ChatGPT

Spotio Field Sales Platform Analysis and Yardura PRD

Spotio Field Sales App: Core Functionalities

Spotio Overview: Spotio is a leading field sales engagement platform designed to help door-to-door and outside sales teams manage territories, optimize their routes, capture sales activities, and track performance in real time [1] [2]. Available on iOS, Android, and web, Spotio provides an integrated solution for reps in the field and managers at headquarters. Below we outline Spotio's core and advanced features:

- **Lead & Prospect Management:** Spotio enables reps to efficiently manage leads and prospects. Reps can add new leads, import prospect lists, and view rich data on potential customers. The platform even provides **"Lead Machine"** data for residential prospects – e.g. whether a home is owner-occupied or rented, homeowner age, and even credit capacity – helping reps qualify leads and focus on high-potential targets [3] [4]. Spotio's **digital business cards** feature allows sharing contact info with prospects seamlessly [5].

- **Sales Territory Mapping:** A hallmark of Spotio is its robust territory management tools. Managers can define sales territories by drawing custom boundaries on a map, selecting areas by city/ZIP code, or stacking multiple ZIP codes [6] [7]. Territories are color-coded and can be assigned to reps to eliminate overlap and ensure full coverage [8]. Spotio even supports territory hierarchies (large areas composed of smaller territories) for organizations that need multi-level territory oversight [9]. These mapping tools prevent two reps from unknowingly canvassing the same area and help identify **gaps or overlaps** in coverage [10]. (Notably, reps are not prevented from logging work outside their area, but they receive a notification reminder if they do [11].)

- **Customer Mapping & Colorization:** Spotio gives teams a visual map of all leads and customers. Reps can drop **pins** on the map for each door knock or customer interaction, and the map updates with outcomes (e.g. sales made) in real time [2]. A **colorization** feature lets users color-code map pins by status or data field – for example, by prospect stage, last visit result, or any custom field [12]. This means at a glance, reps and managers can see which homes showed interest, which need follow-up, and which areas are most promising [12]. Colorization greatly enhances the map's usefulness in planning and review.

- **Route Planning & Optimization:** Spotio's built-in **route planner** helps field reps minimize driving time and increase selling time. Reps can create optimized multi-stop routes by selecting a set of addresses (manually, via lasso tool on the map, or by filtering a list) [13] [14]. The system then computes the fastest or most efficient route, taking into account real-time traffic data. Spotio even allows **recurring routes** for reps who canvass the same area regularly [15]. Mileage is automatically tracked for each trip, eliminating manual expense reports [16]. Routes can be planned on the web app (e.g. by a manager or the rep from a desktop) and then executed on the mobile app, thanks to

hybrid planning support [17]. Conversely, routes created on mobile (including ad-hoc routes for that day) sync back to the web for visibility [18]. Spotio's routing tool is flexible: users can add stops on the fly, reorder stops, set start/end locations, and even choose to avoid tolls or highways when optimizing [19]. Completed routes are saved as **"Trips"** for reporting, giving managers insight into where reps went and when [20]. Overall, route optimization is a flagship capability of Spotio, aimed at reducing "windshield time" and increasing face-to-face time with customers [21].

- **Geolocation Tracking & GPS Breadcrumbs:** To enhance accountability and insight, Spotio offers continuous geolocation tracking of reps (with privacy controls). Managers can **"find my team"** on the map with a single click, seeing each rep's last known location [22]. As reps move and log visits, the system can display **breadcrumbs** – the path of travel – to verify routes taken [23]. Each logged visit is **GPS-verified**, so managers know a pin drop truly represents an in-person visit at that address [23]. If desired, this live tracking can be turned off for privacy, but when enabled it provides real-time visibility into field operations [22]. Reps' door knocks or check-ins are time-stamped and geotagged, building a trustworthy activity trail [2]. This is especially useful for verifying that territories are being covered methodically.

- **Activity Logging & Multi-Channel Communication:** Spotio automates the capture of sales activities across multiple channels, reducing administrative burden on reps. Every in-person **visit** (door knock or meeting) can be logged with one tap, and Spotio also tracks **calls, emails, and texts** with prospects or customers [24]. Through integrations (e.g. with phone and email providers), these communications are logged **100% automatically** – reps don't need to manually input that they made a call or sent an email [24] [25]. This seamless multi-channel tracking ensures the CRM or system of record is always up to date with engagement data, without distracting reps from selling [26]. Reps can even send emails or texts from within the app, using templates if needed, and have those interactions recorded. The benefit is **complete data**: managers get a full picture of how each lead is being worked (not just the first door knock) [27]. In fact, Spotio emphasizes that it **"tracks ALL sales activity – both in-person and virtual"**, whereas some competitors only log the initial visit and miss follow-ups via phone or email [28]. By capturing every touchpoint automatically, Spotio helps ensure no lead falls through the cracks due to poor logging.

- **Sales Pipeline & Opportunity Tracking:** Beyond just mapping and visits, Spotio includes a **pipeline management** interface. This pipeline view allows reps and managers to track deals as they move through stages (e.g. Contacted, Quote Given, Closed-Won, etc.) [29]. For each territory or rep, Spotio can display how many opportunities are in each stage and the total potential value in the pipeline [30]. Managers can filter the pipeline by rep, territory, date, stage, and even custom fields to zero in on particular segments [31] [32]. This gives leadership a clear view of where deals stand and who might need coaching. Spotio's pipeline is essentially a **visual sales funnel** (similar to a CRM) embedded in the field app [1]. Notably, Spotio supports **custom stages** and fields – the pipeline view appears only for record types configured with stages, which can be customized to match the company's sales process [33]. By having a live pipeline, outside sales teams can forecast more accurately and identify bottlenecks (e.g. lots of deals stuck in a mid-stage). MapMyCustomers' review highlighted Spotio's **"visual pipeline building and tracking"** as a major capability for sales managers [1]. Reps themselves benefit by staying organized on who to follow up with next.

- **Task Automation & Autoplays:** Spotio boosts rep productivity through automation of repetitive tasks. **Auto-logging** of activities (as mentioned) is one form of task automation, eliminating time

wasted on data entry [34]. Spotio also provides templates for common tasks – for example, email templates or text templates – so reps can fire off communications quickly with consistent messaging [35]. One of Spotio's standout advanced features is **Autoplays**, which are essentially automated sequences or cadences of follow-up activities [36]. A sales manager can define an Autoplay sequence for a new lead (e.g. Day 1: initial door knock, Day 3: follow-up text, Day 5: call, Week 2: second visit) – and the app will guide the rep through that sequence for each lead [37]. Autoplays ensure that every prospect receives a complete series of touchpoints and **"never falls through the cracks"**, even if an individual rep gets busy [36]. Reps are shown exactly who to engage, when, and how, based on the predefined successful cadence [38]. This not only increases conversion rates by maintaining consistent follow-up, but also helps **onboard new reps faster** by giving them a proven playbook to follow [39]. In essence, Autoplays bring some of the benefits of inside sales cadence tools to field sales. (Competitors like SalesRabbit do *not* offer an equivalent to Autoplay sequences, which Spotio touts as a differentiator [40] [41].)

