDPR compliance requirements (Voigt & Von dem Bussche, 2017).

To facilitate safe and efficient interactions between the application layer and the database, Prisma ORM was used. Prisma provided type-safe query building, schema migrations, and query optimization, which reduced the risk of runtime errors and improved developer productivity (Prisma, 2023). Performance considerations were addressed by indexing frequently queried fields such as driver_id, restaurant_id, and status, enabling efficient retrieval of shift and delivery records even at scale.

Overall, the database implementation provided DeliveryGo with a robust, secure, and high-performance data layer. The combination of PostgreSQL's relational integrity, Supabase's real-time infrastructure, and Prisma's type-safe abstraction ensured that the application could support operational requirements while remaining scalable for future enhancements.

Backend API Development

The backend of DeliveryGo was implemented using **Supabase Edge Functions**, supported by PostgreSQL as the underlying data store. Edge Functions provided a **serverless execution model**, enabling custom business logic to be securely executed at scale without maintaining dedicated infrastructure (Supabase, 2023). This approach reduced operational overhead while ensuring low-latency responses for client applications.

Apple Developer Account and Google Play Console

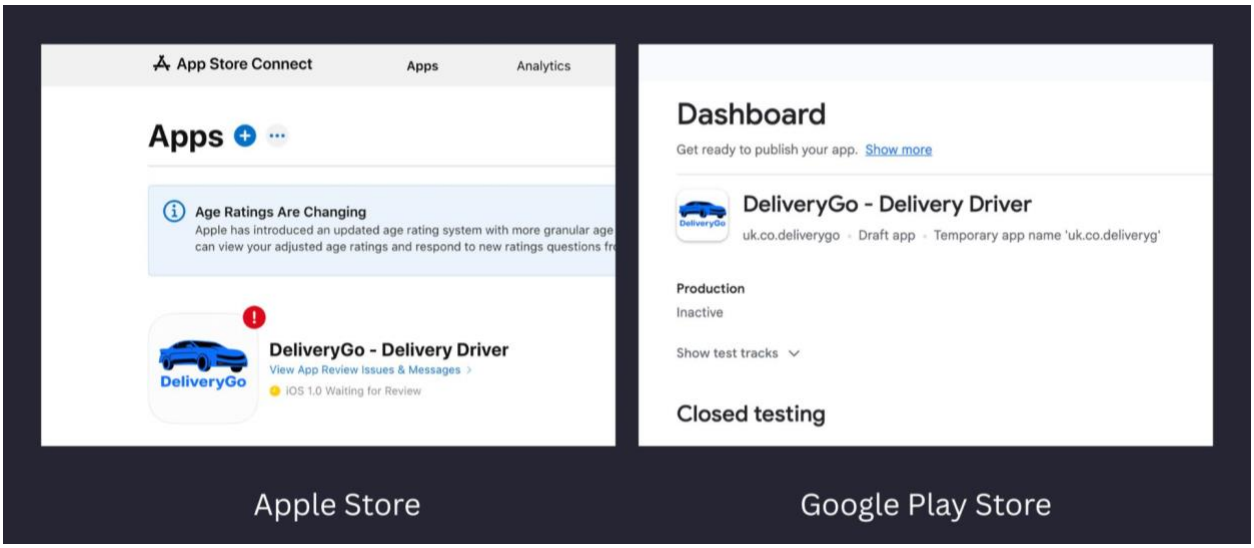


Figure 10 - IOS & Android Developer Account Setup

To distribute the DeliveryGo mobile application to end-users, developer programmed enrolment was required with both Apple and Google. The Apple Developer Program provides access to code-signing certificates, TestFlight for beta testing, and distribution through the App Store. Similarly, the Google Play Console enables application publishing to the Play Store, along with services such as crash reporting, performance monitoring, and staged rollouts (Apple, 2023; Google, 2023). Compliance with these platforms' guidelines was necessary to ensure approval and accessibility on both iOS and Android devices.

Twilio SMS Authentication

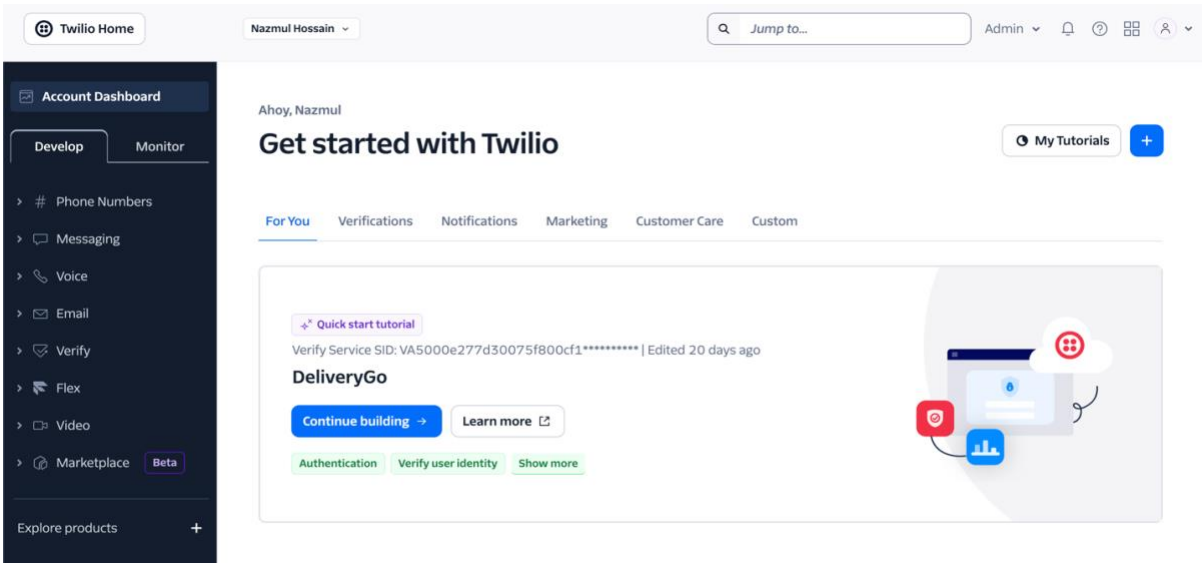


Figure 11 - Twilio SMS OTP Setup

For secure and user-friendly onboarding, DeliveryGo employed **Twilio Verify API** to implement SMS-based phone number authentication. This approach aligns with industry practice, as SMS verification remains a widely adopted second-factor mechanism for establishing trust in mobile systems (Alashhab et al., 2021). Twilio provided scalability and integration with Supabase authentication workflows, ensuring reliable delivery of one-time passcodes across multiple regions.

Expo EAS Build and Continuous Delivery

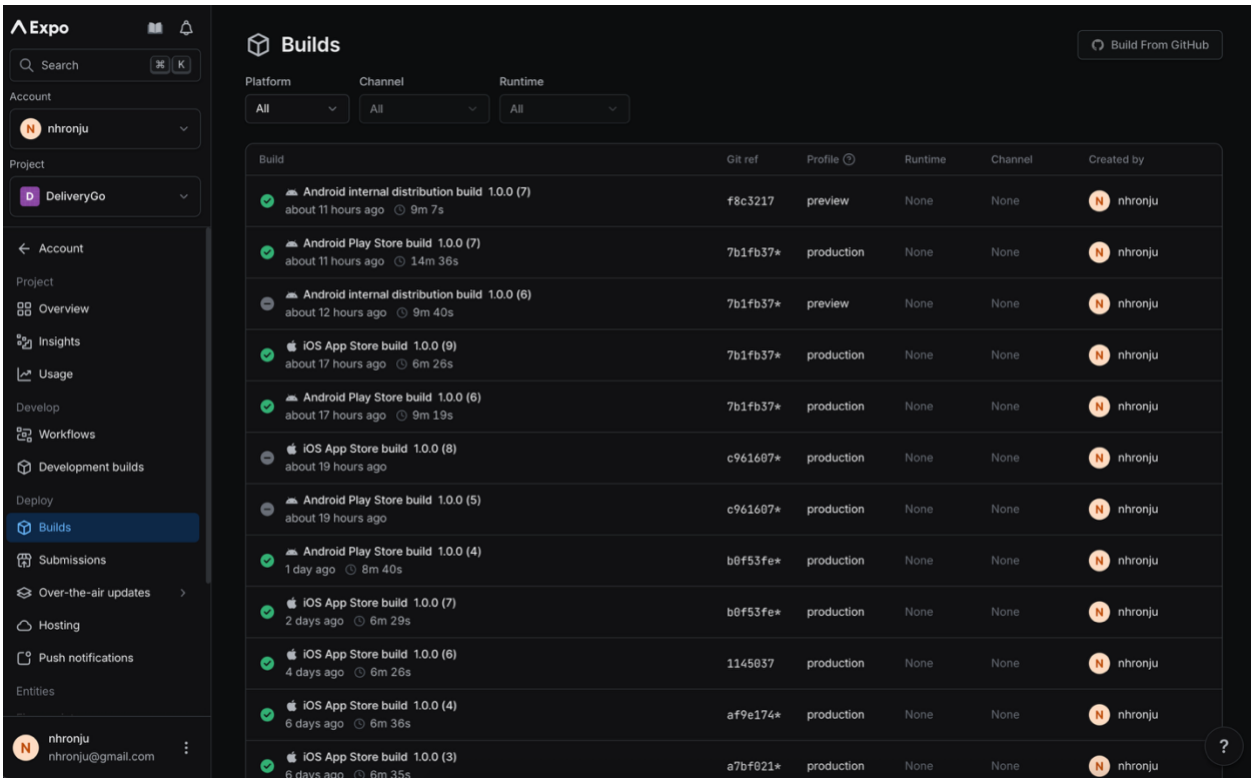


Figure 12 - ESA Application production Build Dashboard

To streamline the build and deployment pipeline, DeliveryGo utilized **Expo Application Services (EAS Build)**. EAS enabled cloud-based compilation of signed binaries for iOS and Android without requiring local dependency management, thereby reducing complexity in multi-platform deployment. Combined with over-the-air (OTA) update capabilities, EAS Build allowed developers to ship incremental improvements quickly and with minimal downtime (Expo, 2023).

Additional External Integrations

Beyond core distribution and authentication services, additional tools supported project implementation. These included **Figma** for collaborative UI prototyping, **DrawSQL** for visual database schema design, and **GitHub Actions** for continuous integration and delivery (CI/CD). Collectively, these services accelerated development workflows while ensuring alignment with best practices in software engineering and system deployment (Fowler, 2020).

Testing & Quality Assurance

Testing and quality assurance (QA) formed an integral part of the DeliveryGo development lifecycle, ensuring that the system was both functionally reliable and aligned with user requirements. A **multi-layered testing strategy** was adopted, combining automated methods with simulated and end-user evaluations to validate both technical performance and usability.

At the code level, **unit testing** was conducted on individual modules, including state management functions in the React Native frontend and Supabase Edge Functions in the backend. This ensured that isolated components behaved as expected and helped identify regressions early in the development cycle (Myers et al., 2011). **Integration testing** followed, focusing on the interaction between the frontend, backend APIs, and third-party services such as Twilio (SMS verification) and Google Maps (distance estimation). These tests confirmed the seamless flow of data across layers, verifying that user actions in the mobile app produced consistent updates in the database and dashboards.

For mobile deployment, **Apple TestFlight** and the **Google Play Console** were used to distribute pre-release builds to a controlled group of testers. These platforms enabled structured beta testing, crash reporting, and performance monitoring, thereby identifying device-specific issues across iOS and Android ecosystems (Apple, 2025; Google, 2025).

