UNIVERSIDAD DE GRANADA

BACHELOR'S THESIS

BACHELOR'S DEGREE IN COMPUTER SCIENCE AND ENGINEERING

AIDiaCAR

Design and development of a sustainability-focused app for optimizing personal vehicle usage

Author
Juan Manuel Segura Duarte

Thesis supervisor
Rosana Montes Soldado

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, September 2025

 Juan Manuel Segura Duarte, 2025

Design and development of a sustainability-focused app for optimizing personal vehicle usage

Juan Manuel Segura Duarte

Keywords: Sustainable Mobility, Vehicle Fleet Management, Collaborative Systems, Persuasive Technology, Green Computing, Vehicle Recommendation, Proof-of-Concept, Software Architecture, Mobile Application.

Abstract:

The widespread ownership of multiple vehicles within a single household represents a significant, yet largely unaddressed, source of economic and environmental inefficiency. The ad-hoc management of these personal "micro-fleets" leads to coordination friction, neglected maintenance, and suboptimal vehicle selection, contributing to excess fuel consumption and greenhouse gas emissions. This thesis addresses this problem by proposing, designing, and implementing *AlDiaCAR*[1], a novel, collaborative mobile application conceived as a "Digital Garage" for the modern family.

The system is architected upon a four-pillar framework designed to holistically tackle the core pain points: Coordination, Vehicle Recommendation, Automated Maintenance, and Analytics. The intellectual centerpiece of this research is the "Proactive Co-Pilot," a multi-factor recommendation engine that acts as a persuasive technology, nudging users toward more sustainable mobility choices by analyzing journey constraints such as Low-Emission Zone (ZBE) regulations.

This work provides a comprehensive architectural blueprint for a full-stack, cross-platform solution, utilizing a technology stack of React Native, Node.js, and MongoDB. The viability of the system's most innovative component was successfully validated through the implementation of a proof-of-concept prototype. This functional artifact demonstrates the core ZBE-aware recommendation logic, confirming the technical feasibility of the proposed solution and establishing a robust foundation for future development towards a feature-complete system. The project's primary contribution is, therefore, a validated design and a functional prototype for a system that synthesizes personal fleet management, collaborative tools, and sustainable decision-making into a single, cohesive platform.

[1]The complete source code for this project is available at: https://github.com/jseg380/Trabajo-Fin-Grado

Diseño y desarrollo de una aplicación para la optimización del uso del vehículo privado con enfoque en sostenibilidad

Juan Manuel Segura Duarte

Palabras clave: Movilidad Sostenible, Gestión de Flotas de Vehículos, Sistemas Colaborativos, Tecnología Persuasiva, Green IT, Recomendación de Vehículos, Prueba de Concepto, Arquitectura de Software, Aplicación Móvil.

Resumen:

La propiedad generalizada de múltiples vehículos dentro de un mismo hogar representa una fuente significativa, y en gran medida desatendida, de ineficiencia económica y medioambiental. La gestión improvisada de estas "microflotas" personales conduce a fricciones de coordinación, un mantenimiento deficiente y una selección de vehículos subóptima, contribuyendo a un exceso de consumo de combustible y a la emisión de gases de efecto invernadero. Esta tesis aborda este problema mediante la propuesta, el diseño y la implementación de *AIDiaCAR*[2], una novedosa aplicación móvil colaborativa concebida como un "Garaje Digital" inteligente para la familia moderna.

El sistema se articula sobre una arquitectura de cuatro pilares, diseñada para abordar de manera integral los principales puntos débiles identificados: Coordinación, Recomendación Inteligente, Mantenimiento Automatizado y Analíticas. El eje intelectual de esta investigación es el "Copiloto Proactivo", un motor de recomendación multifactorial que actúa como una tecnología persuasiva, incentivando a los usuarios a tomar decisiones de movilidad más sostenibles mediante el análisis de restricciones del trayecto, como las regulaciones de las Zonas de Bajas Emisiones (ZBE).

Este trabajo aporta un plan arquitectónico completo para una solución *full-stack* y multiplataforma, utilizando un conjunto de tecnologías compuesto por React Native, Node.js y MongoDB. La viabilidad del componente más innovador del sistema fue validada con éxito a través de la implementación de una prueba de concepto. Este prototipo funcional demuestra la lógica central de recomendación consciente de las ZBE, confirmando la factibilidad técnica de la solución propuesta y estableciendo una base robusta para el futuro desarrollo hacia un sistema con todas las funcionalidades. La contribución principal del proyecto es, por tanto, un diseño validado y un prototipo funcional para un sistema que sintetiza la gestión de flotas personales, las herramientas colaborativas y la toma de decisiones sostenibles en una única plataforma cohesionada.

[2]El código fuente completo de este proyecto está disponible en: https://github.com/jseg380/Trabajo-Fin-Grado

Acknowledgements

I would like to take this opportunity to express my heartfelt gratitude to all those who have supported me throughout my journey in completing this thesis.

First and foremost, I would like to thank my family for their unwavering love and encouragement. Your belief in my abilities has been a constant source of motivation, and I am forever grateful for your sacrifices and support.

I am also grateful to my friends and peers who have stood by me during this challenging process. Your camaraderie, late-night study sessions, and shared laughter have made this journey not only bearable but also enjoyable. Thank you for being my sounding board and for always believing in me.

Additionally, I would like to acknowledge the faculty and staff of the University of Granada for providing a nurturing academic environment. Your dedication to teaching and mentorship has profoundly impacted my educational experience.

This thesis is not just a reflection of my efforts but a testament to the collective support of all those around me. Thank you for being part of this journey.

Contents

List of Figures

List of Tables

Glossary

A

Autónomo An autónomo is the Spanish legal and fiscal term for a self-employed individual or sole trader, roughly equivalent to a "freelancer" or "independent contractor" in anglophone contexts. This is a highly prevalent form of employment in Spain, encompassing a vast range of professions from skilled tradespeople like plumbers and electricians to professional service providers such as consultants, designers, and lawyers. Unlike employees with fixed 9-to-5 schedules, autónomos typically manage their own working hours and client engagements, resulting in a highly variable and often unpredictable daily schedule. This lack of a fixed routine is a critical factor in the user persona for this thesis, as it complicates household resource planning—particularly the availability and use of shared vehicles—necessitating a dynamic, real-time coordination tool rather than a static, schedule-based system.. 4

D

Distintivo Ambiental The Distintivo Ambiental is an official vehicle classification system implemented by Spain's Dirección General de Tráfico (DGT). It categorizes vehicles based on their pollutant emission levels, assigning a corresponding colored sticker that must be displayed on the vehicle. This system is the primary mechanism used by municipalities to regulate traffic in Low Emission Zones (ZBEs). The badges are not a measure of CO_2 emissions but of local air pollutants like NOx and particulate matter. The main classifications for passenger cars are:

- **0 Emisiones (Blue Badge)**: Reserved for the cleanest vehicles, primarily Battery Electric Vehicles (BEVs), extended-range electric vehicles (REEVs), plug-in hybrids (PHEVs) with a range of over 40km, and fuel cell vehicles. They enjoy the most privileges, including unrestricted access to all ZBEs.
- **ECO (Green and Blue Badge)**: This category includes standard hybrid vehicles (HEVs), plug-in hybrids with a range under 40km, and vehicles powered by compressed natural gas (CNG) or liquefied petroleum gas (LPG). They

have significant privileges, though slightly fewer than '0 Emisiones'.
• **C (Green Badge)**: For gasoline vehicles registered from 2006 onwards (Euro 4, 5, 6 standards) and diesel vehicles registered from 2014 onwards (Euro 6). This is a very common category for modern internal combustion engine vehicles.
• **B (Yellow Badge)**: For gasoline vehicles registered from 2001 to 2005 (Euro 3) and diesel vehicles from 2006 to 2013 (Euro 5). These vehicles face the most significant access restrictions in ZBEs.
• **Sin Distintivo (No Badge)**: The oldest and most polluting vehicles, which are typically barred from entering any ZBE. . 4, 15

Z

Zona de Bajas Emisiones (ZBE) A *Zona de Bajas Emisiones (ZBE)* is a geographically defined urban area where access for certain polluting vehicles is restricted to improve air quality. Mandated by Spain's Climate Change and Energy Transition Law, all municipalities with over 50,000 inhabitants are required to establish ZBEs. The implementation and severity of these restrictions are at the discretion of the local city council, leading to a fragmented and often confusing regulatory landscape for drivers.
Access restrictions are enforced based on the vehicle's Distintivo Ambiental. A ZBE may, for example:
• **0 Emisiones (Blue Badge)**: Reserved for the cleanest vehicles, primarily Battery Electric Vehicles (BEVs), extended-range electric vehicles (REEVs), plug-in hybrids (PHEVs) with a range of over 40km, and fuel cell vehicles. They enjoy the most privileges, including unrestricted access to all ZBEs.
• **ECO (Green and Blue Badge)**: This category includes standard hybrid vehicles (HEVs), plug-in hybrids with a range under 40km, and vehicles powered by compressed natural gas (CNG) or liquefied petroleum gas (LPG). They have significant privileges, though slightly fewer than '0 Emisiones'.
• **C (Green Badge)**: For gasoline vehicles registered from 2006 onwards (Euro 4, 5, 6 standards) and diesel vehicles registered from 2014 onwards (Euro 6). This is a very common category for modern internal combustion engine vehicles.
• **B (Yellow Badge)**: For gasoline vehicles registered from 2001 to 2005 (Euro 3) and diesel vehicles from 2006 to 2013 (Euro 5). These vehicles face the most significant access restrictions in ZBEs.
• **Sin Distintivo (No Badge)**: The oldest and most polluting vehicles, which are typically barred from entering any ZBE. . 4

1 Introduction

1.1 Context and relevance: global emissions and personal vehicle management

The escalating climate crisis, a defining global challenge of the 21st century, necessitates profound and immediate transformations across all sectors of society. This urgency is formally encapsulated in the United Nations' Sustainable Development Goal 13 (SDG 13), 'Climate Action,' which calls for immediate and concerted efforts to combat climate change and its impacts. However, progress towards the 2030 targets remains alarmingly insufficient, with recent analyses from intergovernmental bodies indicating a significant gap between current national commitments and the required emissions reduction trajectory to limit global warming. It is within this context of necessary, accelerated action that the transportation sector emerges as a particularly critical domain for intervention due to its substantial contribution to anthropogenic greenhouse gas (GHG) emissions. According to a comprehensive analysis of global climate data, the transport sector was directly responsible for approximately 16.2% of all greenhouse gas emissions recorded in 2016 (see Figure 1.1).

A more granular examination of this figure reveals that road transport—encompassing a wide array of vehicles such as cars, motorcycles, buses, and trucks—constituted the single largest contributor, accounting for a significant 11.9% of total global emissions. Most notably, a striking 60% of these road transport emissions can be attributed directly to passenger travel, a figure dominated by the widespread use of private vehicles [1]. This disaggregation underscores the disproportionate environmental impact of individual mobility choices.

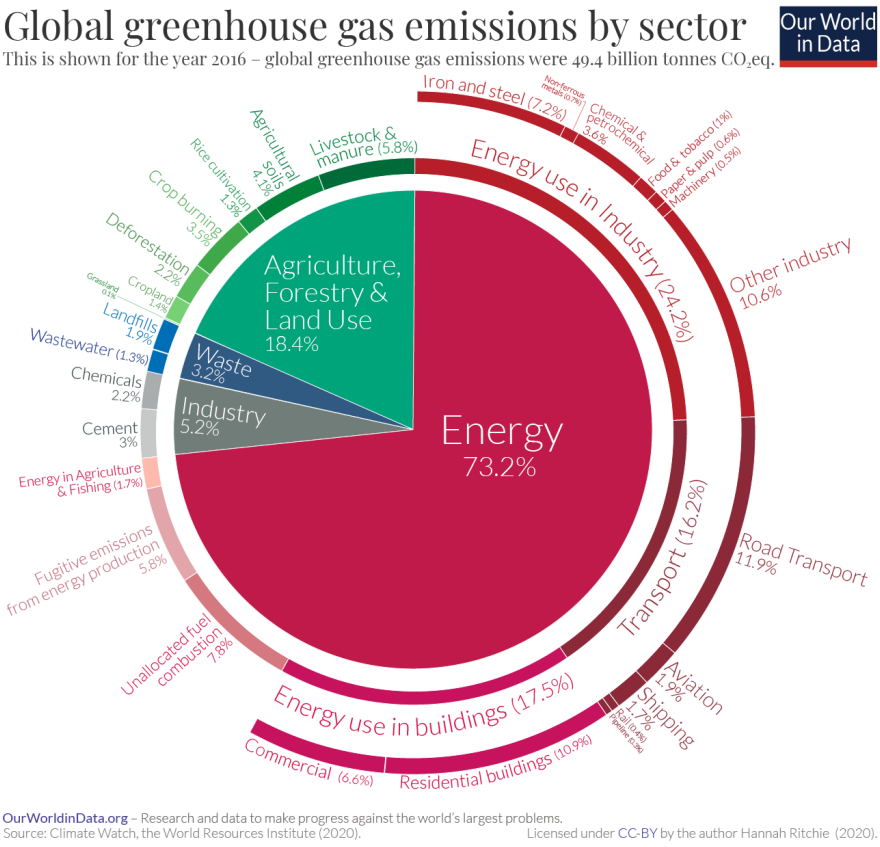


Figure 1.1: Breakdown of global greenhouse gas emissions by sector. Source: Our World in Data [1].

This pronounced reliance on private automobiles is a hallmark of developed economies. In the European Union, for example, statistical trends indicate a persistent pattern of increasing private vehicle ownership, with the number of passenger cars per thousand inhabitants reaching a new peak of 560 in the year 2022 [2]. This saturation of private vehicles not only places immense strain on public infrastructure but also complicates efforts to meet ambitious regional and national emissions reduction targets.

This thesis posits that the millions of multi-car households globally constitute a vast, decentralized, and inefficiently managed network of small-scale vehicle fleets. It is within this context of the fragmented, privately-managed household "micro-fleet" that this research identifies a unique and critically underserved technological niche. The cumulative impact of suboptimal vehicle selection, uncoordinated maintenance, and inefficient usage patterns within these micro-fleets represents a substantial, yet largely unaddressed, opportunity for environmental and economic improvement.

Consequently, this thesis directly confronts these widespread inefficiencies by championing a framework of holistic sustainability for private vehicle utilization. This comprehensive concept necessarily extends beyond the singular act of purchasing an electric vehicle. True sustainability in this context must encompass a multi-faceted approach, including the disciplined and diligent maintenance of all vehicles to ensure they operate at their peak design efficiency. It is well-documented that poorly maintained vehicles, regardless of their powertrain, can experience a significant degradation in performance, leading to increased fuel consumption and a corresponding rise in harmful pollutant emissions [3]. Furthermore, this holistic view involves fostering more conscious and data-informed vehicle selection for each specific journey, as well as promoting the consistent adoption of eco-driving habits among all users within a household.

Furthermore, the theoretical underpinnings of this project are firmly grounded in the principles of Green Computing, with a specific focus on the paradigm known as "ICT for Sustainability" [1]. This particular pillar of Green IT is oriented towards the strategic application of information and communication technologies to monitor, model, and ultimately improve the efficiency of real-world physical processes. By architecting intelligent software solutions, it becomes possible to empower end-users to make more informed and environmentally conscious decisions in their daily lives. This project hypothesizes that a well-designed mobile application can function as a potent form of persuasive technology, a term defined by B.J. Fogg as technology that is intentionally designed to change a person's attitudes or behaviors without resorting to coercion or deception [4]. By providing timely feedback, relevant information, and facilitating collaborative planning, such a system can effectively nudge users towards more sustainable mobility patterns, thereby transforming abstract environmental goals into concrete, actionable behaviors.

1.2 The problem statement: the management gap for the modern vehicle owner

1.2.1 User *persona*: The Martínez household

To precisely articulate and contextualize the real-world challenges this thesis aims to resolve, we employ the methodological tool of a user *persona* [2]. We introduce the Martínez family—a carefully constructed archetype of a modern, middle-class household comprising four individuals. This family unit consists of two parents and their two adult children, all residing in the same home. Their professional and academic lives

[1] ICT4S 2026: International Conference on ICT for Sustainability
[2] What Are User Personas?

reflect contemporary societal structures: one parent is a self-employed professional (autónomo) whose work demands a highly variable and often unpredictable schedule. The other parent maintains a traditional 9-to-5 office job with a regular commute. One of the adult children works primarily from home in a remote capacity, while the other is a full-time university student with their own distinct travel requirements.

Driven by both economic prudence and a genuine concern for their environmental footprint, the family has made a conscious decision to manage a shared pool of three vehicles, correctly identifying the acquisition of a fourth car as an unnecessary financial and ecological burden. This shared 'household fleet' is deliberately diverse, designed to mirror common vehicle ownership patterns observed in contemporary Spain and other parts of Europe:

- **Vehicle 1: The City runner.** This vehicle is a 2014 diesel-powered hatchback. Its primary attributes are its compact size, superior fuel efficiency in urban environments, and ease of parking. However, its age and engine type mean it has been assigned a Distintivo Ambiental B, an environmental classification that legally restricts its access to designated Low Emission Zones (Zonas de Bajas Emisiones - ZBE) now prevalent in major city centers. This regulatory constraint creates a significant operational limitation.

- **Vehicle 2: The All-rounder.** A 2017 gasoline sedan, this car holds a more favorable Distintivo 'C'. It represents a compromise, offering a satisfactory balance of passenger comfort, luggage capacity, and acceptable fuel efficiency. Crucially, its environmental classification grants it full access to many of the ZBEs that restrict the City Runner, making it a more versatile asset for a wider range of journeys.

- **Vehicle 3: The Workhorse.** The newest vehicle in the fleet is a 2021 diesel SUV, which also possesses a Distintivo 'C'. This vehicle is reserved for specific use cases such as long-distance family trips, transporting bulky items, or when maximum passenger comfort is a priority. Despite its utility and modern features, it inherently has the highest fuel consumption, greatest carbon emissions, and most expensive running costs of the three.

The confluence of the family members' unpredictable, often overlapping schedules with the distinct operational and regulatory characteristics of their diverse vehicle set generates a complex, daily logistical puzzle. The management of this micro-fleet thus transcends simple maintenance scheduling and becomes a significant source of recurring inefficiency and household friction.

1.2.2 A day in the Martínez's household: The cascading inefficiencies

To illustrate this dynamic, let us consider a typical Wednesday morning scenario. The *autónomo* parent must attend a critical client meeting located deep within the city center's primary ZBE, a destination that immediately renders the 'B' rated City Runner unusable for this specific trip. Concurrently, the university student needs to travel to campus for a lecture and, given the choice, would naturally prefer the small, fuel-efficient 'B' car for its low running cost and ease of parking near the university. Meanwhile, the parent with the 9-to-5 job has already departed for their office, having taken the 'C' rated sedan, the 'All-Rounder,' as is their daily routine.

This seemingly mundane situation triggers a cascade of logistical queries that, in the absence of a centralized management system, must be resolved through a flurry of ad-hoc, inefficient communication and physical verification. Questions arise instantly: Is the SUV's key readily available, or did someone misplace it? Is the vehicle's mandatory technical inspection (ITV) due in the coming week, making a long journey inadvisable and potentially illegal? Does the SUV have sufficient fuel for the required trip, or will it necessitate an unplanned stop?