- **Calendar & Appointment Management:** Spotio integrates with reps' calendars to help schedule and manage appointments with prospects. Reps can **create appointments** in Spotio (e.g. schedule a demo or follow-up meeting), which are then synced with their Google or Outlook calendar [42]. Similarly, existing calendar events can be pulled into Spotio for visibility. All meeting notes and prospect details are linked, so a rep can see context for each appointment right in the app [43]. The two-way **calendar integration** ensures that a rep's schedule in Spotio is always up-to-date [44] [45]. Managers can also view team calendars if needed to coordinate ride-alongs or assign leads. By handling appointments in-app, Spotio keeps the rep's workflow consolidated (no need to jump out to a separate calendar app to schedule something) and ensures meetings with customers are logged as part of the sales process. Calendar integration is facilitated via linking the user's email account (e.g. through Nylas API), allowing **sync of both appointments and emails** for a complete picture [46] [44].

- **Team Management & Performance Insights:** Spotio provides managers with tools to manage their team structure and monitor performance. **User roles and permissions** can be configured – e.g. Sales Reps have access only to their own records or territories, while Managers and Admins have broader access. Territories can be assigned to reps, and if a rep is limited to their territory, Spotio will even alert them if they log activity outside of it [47] [11]. Management can also organize reps into **teams**, assign team leaders, and more (useful for larger orgs or channel sales) [48] [47].

  Beyond structure, Spotio emphasizes **real-time performance tracking and coaching**. Managers receive email notifications highlighting team and individual progress toward targets [49]. For example, if a rep is falling behind on activity or quota, Spotio can notify the manager so they can intervene early [50]. Reps themselves can get automated alerts on their status, helping them self-correct before end of month [51]. Spotio includes **leaderboards** to tap into the team's competitive spirit [52] – reps can see how they rank on key metrics like knocks, meetings, or sales, which often motivates extra effort. MapMyCustomers noted that Spotio provides **visualization of sales activities and performance in real time, accessible to everyone on the team** [53]. This transparency keeps the team aligned and accountable.

- **Reporting & Analytics:** With so much data captured, Spotio offers robust reporting tools for analysis. The platform's **"My Reports"** feature lets users (especially managers or ops roles) build custom reports using any metrics and KPIs that matter to them [54]. They can apply filters, group

data, and then save these reports for repeated use [55]. Examples might include a report on conversion rate by territory, or activity count by rep per week, etc. The reports update in real-time and are accessible on web or mobile [56]. Spotio also likely provides standard dashboard views – e.g. a manager's dashboard of team performance, pipeline health, etc. However, some reviews suggest that Spotio's analytics are a strong point relative to simpler competitors: for instance, Spotio can generate conversion reports, visit report summaries, and territory performance breakdowns, which competing apps often lack [57] [58]. Overall, these analytics give management the **"end-to-end visibility"** needed to identify what's working and scale best practices [59] [60]. Spotio positions itself as a platform to **"predict pipeline & scale your process"**, not just track dots on a map [61] [62].

- **Integrations and CRM Sync:** Spotio is designed to complement a company's existing CRM and tech stack. It offers native or API integrations with major CRMs like Salesforce, HubSpot, Zoho, and Microsoft Dynamics [63] [64]. This means data collected in the field (new leads, notes, activity logs, etc.) can flow back into the primary CRM automatically, avoiding double entry. Spotio can both import data (e.g. pulling in a list of accounts or prospects for mapping) and export updates (e.g. logging a meeting as an event in CRM) [65]. In addition to CRM, Spotio connects via Zapier and has an open API for custom integrations [63]. They also integrate with tools like Zapier, spreadsheets (for import/export), and phone/email providers. A key value is that reps **"automatically capture field data in their normal flow of work, seamlessly passing data to your CRM"**, yielding better forecasts and performance data without extra effort [66]. Spotio's integration ecosystem (2000+ connections via Zapier) ensures it can fit into most sales tech stacks [67]. This extensibility is crucial for companies that want to use Spotio as a specialized field tool alongside their central CRM.

- **Reliability and Scale:** Spotio touts its technical robustness – claiming **99.998% uptime** and the ability to handle large data volumes [68] [69]. This addresses a pain point in the industry: some apps become sluggish or crash when thousands of leads or records are loaded (for example, Spotio claims that at ~100k leads, certain competitors' apps have stability issues) [68] [69]. Spotio's infrastructure is built to scale with growing teams and data, so reps aren't hampered by a "broken app" as their company expands. This reliability is a significant advanced feature, albeit behind the scenes, that ensures adoption sticks (reps will abandon a tool that is slow or crashes frequently).

In summary, Spotio is a comprehensive platform covering **lead generation, territory management, route optimization, activity tracking (visits, calls, texts, emails), sales pipeline management, task automation, and team performance reporting**. It's **mobile-first** (with a robust app for reps in the field) but also offers a full web experience for managers [70] [71]. The combination of mapping, CRM-like capabilities, and automation sets Spotio apart. It empowers field sales reps to be more productive (some users report it **"increased my productivity 10x"** and they "couldn't do my job without it" [72] [73]) and gives managers the visibility to coach and scale the team.

Competitive Landscape: Spotio vs. Alternatives

To design a best-in-class field solution for Yardura, it's important to understand how Spotio compares to other field sales apps and identify opportunities. Two notable competitors are **Badger Maps** and

SalesRabbit, which, like Spotio, cater to outside sales teams (especially door-to-door and route-based sales). Below is a brief benchmark of these platforms:

- **Badger Maps:** Badger Maps is primarily known as a **route planning and sales mapping** tool. The platform focuses on four key areas: **route optimization, field activity tracking, team management, and CRM connectivity** [74] [75]. Its strength lies in visualization and saving reps time on the road. Reps can easily import a list of customers or prospects and have Badger plot them on an interactive map, then generate **optimized routes** in minutes [76]. Users can create routes that minimize drive time and even export those routes to their calendar [77] [78]. Badger also offers basic **check-in functionality** – reps log visits with notes or outcomes at each stop, providing managers a breadcrumb trail of activity [79]. It supports territory assignment and has mobile apps to guide reps in the field. Integration with CRMs like Salesforce and HubSpot is available to keep data in sync [65].

Where Badger falls short: While Badger Maps is user-friendly for routing, its feature set is not as deep as Spotio's in other areas. Reviews note that **route optimization, while useful, has limitations** – for example, it doesn't always account for complex scheduling constraints or appointment priorities, and it struggles with last-minute route changes [80]. It lacks advanced territory balancing tools that some larger teams need [80]. Activity tracking in Badger is mostly manual (the rep must "check in" and input notes), and the system provides **fewer automation capabilities** – there are no automatic multi-channel logs or follow-up cadences like Spotio's Autoplays [81]. Reporting and analytics in Badger are quite basic; managers can get a weekly report or simple dashboard, but **granular performance metrics or predictive insights are missing** [82] [83]. In short, Badger Maps shines for mapping and routing, but teams that require comprehensive sales engagement features often find it limited [82]. This suggests an opportunity for Yardura's platform to combine Badger's mapping strength with deeper automation and reporting.

- **SalesRabbit:** SalesRabbit is an all-in-one field sales app popular with door-to-door teams (e.g. home security, solar). It offers canvassing tools, a mobile CRM for lead capture, and team management features. SalesRabbit provides an intuitive mobile interface and even industry-specific tools (like homeowner data for roofing or pest control leads) [84]. Users praise its simplicity and mobile-first design. Key features include lead forms, area management, leaderboards, and a built-in training content module for reps. SalesRabbit also supports **routing** and a "knock tracker" mode to systematically canvass neighborhoods.