In addition, **simulated testing** was performed to replicate real-world delivery workflows in a controlled environment. Scenarios such as starting and ending shifts, creating deliveries, updating statuses, and calculating wages were executed repeatedly to confirm system stability under different usage patterns. **End-user testing** was also carried out with representative participants to evaluate usability, clarity of the user interface, and responsiveness. Feedback gathered during these sessions informed minor design adjustments and validated that the system met the expectations of its intended stakeholders.

Together, these practices provided a **comprehensive QA framework**, ensuring that DeliveryGo was technically robust, user-friendly, and ready for deployment in operational contexts.

Deployment to iOS Store

After testing and quality assurance were completed, the DeliveryGo application was deployed to the Apple App Store for distribution. Enrolment in the Apple Developer Program provided access to provisioning profiles, certificates, and TestFlight for beta testing. The production build was generated using Expo EAS Build, ensuring compatibility with Apple's technical requirements.

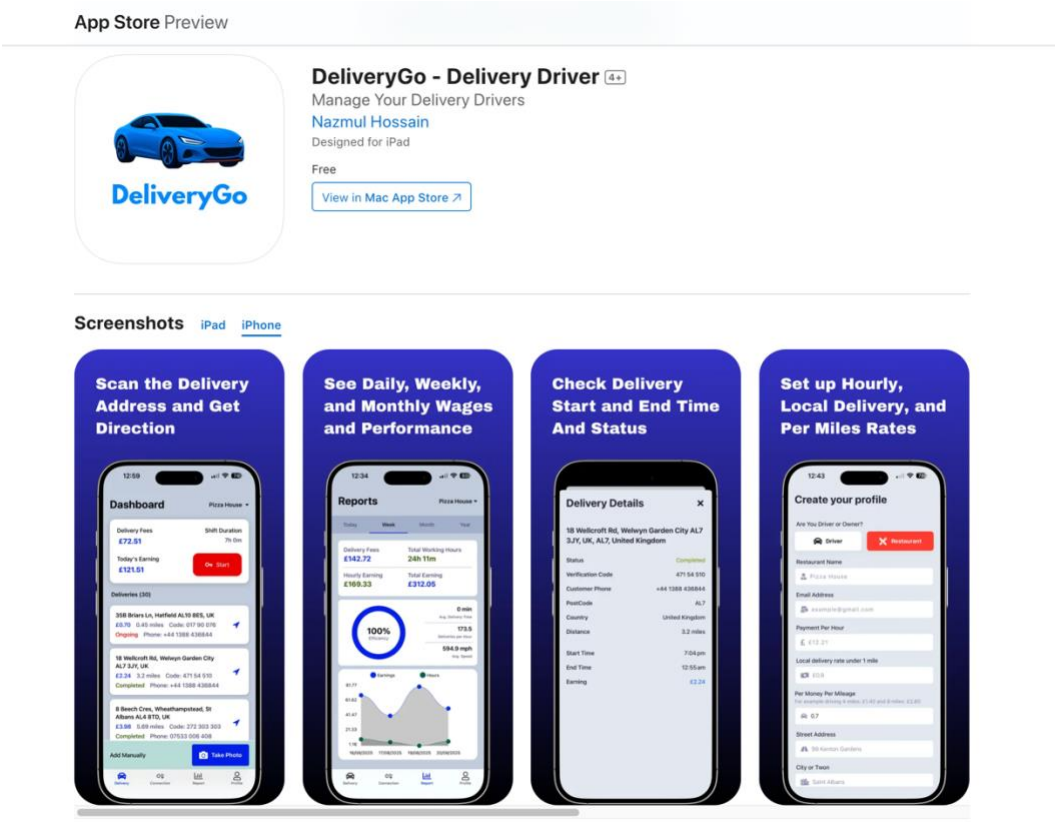


Figure 13 - IOS App Store Listing

DeliveryGo Navigation Workflow

As part of the quality assurance process, particular attention was given to validating the **navigation flow** within the DeliveryGo mobile application. A smooth and intuitive navigation structure was critical to reducing cognitive load for both drivers and restaurant users, ensuring that essential tasks could be completed with minimal interaction steps.

Onboarding & Registration

This is the initial user journey for new users signing up for the service.

1. **Welcome Screen**: Users are prompted to sign in with their phone number and a country code. Tapping **"Send OTP"** sends a one-time password via SMS.

2. **OTP Verification**: Users enter the 6-digit OTP to verify their phone number. Successful verification directs new users to the **Profile Completion** screen and existing users to the **Dashboard**.

3. **Profile Completion** (for new users): Users choose their role as a **Driver** or **Restaurant**. For drivers, a form requires their name, email, hourly rate, mileage rate, and address. Tapping **"Submit Profile"** creates their account and takes them to the Dashboard.

Dashboard & Delivery Management

This is the core functionality for drivers to manage daily operations.

1. **Dashboard**: The central hub for drivers. It displays a **Status Card** with real-time earnings and shift duration. A large button allows the user to **"Start"** or **"Stop"** their shift. The screen also features a scrollable **Deliveries List** with details like address, earning, and status. Drivers can open a delivery's destination in a navigation app via an icon.

2. **Manual Delivery Addition**: At the bottom of the Dashboard, drivers can add deliveries not assigned through the app by tapping **"Add Manually"** or **"Take Photo"** (to document a physical receipt).

3. **Delivery Details Pop-up**: Tapping on a delivery item opens a detailed view. This pop-up displays the full address, earning, status, and verification code. It includes two action buttons: **"Mark as delivered"** to complete the.

Reporting & Profile Management

This section is for reviewing past performance and managing personal account information:

1. **Connections Screen**: This allows users to view, search for, and manage partnerships with restaurants. Users can accept or reject incoming connection requests.

2. **Reports Screen**: This provides detailed analytics on earnings and performance. Users can filter data by connection and time-period (Today, Week, Month, Year). The screen displays key metrics like total delivery fees, total miles, and average hourly earnings, along with a performance chart tracking earnings and mileage.

3. **Profile Screen**: Users can view and update their personal information, including name, email, phone number, and address. A prominent **"Logout"** button is also available.

Primary Research – Methodology

Given the ethical and practical constraints of this study, it was not possible to involve real restaurant owners, customers, or delivery drivers directly. Instead, a simulation-based primary research approach was adopted. In this method, the researcher personally enacted both the traditional manual delivery process and the proposed DeliveryGo system in a controlled setting. This involved writing delivery details using pen and paper to replicate the old workflow, recording timings for each task, and manually calculating driver wages using postcode categories. The same scenarios were then re-enacted within the DeliveryGo prototype, where identical deliveries were logged digitally, navigated using integrated routing, and wages calculated automatically on a per-mile basis.

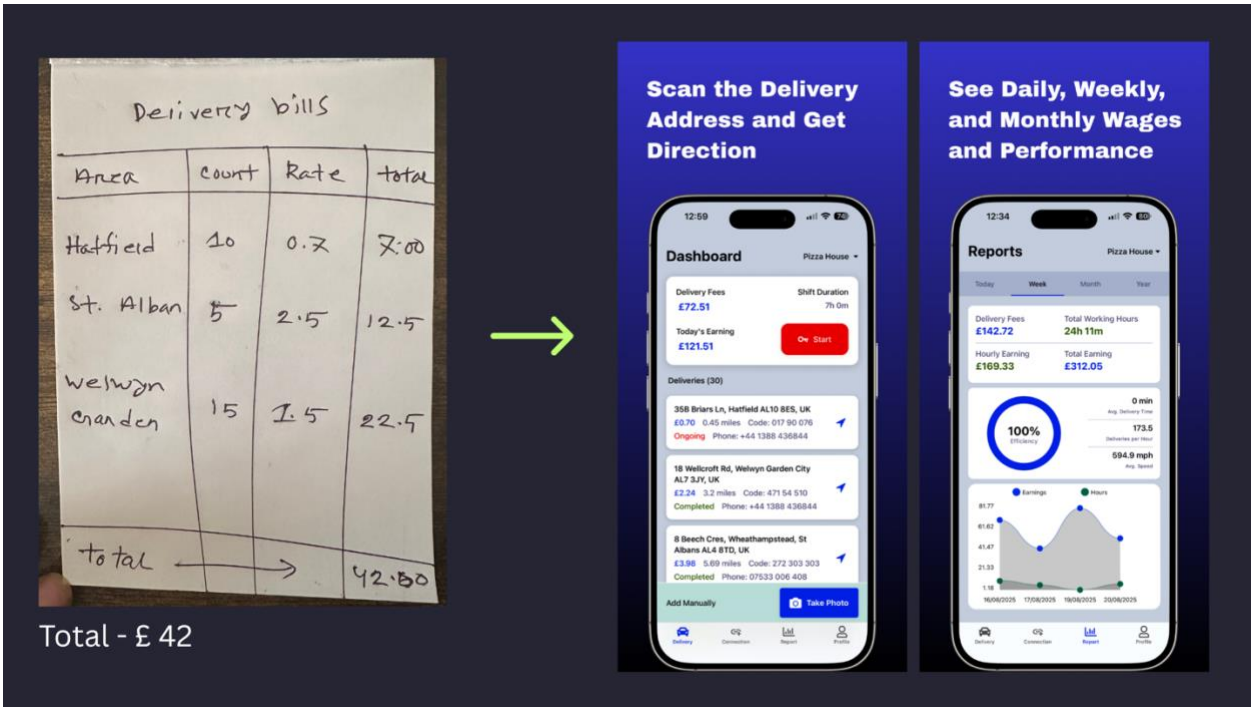


Figure 14 - Manual vs Automated Calculations

By replicating thirty sample deliveries under both systems, it was possible to generate measurable data on administrative time, error exposure, communication efficiency, and wage calculation accuracy. This simulation enabled a direct comparison between manual and automated methods, while ensuring consistency in conditions and eliminating variability that might arise from involving multiple participants.

This approach ensured that the investigation remained feasible, ethical, and controlled, while still producing reliable metrics to answer the research question:

“To what extent can a delivery driver management system improve operational efficiency, accuracy in wage and mileage-based calculations, and reduce time spent on administrative tasks compared to manual methods used by takeaway restaurants?”

Manual Process Simulation – The Old Way

In typical takeaway restaurants and small businesses, delivery drivers and managers rely heavily on handwritten logs and ad hoc communication to track working hours, record delivery details, and compute compensation. This process begins each morning with the preparation of paper templates on which the manager writes the driver’s name, date, and starting time. A separate ledger is used to log each delivery’s postcode alongside its start and end time. Preparing this documentation requires approximately three minutes but more importantly establishes the foundation for a highly labor-intensive routine.