Lacking a unified, digital source of truth for the status of their shared assets, the parent is compelled to make a decision based on incomplete information. In this instance, they take the high-emission, high-cost SUV for what is essentially a short urban errand. This represents a demonstrably suboptimal choice, a direct consequence of a systemic coordination failure. This single, commonplace event serves to crystallize the core pain points that this thesis seeks to address.

1.2.3 Core pain points

The daily challenges faced by the Martínez household can be deconstructed into four distinct, yet interconnected, core pain points. These issues represent significant barriers to the efficient, economical, and environmentally responsible management of a shared household vehicle fleet.

The first and most immediate issue is **coordination friction**. The current management method relies entirely on synchronous, manual communication channels such as text messages, phone calls, and face-to-face conversations. This constant need to query the status and location of vehicles and their keys introduces a significant cognitive overhead into the daily lives of the family members. It consumes valuable time, generates unnecessary stress, and frequently leads to minor conflicts and misunderstandings, creating a persistent source of domestic friction.

Secondly, the family suffers from **shared maintenance blindness**. In a multi-driver, multi-vehicle environment, responsibility for vehicle upkeep becomes dangerously diffused. No single individual possesses a complete, accurate, or real-time overview of the comprehensive maintenance status—including mandatory inspections (ITV), scheduled oil changes, tire wear and pressure, and other critical service intervals—across all three vehicles. This lack of centralized oversight inevitably leads to critical and potentially hazardous oversights, increasing the risk of mechanical failures, legal infractions, and avoidable repair costs.

The third pain point is **inefficient, constraint-unaware selection**. In the heat of the moment, vehicle choices are predominantly dictated by immediate physical availability rather than a rational, holistic assessment of all relevant constraints. An optimal decision would weigh multiple factors simultaneously: ZBE access restrictions, real-time fuel costs, the specific emissions profile of each car, upcoming maintenance deadlines, and the nature of the journey itself. The absence of a tool to facilitate this multi-criteria analysis ensures that the family consistently makes suboptimal choices, leading to higher expenses and a greater environmental impact than necessary.

Finally, the household operates with a complete **lack of shared visibility** into their collective mobility behavior. There exists no mechanism for the family to aggregately track and visualize their total transportation expenditures, fuel consumption, or cumulative environmental impact over time. This information vacuum prevents the formation of effective feedback loops, which are essential for behavioral change. Without access to clear, quantifiable data on their collective habits, the family is unable to make informed group decisions, set meaningful goals for improvement, or objectively measure their progress towards becoming more sustainable and economically efficient.

2 | Background and related work

To adequately contextualize the novel contribution of this thesis, this chapter undertakes a comprehensive review of the existing landscape of software solutions, theoretical frameworks, and academic literature pertinent to vehicle management and sustainable mobility. The primary objective of this chapter is to meticulously map the state of the art, thereby establishing a clear and defensible rationale for the development of the *AIDiaCAR* system. We systematically examine the current technological offerings across three distinct and highly segmented categories of applications: enterprise-grade commercial fleet management systems, consumer-focused personal car maintenance trackers, and specialized eco-routing navigation tools. This analysis not only details the functionalities of existing systems but also critically evaluates their inherent limitations when applied to the specific problem domain of a multi-vehicle, multi-driver household.

Furthermore, this review delves into the theoretical foundations that provide the intellectual scaffolding for this project's design philosophy. Specifically, we explore the principles of gamification as a mechanism for behavior change and the broader paradigm of Green Information Technology (Green IT), particularly the concept of "Green through IT"[1] or Sustainable Human-Computer Interaction (HCI). By grounding our practical design choices in established academic theory, we aim to enhance the project's robustness and potential for real-world impact. The chapter will culminate in a detailed gap analysis, synthesizing the findings from our market and literature review to precisely identify the unoccupied technological and conceptual niche that the *AIDiaCAR* project is designed to fill. This final section serves as the definitive justification for the thesis, articulating its unique value proposition and contribution to the field.

[1] ICT4S 2026: International Conference on ICT for Sustainability

2.1 State of the art in vehicle management applications

The contemporary market for vehicle-related software is both mature and extensive, yet it remains conspicuously fragmented. Existing solutions are typically characterized by a high degree of specialization, with software platforms meticulously engineered to target either large-scale commercial enterprises or individual users. This segmentation has resulted in a landscape where tools are powerful within their intended vertical but fundamentally ill-equipped to address the hybrid challenges faced by a modern household managing a small, private fleet of vehicles. The following subsections dissect each major category, highlighting their capabilities and, more importantly, their shortcomings in relation to the problem statement of this thesis.

2.1.1 Commercial fleet management systems

The Business-to-Business (B2B) sector of fleet management is dominated by powerful, data-intensive software-as-a-service (SaaS) platforms such as *Fleetio*[2] and *Samsara*[3]. These systems are engineered to provide corporations with granular control and comprehensive oversight over large, complex, and geographically dispersed vehicle fleets. Their core value proposition lies in optimizing operational efficiency, ensuring regulatory compliance, and minimizing costs at scale.

The feature set of these platforms is consequently vast and sophisticated. Key functionalities typically include:

- **Real-time GPS tracking and geofencing:** Constant monitoring of vehicle location, speed, and heading, coupled with the ability to define virtual perimeters (geofences) that trigger alerts when crossed.
- **Advanced telematics:** Integration with onboard vehicle hardware (often via the OBD-II port) to capture a rich stream of diagnostic data, including engine RPM, fuel consumption, fault codes (Diagnostic Trouble Codes, Diagnostic Trouble Codes (DTCs)), and harsh braking or acceleration events.
- **Comprehensive maintenance scheduling:** Automated workflows for preventative maintenance based on mileage, engine hours, or calendar intervals, including work order generation and service history logging.
- **Fuel management:** Detailed tracking of fuel purchases, consumption analysis, and integration with fuel card providers to detect fraud or inefficiency.

[2] Fleetio
[3] Samsara

- **Driver behavior monitoring:** Scoring and reporting on driver performance to identify unsafe or inefficient habits, often used for training and safety programs.
- **Regulatory compliance:** Features designed to automate compliance with regulations such as the Electronic Logging Device (ELD) mandate for hours of service and International Fuel Tax Agreement (IFTA) reporting.

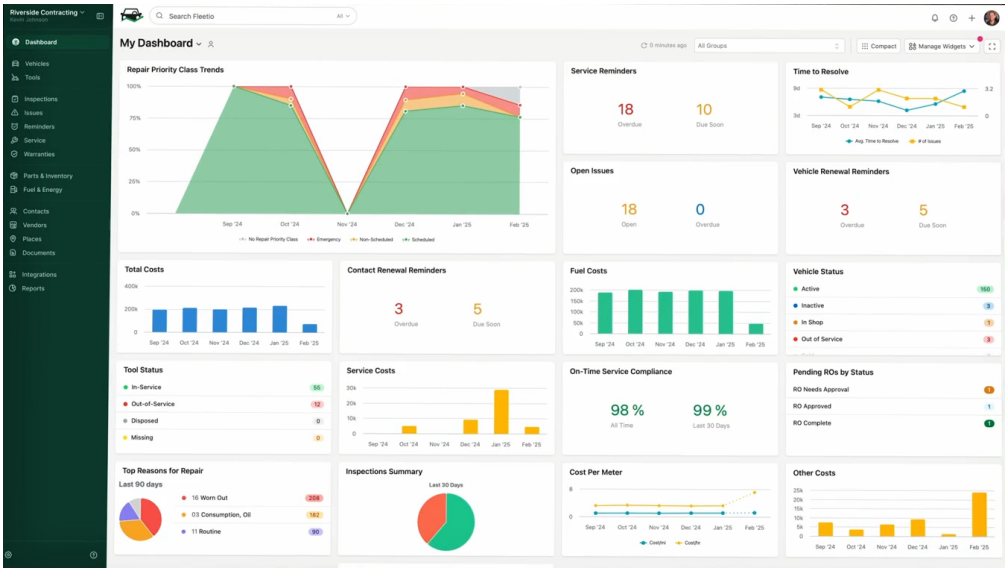


Figure 2.1: Illustrative example of a typical commercial fleet management dashboard (Fleetio), showcasing the complexity and data-density designed for professional logistics managers. Source: Fleetio.

While these systems represent the pinnacle of vehicle management technology, their direct application to the context of a private household, such as the Martínez family *persona*, is fundamentally unviable for a confluence of reasons:

- **Prohibitive complexity and cost:** These are enterprise-grade platforms. Their pricing models are typically structured on a per-vehicle, per-month basis, often with substantial setup fees. For a family managing three cars, the cost would be unjustifiable. Moreover, the feature set is overwhelmingly complex, presenting a steep learning curve and a significant cognitive burden for non-technical users.

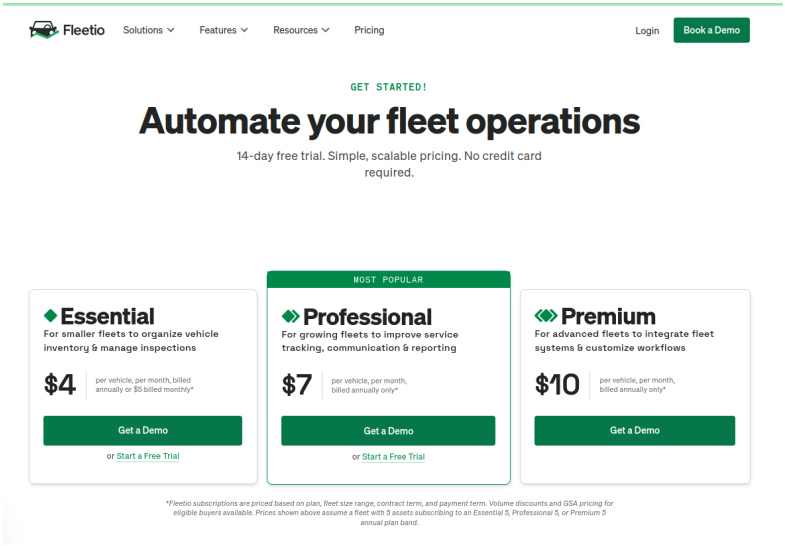


Figure 2.2: Cost of Fleetio starts at 5$ per vehicle per month if billed monthly, 4$ if billed annually. Totalling at about 96$ for a 2 vehicle fleet in the best of the cases. Source: Fleetio Pricing.

- **Misaligned user experience (UX):** The user interface and overall experience are meticulously crafted for a professional fleet manager or dispatcher whose primary job function is to analyze data and manage logistics. The UX prioritizes data density and administrative control over the simplicity, accessibility, and collaborative ease-of-use required for a family context.

- **Fundamentally different focus:** The teleological purpose of a commercial fleet system is the optimization of business assets to maximize profit and minimize risk. Their design philosophy is rooted in surveillance and control. In contrast, the goal of a personal fleet management tool should be to facilitate collaboration, reduce domestic friction, and promote positive, sustainable habits among trusted users. The underlying motivations are fundamentally different.

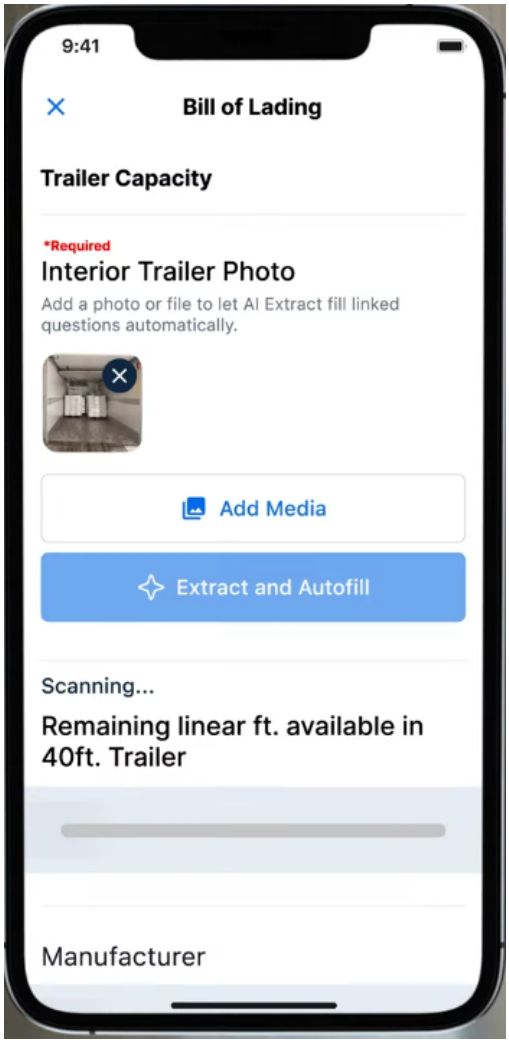


Figure 2.3: Example of a mobile interface for a commercial fleet application (Samsara). The focus remains on asset tracking and driver logs, which is misaligned with the needs of a family. Source: Samsara.

- **Data privacy and social dynamics:** The level of granular tracking (e.g., real-time location, driving habits) that is standard in a corporate setting is often socially unacceptable and raises significant privacy concerns within a family unit. Implementing such a system could introduce an uncomfortable dynamic of surveillance rather than cooperation.

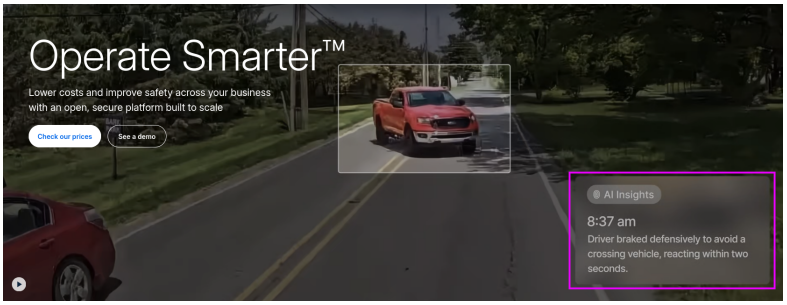


Figure 2.4: Every action of the driver is monitored and analysed. In this demo video from Samsara we can see that the driver reaction was stored and analysed. Source: Samsara.

2.1.2 Personal car maintenance applications

On the other end of the spectrum are applications developed for the individual car owner. This category includes popular apps like *Drivvo*[4], *Fuelly*[5], and *aCar*[6]. These tools are designed to serve as digital logbooks, empowering a single user to meticulously track the health, expenses, and history of their personal vehicle.

Their core competency lies in the detailed logging of specific data points. Users can manually record every fuel-up, calculate fuel economy (MPG or L/100km), log all maintenance and repair expenses, and set personalized reminders for crucial service intervals, such as oil changes, tire rotations, or mandatory technical inspections like the Spanish ITV. Many of these applications also offer basic reporting features, allowing users to visualize their spending and fuel consumption over time.

[4] Drivvo
[5] Fuelly
[6] aCar: Play Store

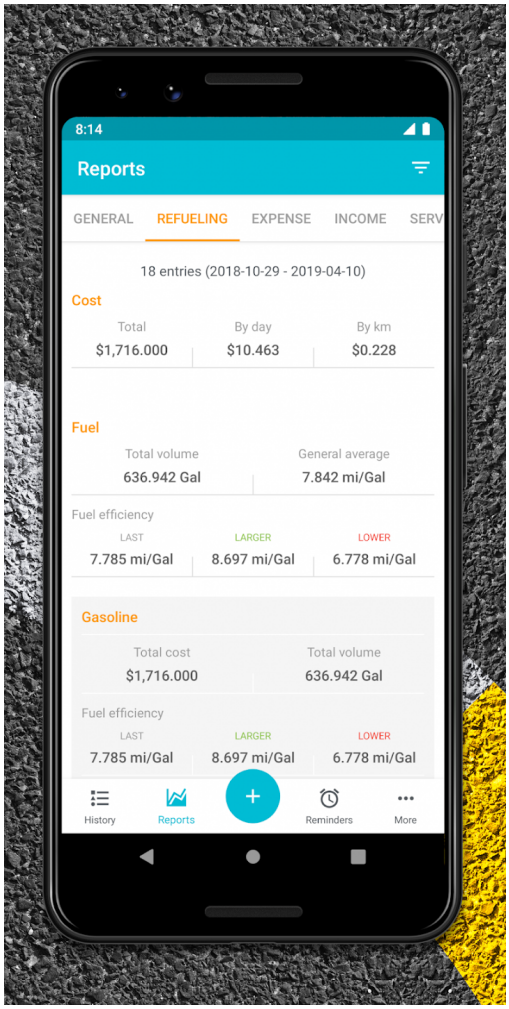


Figure 2.5: Typical user interface for a personal car maintenance app like Drivvo, centered on logging fuel, services, and expenses for a single vehicle. Source: Drivvo.

However, despite their utility for the dedicated single-car enthusiast, their effectiveness rapidly diminishes when applied to the multi-vehicle, multi-driver scenario that this project specifically addresses. Their primary drawbacks include:

- **Inherent single-user, single-vehicle focus:** These applications are architected around the mental model of one person managing one primary vehicle. While some may allow the user to add multiple vehicle profiles, the interface is rarely optimized for comparing them or managing them as a collective "fleet". They lack a centralized dashboard that provides an at-a-glance overview of the entire household's automotive assets.

- **Absence of collaborative features:** Crucially, these apps are not designed for shared access or collaborative management. They function as data silos on a single user's device. There is no mechanism for multiple family members to view and update the status of a shared vehicle in real-time. This completely fails to address the core problem of "shared maintenance blindness" identified in Chapter 1.

- **Superficial approach to sustainability:** While fuel economy calculation is a standard feature, its purpose is almost exclusively financial—to help the user track their fuel expenses. These applications do not leverage this data to provide intelligent, forward-looking recommendations. They cannot, for example, advise a user on which of their three cars would be the most ecologically and economically sound choice for an upcoming journey based on its specific parameters.

- **Passive and utilitarian engagement model:** The vast majority of these apps are functional utilities that require disciplined, manual data entry. They are passive tools that depend entirely on the user's intrinsic motivation. They do not incorporate modern engagement techniques, such as gamification, to actively encourage positive behaviors like performing timely maintenance or adopting more efficient driving habits.

2.1.3 Eco-routing and navigation tools

A third category of relevant software has emerged within mainstream navigation applications. In response to growing environmental awareness, technology giants have begun to integrate sustainability-focused features into their platforms. The most prominent example is *Google Maps*, which introduced an "eco-friendly routing" option (see Figure 2.6) . This feature's algorithm analyzes various factors, including road incline, traffic congestion, and the number of stops, to suggest a route that is optimized for the lowest possible fuel or energy consumption, often presenting it alongside the fastest route.

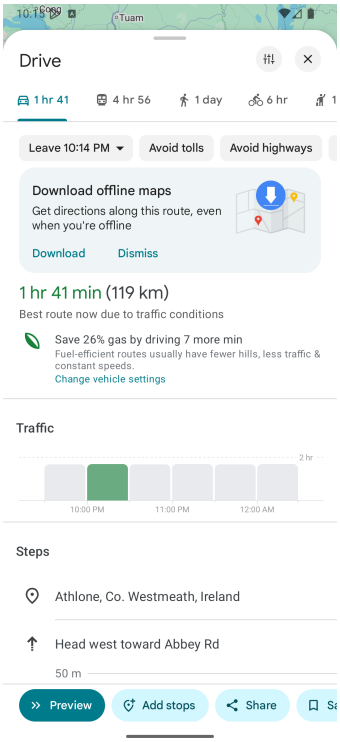


Figure 2.6: An illustration of the eco-friendly routing feature in Google Maps. Source: Google Maps.