Spotio vs SalesRabbit: SalesRabbit is often described as great for tracking the basics of door knocking, but less robust when it comes to data depth and analytics. Spotio's own comparison highlights that **SalesRabbit only tracks initial visits**, whereas Spotio tracks all follow-up activity (calls, emails, etc.) in real time [27]. SalesRabbit also lacks integrated phone/email tracking – it gives **"zero visibility into text, email, and voice"** communications, focusing only on the face-to-face interactions [28]. In addition, Spotio points out that SalesRabbit's infrastructure may not scale well (the app can crash when handling very large lead volumes) [85] [86]. Advanced capabilities like custom automations, territory hierarchies, or tailored analytics are not SalesRabbit's strong suit [87] [88]. On the other hand, SalesRabbit has some advantages: it's very **mobile-centric (built mobile-first) and user-friendly**, and it offers rich homeowner data (in some aspects, even more than Spotio, such as credit capacity scores) [89] [90]. SalesRabbit also includes features like in-app **team chat** and a content library (for training or sales collateral) which Spotio only partially addresses [91] [92].

Implications: The competitive gap suggests that a successful platform should combine **Spotio's breadth of features and analytics with the simplicity and strong mobile UX of tools like SalesRabbit**. None of the

existing players specifically cater to recurring field service operations (they are sales-focused), which is a niche Yardura can fill. Spotio's emphasis on data and automation is a model to emulate, while ensuring the app remains **reliable and easy to use in the field** (a lesson from SalesRabbit's popularity). Additionally, features like offline access and advanced scheduling (which neither Spotio nor SalesRabbit excel at, since they're not built for managing recurring service visits) will differentiate Yardura's solution.

Opportunities for Yardura: Yardura can stand out by **blending field sales and field service management**. From Spotio we learn the value of territory management, lead tracking, and multi-channel logging. From Badger and SalesRabbit we see the importance of intuitive route planning and mobile-first design. However, Yardura's scope goes further: our pooper scooper industry app must handle not just one-off sales visits, but ongoing service visits, dispatching, and operational logistics that sales apps don't cover. This is a gap in the market – traditional field sales tools don't manage recurring jobs, and generic field service apps don't have robust canvassing/lead features. By building an integrated platform, Yardura can leverage Spotio-like capabilities (for acquiring new customers and planning routes) and extend them into **scheduling recurring services, coordinating field technicians ("scoopers"), capturing service details (photos, notes), and syncing with billing**. In the next section, we detail a Product Requirements Document (PRD) for such a solution, aligning with Yardura's existing architecture and business needs.

PRD: Yardura Field Service App (Pooper Scooper Industry)

Product Vision: Create a unified field operations platform for the pet waste removal industry ("pooper scooper" services) that combines robust field sales tools (lead management, territory-based prospecting, route optimization) with advanced field service management (recurring job scheduling, dispatching, and on-site service tracking). This product will empower **multiple user roles** – Sales Reps, Field Technicians (Scoopers), and Admins – to collaborate in growing and serving a customer base efficiently. The app should seamlessly integrate with Yardura's existing systems (customer database, subscription billing via Stripe, etc.) and enhance the **Yardura/Sweep&Go** app architecture (per the provided spec) with Spotio's proven capabilities tailored to our industry.

1. Overview

In the pet waste removal business, success hinges on two fronts: acquiring new customers (sales) and reliably executing recurring service visits (operations). Yardura's Field Service App will be a **cross-platform Progressive Web App (PWA)** that supports both functions in one system. Field **Sales Reps** will use the app to canvass neighborhoods or follow up on inquiries – mapping out territories, identifying potential customers (e.g. homes with dogs), and converting leads into subscriptions. Field **Technicians (Scoopers)** will use the app to receive their daily route of service visits, navigate efficiently to each location, log their work (with photos and notes), and handle exceptions (skipped visits, rescheduling). **Administrators** (and dispatchers) will oversee the entire operation via a web dashboard – managing customer accounts, scheduling and dispatching jobs, tracking field staff progress, and ensuring billing and customer satisfaction are handled.

This PRD aligns with Yardura's modern tech stack (Next.js/React frontend, Prisma/Postgres backend, Stripe for payments, Tailwind UI) and extends it. The goal is to **replicate and enhance Spotio's field-sales functionality for Yardura's domain**, meaning we'll include lead management, mapping, routing, tracking, etc., *plus* industry-specific needs like recurring service scheduling, offline usage in backyards with poor signal, and direct integration to billing and the client portal. By doing so, Yardura will have a single system

to **track the entire customer lifecycle** – from initial lead, to signed client, to ongoing service delivery and payment – providing a competitive edge in efficiency and customer experience.

2. Personas and User Roles

Our system will be used by three primary personas, each with distinct goals and feature needs:

- **Sales Representative (Sales Rep):** Focuses on acquiring new customers for the service. This could be door-to-door canvassers or sales team members responding to inbound leads. Sales Reps need tools to identify promising prospects (e.g. homes with pets in a territory), record leads, schedule follow-ups, and ultimately sign up customers for recurring service plans. They care about **territory coverage, lead qualification, pipeline management, and conversion metrics**. A Sales Rep may use a mobile device while canvassing neighborhoods or a desktop when doing phone/email follow-ups.

- **Field Technician ("Scooper"):** Delivers the actual pet waste removal service in the field. Scoopers are assigned daily routes of customer yards to service. They need a **mobile app (PWA)** that works reliably outdoors (often with spotty internet) and allows them to see their schedule, get driving directions, log their arrival (GPS check-in), note any issues, take before/after **photos**, and mark the job complete. They may also need to trigger notifications (like sending an "on the way" text to the customer) and possibly upsell additional one-time services on site. Their priority is **efficient route navigation, accurate service logging, and staying on schedule**. They are less concerned with sales data and more with task execution and communication back to the office.

- **Administrator (Admin/Dispatcher/Manager):** Oversees business operations and the app's configuration. In a small company, this might be one person (the owner) who does everything from marketing to scheduling; in a larger one, there could be separate roles (a dispatcher for scheduling, a sales manager for leads, etc.). For our purposes, "Admin" represents any backend office user with broad permissions. Admins need a **web dashboard** to manage both sales and service workflows: they will import or assign leads to reps, define territories, configure pricing and service offerings, schedule customer visits, and handle day-to-day adjustments (like reassigning jobs if a tech calls out sick). They also will use the system for **monitoring** – checking where techs are in the field, ensuring jobs are done on time, reviewing performance reports, and maintaining customer satisfaction. Additionally, admins interface with other systems: ensuring completed jobs get billed (Stripe) and that the client portal is updated with service logs and photos. Key concerns for Admins are **visibility, control, and seamless integration** of data across sales, operations, and finance.

(Note: The Yardura client (the actual customer who pays for the scooping service) is an external persona who interacts via the Client Portal – scheduling, notifications, and seeing service history – but they are outside the scope of this field app PRD except where data flows to the portal.)

3. Functional Requirements

This section describes in detail the required features and functionalities of the Yardura Field Service App, grouped by thematic area. The features draw inspiration from Spotio's capabilities, adapted to include the needs of recurring service and dispatch. Each requirement is annotated with the primary user role(s) involved.

3.1 Lead Management & Sales Pipeline (Role: Sales Rep, Admin)
Yardura's platform will incorporate a full lead management module to track potential customers from initial interest through conversion. Core capabilities:

- *Lead Capture & Import:* Allow Sales Reps to quickly create new lead records on the fly (e.g. after a door knock or inbound phone call). A lead record includes contact info, address (with geolocation), and relevant notes (e.g. number of dogs, current waste situation). Admins should also be able to **import leads** in bulk (CSV upload or via integration) for targeting campaigns [65]. For example, if we acquire a list of dog owners in a neighborhood, we can import them as leads with addresses ready for mapping.