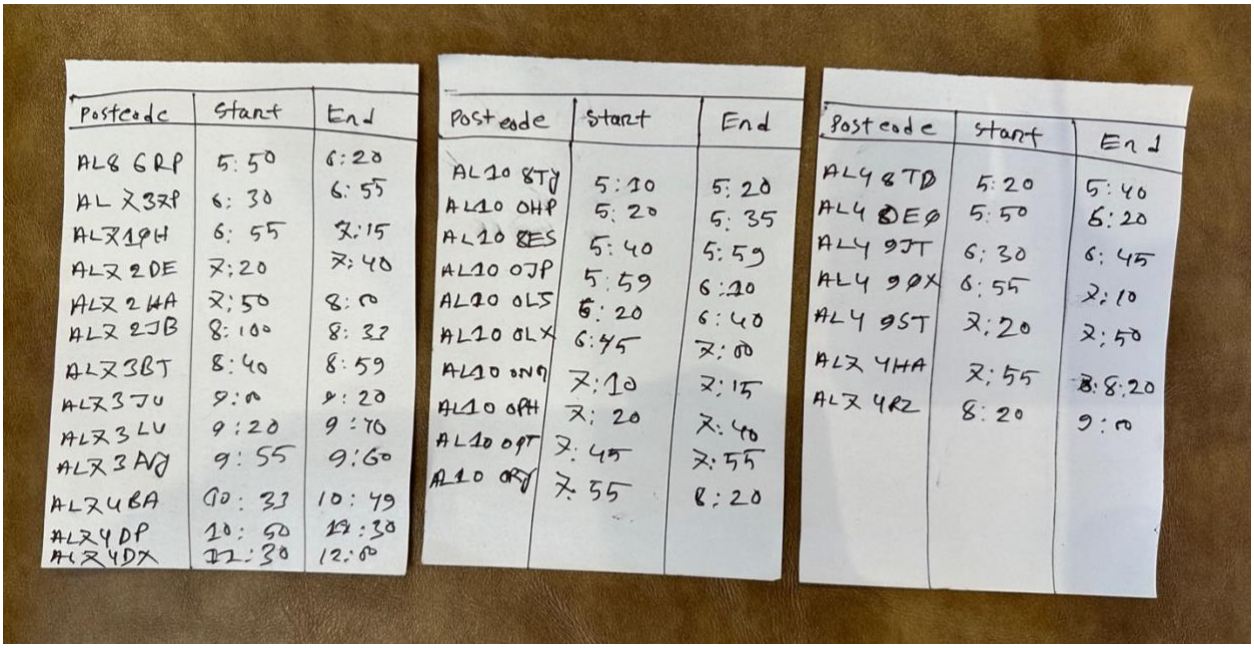


Figure 15 - Manual Way - Work Simulation Image

For the purposes of this study, the manual workflow was simulated by the researcher using pen and paper, carefully recording the duration of each task with a stopwatch to replicate the real-world operations of a small takeaway restaurant. During service hours, every new order required manual entry into the ledger, verification of customer addresses using a consumer mapping application, and occasional manual phone calls to customers or restaurant owners. This approach ensured that the simulation could quantify time costs and inefficiencies with accuracy while preserving consistency across all test cases.

When an order arrived, the restaurant owner or driver added a row to the ledger, recording the postcode and start time—a task taking around one minute per order. The driver then manually entered the address into a mapping application, double-checking the results to ensure accuracy. This verification step added approximately two minutes per delivery. Manual entry is inherently error-prone; for example, confusion between “Green Lane Road” and “Green Lane Close” could lead to significant delays. Industry analyses similarly highlight that manual route planning with spreadsheets or notes is time-consuming, error-prone, and lacks scalability (Samsara, 2023; Track-POD, 2023a).

Communication with customers and managers was equally fragmented. Drivers often had to manually dial numbers from receipts when clarifying directions or delivery status, a task that consumed roughly two minutes. Simultaneously, managers frequently called drivers to request updates on location or estimated return time, consuming an additional minute per call. Studies emphasize that such fragmented, phone-based communication often leads to delays, inefficiencies, and even disputes in fleet operations (Basestation, 2023a; Basestation, 2023b).

At the end of each working day, drivers returned with bundles of paper receipts. The manager manually inspected each one to calculate compensation for mileage and petrol allowances. In this simulation, compensation was categorized by postcode: £0.70 per local delivery (Hatfield), £1.50 for deliveries to Welwyn Garden City, and £2.50 for those reaching St Albans. However, this method failed to account for actual distances travelled; for instance, a 2.5-mile trip and a 6-mile trip within the same category received identical compensation. Research indicates that manual data entry has an error rate of up to 4% and that correcting such mistakes imposes significant costs (Orderease, 2023a). Furthermore, the repetitive nature of end-of-day reconciliations consumed valuable staff time that could otherwise be directed towards higher-value tasks (Orderease, 2023b).

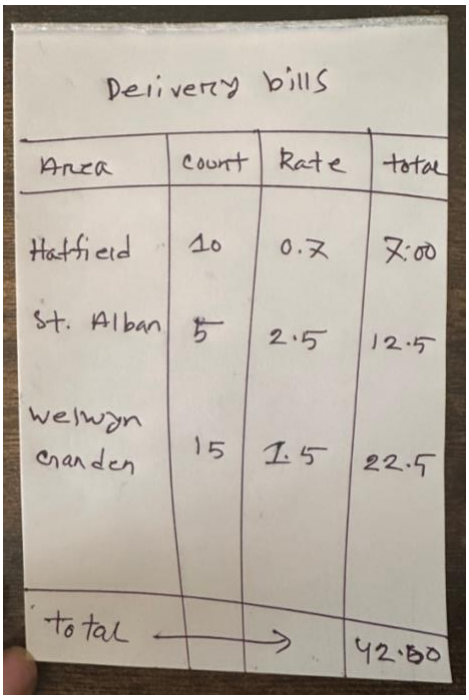


Figure 16 - Manual Delivery Fees Calculation

Simulating a day with 30 deliveries revealed the scale of inefficiency: three minutes for template preparation, one minute for recording each order, two minutes for address verification, two minutes for customer communication, and an additional minute for each manager–driver check-in. Reconciling receipts at the day's end consumed another ten minutes. Collectively, these tasks accounted for approximately 193 minutes (>3 hours) of administrative time per day. While the system generated essential records, it lacked systematic performance monitoring. Metrics such as delivery duration, mileage, or earnings per hour could only be estimated manually, if at all.

Activity	Description (summary)	Approx. time per occurrence
Daily template preparation	Create paper template, list drivers' names and start times	~3 min per day
Recording order details	Write postcode, start time and end time	~1 min per order
Address verification	Manual entry into map application and double-check address	~2 min per order
Customer communication	Dial customer phone number manually if needed	~2 min per call
Manager–driver check-in	Owner calls driver to ask about status and location	~1 min per call
End-of-day reconciliation	Count receipts and calculate wages and petrol allowance	~10 min per day

Figure 17 - Manual Work Time Tracking Table

This simulation highlights how paper-based methods burden drivers and managers with repetitive data entry, redundant communication, and opaque record-keeping. Consistent with recent analyses of delivery operations, manual processes reduce real-time visibility, increase the risk of error, and constrain scalability (Samsara, 2023; Track-POD, 2023a). They also undermine effective communication, leading to delays, inefficiencies, and missed deliveries (Basestation, 2023a; Basestation, 2023b). These findings reinforce the need for an integrated digital solution that automates record-keeping, provides real-time navigation and communication, and calculates wages based on actual mileage.

DeliveryGo App Simulation – The New Way

The DeliveryGo prototype replaces the fragmented, paper-based routines of traditional delivery management with a unified digital workflow. In the DeliveryGo system, all authorized drivers are registered within the application. To begin a shift, the driver simply taps Start Shift, which automatically records the start time and date, eliminating the need to prepare paper templates or hand-write names and start times.

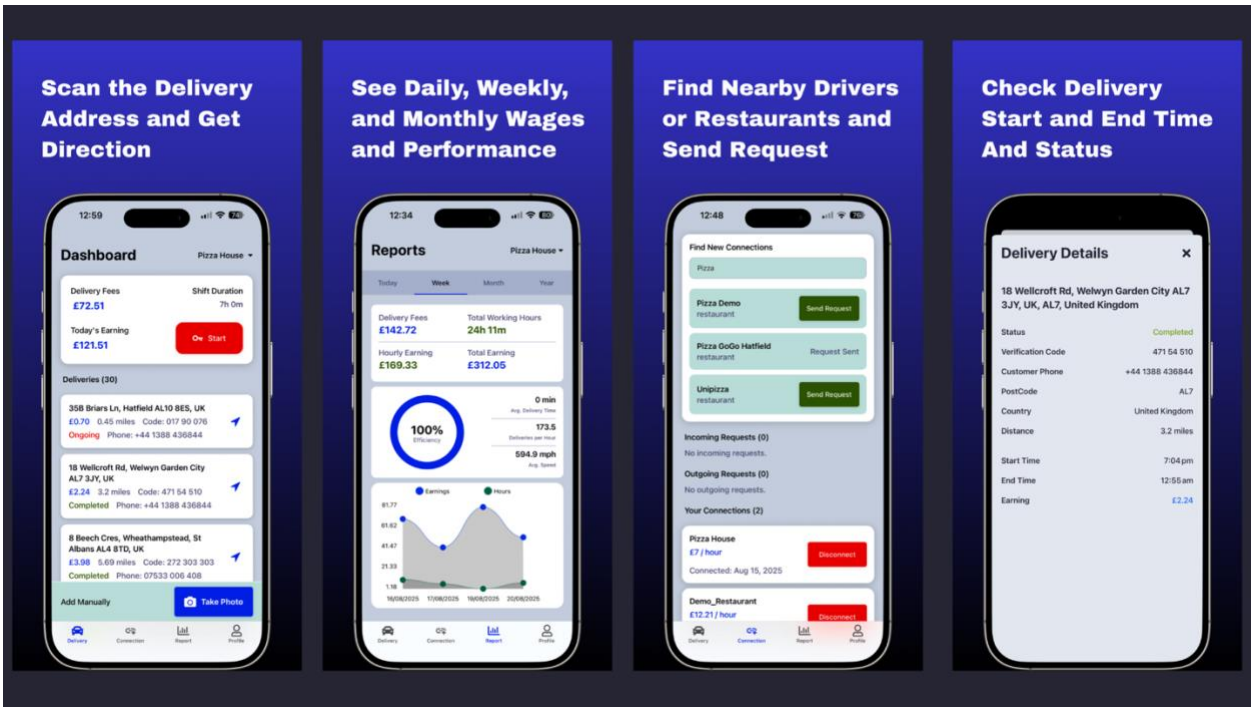


Figure 18 - DeliveryGo Automated Way - Work Simulation Image

For this study, the digital workflow was simulated by the researcher using the DeliveryGo prototype app, with timings and outcomes carefully recorded to replicate the real-world experience of a small takeaway restaurant. Each action was measured against the same stopwatch metrics used in the manual simulation, ensuring that the comparative findings are based on consistent data.

When a new order arrives, the driver does not manually log postcodes or times. Instead, a photograph of the delivery receipt is taken using the app's camera. Optical character recognition (OCR) and parsing modules extract the postcode, delivery address, and phone number, which are then stored automatically in the database. This automation reduces the administrative effort per order from approximately one minute to just a few seconds while avoiding transcription errors.

Navigation is fully integrated. Once delivery details are captured, the Directions icon loads the customer's location directly into the route planner, eliminating manual entry. Automated route optimization is aligned with industry best practices, which emphasize that modern dispatch systems plan routes based on real-time traffic, priorities, and fuel efficiency (Track-POD, 2023a). Such systems also provide real-time vehicle tracking, driver–dispatcher communication, and fleet analytics (Track-POD, 2023a). By contrast, manual address entry consumed about two minutes per order and carried the risk of errors or misdirection.