While this represents a positive step towards promoting sustainable mobility on a massive scale, the limitation of these tools lies in their profound lack of integration with the user's specific context and available resources. They operate as isolated, single-purpose functions and are consequently incapable of answering the more complex, multi-faceted questions that are central to the problem this thesis addresses. The core deficiencies are:

- **Vehicle agnosticism:** The eco-routing algorithm operates on a generic vehicle model or a very basic user selection (e.g., gasoline, diesel, electric). It is entirely unaware of the specific set of vehicles a user actually owns, their individual efficiencies, their real-time maintenance status, or their regulatory constraints (like a Distintivo Ambiental).

- **Inability to recommend a vehicle:** The system can optimize the route for a given, abstract car, but it fundamentally cannot help the user choose the optimal car for that route from their personal fleet. It cannot advise the Martínez family

on whether to take the diesel hatchback or the gasoline sedan for a specific trip into the city.

- **Disconnection from broader management goals:** The feature is transactional and ephemeral. It provides a recommendation for a single journey and is completely disconnected from the user's long-term vehicle management goals, such as managing upcoming maintenance, or tracking a cumulative household carbon footprint.

The decision-making process for a multi-vehicle household is a complex tree of interdependent variables, which current eco-routing tools are not equipped to navigate. The following diagram illustrates a simplified version of this logic.

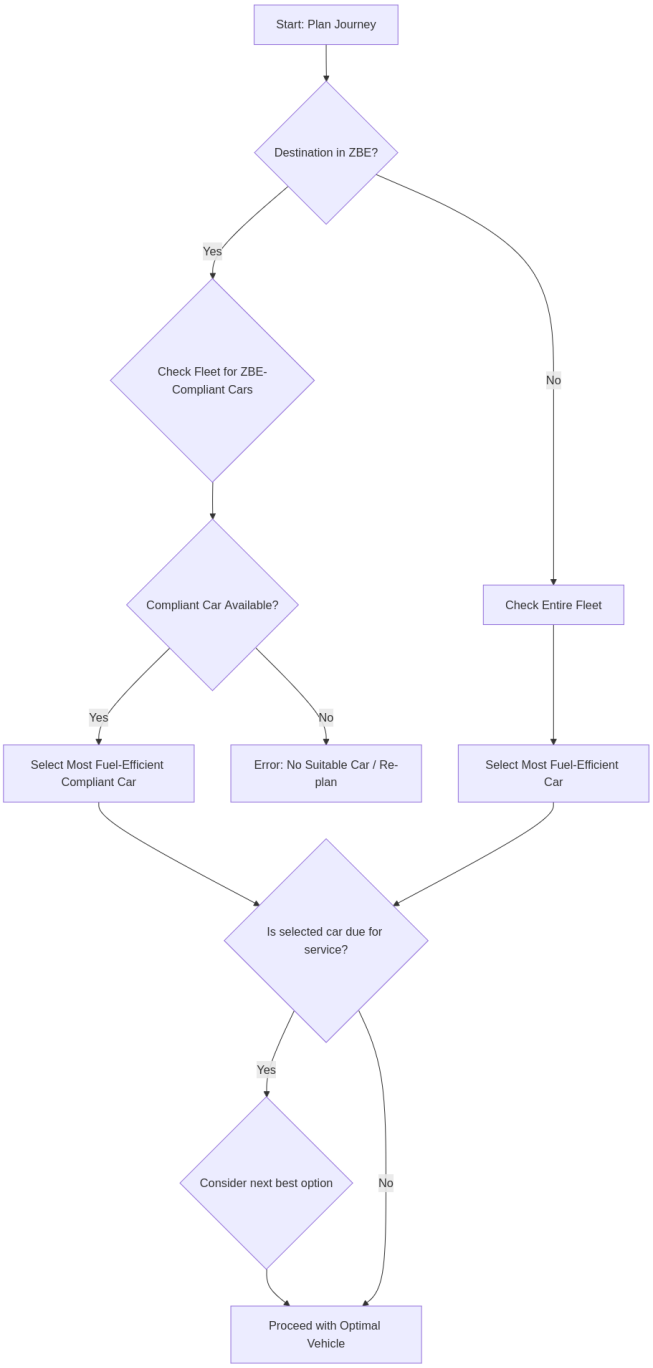


Figure 2.7: A simplified decision flow for selecting the optimal vehicle for a given trip, illustrating the multi-factor considerations that current eco-routing tools cannot address. Source: Author's own work.

2.2 Theoretical foundations

The design and implementation of the *AIDiaCAR* system are not based merely on functional requirements but are deeply informed by established theoretical principles from the fields of human-computer interaction (HCI) and sustainable computing. By leveraging these academic frameworks, the project aims to create a solution that is not only useful but also engaging and genuinely effective at influencing user behavior.

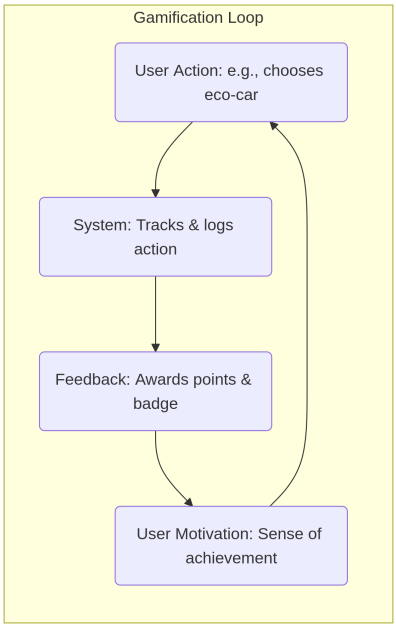
2.2.1 Gamification for behavior change

Gamification is a concept that has gained significant traction in HCI research and practice over the past decade. It is formally defined as "the use of game design elements in non-game contexts" [5]. The core premise is that the motivational techniques that make games so compelling—such as points, badges, leaderboards, and progress bars—can be extracted and applied to real-world applications to increase user engagement and encourage specific, desirable behaviors.

In the context of *AIDiaCAR*, gamification is not a superficial addition but a core mechanic designed to foster long-term habit formation. The application aims to provide positive reinforcement for actions that align with the goals of sustainability and responsible vehicle ownership. For example:

- **Achievements and badges:** Users can earn badges for achievements like "Eco-Warrior" (consistently choosing the lowest-emission vehicle), "Maintenance Master" (logging all service on time), or "Planner Pro" (coordinating vehicle usage ahead of time).
- **Progress statistics:** The application will provide clear, visual feedback on the family's collective progress, showing trends in fuel savings, CO_2 reduction, and money saved. This makes the positive impact of their choices tangible and rewarding.

This approach is supported by a large body of empirical research. A comprehensive meta-analysis by Hamari et al. [6] concluded that gamification is effective in a variety of domains, with significant positive outcomes for user engagement and behavior change, provided it is thoughtfully implemented and aligned with user motivations. The following diagram illustrates the intended motivational loop within *AIDiaCAR*.



2.2.2 Green IT and sustainable HCI

This project is fundamentally an exercise in Green Information Technology (Green IT), a field that explores the intersection of computing and environmental sustainability. Green IT is often bifurcated into two distinct sub-domains:

1. **Green in IT:** This focuses on reducing the direct environmental impact of the computing industry itself, through measures like creating energy-efficient data centers, designing recyclable hardware, and reducing e-waste.

2. **Green through IT (or ICT for Sustainability):** This focuses on leveraging information and communication technologies (ICT) as a tool to effect positive environmental change in other domains of society. This project is a clear exemplar of this second category.

AIDiaCAR uses software as a persuasive technology to influence user behavior in the physical world—specifically, their transportation choices—leading to tangible environmental benefits such as reduced fuel consumption and lower greenhouse gas emissions. This approach directly aligns with research by scholars like Berkhout and Hertin, who analyzed the complex role of digital technologies in mediating environmental impacts. They argue that while technology can help to "de-materialise" economic activity (e.g., by replacing physical travel with teleconferencing), it can also lead to "re-materialise" effects or rebounds, where efficiency gains are offset by increased consumption [7]. A

well-designed system like *AIDiaCAR* must be conscious of these potential rebounds and focus on promoting a net reduction in environmental impact, not merely more efficient consumption. This project, therefore, contributes to the field of Sustainable HCI by providing a practical case study in designing digital interventions for real-world ecological benefit.

2.3 Gap analysis and niche identification

The comprehensive review of the state of the art in both commercial and personal vehicle management software, as well as the underlying theoretical frameworks, reveals a distinct and underserved gap in the existing technological landscape. While highly specialized tools exist for corporate fleets, individual car enthusiasts, and generic navigation, no current solution holistically integrates their respective functionalities into a single, cohesive platform designed specifically for the collaborative context of a private, multi-vehicle household. This synthesis of capabilities represents the core contribution and unique value proposition of the *AIDiaCAR* project, as illustrated conceptually in Figure 2.8.

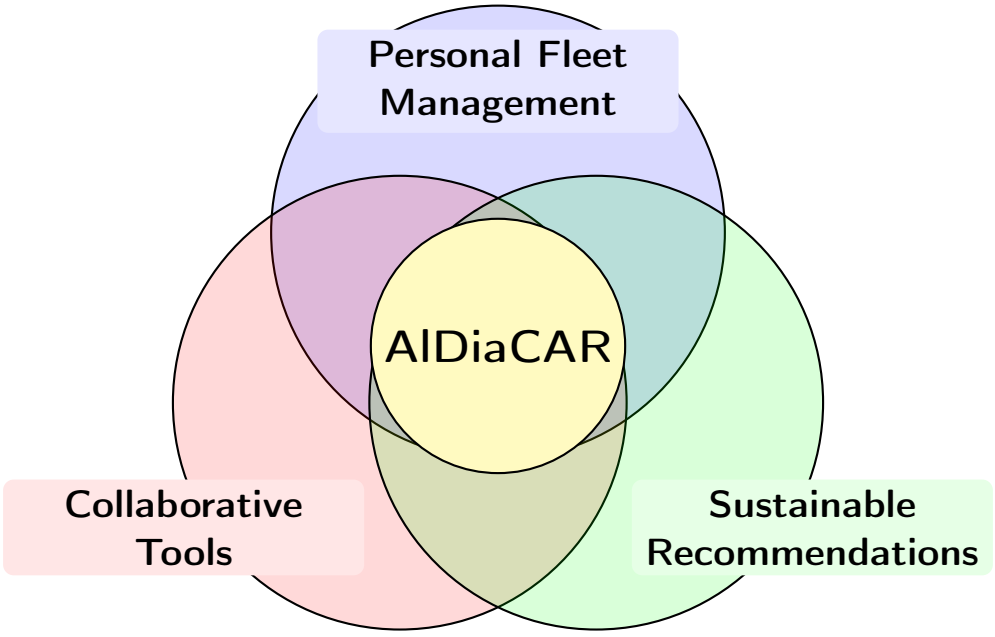


Figure 2.8: Visual representation of the market gap. AIDiaCAR is positioned at the intersection of Personal Fleet Management, Sustainable Recommendations, and Collaborative Tools, a niche not currently occupied by existing solutions. Source: Author's own work.

To further delineate this unoccupied niche, Table 2.1 provides a detailed comparative analysis, summarizing the limitations of existing application categories against the key requirements derived from our problem statement.

Table 2.1: Detailed gap analysis of existing application categories

Application category	**Target user**	**Core functionality**	**Sustainability focus**	**Identified gaps for AIDiaCAR Project**
Commercial Fleet Management Systems	Professional Fleet Manager (B2B)	Logistics, cost control, driver surveillance, large-scale analytics.	Primarily financial (fuel cost reduction); operational efficiency.	Prohibitively expensive and complex for personal use; lacks collaborative, non-surveillance model; misaligned UX and social context.
Personal Maintenance Apps	Individual Car Enthusiast (B2C)	Manual logging of fuel, expenses, and service for a single vehicle.	Minimal; limited to calculating fuel economy for financial tracking.	No collaborative features for shared management; poor multi-vehicle comparison; passive, utilitarian engagement model.
Eco-Routing & Navigation Tools	General Driver / Commuter (B2C)	Calculating a fuel-efficient route for a single, generic journey.	High; focused on optimizing a single trip's route for a generic vehicle model.	Disconnected from user's actual vehicle fleet; cannot recommend the optimal vehicle, only the optimal route; lacks maintenance context.

Therefore, this thesis project, *AIDiaCAR*, is strategically positioned to fill this clearly defined niche. It is conceived as the first mobile application to holistically address the complex needs of an individual or family managing multiple vehicles by uniquely unifying three critical pillars of functionality into a single, user-friendly experience.

First, it provides robust **Personal fleet management** capabilities. This goes beyond the single-vehicle logbook model by offering a centralized, collaborative dashboard where all family members can view the real-time status, location, and maintenance needs of every vehicle in the household fleet. This directly addresses the critical pain point of "shared maintenance blindness" and reduces "coordination friction".

Second, the system incorporates **Integrated sustainability recommendations**. Unlike generic eco-routers, *AIDiaCAR* leverages the detailed data of the user's specific

fleet. Its recommendation engine will consider not only the destination but also the unique emissions profile, fuel efficiency, maintenance status, and regulatory constraints of each available vehicle to suggest the holistically optimal car for every journey. This transforms the application from a passive data logger into an active decision-support tool.

Third, it implements a thoughtful **Behavioral reinforcement** layer through gamification. By embedding game-like elements such as badges, progress tracking, and positive feedback, the application aims to make sustainable choices and diligent vehicle care intrinsically rewarding. This actively encourages long-term habit formation, moving beyond mere utility to foster genuine user engagement and a collective household commitment to more sustainable mobility.

By seamlessly combining these three pillars, the *AIDiaCAR* project provides a novel and significant contribution to the field of Sustainable HCI. It offers a practical, user-centric tool designed to empower individuals and families to systematically reduce their personal transportation footprint, mitigate household friction, and manage their shared assets more efficiently and responsibly. This chapter has established the clear need and opportunity for such a solution, setting the stage for the subsequent chapters which will detail its design, implementation, and evaluation.

3 | AIDiaCAR: System definition and phased development

Having established the theoretical foundations and identified a distinct gap in the state-of-the-art of vehicle management applications in the preceding chapter, this chapter provides a comprehensive definition of the proposed solution: *AIDiaCAR*. This chapter will serve as the architectural and functional blueprint for the project, articulating its core mission, foundational principles, and detailed feature set. The objective is to translate the conceptual framework into a tangible product specification that directly addresses the coordination frictions, maintenance blindness, and suboptimal decision-making prevalent in the modern multi-vehicle household, as exemplified by the Martínez family *persona*.

The project's vision is ambitious, encompassing a wide range of functionalities from real-time coordination to *intelligent*[1], data-driven recommendations and behavioral reinforcement. To manage this complexity and ensure a methodologically sound development process, this chapter also introduces a phased implementation roadmap. This iterative approach deconstructs the complete vision into three distinct, sequential stages: the Alpha version, representing the core viable product; the Beta version, which enhances the system with *intelligent* features and user engagement mechanics; and finally, the Zenith release, which embodies the fully realized, feature-complete expression of the AIDiaCAR concept. This structured methodology allows for focused development, iterative testing, and the progressive validation of the project's hypotheses at each stage.

[1]Within the context of this thesis, the term "intelligent" is not used in the strong AI sense of consciousness or general problem-solving. Rather, it refers to the system's ability to synthesize multiple, disparate data sources (vehicle specifications, regulatory constraints, user input) into a coherent, actionable recommendation. It denotes a system that provides data-driven decision support, moving beyond simple data retrieval.

3.1 The AIDiaCAR vision: A holistic framework

The core mission of AIDiaCAR is to serve as the *intelligent*, collaborative "Digital garage" for the modern household. It is designed to seamlessly integrate vehicle management, user coordination, and sustainable decision-making into a single, cohesive, and effortless user experience. The system is architected upon four foundational pillars, each conceived to address a critical point of friction and inefficiency in the management of a shared, personal vehicle fleet. These pillars function interdependently to create a holistic solution that is more than the sum of its parts.

3.1.1 Visual identity and design principles

Before detailing the functional pillars, it is important to define the visual and interaction design philosophy that underpins the user experience. A consistent and thoughtful visual identity is critical for establishing user trust, ensuring usability, and creating an intuitive interface. The design of AIDiaCAR is guided by three core principles: clarity, accessibility, and encouragement. All visual elements, from the color palette to the typography, are selected to support these principles, creating a clean, modern, and non-intrusive user environment.

Figure 3.1 presents the project's style guide, which specifies the color palette, the chosen fonts for headings and body text, and the application logo. The color scheme utilizes a base of calming blues and greens to evoke a sense of reliability and sustainability, with vibrant accent colors used purposefully to draw attention to key actions and notifications.

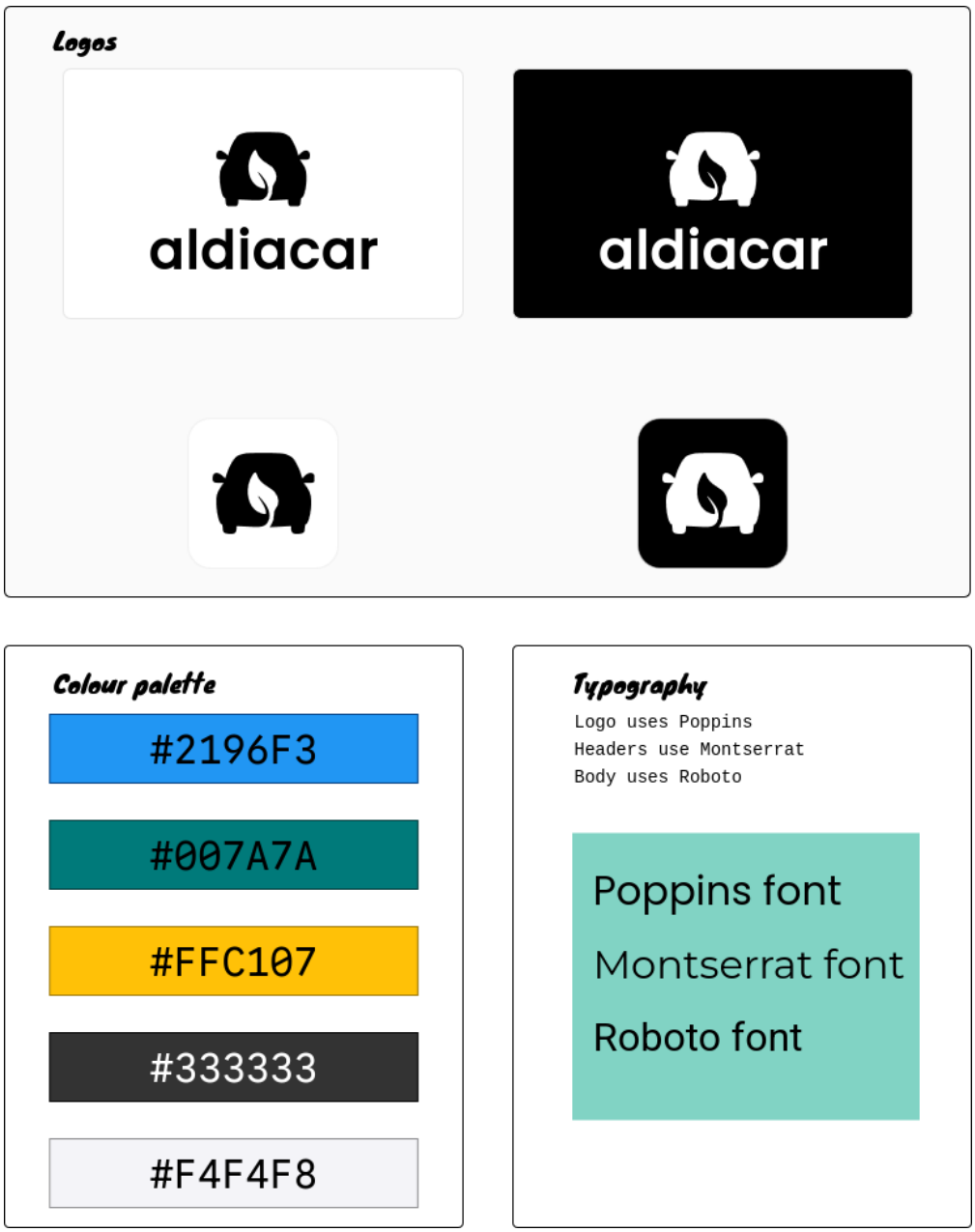


Figure 3.1: AIDiaCAR visual identity guide, defining the color palette, typography, and logo.