- *Territory Assignment for Leads:* Every lead is associated with a territory or zone. The system should support assigning leads to sales territories (either automatically based on address or manually by an Admin). Sales Reps with "limited" access see only leads in their assigned territory [47]. Admins can reassign territories or leads among reps as needed (e.g. if a rep leaves, their leads can be transferred).

- *Lead Qualification Data:* To maximize efficiency, integrate external data about leads (similar to Spotio's homeowner data). For instance, mark if an address is a single-family home vs. apartment, or if we have data on pet ownership in that area. While we may not get credit scores, any data like presence of a fenced yard or past service inquiries could be useful. The system will highlight **"qualified" vs "unqualified" prospects** so reps prioritize the best leads [93]. (Potential integration: a third-party data source or a questionnaire that the rep fills to score the lead).

- *Lead Status & Stage Tracking:* Implement a configurable **sales pipeline** for leads. Example stages: New Lead → Contacted → Quoted → Closed-Won (became customer) or Closed-Lost. Sales Reps update a lead's status as they progress (with the app automating some of this where possible). The pipeline view should show, for each rep or territory, how many leads are in each stage and the potential revenue (e.g. expected subscription value) [29] [30]. Admins need a **"Pipeline Dashboard"** to monitor all reps – seeing who might need help or which areas are underperforming.

- *Follow-ups & Reminders:* Provide task management or reminders for leads. Reps can schedule a follow-up (call, visit, or email) for a lead, which syncs to their calendar. The system should notify the rep when a follow-up is due so that **no prospect is forgotten**. This can be enhanced by **Autoplay sequences** (see below) for standard follow-up cadences. At minimum, implement the ability to create a sequence of tasks for a lead and check them off as completed (e.g. initial visit done, follow-up call scheduled, etc.) [36] [94].

- *Conversion to Customer:* When a lead agrees to service, the platform should support a smooth conversion. This includes capturing which **service plan** they chose (using the existing Yardura subscription schema) and scheduling their first service visit during the onboarding [95]. The Sales Rep should be able to initiate this in the app: for example, fill in a digital signup form or contract, possibly have the customer e-sign on the spot (if required), and instantly convert the lead record into a **Customer account** and a **Job schedule**. We will integrate with Yardura's existing onboarding flow (F-002 in the spec) to handle plan selection, Terms of Service, Stripe payment setup, etc. [95]. Essentially, the app will tie into the `/api/quote` and subscription creation endpoints so that a sales rep in the field can sign someone up on their tablet/phone just like a customer could on the

website [96]. Once converted, the customer profile (with address, plan, billing info) is accessible to Admin and Scooper roles.

- *Digital Contracts & Signature:* (Optional for v1, but desirable) If the business requires a service agreement, allow sales reps to generate a digital contract within the app for the new customer. This could be a templated PDF that auto-fills customer details and can be signed on screen. Spotio has **Digital Contracts/E-Contracts** capability that we may emulate [97]. It adds professionalism and speeds up closing deals on-site.

- *Lead Analytics:* Provide reports on lead metrics: conversion rate by territory, number of leads added per week, time from lead to close, etc. This will help Admins identify which campaigns or reps are most effective. Many of these can be achieved via the custom reporting feature (see Reporting section) using the captured lead data.

3.2 Territory Management & Geospatial Mapping (Role: Admin, Sales Rep)
Borrowing heavily from Spotio's mapping prowess, our system will include robust territory and mapping features:

- *Territory Creation:* Admins can create and manage sales territories through an interactive map UI (web interface). They can define territories in multiple ways, identical to Spotio's methods: drawing a custom polygon on the map [6], selecting predefined areas like a city or ZIP code [98], or entering a list of ZIP codes to combine [7]. Each territory will have a name, a distinct color, and optionally assigned reps [99]. The interface should allow editing boundaries (e.g. move points on a drawn shape) and deleting territories, with appropriate safeguards if sub-territories exist [100].

- *Territory Hierarchies:* Support a parent-child structure of territories [9]. For example, an Admin could have a "Metro Area" territory that contains several smaller neighborhood territories, each assigned to different reps. This hierarchy is useful for managers overseeing large regions with team leads. The system will enforce logical rules (e.g. you cannot delete a parent territory until sub-territories are removed, as Spotio does [9]).

- *Territory Assignment & Permissions:* When creating or editing a territory, Admin can assign one or more Sales Reps to it [99]. Reps by default will have access only to leads/customers in their own territory ("Limited Access"). The app should notify a rep if they attempt to add a record outside their territory (as a friendly warning) [11], but allow override in case territories change or overlap slightly. Admins can override permissions (some reps could be given access to all data if needed). This ensures reps focus on their zone and we maintain data organization by area.

- *Map Visualization of Leads & Customers:* Both Sales Reps and Admins will have access to a **Map View** that plots all relevant records as pins. Sales Reps on mobile can open the map to see leads around them (filtered to their territory or assigned leads). Admins on web can see a global view or filter by rep/territory. The pins should use different colors or icons for **lead vs customer**, and possibly additional color coding by status (for leads) or service status (for customers). Using Spotio's **colorization** concept, allow the user to apply a filter that colors map pins based on a chosen field [12]. For example, an Admin could colorize by "lead stage" to see which prospects are new vs in negotiation [101]; or a dispatcher could colorize customers by service frequency or last service date to visualize coverage. This at-a-glance insight is valuable for planning.

- *Record Detail from Map:* Users can tap/click a pin to see a quick info card (name, address, last visit or status, etc.) and then drill into full details or actions (e.g. "Add Note", "Mark as Closed", "Add to Route"). This map interactivity should mirror Spotio's behavior where clicking a pin offers options like adding that location to a route or updating its info [102] [103].

- *Territory Performance Tracking:* Provide views or reports that aggregate data by territory. For example, an Admin should be able to see how many customers and leads are in each territory, revenue per territory, and conversion rates. Spotio highlights the ability to **"view each territory's performance… and get the most value from each"** [8]. We will include a **Territory Performance Report** that shows metrics like leads converted, services completed, etc., by territory. This can guide decisions on rebalancing territories or focusing efforts on high-potential areas.

- *GIS/Mapping Integrations:* Use a reliable mapping service (Google Maps or Mapbox) for geocoding addresses and rendering the map. We'll need integration for features like the drawing of shapes and possibly layering (e.g. showing KML layers if needed, though not priority). Ensuring the map loads quickly and can handle thousands of pins without crashing (Spotio's emphasis on scalability [104]) is a non-functional requirement. We may leverage clustering of pins when zoomed out for performance.

3.3 Scheduling, Dispatch & Recurring Jobs (Role: Admin, Scooper, *Sales Rep indirectly*)
This is a critical section that extends beyond Spotio's scope, focusing on service operations:

- *Job & Visit Definitions:* In Yardura's context, a **Job** represents a recurring service commitment for a customer (e.g. "Weekly scooping service for Customer X"), and a **Visit** is an instance of that job (e.g. "Service visit on Jan 10th at 123 Maple St."). The system must manage recurring schedules as first-class data. This includes storing job details like frequency (weekly, bi-weekly, etc.), service day preferences, assigned technician, and any specific notes or add-ons (e.g. "also spray deodorizer every visit"). These correspond to the Yardura data model (e.g. `Job`, `ServiceVisit` models) which will be extended or utilized.