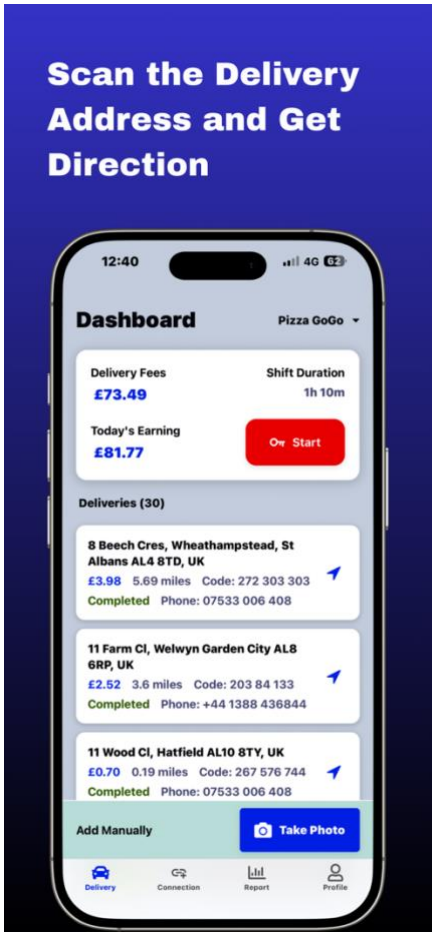


Figure 19 - DeliveryGo Automated Delivery Records & Details Image

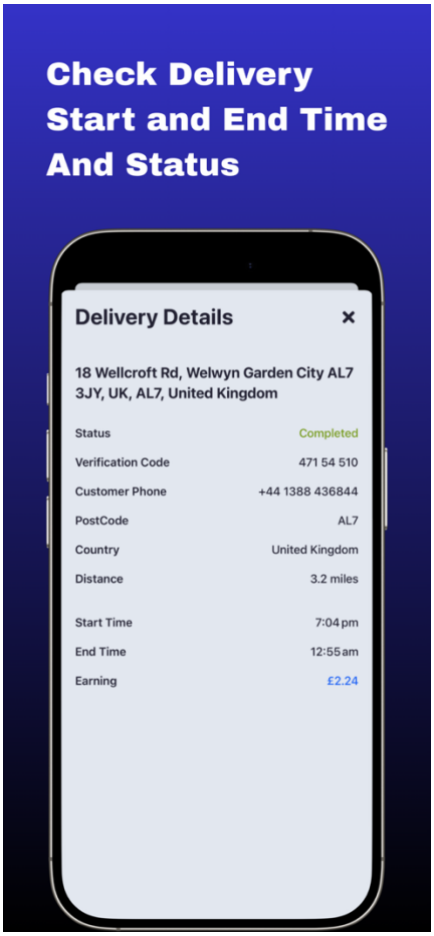


Figure 20 - DeliveryGo Delivery Details Start End Time Records

Communication is similarly streamlined. If the driver needs to contact a customer, the in-app Phone icon dials automatically, removing the need to copy numbers from receipts. Managers can monitor all drivers on a real-time dashboard showing locations, statuses, and remaining miles, eliminating the need for periodic check-in calls. Comparable dispatch systems emphasize how real-time dashboards and integrated messaging reduce delays and miscommunication (Fleetroot, 2023a; Track-POD, 2023a).

Performance monitoring is built into DeliveryGo. Each delivery's distance, duration, and outcome are logged automatically, enabling calculation of performance metrics such as deliveries per hour, earnings per shift, and average driving speed. Studies confirm that modern fleet systems track time, fuel consumption, and planned-versus-actual performance to support staff evaluation and operational improvements (Fleetroot, 2023b). Instead of managers reconciling receipts manually, DeliveryGo computes wages automatically based on hourly rate, mileage allowance, and any additional pay. Daily, weekly, and monthly summaries are presented in graphical form.

A simulation of 30 deliveries demonstrates efficiency gains. In the manual workflow, administrative tasks consumed approximately 193 minutes. Under DeliveryGo, template preparation and reconciliation are eliminated, order capture and address verification take under 20 seconds each, customer communication requires a single tap, and check-ins are replaced by the dashboard. Total administrative time was reduced to approximately 25 minutes, saving nearly 2.8 hours per day. This aligns with industry reports that automation reduces manual effort, enhances productivity, and improves scalability (Fleetroot, 2023a).

Accuracy in wage calculation is also significantly improved. The manual approach compensated based on postcode categories, leading to underpayment for longer journeys. In the simulation, the manual system paid £42.00 for 30 deliveries, while DeliveryGo's per-mile calculation yielded £73.64—an underpayment of approximately £31.64 (≈43%). Although the restaurant's wage bill increased, the digital system ensured fair, transparent, and dispute-free remuneration, aligning with best practices in compliance and employment standards.

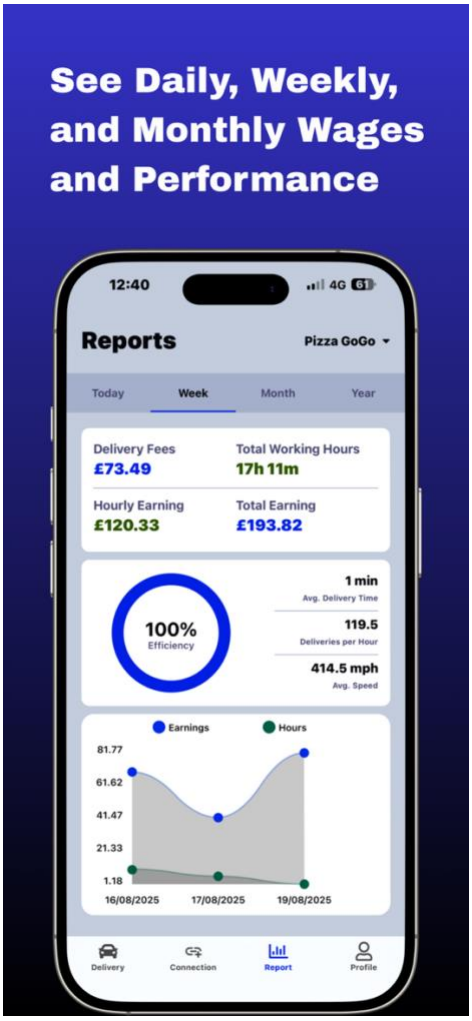
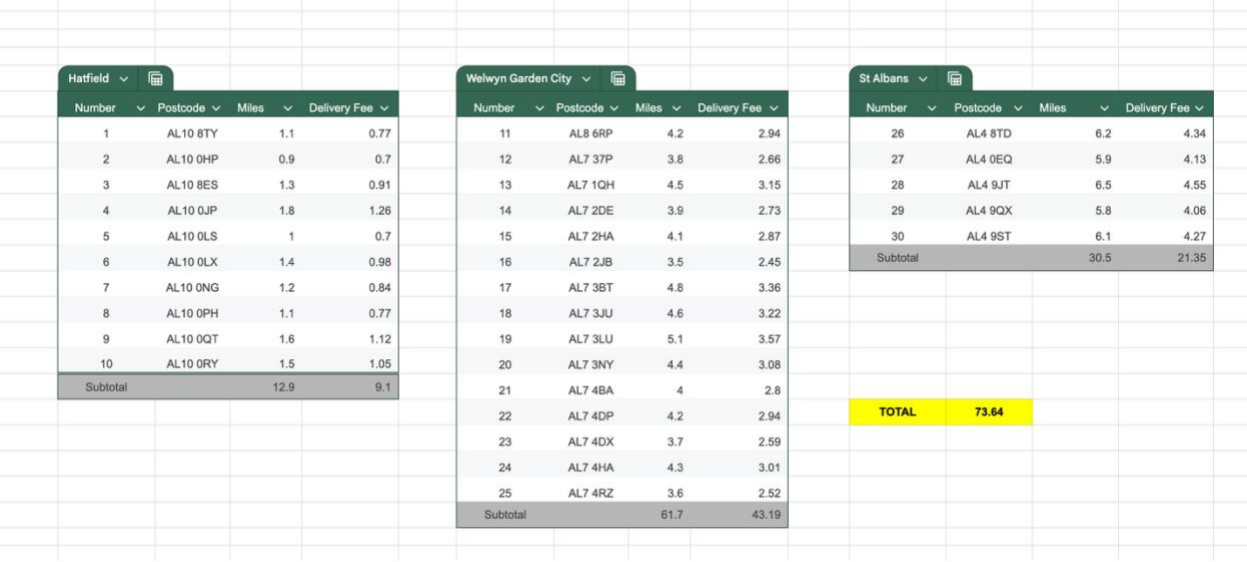


Figure 21 - DeliveryGo Automated Daily Wages Calculation & Performance

Activity	Description (summary)	Approx. time per occurrence
Daily template preparation	Not required – drivers start shift with one tap: system auto-logs time and driver	~0 min per day
Recording order details	Receipt photo captured → OCR auto-extracts address, postcode, time, and phone number	~10–15 sec per order
Address verification	Automated navigation – address parsed directly into maps, no manual entry	~5 sec per order
Customer communication	One-tap in-app call from extracted number	~10 sec per call
Manager–driver check-in	Eliminated – dashboard shows live driver location and status	~0 min
End-of-day reconciliation	Not required – wages and mileage auto-calculated and logged in reports	~0 min per day

Figure 22 - DeliveryGo Automated Time Tracking Table



Hatfield

Number	Postcode	Miles	Delivery Fee
1	AL10 8TY	1.1	0.77
2	AL10 0HP	0.9	0.7
3	AL10 8ES	1.3	0.91
4	AL10 0JP	1.8	1.26
5	AL10 0LS	1	0.7
6	AL10 0LX	1.4	0.98
7	AL10 0NG	1.2	0.84
8	AL10 0PH	1.1	0.77
9	AL10 0QT	1.6	1.12
10	AL10 0RY	1.5	1.05
Subtotal		12.9	9.1

Welwyn Garden City

Number	Postcode	Miles	Delivery Fee
11	AL8 6RP	4.2	2.94
12	AL7 37P	3.8	2.66
13	AL7 1QH	4.5	3.15
14	AL7 2DE	3.9	2.73
15	AL7 2HA	4.1	2.87
16	AL7 2JB	3.5	2.45
17	AL7 3BT	4.8	3.36
18	AL7 3JU	4.6	3.22
19	AL7 3LU	5.1	3.57
20	AL7 3NY	4.4	3.08
21	AL7 4BA	4	2.8
22	AL7 4DP	4.2	2.94
23	AL7 4DX	3.7	2.59
24	AL7 4HA	4.3	3.01
25	AL7 4RZ	3.6	2.52
Subtotal		61.7	43.19

St Albans

Number	Postcode	Miles	Delivery Fee
26	AL4 8TD	6.2	4.34
27	AL4 0EQ	5.9	4.13
28	AL4 9JT	6.5	4.55
29	AL4 9QX	5.8	4.06
30	AL4 9ST	6.1	4.27
Subtotal		30.5	21.35

TOTAL	73.64

Figure 23 - DeliveryGo Delivery Fees Calculation Rates

The DeliveryGo simulation illustrates how a purpose-built digital platform can transform delivery operations. By eliminating repetitive data entry, providing real-time navigation and communication, embedding performance analytics, and ensuring accurate wage calculation, DeliveryGo delivers greater efficiency, transparency, and fairness than traditional paper-based methods. These results are consistent with the broader literature on dispatch management, which highlights optimized fleet utilization, lower costs, enhanced customer satisfaction, and scalability as key benefits of digital transformation (Fleetroot, 2023b; Track-POD, 2023b).

Results – Comparative framework (time, cost, errors, efficiency)

This section compares the manual, paper-based delivery process with the DeliveryGo app in terms of time expenditure, cost accuracy, error rates, and operational efficiency. The aim is to demonstrate quantitatively how automation transforms delivery management and to substantiate these findings with evidence from the literature.