3.1.2 The four functional pillars

The functionality of the AIDiaCAR system is structured around four distinct, yet interconnected, pillars. Each pillar is designed to address a specific set of pain points

identified in the problem statement.

3.1.2.1 Pillar 1: The shared digital key rack (Coordination)

This foundational pillar is designed to definitively eradicate the persistent and friction-inducing "*who has what car?*" problem. It transforms the physical, opaque state of the household's vehicles into a transparent, digital, and universally accessible source of truth. Key functionalities include real-time vehicle status tracking, a seamless check-out/check-in protocol, and a proactive vehicle reservation system.

3.1.2.2 Pillar 2: The proactive co-pilot (Recommendation)

This pillar represents the core intelligence of the system, designed to answer the complex question, "*What is the smartest vehicle choice for this specific trip?*" Its multi-vector analysis engine evaluates journeys against regulatory constraints (ZBEs), economic cost, environmental impact (CO_2 emissions), and maintenance awareness to provide a simple, ranked list of optimal choices.

3.1.2.3 Pillar 3: The omniscient mechanic (Automated maintenance)

This pillar acts as the household's collective, digital memory for vehicle health, ensuring that all automotive assets are consistently maintained. It combats "shared maintenance blindness" through smart maintenance profiles that track service intervals based on both time and mileage, and a shared alert system for upcoming tasks.

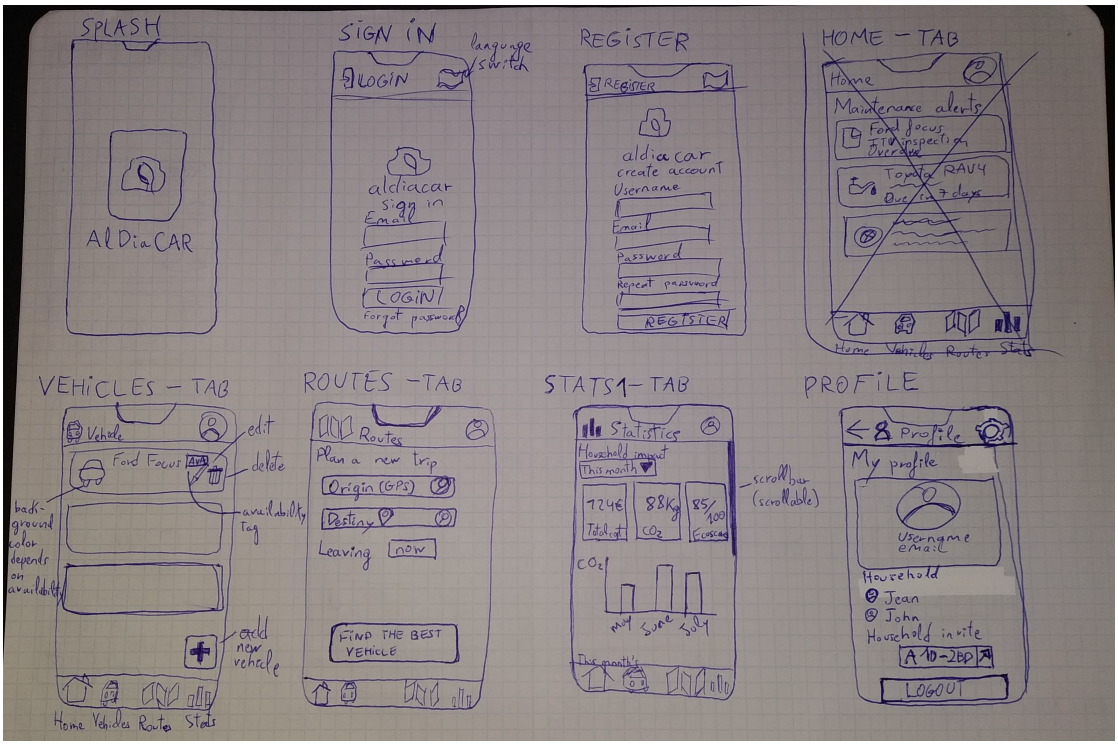
3.1.2.4 Pillar 4: The household impact dashboard (Gamification & analytics)

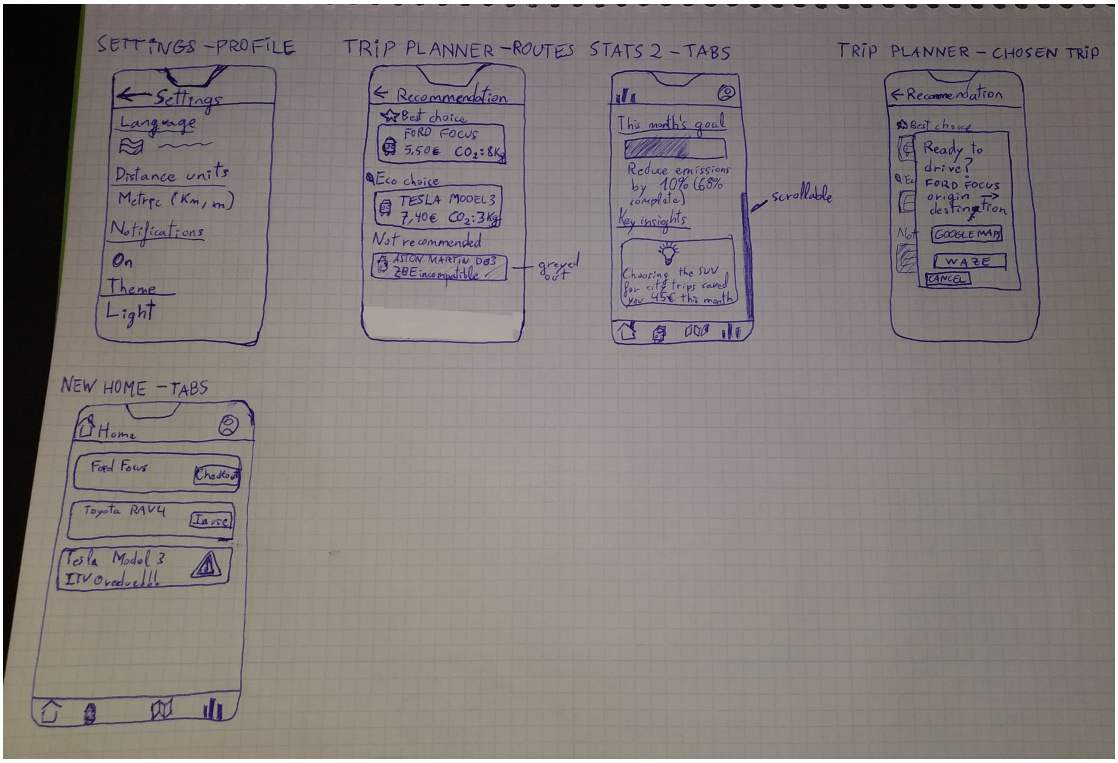
This final pillar addresses the motivational aspects of sustainable mobility. It is designed to transform abstract data from daily use into tangible insights and positive reinforcement through a unified dashboard, personal and team-based achievements, and actionable analytics.

3.2 Conceptual design and user flow wireframes

To translate the abstract functional pillars into a concrete and user-centric interface, this section presents a series of low-fidelity wireframes. These conceptual sketches visualize an initial, high-level vision for the user experience of the Zenith release, illustrating the potential information architecture, component layout, and user flows for the application's most critical tasks. This collection of early-stage designs, presented

across Figure 3.2 and Figure 3.2, serves as a visual blueprint that connects the system's defined features to its ultimate design goals.





3.3 Phased implementation roadmap

To systematically construct the comprehensive system described above, a phased development roadmap is proposed. This iterative methodology deconstructs the full feature set into three manageable and logically sequential versions. Each phase builds upon the last, allowing for focused development, targeted testing, and the potential for user feedback to inform subsequent stages. The three phases are designated as the Alpha version, the Beta version, and the Zenith release.

3.3.1 The Alpha version: Establishing the core viable product

The primary objective of the Alpha phase is to build and validate the foundational functionality of AIDiaCAR. This version is conceived as the Minimum Viable Product (MVP), focusing exclusively on solving the most acute and immediate pain points: coordination friction and shared maintenance blindness.

Features of the Alpha version:

- **Pillar 1 (Coordination):**
  - User and multiple vehicle registration.
  - A simple, clear dashboard showing the status of each vehicle.
  - Manual, one-tap "Check-Out" and "Check-In" functionality to update a vehicle's status between At Home and In Use. The user who checked out the vehicle is displayed.
- **Pillar 2 (Recommendation):** This pillar is **not implemented** in the Alpha version.
- **Pillar 3 (Maintenance):**
  - A dedicated section for each vehicle to manually log key maintenance dates.
  - Simple, time-based push notifications when a logged date is approaching.
- **Pillar 4 (Gamification):** This pillar is **not implemented** in the Alpha version.

The Alpha version is a digital replacement for the family's whiteboard, validating the core need for a centralized coordination hub.

3.3.2 The Beta version: Enhancing intelligence and engagement

Building upon the stable foundation of the Alpha version, the Beta phase introduces the "smart" features that define AIDiaCAR's unique value proposition.

Features of the Beta version (Includes all Alpha features, plus):

- **Pillar 1 (Coordination):** Introduction of the vehicle **Reservation System**.
- **Pillar 2 (Recommendation):** Implementation of an **initial Recommendation Engine** limited to **ZBE constraint analysis**.
- **Pillar 3 (Maintenance):** Enhanced tracking with **mileage-based intervals**.
- **Pillar 4 (Gamification):** A **first layer of gamification** with individual achievements and badges.

The Beta version transforms AIDiaCAR from a simple utility into an *intelligent* assistant.

3.3.3 The Zenith release: The complete vision

The Zenith release represents the culmination of the development roadmap, a feature-complete version of the application that fully realizes the initial four-pillar vision.

Features of the Zenith release (Includes all Beta features, plus):

- **Pillar 1 (Coordination):** Polished UI/UX with potential integration into third-party calendar applications.
- **Pillar 2 (Recommendation):** The **full, multi-factor Recommendation Engine** with cost, CO_2, and maintenance awareness.
- **Pillar 3 (Maintenance):** A shared, collaborative **Household Task List** for maintenance items.
- **Pillar 4 (Gamification):** The complete **Household Impact Dashboard** with advanced analytics and collaborative goals.

The Zenith release is the ultimate expression of AIDiaCAR: an *intelligent* layer that removes friction, saves money, enhances safety, and systematically guides the family toward more sustainable mobility.

4 System design and architecture

This chapter provides a detailed exposition of the architectural design and technical framework for the *AIDiaCAR* application. The design is fundamentally guided by the project's core objectives as defined in the preceding chapters: to deliver a cross-platform, scalable, and maintainable software solution that empowers households to manage and optimize their personal vehicle usage with a primary focus on enhancing sustainability and reducing operational friction.

The following sections will deconstruct the system from a high-level perspective down to its constituent components. We will begin by outlining the overarching client-server architecture, providing a rationale for this fundamental design choice. Subsequently, we will conduct a deep dive into the specific architectural patterns employed within both the frontend and backend, justifying the decisions made to balance complexity, performance, and developer ergonomics. The chapter will also thoroughly define the data model that enables the system's collaborative features, detail the communication protocols and security measures governing the data flow, and provide a comprehensive justification for the selected technology stack. This architectural blueprint serves as the technical foundation upon which the application's features, detailed in Chapter 5, are built.

4.1 High-level architecture: A client-server model

The system is architected following a classic, robust client-server model, a paradigm in which a client requests resources and a server provides them, forming the basis of modern distributed applications [8]. This approach was chosen for its clear separation of concerns, which effectively decouples the presentation layer (the user interface) from the application's core business logic and data persistence layers. This separation is paramount for achieving several key non-functional requirements. It enhances security by abstracting the database behind a controlled Application Programming Interface

(API), improves scalability by allowing the client and server to be scaled independently, and provides strategic flexibility for future development, such as the creation of a web-based client or third-party integrations, without necessitating any modifications to the backend infrastructure. This decoupling is achieved through the use of well-defined APIs, which act as stable architectural boundaries that allow each component to evolve independently [9].

The system is composed of four primary, logically distinct components:

- **Frontend Client:** A cross-platform mobile application that serves as the sole point of interaction for the end-user. It is responsible for rendering the user interface, capturing user input, managing client-side state, and handling all communication with the backend server. The use of React Native and the Expo framework enables a single codebase to target both iOS and Android platforms.

- **Backend Server:** The central nervous system of the application. Implemented as a Node.js application using the Express.js framework, the backend serves as the authoritative hub for all operations. Its responsibilities include handling business logic, processing data, managing user authentication and authorization, and serving as the exclusive intermediary for all database interactions.

- **Database:** A MongoDB NoSQL database instance is utilized for all persistent data storage. This includes household profiles, user accounts, detailed vehicle specifications, maintenance schedules, and trip logs. Its flexible, document-oriented nature provides the agility required for an evolving data model.

- **Mock API (Development Environment):** To facilitate robust and isolated development, a secondary Node.js server is included in the development environment. This server simulates a third-party API for retrieving vehicle specifications (e.g., emission factors), allowing the core application to be developed and tested with deterministic data without reliance on external services.

Figure 4.1 provides a visual representation of this architecture, illustrating the primary components and the standardized data flow protocols that connect them within the development environment.

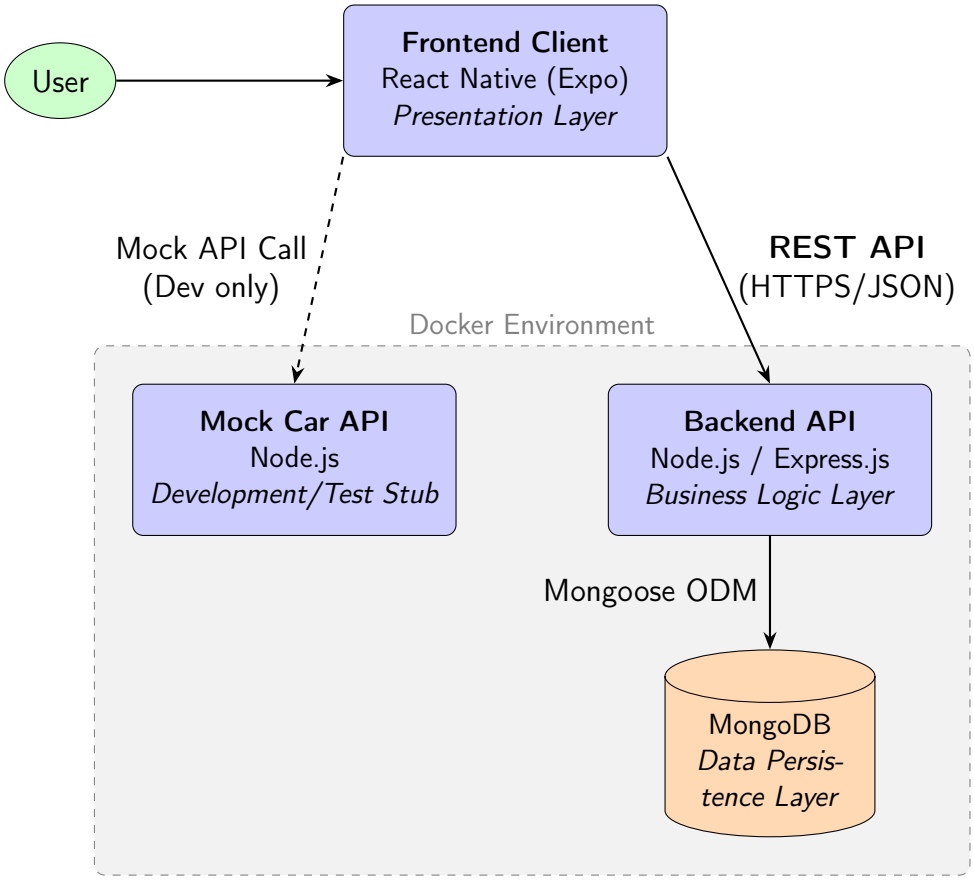


Figure 4.1: High-level system architecture, illustrating the separation of layers and data flow.

4.2 Backend architecture: A Model-View-Controller approach

The backend is architected as a monolithic RESTful API adhering to the Model-View-Controller (MVC) design pattern, a well-established architectural pattern for separating application concerns and improving modularity, as influenced by the principles of object-oriented design [10]. This classical pattern was selected for its proven efficacy in organizing server-side applications, promoting a clear separation of concerns that directly maps to the project's directory structure. This organizational clarity enhances maintainability and simplifies development.

The request-response lifecycle within the backend follows a well-defined path, as illustrated in the sequence diagram in Figure 4.2. An incoming HTTP request from

the client is first received by the Express.js router. The router identifies the appropriate endpoint and forwards the request through a chain of middleware functions, primarily for authentication. Upon successful validation, the request is handed to the designated controller function, which contains the core business logic. The controller interacts with the Mongoose model to perform necessary database operations (Create, Read, Update, Delete - CRUD). Finally, the controller formats the response and sends it back to the client.

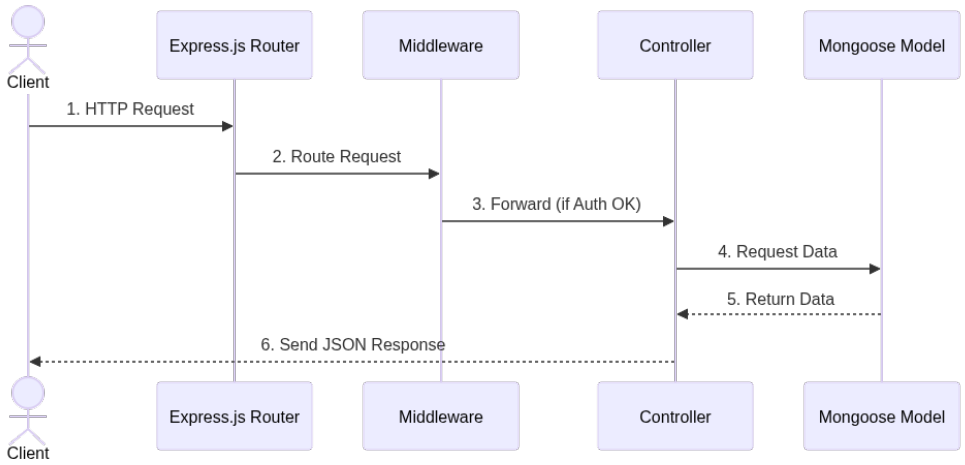


Figure 4.2: Sequence diagram of the backend request-response lifecycle.