- *Recurring Schedule Generation:* Once a customer is onboarded (via sales or directly via portal) with a given frequency, the system auto-generates a series of scheduled visits. For example, a weekly customer might have visits every Tuesday. Admins need an interface to view and adjust these schedules. The **Dispatch Board/Calendar** will show all upcoming visits, likely in a calendar or a board format grouped by date or by technician. The spec calls for a **drag-drop board** for dispatch [105] – we envision a scheduling board where each column is a date (or a tech) and visits are cards that can be dragged to reschedule or reassign. For instance, a dispatcher could drag a visit from Jan 10 to Jan 11 if the customer requests a change, or drag it from Technician A's column to Technician B's if swapping assignments.

- *One-Time & On-Demand Jobs:* In addition to recurring plans, the system should handle one-time service requests (e.g. a customer wants a special clean outside their normal schedule, or a trial single visit for a new client). Admin or Sales Rep can create a one-time job, schedule it on an open date, and assign it. These one-time jobs are treated similarly in the dispatch view but clearly marked as one-off.

- *Dispatching & Assignment:* The Admin (dispatcher) can assign jobs to Field Technicians. Typically, a recurring job will be assigned to a primary Scooper (e.g. John Doe handles all weekly visits for

Customer X). The system should default to that assignment for each generated visit, but also allow changes. If John is out sick one day, the dispatcher can reassign that day's visits to another tech with a few clicks (multi-select visits and assign to a different user). When doing so, the **affected Scoopers get notified** of changes (see Notifications section). Each technician should have a manageable workload per day – to assist, the dispatch board can display total visits or estimated hours per tech per day. Yardura's spec F-003 explicitly lists **"reassign/reschedule"** as needed functionality [105], which we will deliver via this interface.

- *Route Optimization for Daily Schedules:* A unique advantage we'll implement: once a technician's set of visits for a day is finalized, the system can automatically **optimize the route order** of those stops for that tech. For example, if a tech has 10 houses to service in a day, the system (via Google Maps Distance Matrix API) can compute the optimal sequence to minimize drive time [106]. This optimization should occur on demand (e.g. an Admin clicks "Optimize Route" for that tech's day) or automatically overnight for the next day's routes. We'll abide by the SLA in the spec: route optimization should handle up to ~100 stops in ≤10 seconds (our use case is likely <20 stops per route, so this is feasible). Technicians will see their route in optimized order when they start their day, but they should remain free to adjust if needed (maybe they want to visit a certain area first due to a time-specific request). So the app will allow manual reordering too. Also, include options to avoid tolls or highways if desired, similar to Spotio's route options [19].

- *Live Dispatch Updates:* If any schedule changes happen during the day (e.g. a customer cancels last minute, or a new emergency job is added), the Admin can update the schedule and the changes should reflect in the Scooper's app in near real-time. For instance, a new job could be pushed to a tech's route at 2pm; the tech receives a notification and sees the stop added. Real-time syncing (perhaps via WebSockets or rapid polling) will keep field and office aligned.

- *Calendar Views:* Provide multiple views for scheduling: a **daily view** (e.g. list of all jobs on a given day, used by dispatch to ensure coverage), and a **technician-centric view** (each tech's agenda). Also allow filtering by territory or service type if needed. The dispatch calendar should integrate with external calendar (iCal feeds or Google Calendar sync) if management wants to subscribe to schedules, but this is secondary.

- *Skipping and Cancelling Visits:* Weather events or other factors can necessitate skipping service visits. The system must support marking a scheduled visit as "Skipped" with a reason code (e.g. "Heavy Rain", "Customer not home/unable to access yard", etc.). According to spec, **"weather skip + notify"** is a requirement [107]. This implies if an Admin marks an entire day's routes as skipped due to weather, an automatic **notification to customers** should be sent (likely via SMS/email) informing them of the skip and possibly rescheduling info. The app should allow bulk skipping: e.g. select all visits on Feb 5th and mark as skipped (reason: snow), triggering notifications. For individual skips (like a tech arrives and gate is locked, so they can't perform the service), the Field Tech should mark that on their app, capturing the reason and perhaps a photo ("could not enter – photo of locked gate"). These skip reasons feed into billing adjustments (see Billing section) – e.g. skip with reason might mean a credit or a partial charge depending on policy [108]. Admins should review skipped visits and decide on follow-up (reschedule later in week or wait till next cycle).

- *Reschedule & Reclean:* For skipped or problematic visits, dispatch should be able to easily **reschedule** them. E.g., drag the missed job to a different day or mark it for an extra visit in the next cycle.

"Reclean" might refer to doing a second pass if a customer is unsatisfied – the system should allow the creation of a corrective one-time job tagged to a recent visit (so we know it's a redo) [107]. The platform should track these recleans for performance metrics (e.g. how often we had to redo a job).

- *Service Visit Details:* Each service visit (job instance) should contain specific info for the technician: customer name, address (with map directions link), gate code or special access instructions, number of dogs, any add-on services (e.g. deodorize yard, fill water bowl – we can incorporate cross-sell items from spec F-010 in future [109]). The tech should see all this at a glance. After completing, the visit record will store outputs: completion timestamp, any notes, photos taken, and perhaps the weight of waste collected or other metrics if we do "Wellness" tracking (the spec's wellness feature involves recording data like weight/frequency of waste [110] – we might not implement that in v1, but we'll keep the door open by allowing techs to enter optional data readings).

- *Capacity Management:* (Optional) If we want to avoid overbooking, the system could maintain an estimate of how long each visit takes (maybe default 10 minutes, configurable per job based on yard size or dogs). Then daily route planning could total the estimated time vs. an 8-hour workday. For v1, we assume dispatchers manage this manually, but we will not exceed reasonable limits (the dispatch board is visual enough to avoid overbooking one person).

3.4 Field Technician Mobile App Features (Role: Scooper, Admin for monitoring)
The Scooper's PWA is their primary tool during the workday. It must be extremely user-friendly, reliable offline, and focused on task execution. Key features include:

- *Daily Route View:* When a Scooper logs in, they see their assignments for the day. This can be a list (ordered list of stops) and a map. The list shows each visit with time (if scheduled at specific times, else just in sequence), customer name, city, and perhaps an indicator if it's a first-time visit or a re-clean. The tech can tap a stop to see full details. There should also be a **"Start Route"** or **"Navigate"** button to initiate turn-by-turn navigation to the first stop. We will integrate with phone's mapping app (Google/Apple Maps) for navigation when needed, since building navigation in-app is beyond scope (just deep link the address).

- *GPS Check-in / Clock-in:* Upon arrival at each job, the Scooper should **check in** to record that they are on site. This can be a simple "Arrive" button on the job details, which records timestamp and current GPS coordinates. The app should validate the coordinate is reasonably close to the customer address (within a threshold) to ensure the tech is at the right location (similar to Spotio's visit verification [23]). This check-in serves multiple purposes: it can trigger an **"On The Way" SMS** to the customer just before arrival, and it provides real-time status to the Admin that the visit is in progress [111]. We will implement an automated SMS such that when the tech taps "Depart for Next Stop" or "Arrive", it can send a templated message via Twilio (e.g. "Hi, this is Your Scooper, I'm on the way to your home now with an ETA of 15 minutes.") [106]. The spec explicitly lists **"on-the-way SMS"** as a feature [111]. We'll allow the tech to cancel or edit the message if needed, but it should be mostly automatic to reduce friction.