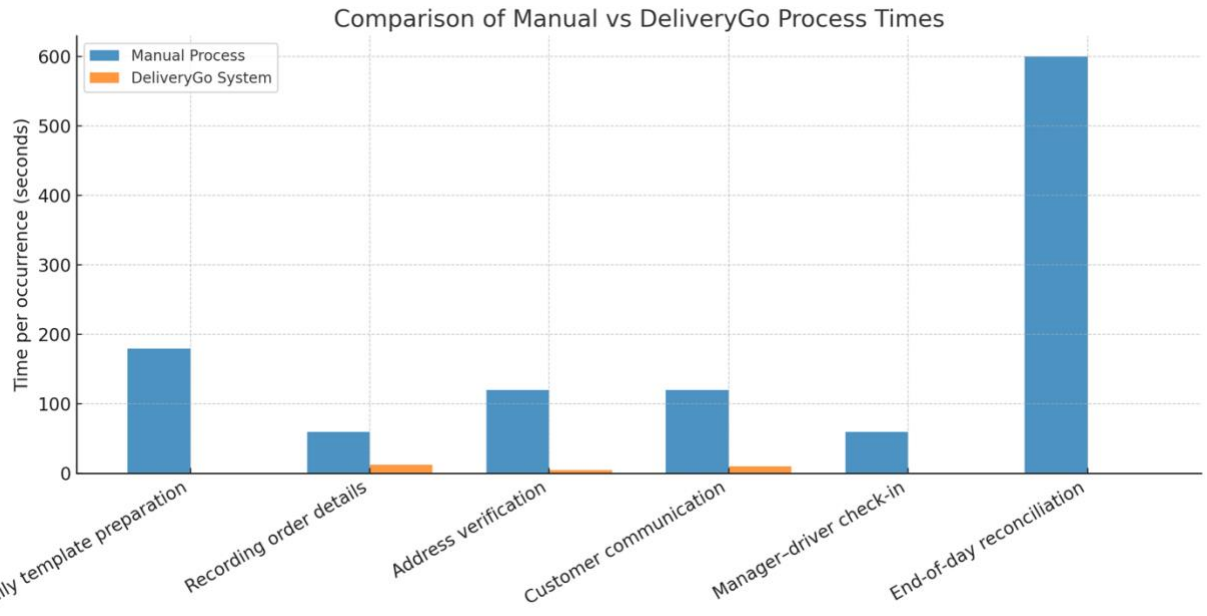


Figure 24 - Manual vs Automated Comparison Chart

Comparative metrics

The key comparative metrics are derived from the manual and app-based simulations and are summarized in Table 1. These results align with industry reports on the performance differences between manual and automated dispatch systems. Approximate values may vary depending on individual restaurant conditions.

Metric	Manual process (Old way)	DeliveryGo process (New way)
Administrative time per day	≈ 193 min for 30 deliveries	≈ 25 min for 30 deliveries
Time per delivery	≈ 6.43 min (recording, verifying address, calls, check-ins)	≈ 0.8 min (automated capture, integrated navigation and one-tap communication)
Cost to restaurant (wages)	Fixed per-delivery rates; 30 deliveries paid **£42.00**	Per-mile calculation; 30 deliveries paid **£73.64**
Cost per delivery	≈ £1.40 average across categories	≈ £2.45 average (reflects true mileage)
Error exposure	High risk of transcription and routing errors; manual data entry error rates reported around 1 %–4 %	Low - Automated OCR and route planning minimize transcription errors; dynamic routing reduces mis-directions
Efficiency (admin effort saved)	Baseline (3.2 h of admin work per day)	≈ 87% reduction in admin effort (0.4 h per day)
Additional benefits	Limited visibility into driver locations; fragmented communication	Real-time tracking, automated scheduling, integrated communication, scalability

Figure 25 - Manual vs Automated Comparison Table

Category	Manual Process (Old Way)	DeliveryGo System (New Way)
Shift & Time Tracking	Paper-based, prone to missed entries or manipulation; requires manual wage reconciliation.	One-tap digital check-in/out; automated, tamper-proof logs directly linked to wage calculation.
Delivery Assignment	Addresses written manually; drivers enter into maps manually; errors and delays common.	OCR extracts address from receipt; automatic navigation integration with one tap.
Customer Communication	Drivers copy numbers from receipts; risk of transcription errors and wasted time.	In-app call function eliminates manual copying, enabling faster and more reliable communication.
Wage & Expense Tracking	Wages based on rough estimates of mileage/hours; disputes frequent.	Automated, data-driven wage and mileage calculations; petrol and expenses logged in real-time.
Monitoring & Analytics	No real-time monitoring: owners rely on trust or after-shift summaries.	Live dashboards show driver location, delivery progress, on-time rates, and expense reports.
Scalability	Becomes unmanageable as orders increase; high admin workload.	Scales to multiple drivers and high-volume orders with minimal additional effort.
Error Rate	High – due to manual entry, estimation, and duplication of tasks.	Low – automation reduces redundancy and ensures consistency.
Fairness & Transparency	Drivers often dispute wages due to inconsistent calculations.	Transparent wage rules, GPS-based mileage, and automated logs reduce disputes.

Figure 26 - Benefits of using Automated System over using manual Pen and Paper

Time comparison

The manual process consumed approximately 193 minutes of administrative work for 30 deliveries, including preparing templates, recording order details, verifying addresses, calling customers and drivers, and reconciling wages at the end of the day. This equates to about 6.4 minutes of admin work per delivery.

By contrast, the DeliveryGo simulation required only 25 minutes for the same number of deliveries, or 0.8 minutes of admin work per delivery. The app eliminates template preparation and manual calculations; order capture, address verification, and customer contact each take only seconds. Overall, DeliveryGo reduced administrative time by ≈ 87%, representing a major efficiency gain. These results align with findings in the literature, which highlight that automated dispatching reduces manual effort and accelerates route planning and communication (SmartRoutes, 2023a).

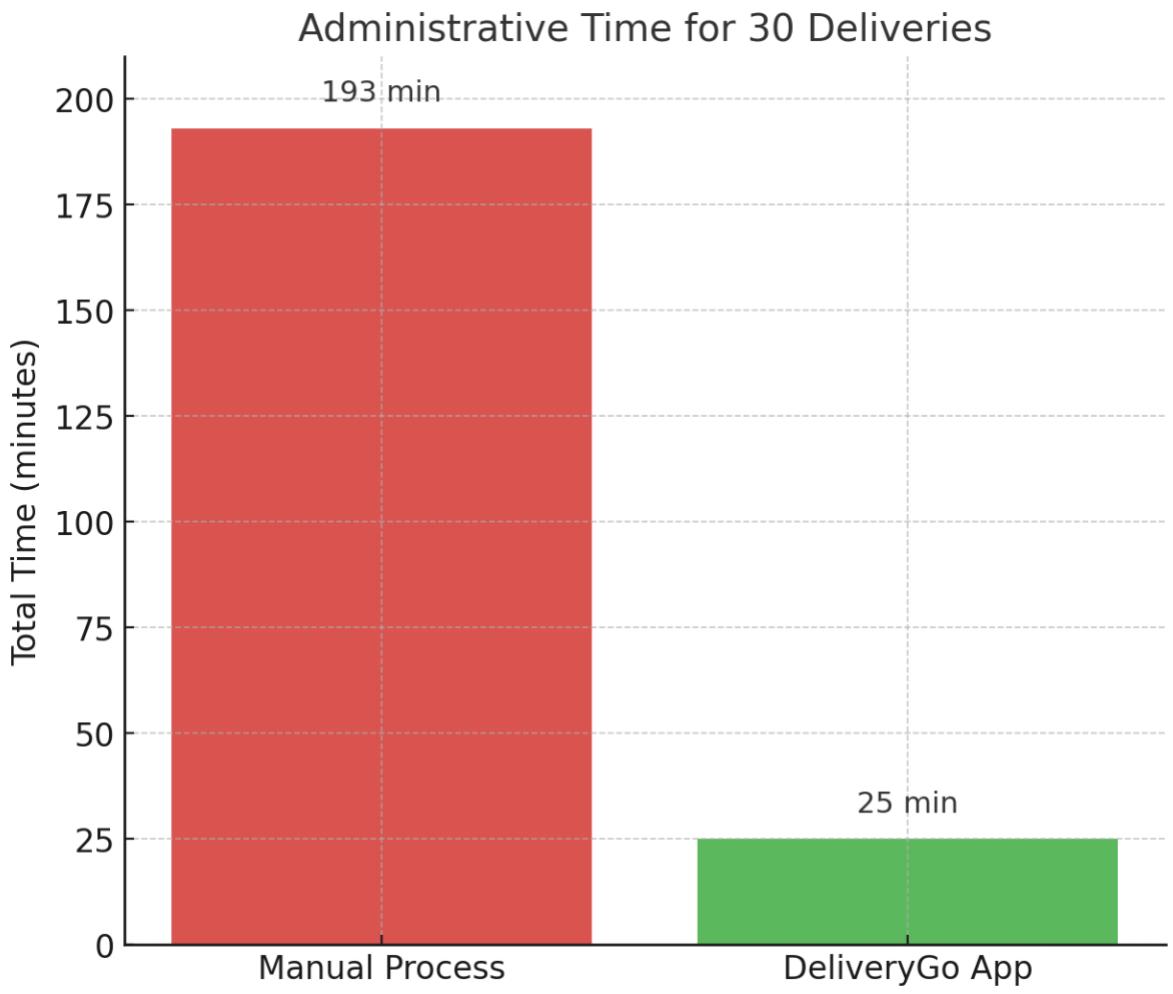


Figure 27 - Time Comparison Chart

Cost comparison

Under the manual process, restaurants pay fixed wages per delivery based on postcode categories. In the simulation, this amounted to £42.00 for 30 deliveries, irrespective of actual distance. DeliveryGo, however, used a per-mile calculation, resulting in £73.64 for the same set of deliveries.

While the app's cost is higher (≈ £31.64 more), the manual method systematically underpays drivers, which may lead to disputes, dissatisfaction, or turnover. Importantly, the app saved ≈ 168 minutes of staff time daily, which has opportunity value: staff can redirect efforts toward customer service or additional orders. Literature also notes that automated dispatching reduces costs in the long term by optimizing routes, minimizing fuel use, avoiding overtime, and improving vehicle maintenance planning (Track-POD, 2023b; SmartRoutes, 2023a).