4.2.1 Backend components

- **Routes (‘/routes‘):** This layer defines the API's endpoints. It maps specific HTTP methods (GET, POST, PUT, DELETE) and URL paths to their corresponding controller functions. The design employs dedicated route files for each logical resource (e.g., `authRoutes.js`, `vehicleRoutes.js`, `householdRoutes.js`), promoting modularity.

- **Controllers (‘/controllers‘):** Controllers form the heart of the business logic. Each function within a controller is responsible for handling a specific request, orchestrating the necessary operations, and formulating the final response. In the current architecture, controllers directly interact with the Mongoose models to execute data operations and contain all business-specific logic, such as calculating maintenance alerts or processing vehicle recommendations.

- **Models (‘/models‘):** This layer defines the data structures (schemas) for the application's entities using Mongoose, an Object Data Modeling (ODM) library

for MongoDB. The primary models are `Household.js`, `User.js`, `Vehicle.js`, and `Trip.js`. These models provide a high-level abstraction for all interactions with the database.

- **Middleware (‘/middleware‘):** Middleware functions are executed between the router and the controller. They are primarily used for cross-cutting concerns. The most critical middleware, `authMiddleware.js`, is responsible for protecting routes by extracting and verifying the user's JSON Web Token (JWT), ensuring that only authenticated users can access protected resources.

4.2.2 Architectural Decision: Controller-Centric Business Logic

A deliberate architectural decision was made to embed all business logic directly within the controller layer, rather than abstracting it into a separate "Service Layer." For the current scope and complexity of the project, this approach offers the benefit of simplicity and reduces developmental overhead. It allows for rapid iteration on the core functionalities of the prototype. However, it is recognized that as an application scales, this approach can lead to "fat controllers" and code duplication. Future iterations of the system would likely require refactoring to introduce a dedicated service layer to encapsulate reusable business logic, thereby improving modularity and testability.

4.3 Frontend Architecture: A Hybrid State Management Strategy

The frontend client, built with React Native and Expo, employs a pragmatic hybrid state management strategy. This approach avoids the dogmatic application of a single pattern, instead choosing the most appropriate tool for each type of state, thereby optimizing for both developer experience and application performance.

4.3.1 Global State Management via React Context

For state that is truly global and required across many disparate parts of the application, the React Context API is employed. The primary use case for this is user authentication. The `AuthContext` provides the entire component tree with access to the current user's data, authentication status, and functions for logging in and out. This approach is ideal for such cross-cutting concerns because it is lightweight, built directly into the React library, and avoids the complexity of larger, third-party state management libraries.

4.3.2 Screen-Level State Management (MVVM-like Pattern)

For state that is local to a specific screen, the architecture follows a pattern that is conceptually aligned with Model-View-ViewModel (MVVM), where each screen manages its own state and logic, a common approach in modern client-side applications for achieving a clean separation of the view from its underlying model [11]. Each screen component is responsible for managing its own state (e.g., list of vehicles, loading status) and data-fetching logic using React's built-in hooks. This encapsulation prevents the global state from becoming a bottleneck, improves performance by preventing unnecessary re-renders, and makes components easier to test in isolation. This approach aligns with the principles of modern, declarative UI development prevalent in the React Native ecosystem [12].

4.4 Data Architecture and Modeling

The design of the data model is critical to supporting the application's core collaborative features.

4.4.1 Rationale for a NoSQL Database (MongoDB)

A NoSQL database, specifically MongoDB, was chosen over a traditional relational database for its flexible data schema. The domain of vehicle management involves entities with diverse attributes; for example, an electric vehicle's data profile differs significantly from that of a diesel vehicle. A NoSQL document-based model allows for these variations to exist within the same collection without requiring complex schema migrations. This approach aligns with the principles of aggregate-oriented databases, which prioritize scalability and development flexibility [13].

4.4.2 Core Collections and Collaborative Relationships

The data model is architected around the central concept of the "household," as shown in Figure 4.3. This design directly reflects the problem statement and enables all multi-user functionalities.

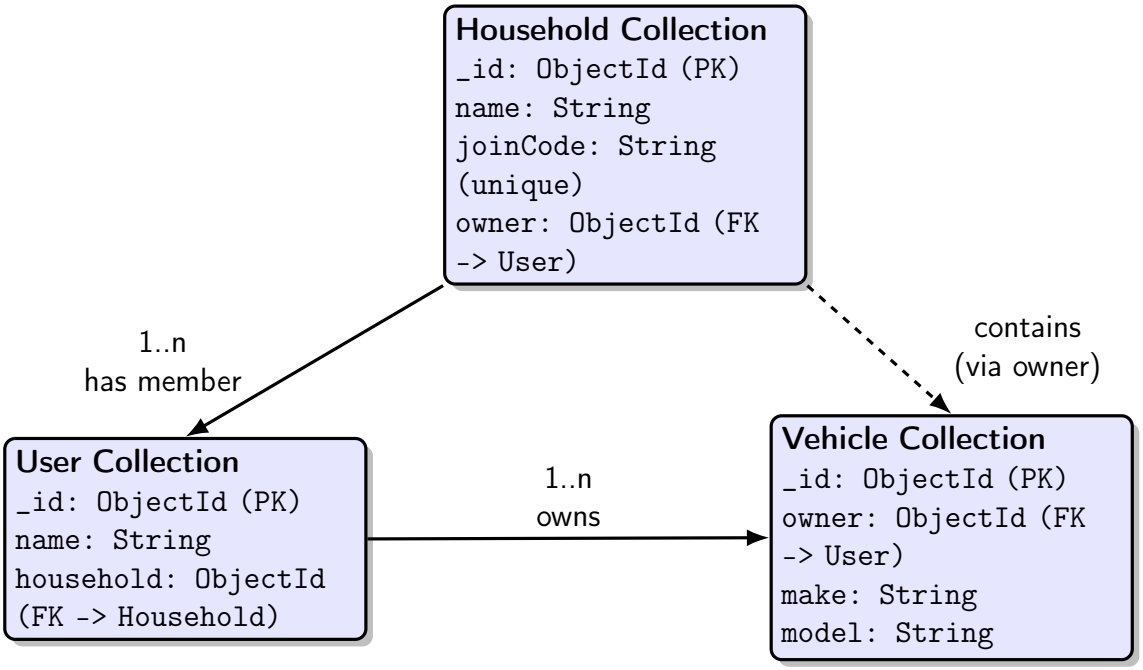


Figure 4.3: Detailed data model illustrating the central role of the Household collection and its relationships with users and vehicles.

The primary collections and their relationships are as follows:

- **Households:** This is the central collection that groups users and their shared assets. Each household has a unique, randomly generated 'joinCode' to facilitate a simple and secure invitation mechanism.
- **Users:** Each user document is linked to a single 'Household' via a reference ID. When a new user registers, a personal household is created for them by default. They can then join other households or invite others to theirs.
- **Vehicles:** Each vehicle is owned by a single user. A vehicle's association with a household is therefore derived implicitly through its owner's membership in that household. This enables authorization logic where any member of a household can view and interact with any vehicle owned by another member of the same household.

A comprehensive definition of all Mongoose data schemas is provided in Appendix B.

4.5 API design and security protocols

The Application Programming Interface (API) is the critical boundary between the client and server, and its design is governed by principles of standardization and security.

4.5.1 API design: RESTful principles

The API is designed following REST (Representational State Transfer) principles, adhering to the constraints of statelessness, resource-based addressing, and a uniform interface as defined in the foundational architectural style [14]. It uses standard HTTP methods (GET, POST, PUT, DELETE) for operations and leverages URL paths to identify resources (e.g., `/api/vehicles`). This adherence to web standards makes the API predictable and easy to consume. A complete specification of all API endpoints is available in Appendix A.

4.5.2 Authentication and authorization

The system employs a robust, token-based security model.

- **Authentication:** User authentication is managed via JSON Web Tokens (JWTs). Upon successful login, the backend server generates a signed JWT and sends it to the client in a secure, **HTTP-Only** cookie. The HTTP-Only flag is a crucial security measure that prevents the token from being accessed by client-side JavaScript, mitigating the risk of Cross-Site Scripting (XSS) attacks.

- **Authorization:** The system's authorization strategy is **Household-Based Access Control**. Once a user is authenticated, the middleware performs a second check for any request that attempts to access a resource (like a vehicle). The logic verifies that the requesting user is a member of the same household as the resource's owner. If they are not, the request is rejected with a '403 Forbidden' status. This model ensures that users can only interact with vehicles within their own household group.

4.6 Technology stack justification

The selection of each technology was the result of a deliberate evaluation of the project's requirements.

- **React Native (with Expo):** Chosen for its primary benefit of cross-platform development, allowing a single codebase to generate native applications for both

iOS and Android. This drastically reduces development time.
Considered Alternative: Flutter. Flutter was a strong contender but was ultimately not selected due to the project's reliance on the vast JavaScript/React ecosystem and the desire for language synergy with the backend.

- **Node.js with Express.js:** Selected for the backend due to its non-blocking, event-driven architecture, which provides excellent performance for an API server. Its use of JavaScript creates a seamless, full-stack development experience.
Considered Alternative: Python with Django/Flask. This stack was passed over to maintain language consistency across the entire project.

- **MongoDB:** As previously detailed, MongoDB was chosen for its flexible document-based schema, highly suited to the evolving data structures of vehicle profiles.
Considered Alternative: PostgreSQL. A relational database was considered for its data integrity features, but the need for schema flexibility was deemed more critical for this project.

- **Docker:** Docker and Docker Compose are utilized to containerize the backend services. This practice is a key enabler of modern Continuous Delivery pipelines, as it eliminates environmental discrepancies and simplifies the deployment process [15].

4.7 Architectural considerations for external integrations

While the current proof-of-concept uses simplified, hardcoded data for features like ZBE detection, the Zenith Release vision requires integration with several third-party services. To manage these future integrations without creating tight coupling, the backend architecture will employ the **Adapter Pattern**. For each external service (e.g., ZBE data, fuel prices, mapping), a dedicated "adapter" module will be created. This module will act as an intermediary, translating the application's internal requests into the specific format required by the external API and translating the response back into a standardized format. This abstraction ensures that if a data provider changes in the future, only the corresponding adapter module needs to be rewritten, leaving the core business logic of the application untouched.

5 | Implementation details

This chapter transitions from the abstract architectural design of Chapter 4 to the concrete technical implementation of the *AIDiaCAR* system. The objective is to provide a detailed walkthrough of the project's source code, explaining the organizational structure, key software modules, and the programming patterns employed to bring the conceptual design to life. This chapter serves as a bridge between the system's architecture and its validation, detailing not only *what* was built but also the quality assurance strategy designed to verify its correctness and robustness.

5.1 Project structure overview

The project is organized as a monorepo, a single repository containing all the code for the system. This approach simplifies dependency management and streamlines development across the different parts of the application, which is particularly advantageous for a full-stack project with tightly coupled frontend and backend components. The high-level directory structure, which promotes a clear separation of concerns, is shown in Figure 5.1.

The `src` directory cleanly separates the `backend` and `frontend` codebases. The `docker` directory contains the Docker Compose configuration, ensuring a reproducible development environment for all services. Finally, the `tests` directory houses the automated quality assurance framework, with distinct subdirectories for backend and frontend testing.

5.2 Backend implementation (Node.js)

The backend is a RESTful API built with Node.js and the Express.js framework. It is the authoritative core of the system, responsible for all business logic, data persistence, and security enforcement.

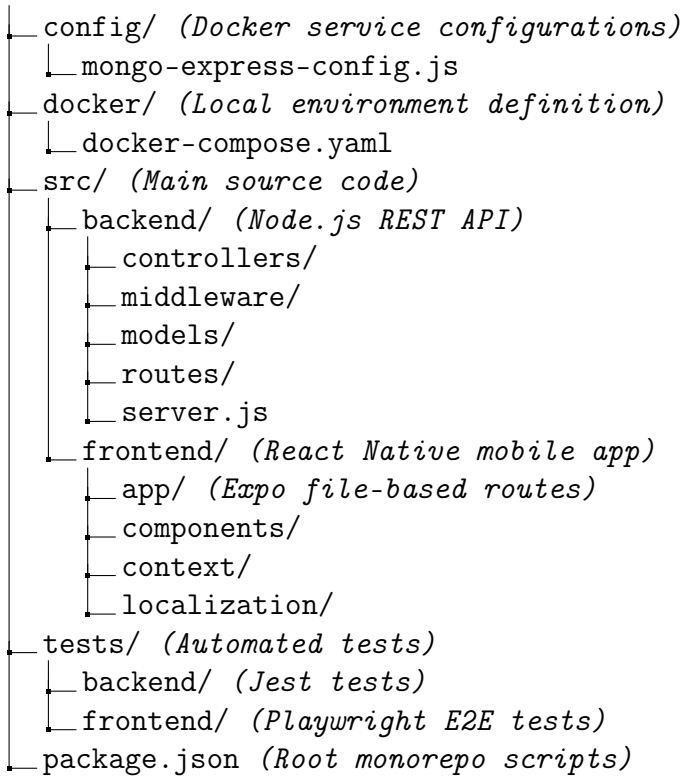


Figure 5.1: High-level project directory structure.

5.2.1 Data models and persistence (Mongoose)

Data schemas are defined in the `src/backend/models/` directory using Mongoose, an Object Data Modeling (ODM) library that provides a robust abstraction layer for interacting with the MongoDB database. The principal models implemented are `Household.js`, `User.js`, `Vehicle.js`, and `Trip.js`.

The collaborative nature of the application is enabled by the `Household` model. A key feature of this schema is the automatic generation of a unique, human-readable 'joinCode' upon creation, facilitated by the 'nanoid' library. This code serves as a simple and secure mechanism for users to join existing households. A snippet of this implementation is shown below.

Listing 5.1: Join Code Generation in `models/Household.js`

```
import mongoose from 'mongoose';
import { customAlphabet } from 'nanoid';

// Using a dictionary-based alphabet for memorable codes
```

```
const alphabet = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789';
const nanoid = customAlphabet(alphabet, 8);

const HouseholdSchema = new mongoose.Schema({
  // ... other fields
  joinCode: {
    type: String,
    required: true,
    unique: true,
    // Automatically generate code on creation
    default: () => nanoid(),
  },
  // ... other fields
});
```

The full, detailed Mongoose schemas for all data models are provided in Appendix B for comprehensive review.

5.2.2 Controllers and business logic

The core business logic of the application resides within the controller files in the `src/backend/controllers/` directory. These modules orchestrate the interactions between incoming API requests, the data models, and the final response sent to the client.

A prime example of this orchestration is the user registration flow within `authController.js`. When a new user registers, the system not only creates a new `User` document but also atomically creates a new, personal `Household` for that user, establishing the foundational link for all collaborative features from the moment of inception.

The logic for managing collaborative groups is encapsulated in `householdController.js`. The `joinHousehold` function handles the complex state changes required when a user moves from one household to another. This includes adding the user to the new household's member list, updating the user's own household reference, removing them from their previous household, and, critically, deleting the old household if the departing user was its last member. This logic, illustrated in Figure 5.2, ensures data integrity and prevents orphaned household documents in the database.

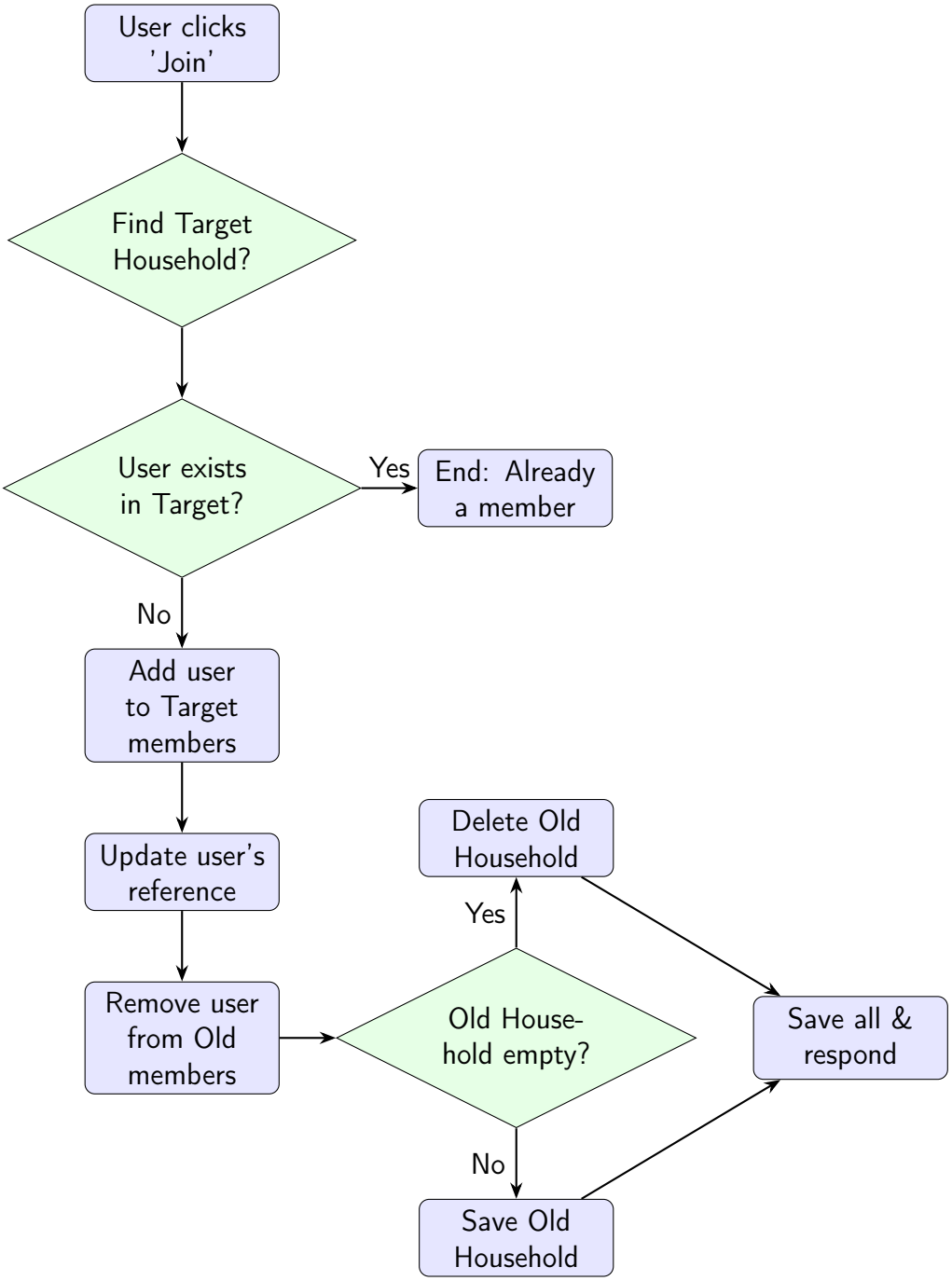


Figure 5.2: Logical flow diagram for the ‘joinHousehold‘ controller function.

5.2.3 API routes and security middleware

API endpoints are defined in the `src/backend/routes/` directory. These files map URL paths and HTTP methods to their corresponding controller functions. Security is enforced at this layer through custom middleware. The most critical of these is the `protect` function in `authMiddleware.js`, which is applied to all sensitive routes. This middleware extracts the JWT from an `HttpOnly` cookie sent with the client's request, verifies its signature against the server's secret key, and, if valid, decodes the user's ID from the token payload. This ID is then attached to the Express request object, making the authenticated user's identity securely available to all subsequent controller logic without needing to pass it in the request body.