- *Task Guidance & Checklists:* For each job, display any predefined checklist or tasks. For example: "Close gate upon exit", "Leave a door hanger if customer not home". The tech can check off items if needed. This ensures consistent service quality. In the future, this could tie to cross-sell (e.g. "Apply deodorizer if included").

- *Photo Capture:* Technicians must be able to take photos within the app and attach to the visit record. We'll prompt for at least one **"after" photo** of the cleaned yard (and perhaps an optional "before" photo if the tech wants to document an exceptionally messy yard or an issue). The photos are uploaded (with offline caching if no signal) and tagged with that visit. This ties into spec F-004 which requires **"photos required"** for field techs [111] and F-005 Client Portal's gallery [112] – customers will later see these photos in their portal as proof of service. We must compress photos for quick upload and meet the spec SLA (photo upload ≤3s ideally). Low reception mode: if offline, the app stores the photo and uploads when back online (with proper retry logic).

- *Service Notes:* The Scooper can add **notes** for any special observations during the visit. E.g. "Dog was outside, very friendly." or "Large amount of waste today, took extra 5 minutes" or "Could not access side yard because gate was locked." These notes are saved to the visit record. Some notes might be tagged as needing office attention (we could include a checkbox like "Follow-up required" if, say, the tech wants the office to call the customer). All notes should be visible to Admins and also possibly to customers in the portal (with some filtering if internal-only notes exist).

- *Job Completion & Outcomes:* After finishing a visit, the Scooper marks it **Completed**. At that moment, if any required fields are missing (e.g. no photo was taken, or a "weight of waste" reading if we do that), the app should prompt to input them. The completion triggers several things:

- timestamp and GPS capture (to mark where and when finished),
- status update to dispatch (so Admin knows it's done),
- possibly an automated **completion notification** to the customer (e.g. via email: "Your yard has been serviced at 2:45pm, see photos in your portal."). This could also be an SMS depending on preferences. Notification preferences are part of client portal settings per spec F-005 [112], and we'll ensure our system can send the appropriate webhook or message to fulfill that.
- If any issues occurred (like partial completion), the tech can mark an exception status (Completed with Issue, or Skipped, etc.). If skipped or incomplete, the job might remain open for reschedule.

- If the tech collected any on-site payment or tips (not typical since we bill via Stripe subscriptions usually), they could note that too, but likely not needed due to Stripe integration.

- *Offline Mode:* A critical requirement is robust offline support, since techs will be in backyards where connectivity might drop. The PWA must use caching and local storage so that the day's route and data is available even if offline. Techs should be able to view all job details, capture notes and photos, and complete jobs without internet. These will be queued locally and **sync automatically** once a connection is re-established [111]. We'll implement a synchronization module that retries sending data (visits, photos) in the background. This ensures no data loss and uninterrupted workflow. Perhaps use Service Workers for background sync. The UI should indicate if the app is offline and what data is pending upload.

- *Time Tracking (Clock-in/Clock-out & Breaks):* For payroll and accountability, include a basic time tracker. At start of day, Scoopers clock in via the app; at end, clock out. They can also log breaks (start break, end break). The app can remind them if they forgot to clock out by day's end. These timestamps feed into the Payroll module (spec F-008) for calculating hours [110]. Also track driving vs on-site time possibly, but initially just total hours is fine. An "odometer" field is mentioned in spec [111] – this suggests capturing vehicle mileage. We could implement: at start of day or each trip, tech enters

odometer reading, and at end of day as well, to compute miles driven (or use GPS distance as proxy). This is helpful for reimbursing mileage or tracking fleet usage, and can feed into payroll or eco metrics.

- *Incident Reporting:* If any accident or unusual incident occurs (e.g. dog escape, property damage), allow tech to flag it in the app so management is alerted immediately. This is more of a safety feature; these incidents can be logged as special records with photos if needed.

- *Navigation Integration:* Each stop will have a one-tap option to open it in Google Maps (or Waze/Apple Maps) for navigation. Alternatively, our app can display a live map with the route polyline and current location of tech (if not too complex). Initially, leveraging external map apps might be simpler and more familiar to users.

- *Push Notifications to Tech:* The tech should receive in-app or push notifications for events like: a new job assigned to you last-minute, a job on your route got canceled or rescheduled, messages from admin, etc. Since it's a PWA, push notifications via the browser might be used for Android; iOS PWA push is now possible as of iOS 16+ as well. This keeps the tech updated without constantly refreshing.

- *Team Communication:* We noted Spotio and SalesRabbit both have team chat features [91]. Including a basic chat or messaging system could greatly help coordination (e.g. tech can message "I'm running 20 minutes behind" to dispatcher, or ask a question about a job). If feasible, implement a **Team Chat** channel or at least a one-to-one messaging between tech and admin. This can be a simple real-time chat using a service (or even Slack integration as a proxy). For v1, this is nice-to-have; at minimum, ensure techs have easy click-to-call for their manager and maybe a way to log that they made a call if off-app.

3.5 Communication & Notifications (Role: All)
Communication features ensure everyone (including the customer) stays informed:

- *Customer Notifications:* As described, automated notifications should be sent for key events:
- **Service Reminders:** e.g. "Your scooper will visit tomorrow between 9-11am." (This can be an email/SMS a day prior, optional if customer opts in).
- **On-the-Way Alert:** When tech starts heading to the customer, trigger SMS ("On the way, arriving by 2:30pm") [111].
- **Completion Notice:** After service, send an email or text confirming completion, possibly with a summary or a link to view photos in the portal.
- **Skip/Cancellation Notice:** If a service is skipped or rescheduled (by weather or other reason), notify the customer immediately with an apology and new schedule info [107].

These hooks tie into Yardura's notification preference center (portal setting) [112] and likely use a provider like Twilio (for SMS) and SMTP/Resend (for emails) [106]. We should use centralized notification services already in the Yardura architecture where possible.

- *Internal Notifications:* Within the app, users receive alerts for events relevant to them:

- Sales Rep: gets notified when a new lead is assigned to them or when a lead in their territory requests info (like if a web lead comes in via the quote form, assign to a rep and notify them immediately to follow up).
- Scooper: notified of schedule changes (new job added, job removed, or any changes to details), as well as when they have overdue tasks (e.g. if they haven't marked a job complete an hour after scheduled).
- Admin: notified if a job is marked skipped or any issues are logged, so they can react. Also, if a new lead comes in without an assignment, admin gets an alert to assign it.

The system can use a notification center pattern or push messages. Also, daily summary emails to Admin (e.g. "Today's Completed Jobs: 95/100, 5 skipped") could be useful.

- *Autoplays & Sequenced Communications:* For sales follow-ups, allow creation of **Autoplay** sequences similar to Spotio [36]. E.g., define a sequence for new web leads: Day 0 email, Day 1 call, Day 3 text, etc. The system can then automatically send or prompt these communications. Initially, we might just implement templated emails/texts that a Sales Rep can trigger with one click (and log the activity). Full automation could be a later feature.

- *Email & Calendar Integration:* Sales Reps should integrate their email (likely via OAuth through a service like Nylas or directly Gmail API) so that sending an email to a lead from our platform logs it. We already allow calendar integration (Spotio style) [46] [113] – if a rep integrates, their **appointments sync** both ways. This is important for sales reps booking appointments, but also possibly for admin dispatch if we want to see tech calendars (though techs will use our internal scheduler primarily). The integration will ensure any meeting scheduled in the app is added to their Google Calendar, and vice versa.