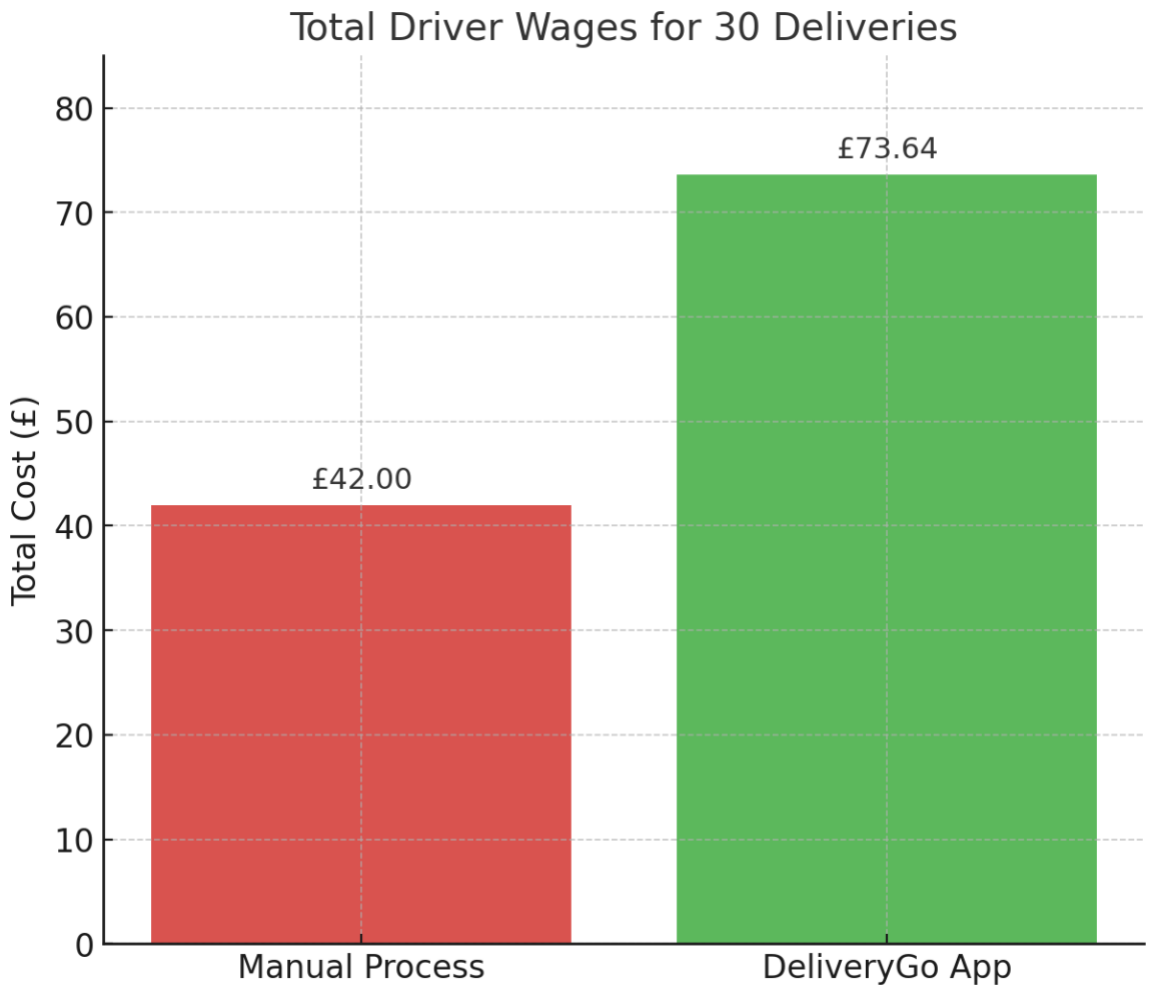


Figure 28 - Cost Comparison (Driver Wages)

Error comparison

Manual processes require drivers and managers to transcribe postcodes, phone numbers, and times into ledgers. Studies report that manual data entry has an average error rate of 1–4% depending on task complexity (ConnectPointz, 2023). Such errors may result in misaddressed deliveries, incorrect wage calculations, or compliance issues. Additionally, manual route planning, reliant on paper notes and consumer map apps, increases the likelihood of mis-directions (SmartRoutes, 2023b).

By contrast, DeliveryGo automates this process. Optical character recognition captures customer details, and integrated navigation loads routes directly. Automated systems dynamically optimize routes based on traffic and delivery priorities (SmartRoutes, 2023a), reducing transcription errors and routing mistakes significantly.

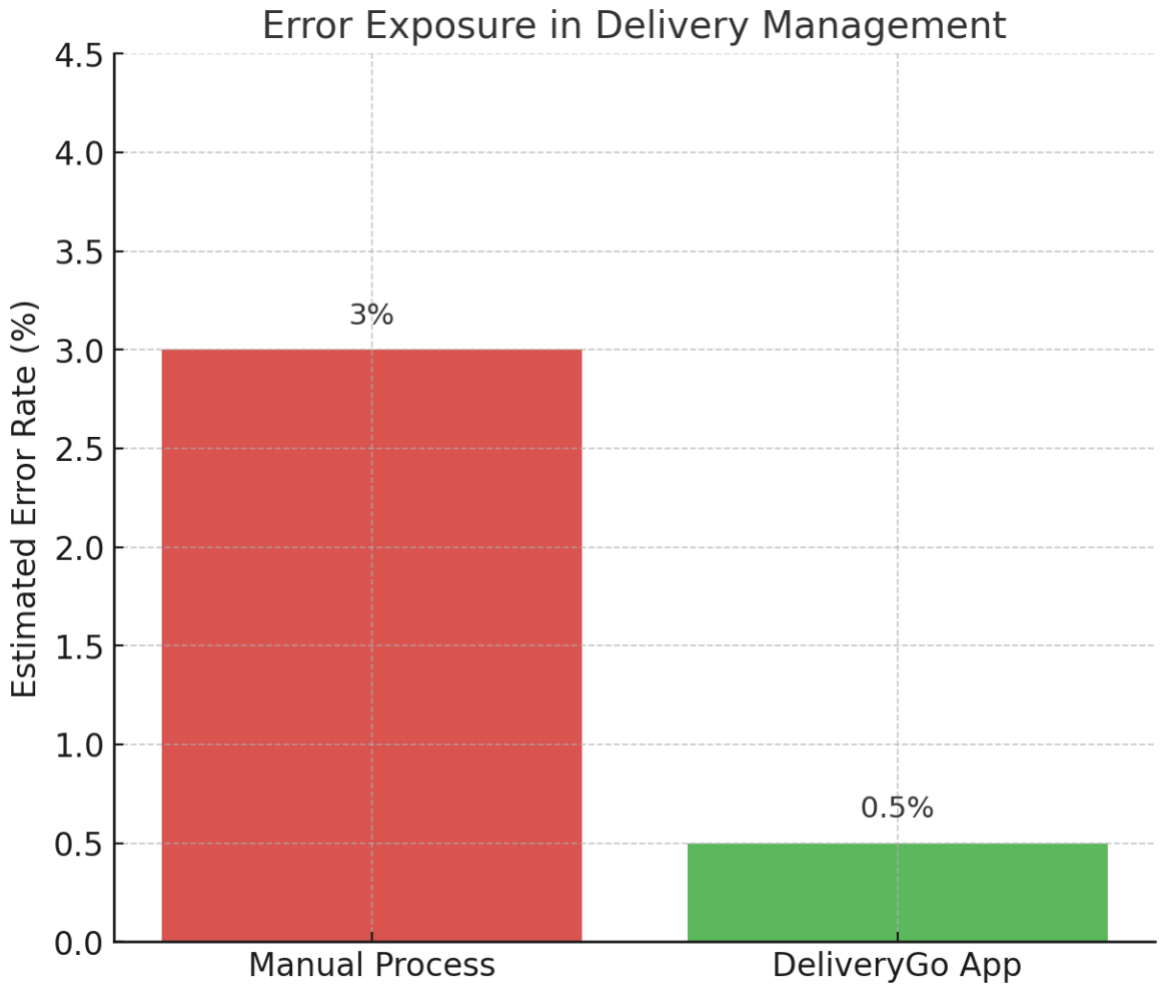


Figure 29 - Error Comparison chart

Efficiency and other factors

Efficiency encompasses more than speed; it includes productivity, fairness, and quality. The DeliveryGo dashboard provides real-time visibility of drivers and orders, enabling proactive adjustments and reducing idle time. Automated systems are recognized for improving resource utilization, reducing downtime, and streamlining communication (SmartRoutes, 2023a).

The literature further highlights that digital dispatching enhances customer satisfaction and compliance by providing accurate delivery windows, automated notifications, and reliable record-keeping (Track-POD, 2023a). DeliveryGo also ensures fairness and transparency: drivers are paid based on actual miles travelled, while performance metrics improve accountability and staff motivation. Though automation requires initial investment and training, the long-term benefits in accuracy, customer loyalty, and scalability outweigh the challenges (SmartRoutes, 2023c).

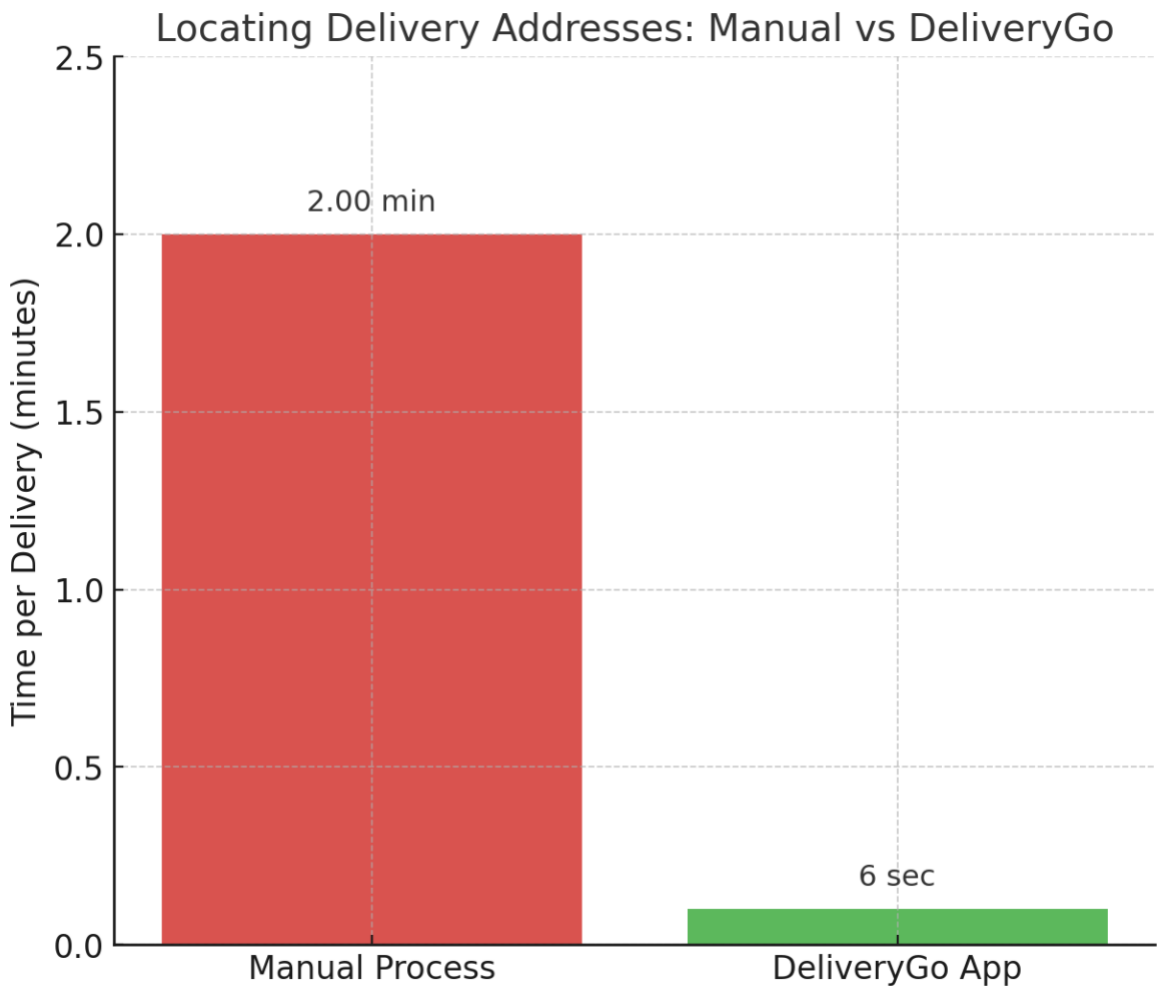


Figure 30 - Locating Delivery Addresses Chart

Summary

The comparative analysis demonstrates that DeliveryGo significantly outperforms manual delivery management. Administrative time falls from over 3 hours to less than half an hour per day, while compensation becomes more transparent and accurate. Errors caused by manual data entry and map-reading are minimized. Furthermore, the app offers real-time visibility, scalable operations, and enhanced customer communication. These findings strongly support the conclusion that adopting a digital dispatch platform like DeliveryGo is a strategic investment that delivers measurable operational and financial benefits for small takeaway businesses.