A complete specification of all API endpoints, including their routes, required HTTP methods, expected payloads, and required authentication level, is available in Appendix A.

5.3 Frontend implementation (React Native)

The frontend is a cross-platform mobile application developed using React Native with the Expo framework and TypeScript for static typing. It provides the user interface and manages all interactions with the backend API.

5.3.1 Navigation and screen architecture

The application's navigation is managed by Expo's file-based router. The directory structure within `src/frontend/app/` directly translates into the app's navigational routes. The main user interface is structured around a tab bar, defined in `app/(tabs)/_layout.tsx`, which provides access to the primary sections: Home, Vehicles, Routes, and Stats.

A key programming pattern employed across the main screens is the `useFocusEffect` hook from Expo Router. This hook is used to re-fetch data from the backend whenever a screen comes into focus. This ensures that the data displayed to the user is always fresh and reflects any changes made on other screens (e.g., refreshing the vehicle list after a new vehicle has been added), providing a seamless and intuitive user experience.

5.3.2 State management and API communication

The frontend employs a hybrid state management strategy.

- **Global State:** Application-wide state, specifically the user's authentication status and profile information, is managed via React's Context API. The implementation in `context/AuthContext.tsx` creates a provider that wraps the entire application, making the user's identity and authentication status accessible to any component without the need for prop drilling.
- **Local State:** Screen-specific state, such as the list of vehicles on the vehicle management screen, is managed locally within each component using React's native `useState` and `useEffect` hooks.

All communication with the backend API is handled by the `axios` library. A global instance of axios is configured with the `withCredentials: true` option. This crucial setting instructs the library to automatically include the secure, `HttpOnly` authentication cookie with every request sent to the backend, seamlessly managing user sessions.

5.4 Quality assurance and test planning

To ensure the correctness and reliability of the implemented proof-of-concept, a multi-layered testing strategy was designed, adopting principles from the classic software testing pyramid. This strategy combines low-level unit and integration tests with high-level end-to-end tests to validate the system's functionality from different perspectives.

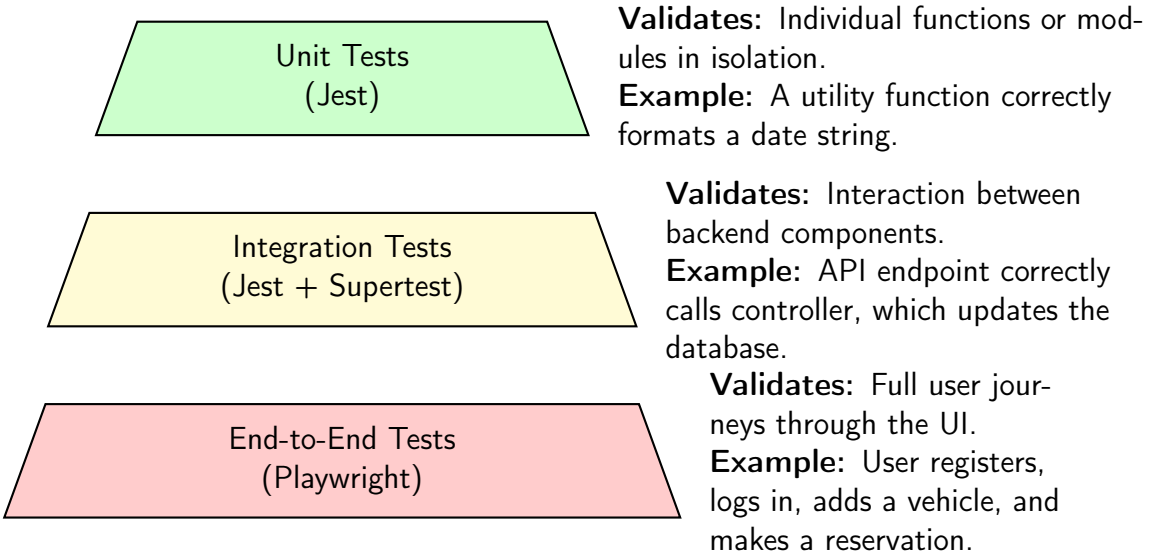


Figure 5.3: The software testing pyramid as applied to the AIDiaCAR project.

As illustrated in Figure 5.3, the strategy is composed of the following layers:

- **Backend testing (Jest & Supertest):** The backend test suite, located in `tests/backend/`, uses the Jest framework. Unit tests are planned for individual utility functions and complex business logic modules in isolation. Integration tests, using the Supertest library, are designed to validate the complete request-response cycle of the API endpoints. The implemented test suite for this thesis focuses on the most critical endpoints, such as user authentication and the vehicle recommendation logic, to validate the core "vertical slice" of the application.

- **Frontend testing (Playwright):** End-to-End (E2E) tests are designed to simulate a real user's journey through the application from start to finish. The test plan for this proof-of-concept, located in `tests/frontend/`, utilizes the Playwright framework to create automated scripts for the most critical user flows: (1) successful user registration and login, (2) adding a new vehicle to a household, and (3) making a reservation for an available vehicle.

This testing strategy ensures that the most critical path of the application is validated at multiple levels of abstraction. The results of these tests are presented in Chapter 6.

5.5 Development effort and cost analysis

While this project was undertaken within an academic context, it is valuable to perform a cost analysis to estimate the resources required to develop a market-ready product based on the AIDiaCAR concept. This section provides a high-level budget estimation, breaking down the costs associated with the development of the "Thesis-Grade Beta" (the scope implemented for this project) and projecting the additional investment needed to achieve the full Zenith release.

5.5.1 Methodology and assumptions

The budget is calculated based on a team of freelance professionals based in Spain, with standard market rates for software development roles. The estimation is based on the hours required for each key phase of the project: design, development, and project management.

The following roles and hourly rates are assumed:

- **Project manager / Product owner:** Responsible for requirements, planning, and oversight. Rate: €50/hour.

- **UI/UX Designer:** Responsible for visual identity, wireframing, and creating high-fidelity mockups. Rate: €35/hour.

- **Backend developer:** Responsible for the server, API, and database architecture. Rate: €40/hour.
- **Frontend (Mobile) developer:** Responsible for the React Native application. Rate: €40/hour.

5.5.2 Estimated development costs

Table 5.1 outlines the estimated hours and costs for the development of the Thesis-Grade Beta (representing the completed Alpha and Beta phases) and the projected additional effort required to complete the Zenith release.

Table 5.1: Estimated development costs by phase

Role	Hours (Beta)	Hours (Zenith add-on)	Rate (€/hr)	Subtotal (Beta)	Subtotal (Zenith add-on)
Project manager	40 hrs	60 hrs	€50	€2,000	€3,000
UI/UX Designer	30 hrs	40 hrs	€35	€1,050	€1,400
Backend developer	120 hrs	100 hrs	€40	€4,800	€4,000
Frontend developer	120 hrs	100 hrs	€40	€4,800	€4,000
Total person-hours	**310 hrs**	**300 hrs**	—	—	—
Subtotal labor costs	—	—	—	**€12,650**	**€12,400**
			Projected total for Zenith release:		**€25,050**

The development of the Thesis-Grade Beta, which includes the critical household data model, the reservation system, and the full multi-factor recommendation engine, is estimated to require approximately 310 person-hours, resulting in a cost of €12,650.

To complete the Zenith release, which would involve integrating live third-party APIs, building the full graphical analytics dashboard, and implementing the collaborative gamification layer, an additional 300 person-hours are projected. This would represent a further investment of €12,400, bringing the total estimated development cost for a market-ready Zenith version to €25,050.

5.5.3 Other associated costs

Beyond direct development labor, a production-grade application would incur other initial and ongoing costs. These are summarized in Table 5.2.

Table 5.2: Other costs annual estimation

Category	Description	Est. cost (p.a.)
Infrastructure	Server hosting (PaaS), managed database (MongoDB atlas)	€600
Third-Party APIs	Google Maps API, ZBE data provider, fuel price API	€500 - €1,500
App store fees	Apple developer program, Google Play developer account	€125
Contingency (15%)	For unforeseen development challenges and scope changes	€3,757 (initial)
Total (First year)		**€4,982 - €5,982**

6 | Results and validation

This chapter presents the results of the implementation phase, validating the functionality of the *AIDiaCAR* prototype against the objectives defined in the preceding chapters. The validation process serves two primary purposes: first, to confirm that the system's key features are operational and have been implemented correctly according to the architectural design; and second, to demonstrate that these features effectively address the core problem statement. This chapter provides visual evidence of the working frontend through a walkthrough of the core user journeys, complemented by a summary of the automated tests that verify the backend logic and the integrity of the system as a whole.

6.1 Frontend validation via core user workflows

The most direct method for validating the application's success is to demonstrate its functionality from the end-user's perspective. The following sections use screenshots from the running application to provide tangible evidence that the primary features have been successfully implemented and that the core user workflows are operational.

6.1.1 User onboarding and household management

The user journey begins with registration. As per the system's architecture, upon creating an account, each user is automatically assigned to a new, personal household. This establishes the foundational collaborative entity from the outset. The user can then manage their household, invite others, or join an existing household. This workflow directly validates the implementation of the 'Household' data model, which is central to the project's collaborative goals.

Figure 6.1 shows the application's login screen, the entry point for authenticated users. Upon successful login, the user can navigate to their profile, as shown in Figure 6.2. This screen serves as the central hub for identity and household management, displaying the user's details, the current household's name, its unique join code for inviting

others, and a list of current members. This screen confirms that the authentication, data retrieval, and household association processes are functioning correctly.

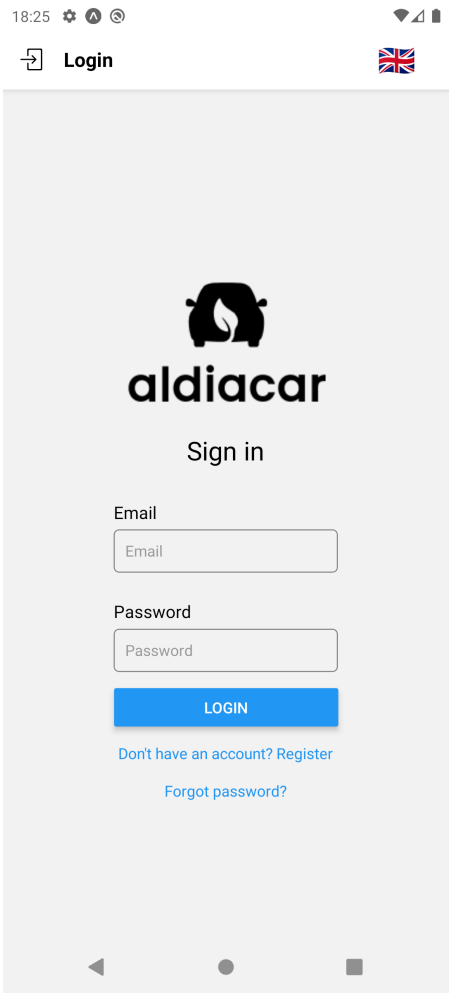


Figure 6.1: The user login screen, serving as the gateway to the application.

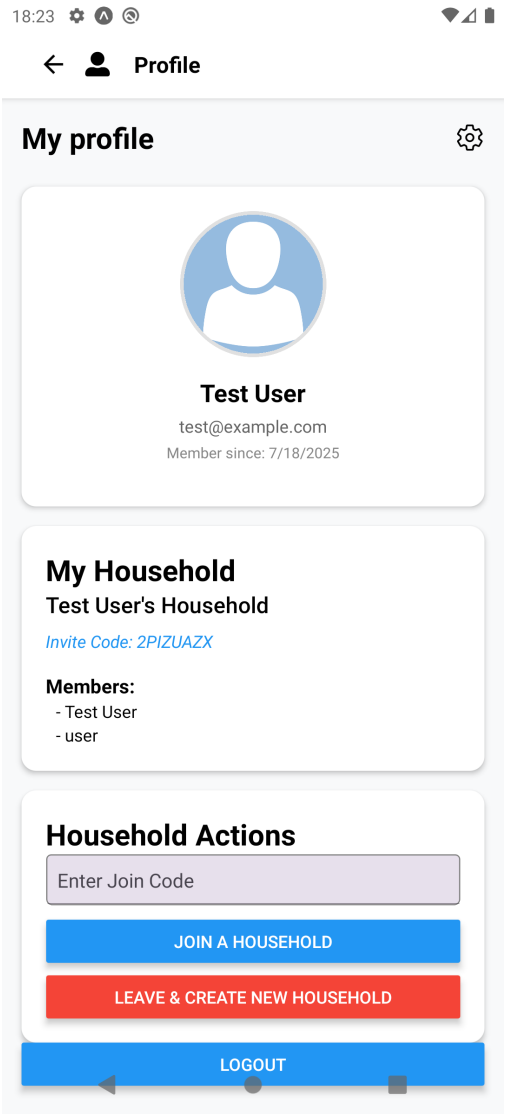


Figure 6.2: The user profile screen, displaying personal details and the household management card with its unique join code and member list.

6.1.2 Vehicle fleet management

A core requirement of the system is to provide users with a centralized tool to manage their personal fleet of vehicles. The "Vehicles" tab provides this functionality, validating the system's CRUD (Create, Read, Update, Delete) capabilities for vehicles. Users can view all vehicles associated with their household, add new vehicles, and edit or delete existing ones.

Figure 6.3 shows the main vehicle list, providing a complete, at-a-glance overview of the household's automotive assets. From this screen, the user can initiate the process of adding a new car, which leads to the form shown in Figure 6.4. This workflow demonstrates a key feature: the system first requires basic vehicle information and then simulates a call to an external service (via the Mock API) to fetch technical specifications like the emission factor, which is crucial for the recommendation engine.

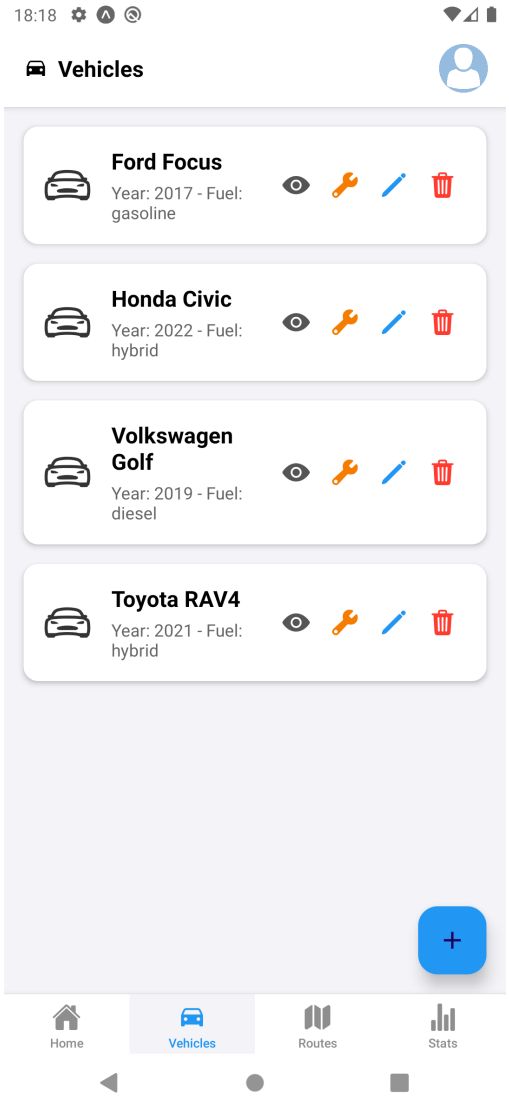


Figure 6.3: The main vehicle management screen, displaying a list of all vehicles registered to the household.

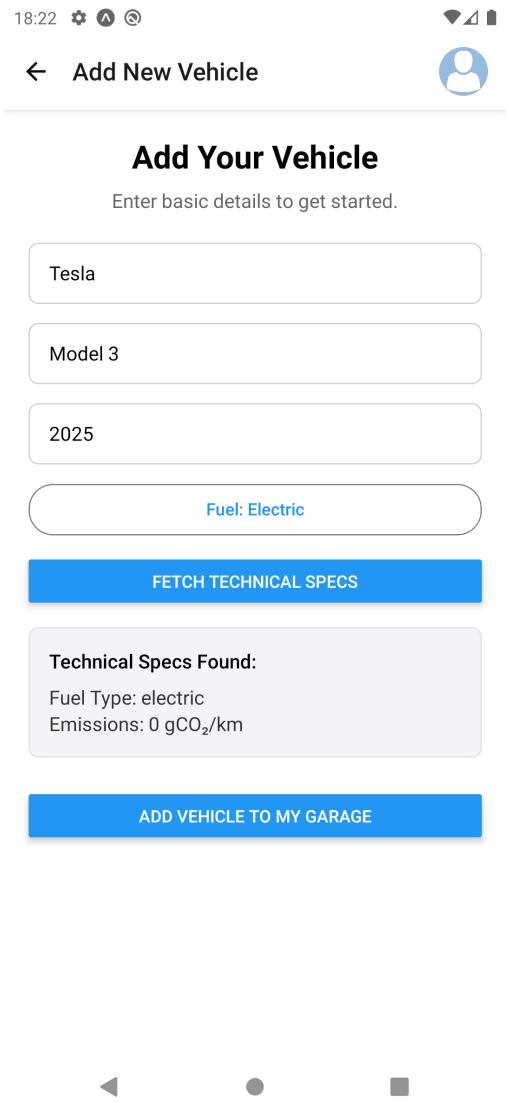


Figure 6.4: The two-step process for adding a new vehicle, including fetching technical specifications from the mock external API.

6.1.3 Coordination and vehicle status dashboard

To directly address the problem of coordination friction, the "Home" tab serves as a real-time vehicle status dashboard. This interface, shown in Figure 6.5, provides all household members with immediate and transparent visibility into the current state of each vehicle: ‘At Home‘, ‘In Use‘, or ‘Reserved‘.

This screen validates the implementation of Pillar 1. Users can perform one-tap

"Check-Out" and "Check-In" actions to update a vehicle's status. Furthermore, the dashboard provides the interface for the vehicle reservation system. As shown in Figure 6.6, a user can select an available vehicle and reserve it for a future time slot, which immediately updates its status for all other household members, preventing scheduling conflicts and enabling proactive planning.