- *Click-to-Call and Call Logging:* Enable a rep to tap a phone number in a lead record to call (on mobile it will use the phone dialer). If we can integrate a calling solution or simply prompt them to log a call result after the phone call, that would capture call activity. Spotio offers integrated calling with a feature called "SPOTIO Numbers" for caller ID and logging [114] – we might skip that complexity at first, but at least log that a call was made and allow marking outcome (Reached, Left VM, No answer, etc.).

- *Team Broadcast:* Admins may need to send a quick message to all reps or all techs (e.g. "Storm rolling in, wrap up and head back"). A broadcast notification feature for certain roles could be included.

3.6 Billing & Finance Integration (Role: Admin primarily)
While much of billing happens in the background (via Stripe subscriptions), the field app must hook into those processes to ensure correct charging for services:

- *Stripe Subscription Sync:* When a lead converts to a customer with a recurring plan, the app will call the existing Yardura backend to create a Stripe subscription (this is handled in onboarding F-002) [95]. We ensure the sales rep workflow triggers the same backend logic as the self-service flow (maintaining parity with the existing quote wizard to generate `leadId` and then subscription) [96].

- *One-Time Charges:* If a one-time job or add-on service is performed (like a one-time deep clean or a deodorizing add-on not included in the subscription), the app should flag it for billing. This could

mean generating a one-time invoice via Stripe or adding to the next invoice. The system should support **per-visit charges** outside the subscription. For example, if a customer requests an extra service visit, an invoice is created upon completion of that visit. Admins need to approve/send these or have them auto-charge if payment method on file. Our app should mark such jobs as billable extras, so the finance integration (spec F-007 covers one-time invoices and extras) can handle it [115].

- *Skip Reason Pricing Adjustments:* When a visit is skipped for a reason, apply business rules for billing. The spec mentions **"skip-reason pricing"** [115] – e.g., if skipped due to weather (our fault), the customer might get a refund or credit for that visit; if skipped due to customer (e.g. they locked gate), maybe still charge or partial charge. We will implement logic such that when marking a skip, Admin can choose whether to credit or not, and the billing system will either not charge for that occurrence or charge a reduced fee (perhaps set as a percentage or flat fee for skip). This info will be sent to Stripe or recorded to be applied on the invoice.

- *Dunning & Payment Failures:* While Stripe handles failed payments (dunning emails, retries), the admin interface should show if a customer's account is past due or a payment failed. That might be more on the client portal side, but dispatch should perhaps see a warning (e.g. "Do not service - account suspended for non-payment"). So integration with Stripe webhooks to flag accounts would be helpful.

- *QuickBooks Online (QBO) Sync:* The spec lists QBO sync [115]. Our system will log all completed jobs and invoices, which can later sync to QBO for accounting. Not directly a field app feature, but data from field (like hours worked, mileage, number of visits) can be part of what's synced for payroll and accounting.

- *Payroll Data:* The field app will generate data for payroll: hours clocked, visits completed, miles driven. Spec F-008 outlines payroll metrics (hourly/bonus/commission, overtime, mileage etc.) [110]. We ensure the app captures the raw data (time stamps, odometer, etc.) to feed those calculations. For instance, a commission could be given per new customer signed (sales rep) or per job completed (tech) – the system can tally those, but actual payroll processing likely external or in admin reports.

3.7 Reporting & Analytics (Role: Admin, Sales Manager)
We will incorporate a powerful reporting module leveraging the data collected:

- *Standard Reports:* Out-of-the-box, provide a set of reports/dashboards:
- **Sales Funnel Report:** Leads by stage, conversion rates, average time to convert.
- **Activity Report:** Number of door knocks, calls, emails by rep (from sales side) – similar to Spotio's activity capture [24].
- **Territory Performance:** Revenue and active customers per territory, leads conversion by territory [8].
- **Technician Productivity:** Jobs completed per day/week by each tech, average time per job, skips per tech, etc. This can highlight if someone is struggling or if routes are imbalanced.
- **Customer Retention:** Churned customers by month (though this might be more portal related).
- **Financial Summary:** Billable visits vs skipped, extra charges, etc., for a period (to reconcile with Stripe reports).

- **Leaderboard:** A live leaderboard showing top performers – could have two: one for sales (e.g. most new subscriptions this month) and one for service (e.g. highest on-time percentage or most five-star feedback if we collect feedback).

- *Custom Reports:* Following Spotio's **"My Reports"** approach, allow users to create custom report queries from the data [116]. This might be ambitious for v1, but at minimum, exportable data or a simple filterable table view of key entities could be provided. For instance, an Admin could filter all jobs in June where skip reason was "Weather" and export that list. We can integrate a tool or use our database to supply these queries.

- *Data Dashboard UI:* Use charts, graphs and summary cards to present data. For example, a dashboard homepage for Admin might show: Total Active Customers, New Leads This Week, Completion Rate, etc., in a glance.

- *Real-time Updates:* The reporting module should reflect data in near real time (as soon as a job is completed or a lead updated) so that managers are always looking at current info [62]. This might require some live refresh or at least up-to-the-hour refresh intervals.

- *Export & Integration:* All reports should be exportable (CSV/PDF) for management meetings, and an API could allow external BI systems to pull data (the spec mentions a public API/Webhooks integration [106]).

3.8 Technical & Cross-Platform Requirements (Role: Developers/IT, applies to all users indirectly)
These requirements ensure the product fits into Yardura's architecture and is delivered effectively on web and mobile:

- *Platform & Architecture Alignment:* The application will be built as a **Next.js/React** front-end with a **Prisma ORM** connecting to a **Postgres** database (migrated from the current SQLite) [117] [118]. We'll leverage the existing schema and extend it for new entities like Lead, Territory, Route, etc., keeping naming consistent and adding relations incrementally. The backend will expose RESTful API endpoints (or Next.js API routes) for all functionalities, documented via OpenAPI, and secure them with existing auth (NextAuth sessions, etc.). We will also implement appropriate webhooks and integration consumers (Stripe, Twilio) as needed [106].

- *Progressive Web App (PWA):* The field app will be a PWA to maximize cross-platform reach. On desktop, it runs in browser for Admin; on mobile, users can "Add to Home Screen" to use it like a native app. It should function offline, as described, using Service Workers for caching key assets and data (e.g. cache the map tiles for territory area, cache the day's route, etc.). The PWA approach means we maintain one codebase for all platforms, aligning with Yardura's web-centric stack.

- *Responsive and Ergonomic UI:* Use Tailwind CSS and ShadCN UI components (consistent with Yardura's design system) [117] to ensure a cohesive look. The UI must be responsive: e.g., an Admin's dispatch dashboard might be best on a large screen, whereas a Sales Rep's territory map and a Scooper's route view should adapt to phone screens. We'll incorporate mobile-friendly touches like larger tap targets, simple navigation (tab bar for Reps/Techs perhaps), and minimal text entry (use buttons, toggles, voice-to-text for notes if possible).

- *Cross-Platform Delivery Options:* While the PWA should suffice, we will also consider wrapping the PWA in a native shell (using tools like Capacitor or React Native WebView) to publish on app stores if needed for easier distribution or if device-specific capabilities (background location, push notifications) are needed. At very least, ensure the PWA passes criteria for iOS and Android installation (serving over HTTPS, having a manifest and service worker). For features like push notifications and geolocation, check compatibility: modern PWAs can handle geolocation and even background sync, which should cover our needs.