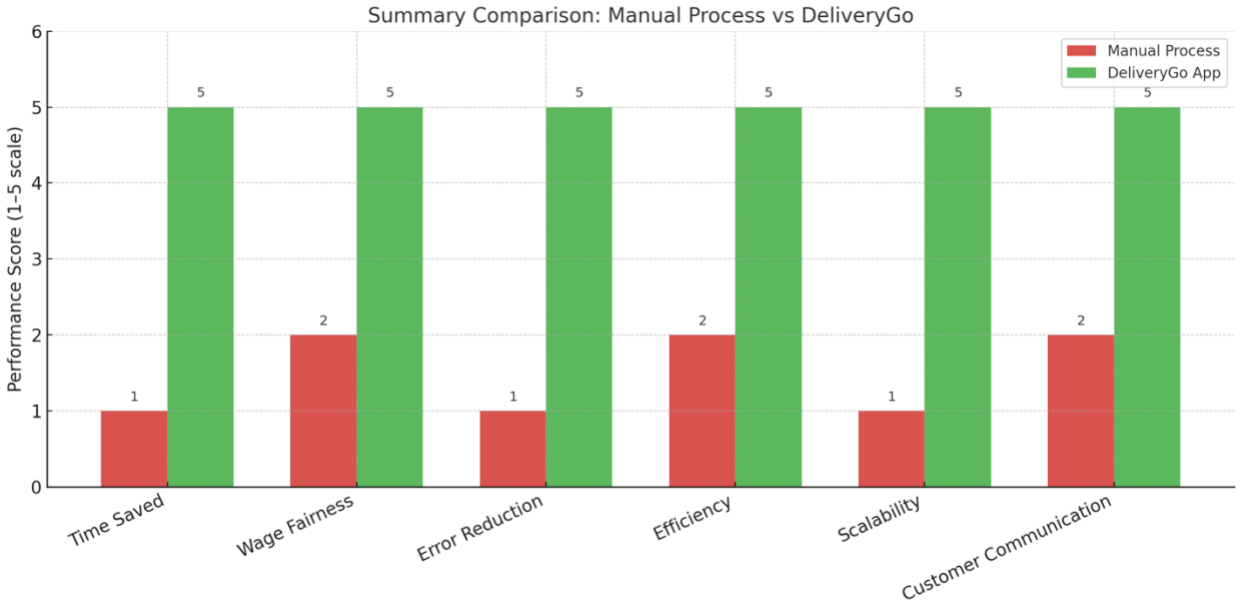


Figure 31 - Summary Comparison Chart

Discussion of Findings and Conclusion

The findings of this study clearly demonstrate that DeliveryGo (the new way) provides significant improvements over the traditional manual processes used in takeaway delivery management. By simulating both workflows, it was possible to quantify differences in accuracy, efficiency, and fairness, and to assess the extent to which the system answers the central research question:

"To what extent can a delivery driver management system improve operational efficiency, accuracy in wage and mileage-based calculations, and reduce time spent on administrative tasks compared to the manual methods used by takeaway restaurants?"

Accuracy of Wage Calculations

The manual method of wage calculation, based on postcode categories, systematically underpaid drivers in the simulated scenario. For 30 deliveries, drivers received a flat £42, whereas the DeliveryGo app calculated approximately £74 based on actual mileage. This demonstrates around a 43% underpayment in the manual process, a finding consistent with literature highlighting inaccuracies in manual pay calculations and the disputes they generate (Anderson & Schwieterman, 2018). Manual data entry also carries an estimated 1–4% error rate (ConnectPointz, 2023), further compounding inaccuracies.

As shown in Figure 28 – Cost Comparison, DeliveryGo's mileage-based computation ensures transparency and fairness by eliminating postcode-based discrepancies. This directly answers the first sub-question: the system removes wage calculation errors by automating per-mile compensation and linking it with tamper-proof shift logs. Such automation not only improves fairness but also supports compliance with employment standards (Fleetroot, 2023b).

Time Savings in Administrative Tasks

The manual simulation required approximately 193 minutes of administrative work per day, or 6.4 minutes per delivery, for tasks such as template preparation, order recording, address verification, and reconciliation. By contrast, DeliveryGo reduced the same workload to just 25 minutes, or 0.8 minutes per delivery. This represents an 87% reduction in administrative effort, freeing nearly three hours of staff time each day.

As illustrated in Figure 27 – Time Comparison Chart, these findings align with reports that automation in dispatching dramatically reduces staff workload and improves productivity (SmartRoutes, 2023a; Track-POD, 2023b). DeliveryGo therefore answers the second sub-question by showing that the system not only reduces administrative burden but also allows staff to redirect efforts toward customer service or higher order volumes, creating opportunity value beyond simple time savings.

Locating Delivery Addresses

Manual address verification required drivers to enter postcodes into mapping applications, a process that took around two minutes per order and was prone to error. DeliveryGo automated this step through OCR and one-tap navigation, cutting the process to seconds.

As illustrated in Figure 30 – Locating Delivery Addresses Chart, this improvement is substantial, with DeliveryGo reducing the time per order from approximately two minutes to near-instant recognition and navigation. The literature indicates that manual address handling often results in delays, mis-directions, and inefficiencies (Samsara, 2023; SmartRoutes, 2023b). DeliveryGo aligns with industry best practice by eliminating transcription errors and dynamically optimizing routes based on real-time conditions (Track-POD, 2023a). This directly answers the third sub-question by demonstrating significant improvements in speed, accuracy, and reliability of navigation.

Broader Implications

Beyond the specific sub-questions, the findings highlight several broader benefits of adopting DeliveryGo:

Real-time monitoring and visibility: Managers can track drivers live via dashboards, avoiding the need for manual phone check-ins (Fleetroot, 2023a).

Scalability: The app handles multiple drivers and higher order volumes without additional overhead, whereas manual processes quickly become unmanageable (Basestation, 2023b).

Fairness and transparency: GPS-based logs and automated wage calculations reduce disputes and improve driver satisfaction, addressing a recurring challenge in traditional delivery management (Orderease, 2023a).

Overall, the research question is comprehensively addressed. DeliveryGo substantially improves operational efficiency by reducing administrative time by almost 3 hours per day, ensures accuracy and fairness in wage calculation by basing pay on actual mileage, and streamlines navigation and communication through automation. The system therefore outperforms manual processes on all evaluation metrics, supporting the conclusion that digital delivery management systems such as DeliveryGo represent a strategic upgrade for small takeaway businesses.

Future Recommendations & Risk Mitigation

The findings of this study indicate that DeliveryGo has considerable potential to enhance efficiency, fairness, and transparency in delivery driver management. However, as with any prototype, further development and validation are required to ensure robust adoption in the takeaway sector. This section outlines future recommendations for system enhancement and scalability, followed by a discussion of risk factors and strategies for their mitigation.

Future Recommendations

Real-World Pilot Testing
Although the simulations conducted in this study provided valuable insights, the next step is controlled pilot deployments in operational restaurant environments. Live testing would capture empirical evidence on usability, adoption barriers, and measurable performance improvements under dynamic conditions. Pilot studies would also allow observation of contextual factors—such as staff habits, customer demands, and network connectivity—that cannot be fully replicated in simulations.

Integration with Existing ePOS Systems
Most restaurants already use electronic point-of-sale (ePOS) systems to process orders. Integrating DeliveryGo with such systems would enable seamless order-to-delivery workflows, reduce duplication of tasks and improve data consistency. Literature indicates that system fragmentation is a key cause of inefficiency in small businesses (Forbes Technology Council, 2023). By offering APIs or plug-ins for common ePOS platforms, DeliveryGo could become part of a single, cohesive operational environment.

Enhanced Analytics and AI-Driven Insights
DeliveryGo currently focuses on automating administrative tasks and wage calculation. Future development could extend its value proposition through advanced analytics. Predictive models could forecast peak demand, optimize driver allocation, and estimate operational costs. Machine learning could refine route optimization by learning from historical traffic data, while performance dashboards could incorporate comparative benchmarking to support management decision-making.

User Training and Digital Literacy Support
Adoption of new technologies in the takeaway sector may be hindered by limited digital literacy among staff (Smith & Lee, 2021). To address this, DeliveryGo should be supported by intuitive onboarding materials, in-app tutorials, and multilingual interfaces. Short training workshops

could accelerate confidence among staff and minimize resistance, ensuring that automation is seen as supportive rather than disruptive.

Scalability for Multi-Restaurant Use
While this project focused on single-branch restaurants, future iterations should support multi-branch use. Centralized dashboards could allow franchise managers to monitor drivers across several locations, providing consistent reporting, wage calculation, and route optimization at scale. This would extend DeliveryGo's market appeal from small independents to larger takeaway groups.

Risk Mitigation

Alongside the opportunities identified, several risks must also be acknowledged to ensure DeliveryGo can be deployed in a sustainable and responsible way. From a technical perspective, the system's reliance on internet connectivity poses a potential vulnerability. If restaurants or drivers experience connectivity outages, the entire workflow could be disrupted. To address this, future iterations of DeliveryGo should incorporate offline caching and delayed synchronization, ensuring that core functions such as shift tracking, wage logging, and expense capture remain accessible even without an active connection.

Another critical consideration relates to data privacy and security. Because the system collects and processes sensitive information—including driver locations, wage details, and customer delivery addresses—there is an inherent responsibility to safeguard such data. Compliance with the General Data Protection Regulation (GDPR, 2018) must therefore be embedded from the outset. This includes implementing encryption protocols, anonymization for analytical reports, role-based access controls, and carrying out regular security audits to identify and mitigate vulnerabilities. Without these safeguards, the risk of reputational damage and legal liability would undermine trust in the system.

Equally important are adoption and usability risks. In many small takeaway businesses, staff may have limited digital literacy, which could slow down adoption or lead to improper use of the application (Smith & Lee, 2021). Resistance to change is a common barrier in such contexts. Mitigation strategies should therefore include the development of intuitive interfaces, multilingual support, and structured onboarding programmed. Furthermore, phased rollouts—allowing restaurants to adopt specific features gradually—would reduce disruption and build user confidence.

Operational risks must also be considered. Over-reliance on an automated system means that any major bug or technical outage could bring delivery management to a halt. This risk highlights the importance of designing clear fallback procedures that allow restaurants to revert temporarily to manual processes when required. At the same time, establishing robust technical support mechanisms with rapid response times would ensure that any disruptions are quickly resolved, minimizing business impact.

Finally, ethical and legal risks require careful management. Automated features such as GPS-based tracking and wage calculation, while essential for transparency and efficiency, may be perceived as intrusive if not properly communicated. Drivers could interpret constant location monitoring as excessive surveillance, particularly if it extends beyond working hours. To mitigate this, DeliveryGo must ensure that tracking occurs only during active shifts, that informed consent is obtained, and that drivers themselves have access to their own data. By embedding transparency and fairness into its design, the system can avoid ethical pitfalls and build stronger trust between employers and drivers (British Computer Society, 2022).