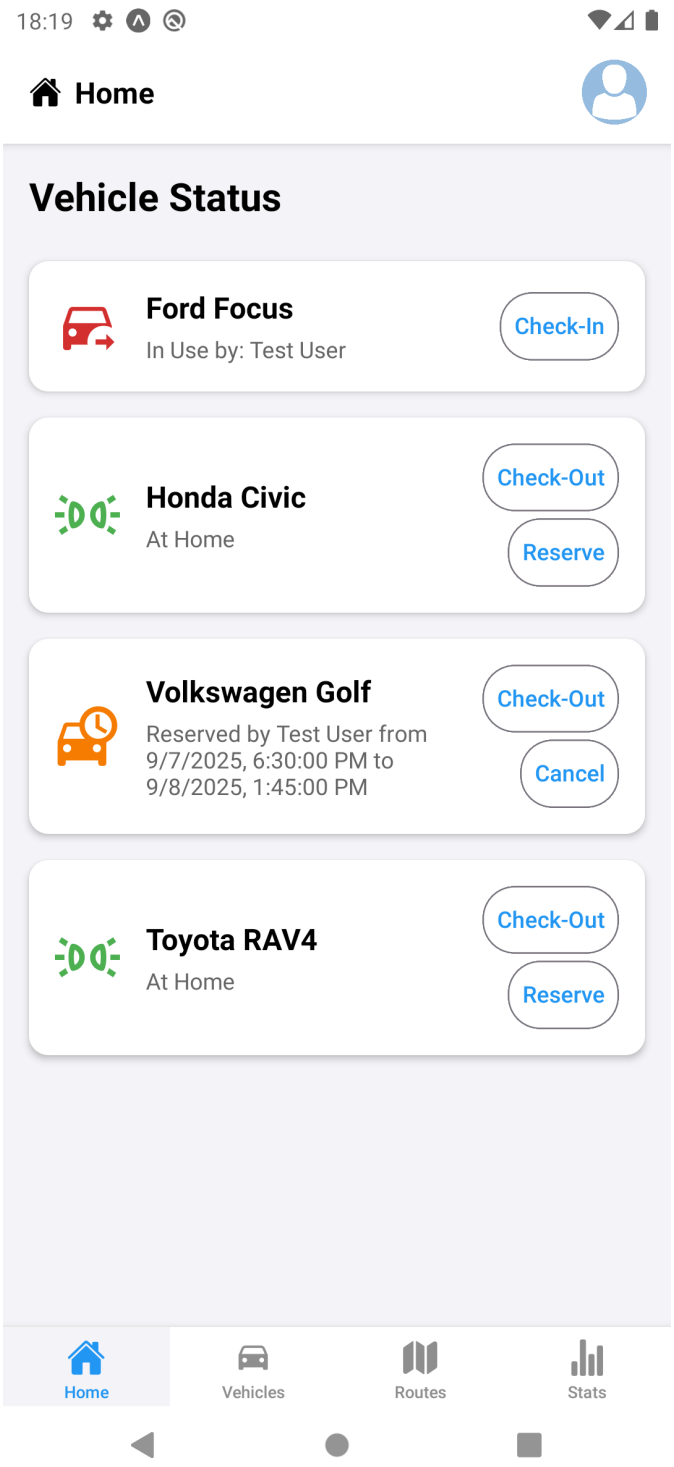


Figure 6.5: The main status dashboard, providing real-time visibility into the availability of each household vehicle.

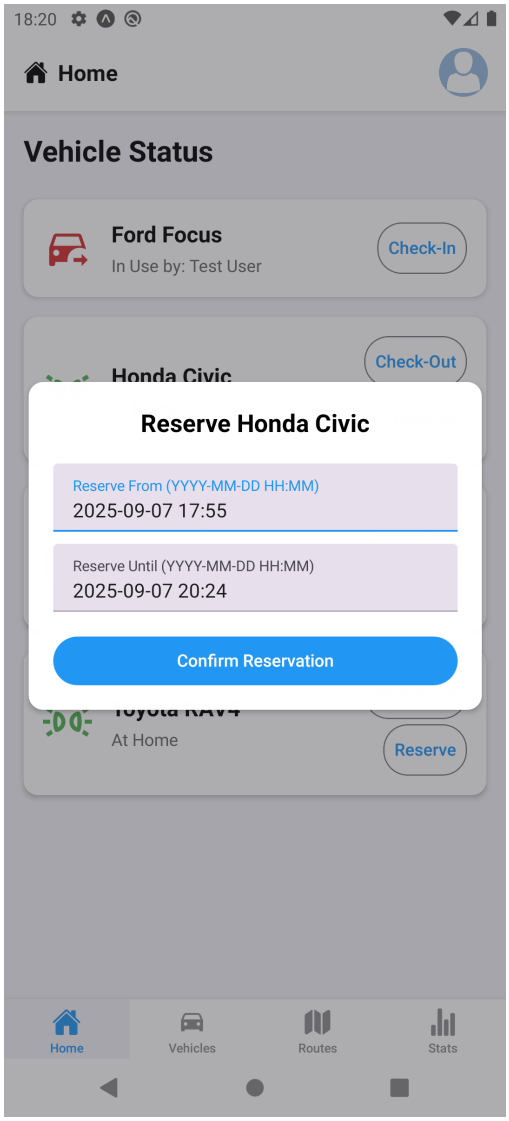


Figure 6.6: The reservation modal, allowing a user to book an available vehicle for a specific future time slot.

6.1.4 Proof-of-concept for recommendation

The validation of the system's core hypothesis—that an intelligent engine can guide sustainable choices—is demonstrated through the "Routes" workflow. As established in the project's scope, this implementation serves as a vertical slice proof-of-concept for the Pillar 2 recommendation engine.

The user first selects a trip from a set of predefined locations, one of which is inside

a hardcoded ZBE, as shown in Figure 6.7. The application then sends this information to the backend, which processes it against the ZBE rules. The result, displayed in Figure 6.8, is a ranked list of available vehicles. In this example, the destination is inside the ZBE, so the system has automatically filtered out non-compliant vehicles and presents only the eligible, hybrid car as the "Best Choice," thus validating the core constraint-based filtering logic.

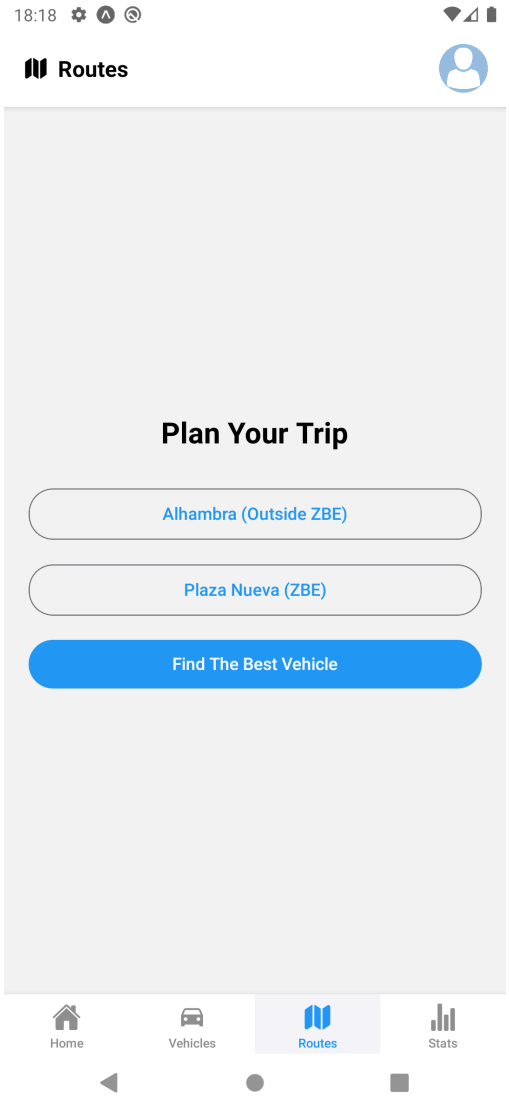


Figure 6.7: The route planning interface, where the user selects a trip from a list of predefined demonstration locations.

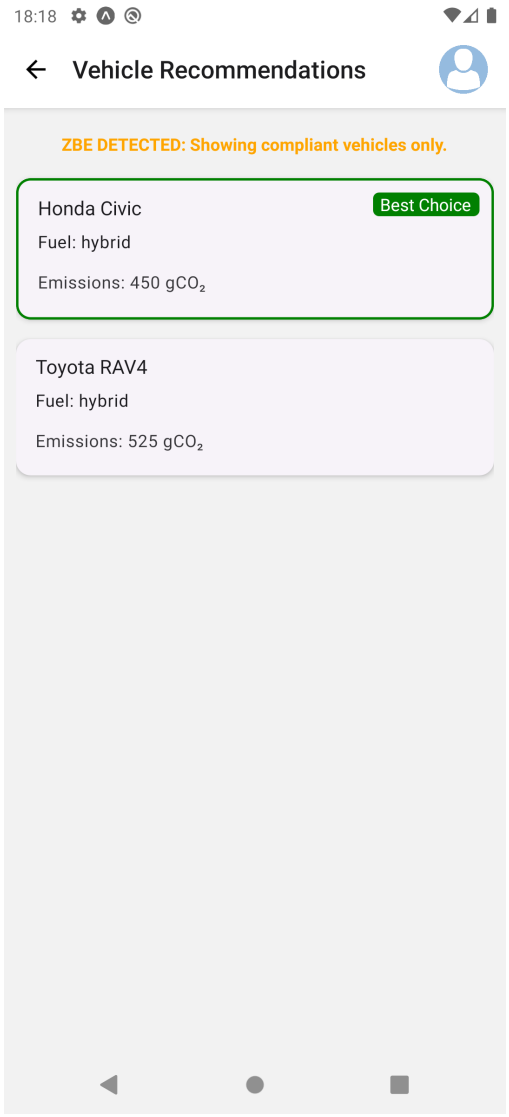


Figure 6.8: The recommendation results screen. A "ZBE DETECTED" warning is displayed, and the list of vehicles is filtered accordingly.

6.1.5 Statistics and gamification

Finally, the "Stats" tab validates the foundational elements of Pillar 4. This screen, shown in Figure 6.9, aggregates data from the user's logged trips to provide key performance indicators, such as total distance traveled and total CO_2 emissions. It also displays the user's earned achievements, confirming that the backend logic for tracking user actions and granting awards is functioning correctly. This serves as the data collection and feedback mechanism upon which the full analytics dashboard and col-

laborative gamification of the Zenith release would be built.

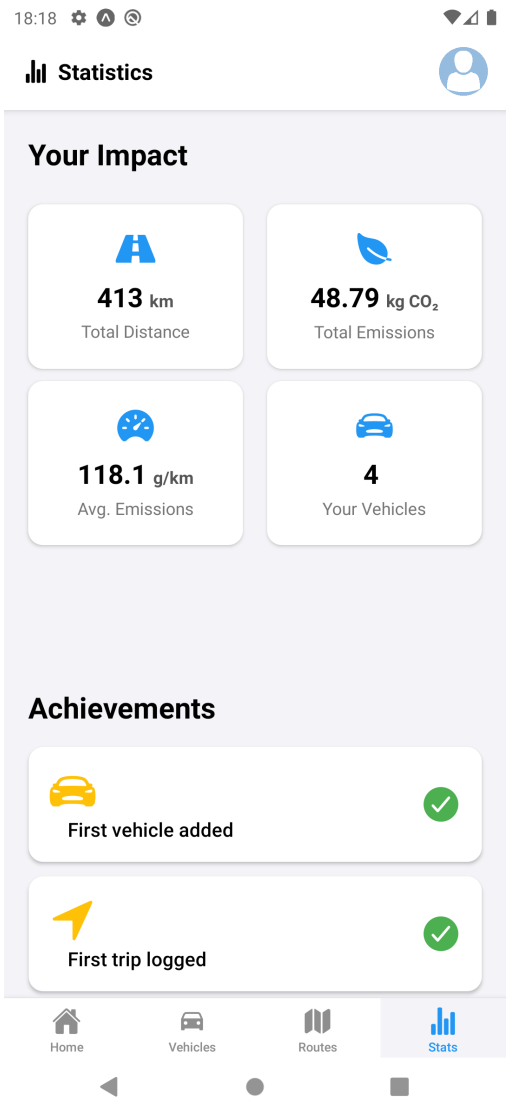


Figure 6.9: The statistics and achievements screen, providing the user with tangible feedback on their activity and progress.

6.2 Backend and End-to-End validation

To validate the system's logic and the integrity of the integration between the frontend and backend, a suite of automated tests was executed.

6.2.1 Backend integration testing

The backend test suite was run using Jest and Supertest to verify the correctness of the API endpoints in a controlled environment. The tests cover critical business logic, including authentication, household management, and the core recommendation algorithm. Figure 6.10 shows the output from the test runner, confirming that all implemented tests passed successfully. This result validates that the server-side logic is robust and performs as expected according to the system design.

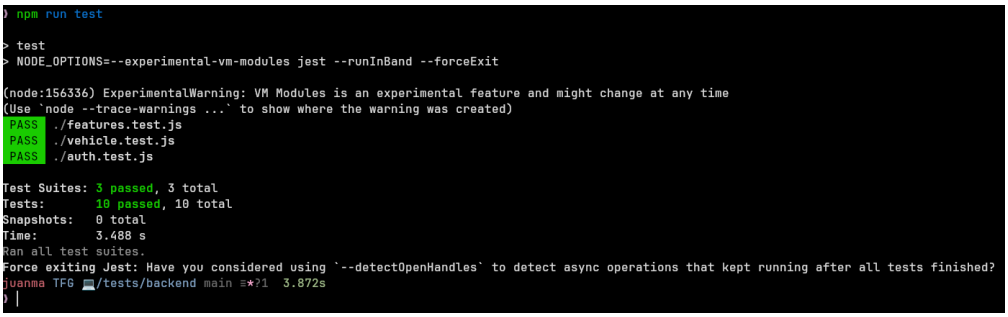


Figure 6.10: Output from the Jest test runner, confirming the successful execution of the backend integration test suite.

6.2.2 End-to-End (E2E) testing

To validate the complete user journey from the user interface to the database and back, E2E tests were conducted using the Playwright framework. These tests automate a web browser to perform critical user flows such as registration, login, and adding a vehicle. Figure 6.11 displays the summary report from the Playwright test execution, indicating that all critical-path user journeys completed without errors. This validates the seamless integration between the frontend and backend components.

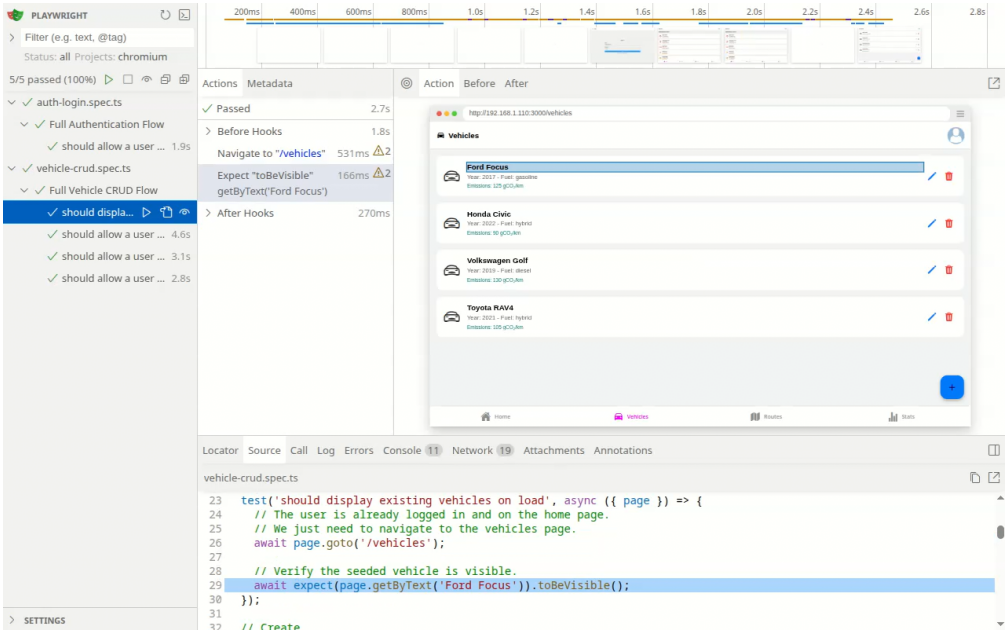


Figure 6.11: Summary report from the Playwright E2E test suite, showing that all critical user flows passed successfully.

6.3 Validation summary

The combination of visual confirmation from the frontend walkthroughs and the successful execution of the automated test suites confirms that the AIDiaCAR prototype successfully implements its core objectives. The system effectively provides a collaborative framework for household vehicle management and validates the technical feasibility of its innovative recommendation engine. Table 6.1 maps the core project objectives to their implemented and validated status.

Table 6.1: Summary of validated project objectives

Objective	**Validation Method**	**Status**
Mitigate coordination friction	Frontend screenshots (Status dashboard, reservation modal), E2E tests (Playwright)	Implemented
Combat maintenance blindness	Frontend screenshots (Maintenance modal), Backend tests (Jest)	Implemented
Address inefficient selection	Frontend screenshots (Recommendation screen), Backend tests (Jest)	Implemented
Establish collaborative framework	Frontend screenshots (Household card), Backend tests (Jest), E2E tests	Implemented
Gamification foundation	Frontend screenshot (Stats screen), Backend tests (Jest)	Implemented

7 | Project management and planning

This chapter details the project management framework, planning methodology, and execution strategy employed throughout the development of the *AIDiaCAR* thesis project. It outlines the processes used for task identification, scheduling, and resource management, and provides a transparent account of the strategic decisions made to navigate the challenges inherent in a research-oriented software development endeavor.

7.1 Methodology

The project was executed following an exploratory and iterative methodology. This approach was deliberately chosen over a more traditional, rigid waterfall model. A waterfall methodology, with its sequential and inflexible phases, was deemed fundamentally unsuitable for this project, where the process of writing the thesis and developing the software artifact were deeply intertwined. Requirements and architectural insights often emerged during the implementation process itself, necessitating a flexible approach that allowed for continuous refinement.

This iterative methodology provided the necessary adaptability. By developing the system in progressive cycles, it was possible to build and validate components incrementally, with the writing of the thesis and the coding of the application informing each other. This progressive development and validation strategy was crucial for ensuring that the final research artifact was both robust and directly aligned with the project's central academic hypotheses, even as the understanding of the problem space deepened over time.

7.2 Project phases and task organization

To structure the project and ensure comprehensive coverage of all necessary activities, the work was logically deconstructed into five primary phases. This approach, anal-

ogous to a Work Breakdown Structure (WBS) [1], provided a clear sequence for the project's execution, grouping activities from initial research to final documentation.

- **Phase 1: Research and state-of-the-art analysis.** This initial and foundational phase involved a comprehensive review of academic literature and existing market solutions. It included a detailed analysis of competing applications, research into the theoretical foundations of persuasive technology and Green IT, and the formal identification of the market gap that this thesis aims to address. The outputs of this work are detailed in Chapter 2.

- **Phase 2: System design and architecture.** This phase encompassed all high-level planning and design activities. Responsibilities included the definition of the client-server architecture, the selection of the technology stack, the formal modeling of the application's data structures, and the conceptual design of the user interface and user experience (UI/UX)[2] through the creation of wireframes and a visual style guide. This work corresponds to the material presented in Chapters 3 and 4.

- **Phase 3: Proof-of-concept implementation.** This phase represented the core software development effort. The primary goal was the implementation of a focused prototype to validate the most novel and complex aspects of the system. This included building the backend API, the frontend mobile application, and the necessary database schemas. The technical details of this implementation are documented in Chapter 5.

- **Phase 4: Testing and validation.** This phase focused on ensuring the quality, correctness, and robustness of the software artifact. It included the implementation of automated tests for backend modules and comprehensive manual validation of the core user workflows to ensure they functioned as designed and met the project's requirements. The results of this work are presented in Chapter 6.

- **Phase 5: Documentation and thesis writing.** This was a continuous, overarching activity that ran in parallel with all other phases. It involved the diligent and ongoing process of documenting all research findings, architectural decisions, implementation details, and validation results, culminating in the production of this final thesis report.

[1] A Guide to Work Breakdown Structure (WBS)
[2] What is UX / UI Design?

7.3 Project timeline and execution

The development and research cycle for this project concluded in September 2024. Given the exploratory nature of the work, where research and implementation were concurrent activities, a formal, detailed Gantt chart with rigid deadlines was not employed. Such a tool would have imposed an artificial linearity on a process that was, by necessity, non-linear and required flexibility.

Instead, the execution strictly followed the logical sequence of the major project phases. This ensured a structured and methodical progression. A significant emphasis was placed on the initial phases of research, design, and architecture. This front-loading of planning activities was a deliberate risk mitigation strategy. By ensuring that a robust theoretical and technical foundation was established before any significant code was written, the project minimized the risk of costly refactoring or architectural changes during the later implementation phase.

7.4 Scope management and proof-of-concept focus

The primary goal of a software engineering thesis is not to deliver a feature-complete commercial product, but to design, build, and validate a proof-of-concept (PoC) that effectively demonstrates a novel solution to a well-defined problem. In line with this academic objective, the scope of the implementation was strategically focused on creating a "vertical slice" that validates the most innovative component of the AIDiaCAR architecture: the recommendation engine.

Rather than implementing a wide array of standard CRUD (Create, Read, Update, Delete) features superficially, the development effort was concentrated on proving the viability of the core hypothesis. The implemented PoC successfully demonstrates the system's ability to take a user's destination, check it against a geographically defined rule (in this case, a simplified ZBE boundary), and filter the user's available vehicles based on this constraint. The use of hardcoded points for demonstration purposes allows for a controlled, repeatable, and clear validation of this core logical flow, from the user interface through the backend API to the database and back.

This focused approach, while a simplification of the full Zenith vision, is a deliberate and strategic choice. It provides a robust and tangible validation of the system's most complex and innovative concept. By proving that the architecture can support this core functionality, the project successfully meets its primary research objective and establishes a strong foundation upon which the full feature set can be built in future

work.

7.5 Resource management

The resources required for the completion of this project were primarily managed in terms of time and technology.

The single most critical resource was developer time. As a solo academic project, all work phases, from research to implementation and documentation, were executed by the author. The project scope and implementation goals were defined and managed with this significant constraint in mind.

No significant financial budget was required for the development of the AIDiaCAR prototype. This economic efficiency was achieved by exclusively adopting a technology stack based on free and open-source software (FOSS). The use of technologies such as React Native, Node.js, Express.js, MongoDB Community Edition, and Docker allowed for the development of a complete, full-stack application without any licensing fees or software acquisition costs. Furthermore, deployment and infrastructure costs were avoided by utilizing a local, containerized development environment for building and testing the prototype. This is a standard and pragmatic approach for an academic proof-of-concept, as it avoids the unnecessary costs and operational complexities associated with deploying to live, cloud-hosted infrastructure, which was outside the scope of this research.

8 | Conclusion and future work

This final chapter synthesizes the outcomes of the research and development efforts undertaken for this thesis. It begins by presenting a concise conclusion, summarizing the core problem addressed, the proposed solution, and the principal contributions of the project. Following this, the chapter outlines a comprehensive roadmap for future work, detailing the logical steps required to evolve the current research prototype into a feature-complete, production-ready system.

8.1 Conclusion

The management of personal vehicles in the modern, multi-car household is fraught with inefficiencies that lead to increased costs, environmental impact, and daily domestic friction. As established in the problem statement, a typical household, exemplified by the Martínez family *persona*, consistently grapples with significant challenges related to coordination friction, shared maintenance blindness, and suboptimal, constraint-unaware vehicle selection. This ad-hoc management of a valuable and environmentally significant set of assets represents a critical and underserved technological niche.

In response to these challenges, this thesis proposed, designed, and architected a novel software solution, *AIDiaCAR*, conceived as an intelligent, collaborative "Digital garage". The system is structured upon a four-pillar framework designed to holistically address the identified pain points. The intellectual centerpiece of this proposed solution is the Pillar 2 "Proactive co-pilot", a conceptual model for a multi-factor recommendation engine that can serve as a persuasive technology to guide users toward more sustainable and economically sound mobility choices.

The primary contribution of this project is a comprehensive and defensible architectural blueprint for a holistic, personal fleet management system. The viability of this architecture's most innovative component—the constraint-based recommendation logic—was successfully validated through the implementation of a focused, 'vertical slice' proof-of-concept. This prototype effectively demonstrated the architectural pathway for a ZBE-aware recommendation, proving the central hypothesis that such a

system is technically feasible and establishing a robust foundation upon which future development can confidently build.

8.2 Future work

The research and development conducted for this thesis have culminated in a successful proof-of-concept that validates the core architectural principles of the *AIDiaCAR* system. The following sections outline a strategic roadmap for evolving this prototype from its current state into the feature-complete Zenith release envisioned in Chapter 3.

8.2.1 Full implementation of the four pillars

The next logical phase of development would focus on building out the full feature set of the remaining pillars to create a truly collaborative and engaging user experience, building upon the foundational elements already in place.

- **Pillar 1 (Coordination):** Future work would focus on enhancing the existing collaborative framework. This includes implementing a seamless user invitation system for the Household entity and building out the full vehicle reservation system with a polished user interface and potential integration with third-party calendar applications (e.g., Google Calendar, Apple Calendar).

- **Pillar 3 (Maintenance):** To more effectively combat "shared maintenance blindness," a shared, collaborative **"Household task list"** would be developed. This would allow any household member to view, take ownership of, and mark as complete any upcoming maintenance item. This feature would be enhanced through integration with mapping services to automatically suggest nearby approved service centers or ITV stations when an alert is triggered.

- **Pillar 4 (Gamification & analytics):** The full **"Household impact dashboard"** would be created, featuring clear, graphical charts and data visualizations for tracking collective transportation costs, total CO_2 emissions, and a composite "Eco-score." The gamification layer would be expanded to include **collaborative household goals** (e.g., "Reduce our collective emissions by 10% this month"), transforming sustainability into a cooperative and rewarding team effort.

8.2.2 Enhancements to the recommendation engine

With the core logical flow validated by the proof-of-concept, the recommendation engine would be evolved into the full, multi-factor model described in the system definition.

- **Integration of live data APIs:** A critical step towards production viability is replacing the hardcoded demonstration data with real-time information from third-party services. This would require integration with:
  - A national or regional **ZBE data service** to provide up-to-the-minute information on complex low-emission zone boundaries and access rules.
  - One or more **local fuel price data services** to ensure that economic calculations are based on current market rates.
  - **Real-time traffic data services** (e.g., Google Maps Directions API) to provide accurate estimations of journey distance and time, which would improve the precision of cost and emissions calculations.

- **Inclusion of maintenance status:** The engine's algorithm would be enhanced to query the status of Pillar 3. It will be programmed to check for upcoming critical maintenance and automatically flag vehicles as "Not Recommended" for long trips if, for example, a tire change or mandatory inspection is imminent.

- **Total Cost of Ownership (TCO) model:** For an even more holistic financial recommendation, a more advanced version of the engine could be developed to incorporate factors beyond immediate operational costs. By factoring in long-term expenses such as scheduled maintenance, vehicle depreciation rates, and insurance premiums, the system could provide a "Total Cost of Ownership" recommendation, offering users a much deeper insight into the true financial implications of their vehicle choices.

8.2.3 Technical and architectural evolution

To support the transition from a research prototype to a scalable, production-grade application, several key technical and architectural advancements would be required.

- **Backend refactoring to a service layer:** As the complexity of the business logic grows with the full implementation of all pillars, the backend would benefit from being refactored. The current controller-centric logic would be migrated

to a dedicated service layer[1]. This would encapsulate and abstract business logic, improving modularity, promoting code reuse, and significantly enhancing the testability of the system as it scales.

- **Production deployment strategy:** A robust deployment strategy would be implemented. This would involve containerizing the Node.js application for deployment to a Platform-as-a-Service (PaaS)[2] like Heroku[3] or a container orchestration platform like Kubernetes[4]. The database would be migrated to a managed, auto-scaling service such as MongoDB Atlas[5] to ensure high availability and data integrity.

- **CI/CD pipeline implementation:** To streamline development and ensure high quality, a Continuous Integration/Continuous Deployment (CI/CD) pipeline would be established (e.g., using GitHub Actions[6] or Jenkins[7]). This pipeline would automate the process of running unit and integration tests on every code commit and would automate the deployment of successful builds to a staging and then production environment, improving the reliability and velocity of the development cycle.

[1] Service layer pattern
[2] What is platform as a service (PaaS)?
[3] Heroku: The AI PaaS For Deploying, Managing, and Scaling Apps
[4] Production-Grade Container Orchestration
[5] MongoDB Atlas
[6] GitHub Actions
[7] Jenkins

Bibliography

[1] Hannah Ritchie. Sector by sector: where do global greenhouse gas emissions come from? *Our World in Data*, 2020. https://ourworldindata.org/ghg-emissions-by-sector.

[2] Eurostat. Passenger cars per 1 000 inhabitants reached 560 in 2022. 2024. https://ec.europa.eu/eurostat/web/products-eurostat-news/w/ddn-20240117-1.

[3] International Energy Agency. Fuel economy in major car markets: Technology and policy drivers 2005-2019. Technical report, IEA, 2021. https://www.iea.org/reports/fuel-economy-in-major-car-markets.

[4] B.J. Fogg. *Persuasive Technology: Using Computers to Change What We Think and Do*. Morgan Kaufmann, 2002.

[5] Sebastian Deterding, Dan Dixon, Rilla Khaled, and Lennart Nacke. Gamification: Toward a definition. *CHI 2011 Workshop*, 2011.

[6] Juho Hamari, Jonna Koivisto, and Harri Sarsa. Does gamification work?–a literature review of empirical studies on gamification. *HICSS*, 2014.

[7] Frans Berkhout and Julia Hertin. De-materialising and re-materialising: digital technologies and the environment. *Futures*, 36(8):903–920, 2004.

[8] Andrew S. Tanenbaum and David J. Wetherall. *Computer Networks*. Prentice Hall, 5th edition, 2011.

[9] Robert C. Martin. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Prentice Hall, 2017.

[10] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1995.

[11] Ankita Garg, Ayushi Gupta, and Anjali. Comparing the performance of mvc, mvp and mvvm pattern. In *2013 1st International Conference on Emerging Trends in Engineering and Technology*, pages 106–109, 2013.

[12] Houssein Djirdeh, Anthony Accomazzo, Sophia Shoemaker, and Devin Abbott. *Fullstack React Native: The complete guide to React Native*. Fullstack.io, 2018.

[13] Pramod J. Sadalage and Martin Fowler. *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. Addison-Wesley Professional, 2012.

[14] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000. https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.

[15] Jez Humble and David Farley. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley Professional, 2010.

A | API Endpoint Specification

This appendix provides a comprehensive specification for the RESTful API that powers the *AIDiaCAR* backend. The API is designed following REST principles, utilizing standard HTTP methods and status codes. All data is exchanged in the JSON format. Access to sensitive endpoints is controlled via a JWT-based authentication system, as detailed in Chapter 4.

All specified routes in the following table are prefixed with `/api`. For example, the route listed as `/auth/register` corresponds to the full endpoint `/api/auth/register`.

Table A.1: AIDiaCAR API Endpoint Definitions

Method	Route	Description	Protected
Authentication and User Management			
POST	`/auth/register`	Registers a new user and creates a personal household for them.	No
POST	`/auth/login`	Authenticates a user and returns a JWT in an HttpOnly cookie.	No
POST	`/auth/logout`	Clears the authentication cookie, logging the user out.	No
GET	`/users/profile`	Retrieves the profile information for the currently authenticated user.	Yes
PUT	`/users/profile/avatar`	Updates the authenticated user's profile picture. Expects multipart/form-data.	Yes
Household Management			
GET	`/households/my-household`	Retrieves the details of the household the current user belongs to, including a list of members.	Yes

Continued on next page

Table A.1 – continued from previous page

Method	Route	Description	Protected
POST	/households/join	Allows the current user to join an existing household using a join code.	Yes
POST	/households/leave	Allows the current user to leave their current household. A new personal household is created for them.	Yes
Vehicle Fleet Management (CRUD)			
POST	/vehicles	Creates a new vehicle and associates it with the authenticated user as the owner.	Yes
GET	/vehicles	Retrieves a list of all vehicles owned by any member of the user's current household.	Yes
GET	/vehicles/:id	Retrieves the details of a single vehicle by its ID.	Yes
PUT	/vehicles/:id	Updates the details of a specific vehicle. Can be used for general info or for updating maintenance schedules.	Yes
DELETE	/vehicles/:id	Deletes a vehicle from the system.	Yes
Vehicle Status and Coordination			
POST	/vehicles/:id/checkout	Marks a vehicle as 'in_use' by the authenticated user.	Yes
POST	/vehicles/:id/checkin	Marks a vehicle as 'at_home', making it available again.	Yes
POST	/vehicles/:id/reserve	Reserves a vehicle for a specified future time slot. Requires 'reservedFrom' and 'reservedUntil' in the body.	Yes
POST	/vehicles/:id/cancel-reservation	Cancels an existing reservation made by the authenticated user.	Yes
Core Application Features			
GET	/maintenance/summary	Retrieves a consolidated and sorted list of all upcoming and overdue maintenance tasks for the user's household fleet.	Yes

Continued on next page

Table A.1 – continued from previous page

Method	Route	Description	Protected
POST	`/recommendations`	The core recommendation engine. Takes trip details (‘origin‘, ‘destination‘, ‘distance‘) and returns a sorted list of recommended vehicles.	Yes
GET	`/stats`	Retrieves aggregated statistics for the authenticated user, such as total distance traveled and total emissions.	Yes
POST	`/trips/log`	Logs the completion of a trip, updating vehicle maintenance counters and user statistics.	Yes
Development and Testing			
POST	`/init-db`	**(Development Only)** Clears and seeds the database with a predefined set of test users, vehicles, and trips.	No

B | Mongoose Data Schemas

This appendix provides the complete source code for the Mongoose schemas that define the data architecture of the *AIDiaCAR* application. These schemas enforce data structure and validation at the application layer before data is persisted to the MongoDB database.

B.1 Household Schema

The `Household` schema is the central entity for enabling collaborative features. It groups users together and uses the `nanoid` library to generate a unique, human-readable code that users can share to invite others.

Listing B.1: Source code for `models/Household.js`

```
import mongoose from 'mongoose';
import { customAlphabet } from 'nanoid';

// Using a dictionary-based alphabet for more memorable
   codes
const alphabet = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789';
const nanoid = customAlphabet(alphabet, 8); // Generates an
    8-character code like 'A3B5D9F1'

const HouseholdSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true,
  },
  joinCode: {
    type: String,
    required: true,
    unique: true,
    default: () => nanoid(),
```

```
  },
  owner: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true,
  },
  members: [{
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User'
  }],
}, { timestamps: true });

export default mongoose.model('Household', HouseholdSchema)
  ;
```

B.2 User Schema

The `User` schema defines the structure for user accounts. It includes a reference to the `Household` the user belongs to. A key feature is the `pre('save')` middleware hook, which automatically hashes the user's password using `bcryptjs` before it is saved to the database, ensuring that plain-text passwords are never stored.

Listing B.2: Source code for `models/User.js`

```
import mongoose from 'mongoose';
import bcrypt from 'bcryptjs';

const UserSchema = new mongoose.Schema(
  {
    name: {
      type: String,
      required: true,
      trim: true,
    },
    email: {
      type: String,
      required: true,
      unique: true,
      lowercase: true,
      trim: true,
    },
```

```
    password: {
      type: String,
      required: true,
      select: false, // Never return password in queries
    },
    household: {
      type: mongoose.Schema.Types.ObjectId,
      ref: 'Household',
    },
    avatar: {
      type: String,
      default: 'images/generic-avatar.png',
    },
    stats: {
      distanceTraveled: { type: Number, default: 0 },
      co2Saved: { type: Number, default: 0 },
      totalVehicles: { type: Number, default: 0 },
    },
    // A simple array to store unique achievement keys
    achievements: {
      type: [String],
      default: [],
    },
  },
  {
    timestamps: true,
  }
);

// Password hashing middleware
UserSchema.pre('save', async function (next) {
  if (!this.isModified('password')) return next();

  const salt = await bcrypt.genSalt(10);
  this.password = await bcrypt.hash(this.password, salt);
  next();
});

// Password verification method
```

```
UserSchema.methods.matchPassword = async function (
    enteredPassword) {
  return await bcrypt.compare(enteredPassword, this.
      password);
};

export default mongoose.model('User', UserSchema);
```

B.3 Vehicle Schema

The `Vehicle` schema is a complex model that defines a vehicle's core attributes, its real-time status, and its maintenance schedule. The `status` field is a sub-document that tracks whether the vehicle is available, in use, or reserved. The `upcomingMaintenance` field is an embedded sub-document that defines default and user-configurable service intervals, which are crucial for the maintenance alert system.

Listing B.3: Source code for `models/Vehicle.js`

```
import mongoose from 'mongoose';

// Default generic maintenance intervals
const maintenanceSchema = new mongoose.Schema({
  tires: {
    date: {
      type: Date,
      default: () => new Date(new Date().setFullYear(new
          Date().getFullYear() + 1)), // One year from now
    },
    distance: {
      type: Number,
      default: 40000,
    },
  },
  brakes: {
    distance: {
      type: Number,
      default: 50000,
    },
  },
  oilChange: {
    distance: {
```

```
      type: Number,
      default: 30000, // Default to 30,000 km for synthetic
          oil (moden cars)
    },
  },
  itv: {
    type: Date,
    default: () => new Date(new Date().setFullYear(new Date
       ().getFullYear() + 2)), // Two years from now
  },
});

const VehicleSchema = new mongoose.Schema({
  make: { type: String, required: true },
  model: { type: String, required: true },
  year: { type: Number, required: true },
  fuelType: { type: String, required: true, enum: ['
     gasoline', 'diesel', 'electric', 'hybrid'] },
  owner: { type: mongoose.Schema.Types.ObjectId, ref: 'User
     ', required: true },
  status: {
    state: {
      type: String,
      enum: ['at_home', 'in_use', 'reserved'],
      default: 'at_home',
    },
    checkedOutBy: {
      type: mongoose.Schema.Types.ObjectId,
      ref: 'User',
      default: null,
    },
    reservedBy: {
      type: mongoose.Schema.Types.ObjectId,
      ref: 'User',
      default: null,
    },
    reservedFrom: {
      type: Date,
      default: null,
    },
```

```
    reservedUntil: {
      type: Date,
      default: null,
    },
    lastUpdated: {
      type: Date,
      default: Date.now,
    },
  },
  emissions: Number,
  upcomingMaintenance: maintenanceSchema,
});

export default mongoose.model('Vehicle', VehicleSchema);
```

B.4 Trip Schema

The `Trip` schema is used to log each journey taken by a user. It stores references to the driver and the vehicle used, as well as key data about the trip such as distance and calculated emissions. This collection provides the raw data that feeds the statistics and gamification features of the application.

Listing B.4: Source code for `models/Trip.js`

```
import mongoose from 'mongoose';

const TripSchema = new mongoose.Schema({
  vehicle: { type: mongoose.Schema.Types.ObjectId, ref: '
    Vehicle', required: true },
  driver: { type: mongoose.Schema.Types.ObjectId, ref: '
    User', required: true },

  locations: {
    start: {
      type: {
        type: String,
        enum: ['Point'],
        required: true,
      },
      coordinates: {
        type: [Number],
```

```
        required: true,
      },
    },
    end: {
      type: {
        type: String,
        enum: ['Point'],
        required: true,
      },
      coordinates: {
        type: [Number],
        required: true,
      },
    },
  },

  distance: { type: Number, required: true }, // in km
  date: { type: Date, default: Date.now },
  calculatedEmissions: { type: Number, required: true }, //
      in grams of CO2
});

export default mongoose.model('Trip', TripSchema);
```