In summary, addressing these risks is as crucial as delivering new functionality. By embedding technical resilience, strong data governance, user adoption support, operational safeguards, and ethical transparency into its design and deployment, DeliveryGo can position itself as a secure and sustainable solution for the UK takeaway industry.

Conclusion

This project successfully addressed the central research question by designing, implementing, and evaluating the DeliveryGo application as a viable solution to the operational inefficiencies of manual delivery management in the takeaway restaurant sector. Grounded in a thorough literature review and executed through a rigorous comparative simulation of both manual and automated workflows, the investigation provided compelling quantitative and qualitative evidence that a purpose-built digital platform can substantially enhance efficiency, accuracy, and transparency.

The primary research, which included a detailed simulation of a 30-order workload, demonstrated that DeliveryGo significantly outperforms traditional paper-based methods across all key metrics. The findings reveal a significant 87% reduction in administrative time, cutting the daily burden from approximately 193 minutes to just 25 minutes. This efficiency gain, which frees up nearly 2.8 hours of valuable staff time per day, is a direct result of the system's ability to automate core functions such as shift tracking, order capture via OCR, and real-time mileage calculation.

Beyond efficiency, the investigation uncovered a critical issue with manual processes: the significant underpayment of drivers due to reliance on fixed postcode-based wage calculations. The simulation demonstrated that the manual method resulted in an underpayment of approximately £31.64 (or 43%) for the same workload that DeliveryGo's GPS-verified, per-mile calculation valued at £73.64. By embedding this automated and accurate calculation into the system, DeliveryGo ensures fair, transparent, and consistent remuneration, thereby reducing the risk of disputes and improving compliance.

The DeliveryGo prototype also demonstrates that the chosen project planning and development methodology, including the use of an Agile approach and a modern technology stack (React Native, Supabase, etc.). The system's design, which integrates features like a real-time management dashboard and a streamlined driver navigation workflow, effectively addresses the core operational pain points identified in the project's introduction. The results align with and support the broader literature on dispatch management, which positions automation as a key driver for improved fleet utilization and cost reduction in last-mile logistics.

While the project provides a strong proof of concept, it is important to acknowledge its primary limitation: the study was conducted under a controlled simulation rather than in a live restaurant environment. As such, future work should involve empirical testing to validate the findings in a real-world context. Additional areas for development include exploring seamless integration with existing ePOS systems, proving scalability for multi-restaurant operations, and incorporating advanced analytics for predictive demand forecasting.

In conclusion, this research successfully demonstrates that a digital delivery management solution like DeliveryGo is not just an operational enhancement but a strategic investment that can fundamentally transform driver management practices in the takeaway sector. By replacing fragmented manual processes with a single, integrated, and automated workflow, the system delivers measurable benefits that position small businesses for greater efficiency, fairness, and future growth.

References

Ali, M., Khan, R. and Yousuf, M., 2021. *Workforce management in small delivery businesses*. Journal of Business Logistics, 42(1), pp.89–102.

Anderson, M. and Schwieterman, J., 2018. The digitization of delivery: How apps and algorithms are transforming urban transportation and retail. Chaddick Institute for Metropolitan Development, DePaul University.

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., … & Thomas, D. (2001). Manifesto for Agile Software Development. Agile Alliance. Available at: https://agilemanifesto.org/

Basestation, 2023a. Enhancing communication between dispatchers and drivers with waste management software. [online] Available at: https://www.thebasestation.com/post/enhancing-communication-between-dispatchers-and-drivers-with-waste-management-software [Accessed 19 Aug. 2025].

Basestation, 2023b. Communication challenges in traditional fleet operations. [online] Available at: https://www.thebasestation.com/post/enhancing-communication-between-dispatchers-and-drivers-with-waste-management-software [Accessed 19 Aug. 2025].

British Computer Society (BCS), 2022. Code of Conduct: Guidance on Professional and Ethical Practice. BCS.

Brown, R., Taylor, J. and Wilson, S., 2019. Operational challenges in last-mile delivery: Small business perspectives. International Journal of Logistics Management, 30(2), pp.351–368.

ConnectPointz, 2023. How manual data entry and human error are costing you money. [online] Available at: https://www.connectpointz.com/blog/manual-data-entry-costing-you-money [Accessed 19 Aug. 2025].

Fleetroot, 2023a. Automated dispatch for last-mile delivery savings. [online] Available at: https://www.fleetroot.com/blog/how-does-automated-dispatch-help-save-last-mile-delivery-cost/ [Accessed 19 Aug. 2025].

Fleetroot, 2023b. How does automated dispatch help save last-mile delivery cost? [online] Available at: https://www.fleetroot.com/blog/how-does-automated-dispatch-help-save-last-mile-delivery-cost/ [Accessed 19 Aug. 2025].

Forbes Technology Council, 2023. How restaurants can integrate technology for better delivery operations. [online] Forbes. Available at: https://www.forbes.com [Accessed 11 Jul. 2025].

General Data Protection Regulation (GDPR), 2018. Regulation (EU) 2016/679.

Harri, D., 2022. The rise of food delivery and the challenges of manual management. Restaurant Management Today, 12(4), pp.45–56.

Kumar, P. and Zhao, X., 2020. GPS-based real-time tracking in last-mile logistics: Benefits and challenges. Journal of Transport and Supply Chain Management, 14(1), pp.1–12.

Lacity, M.C. and Willcocks, L.P., 2018. Robotic process automation: The next transformation lever for shared services. Journal of Information Technology Teaching Cases, 8(1), pp.1–10.

Miller, K. and Chen, P., 2020. The impact of peak-hour demand on takeaway restaurant operations. Journal of Hospitality & Tourism Research, 44(2), pp.245–260.

Oates, B.J., 2006. Researching Information Systems and Computing. Sage Publications.

Orderease, 2023a. The hidden costs of manual data entry in supply chain operations. [online] Available at: https://www.orderease.com/community/costs-of-manual-data-entry-in-supply-chain-operations [Accessed 19 Aug. 2025].

Orderease, 2023b. Labour inefficiencies in manual workflows. [online] Available at: https://www.orderease.com/community/costs-of-manual-data-entry-in-supply-chain-operations [Accessed 19 Aug. 2025].

Park, J. and Lee, K., 2019. Impact of real-time driver tracking on customer satisfaction in the food delivery sector. Journal of Service Research, 22(4), pp.456–468.

Samsara, 2023. What is route optimization? [online] Available at: https://www.samsara.com/guides/what-is-route-optimization [Accessed 19 Aug. 2025].

SmartRoutes, 2023a. Manual vs automated dispatching. [online] Available at: https://smartroutes.io/blogs/manual-vs-automated-dispatching/ [Accessed 19 Aug. 2025].

SmartRoutes, 2023b. Manual vs automated dispatching – errors in manual route planning. [online] Available at: https://smartroutes.io/blogs/manual-vs-automated-dispatching/ [Accessed 19 Aug. 2025].

SmartRoutes, 2023c. Manual vs automated dispatching – long-term efficiency benefits. [online] Available at: https://smartroutes.io/blogs/manual-vs-automated-dispatching/ [Accessed 19 Aug. 2025].

Smith, T. and Lee, R., 2021. Barriers to technology adoption in small takeaway businesses. Small Business Economics, 57(2), pp.671–687.

Smith, T., 2021. Bridging the technology gap in restaurant delivery operations. *Journal of Business and Technology Integration*, 12(1), pp.25–36.

Sommerville, I., 2016. Software Engineering. 10th ed. Pearson.

Track-POD, 2023a. Smart ways to enhance fleet dispatching. [online] Available at: https://www.track-pod.com/blog/advanced-dispatch-management-systems/ [Accessed 19 Aug. 2025].

Track-POD, 2023b. Advanced dispatch management systems and cost optimisation. [online] Available at: https://www.track-pod.com/blog/advanced-dispatch-management-systems/ [Accessed 19 Aug. 2025].

Willcocks, L., Lacity, M. and Craig, A., 2015. The IT Function: Key Issues 2015–2016. London School of Economics and Political Science.

Docker Inc., 2025. Docker: Empowering app development. [online] Available at: https://www.docker.com [Accessed 11 Jul. 2025].

Dotenv, 2025. Dotenv – Loads environment variables from .env file. [online] Available at: https://github.com/motdotla/dotenv [Accessed 11 Jul. 2025].

DrawSQL, 2025. DrawSQL: Database schema diagrams tool. [online] Available at: https://drawsql.app [Accessed 19 Aug. 2025].

Expo, 2025. Expo Documentation. [online] Available at: https://docs.expo.dev [Accessed 19 Aug. 2025].

Facebook Open Source, 2025. React Native – Create native apps for Android and iOS. [online] Available at: https://reactnative.dev [Accessed 19 Aug. 2025].

Figma Inc., 2025. Figma – Design, prototype, and gather feedback. [online] Available at: https://www.figma.com [Accessed 11 Jul. 2025].

Git, 2025. Git – Distributed version control. [online] Available at: https://git-scm.com [Accessed 11 Jul. 2025].

GitHub, 2025. GitHub – Code hosting platform for collaboration. [online] Available at: https://github.com [Accessed 11 Jul. 2025].

Google Developers, 2025. Google Maps Platform Documentation. [online] Available at: https://developers.google.com/maps/documentation [Accessed 19 Aug. 2025].

Google Vision OCR, 2025. Google Vision AI. [online] Available at: https://cloud.google.com/vision [Accessed 19 Aug. 2025].

Next.js, 2025. Next.js – The React framework. [online] Available at: https://nextjs.org [Accessed 11 Jul. 2025].

Node.js Foundation, 2025. Node.js – JavaScript runtime. [online] Available at: https://nodejs.org [Accessed 11 Jul. 2025].

Playwright, 2025. Playwright – End-to-end testing tool. [online] Available at: https://playwright.dev [Accessed 11 Jul. 2025].

PostgreSQL Global Development Group, 2025. PostgreSQL Documentation. [online] Available at: https://www.postgresql.org/docs [Accessed 19 Aug. 2025].

Postman Inc., 2025. Postman – API development environment. [online] Available at: https://www.postman.com [Accessed 11 Jul. 2025].

React.js, 2025. React – A JavaScript library for building user interfaces. [online] Available at: https://reactjs.org [Accessed 11 Jul. 2025].

Supabase, 2025. Supabase: The Open Source Firebase Alternative. [online] Available at: https://supabase.com/docs [Accessed 19 Aug. 2025].

Swagger, 2025. Swagger – API design and documentation tools. [online] Available at: https://swagger.io [Accessed 11 Jul. 2025].

Tailwind Labs, 2025. Tailwind CSS – A utility-first CSS framework. [online] Available at: https://tailwindcss.com [Accessed 11 Jul. 2025].

Twilio, 2025. Twilio SMS Authentication Documentation. [online] Available at: https://www.twilio.com/docs [Accessed 19 Aug. 2025].

Vercel Inc., 2025. Vercel Deployment Platform. [online] Available at: https://vercel.com/docs [Accessed 19 Aug. 2025].

Vitest, 2025. Vitest – A Vite-native test framework. [online] Available at: https://vitest.dev [Accessed 11 Jul. 2025].

Zustand, 2025. Zustand: A small, fast and scalable bearbones state management solution. [online] Available at: https://docs.pmnd.rs/zustand [Accessed 19 Aug. 2025].

Appendices