- *Security:* Follow security best practices. Leverage Yardura's existing auth system (NextAuth) for user login and role-based access. Ensure data partitioning (one org's data vs another if multi-tenant) is respected via the Org model. Sensitive personal data (PII, customer addresses, etc.) should be encrypted in the database where appropriate, as noted in spec (column-level encryption, audit trails). All external integrations (Stripe, Twilio, Maps) must use secure tokens/keys stored in env configuration.

- *Performance & Offline SLAs:* The app should be performant: e.g., **job list load < 200ms**, **portal interactions < 500ms** as per spec targets. Maps and routing computations should be optimized, leveraging caching of results (e.g., don't recompute the same route repeatedly; possibly store optimized route order with each day's schedule). On low-end mobile devices, ensure the app remains snappy by minimizing heavy computations on the client (offload to server if needed, e.g., route optimization could be done server-side via a Maps API call and send the ordered list to client). Photos should be resized on client before upload to meet the <3s upload goal. We will also implement graceful degradation if something fails (e.g., if route optimization API is down, still show un-optimized list with a warning).

- *Integration Points:* The architecture needs to integrate with:

- **Google Maps / Distance Matrix API** for geocoding addresses and optimizing routes [106].
- **Twilio (or similar SMS API)** for sending text messages (on-the-way alerts, etc.) [106].
- **Stripe API** for any billing triggers (likely minimal since subscriptions auto-bill, but for one-time charges or creating Checkout sessions if needed) [106].
- **Resend/Email service** for emailing customers (we have that in current stack) [106].

- Potential **Zapier or open API** for future integration (e.g., connecting to other systems, but not critical for v1 aside from what's listed).

- *Testing and Reliability:* Implement thorough testing, especially for offline scenarios and data sync conflicts. Use test automation for route calculations (ensuring correct optimization) and permission checks (a rep shouldn't see others' leads, etc.). Monitor app performance and crashes (integrate an error reporting SDK since PWA can use something like Sentry). We aim for high reliability like Spotio's touted uptime; any critical bug that causes app crashes or data loss will erode user trust, so QA is key.

In summary, the technical approach is to leverage the existing Yardura web app foundation and extend it with new modules (Lead management, Territory, Scheduling, etc.), keeping everything cohesive in one system. This avoids siloing data and ensures users (especially Admin) have one place to log in and do everything.

4. Feature Matrix by User Role

The following table summarizes major features and which user roles they are available to, highlighting differences in usage:

Feature	**Sales Rep** (Sales)	**Field Tech** (Scooper)	**Admin/Manager**
Lead Creation & Import	Create new leads in app; view/edit own leads. Cannot bulk import.	*Not applicable* (no lead focus)	Bulk import leads; view/edit all leads; assign leads to reps.
Territory Mapping	View own territory on map; see leads/ customers in territory. Cannot create territories.	View service areas (if needed), mostly not used.	Create/edit territories; assign territory to reps; view all territories on map.
Lead Qualification Data	See enriched data on leads (homeowner info, etc.) to prioritize.	N/A	Configure which data is shown; view lead quality metrics across territories.
Pipeline & Stages	View and update stages of own leads; personal pipeline stats.	N/A	View pipeline of all leads or by rep/territory; analyze conversion rates.
Appointments & Calendar	Schedule follow-ups/ meetings with leads; sync to personal calendar.	N/A (except maybe team meetings)	Oversee sales appointments if needed; possibly schedule sales events for reps.
Autoplay Sequences	Execute assigned follow-up sequences (e.g. send template emails, make calls as prompted). Cannot create sequences.	N/A	Create/modify autoplay sequences (cadences) for leads; assign sequences to reps or lead sources.
Route Planning (Sales)	Create optimized routes to visit multiple leads in a canvassing session [21] ; use mobile map for navigation.	N/A (their routes are jobs, handled below)	Possibly create/edit routes for reps (e.g. suggest a canvassing route for a new rep); mostly rep-driven.
Dispatch Scheduling Board	N/A (sales reps don't manage schedule of techs).	N/A (techs don't manage overall scheduling).	Full access to scheduling interface: create jobs, assign/reassign visits, drag-drop reschedule, view all techs' calendars.

Feature	**Sales Rep** (Sales)	**Field Tech** (Scooper)	**Admin/Manager**
Daily Job List/ Calendar	N/A (sales may have tasks, but not service jobs).	View personal daily route of assigned jobs; see upcoming schedule.	View any tech's schedule; master calendar of all jobs. Can adjust any entry.
Route Optimization (Service)	N/A (or minimal if rep uses for leads)	Use "Optimize Route" on daily jobs list to reorder stops efficiently. Can also manually reorder if needed.	Optimize routes for any tech or day; configure optimization rules (avoid tolls, etc.) [19] .
GPS Tracking & Breadcrumbs	Location optionally tracked during canvassing; used for verification of visits. Reps can view their own past route but not others.	Location tracked for route progress; tech sees own current GPS, possibly a trail of completed stops.	Track all reps and techs on map in real-time [22] ; view breadcrumb trails of where each has been. (Privacy controls to disable as needed.)
"Find My Team" Map	N/A (rep sees only self on map, not needed).	N/A (tech sees only self).	Yes – one-click to locate all active field users on the map with their last known position [22] .
Check-In / Visit Logging	Log visits to leads (door knocks) with one tap; auto timestamp/GPS.	Check in at each job (arrival time, GPS); mark completion with timestamp.	View logs of all visits and knocks; verify GPS for accountability.
Photos Attachment	Can attach photos to a lead (e.g. picture of house or a signed contract). Not required but possible.	Must take photos for each service job (before/after as required) [111] ; view past photos for reference.	Access all photos via admin interface or client timeline [112] ; use for QA or customer inquiries.
Notes & Tags	Add notes on leads (conversation details, preferences); tag priority leads.	Add notes on jobs (service observations, issues); record any on-site upsells or customer feedback.	View all notes; add internal notes on customers (visible only to staff); highlight notes that need follow-up.
Skip/Issue Reporting	N/A (no concept of skip in sales beyond follow-up dropped).	Mark job as skipped (with reason code) if unable to complete; report issues (dog not home, etc.) via app.	Receive alerts on skips; override or reschedule as needed; see summary of skip reasons for billing adjustment [115] .

Feature	**Sales Rep** (Sales)	**Field Tech** (Scooper)	**Admin/Manager**
Messaging/ Chat	Use in-app chat to communicate with team/manager (e.g. need marketing material, etc.).	Use chat to communicate with dispatch/admin (e.g. "delayed at site, will be late").	Broadcast messages or chat individually with reps/techs. Possibly maintain group channels (All Techs, All Sales).
Customer Info Access	View customer profiles once lead converts (maybe limited info: name, address, plan). Sales might revisit a current customer for upsell.	View assigned customer profiles (address, contact, pet details, service notes) to perform service correctly. Not all billing info.	Full access to all customer profiles, including billing status, service history, etc. Manage customer data.
Billing Actions	Possibly record payments if taking deposit in field (but normally not, since Stripe subscription).	N/A (techs do not handle billing). May just see if customer is prepaid or not.	Manage billing: set prices, handle refunds or extra charges (if a tech logged extra service, admin finalizes the invoice) [115] . Reps and techs largely do not deal with Stripe directly.
Analytics/ Reports Access	Limited – a sales rep might see a personal dashboard (e.g. their lead conversion rate, how many knocks they did vs target).	Limited – a tech might see basic personal stats (jobs done this week, on-time percentage, etc.) to self-evaluate.	Full analytics access. Generate all reports (sales and operations). Download data. Use custom reporting tools [54] .
User Management	N/A (cannot create users or teams).	N/A.	Admin can create/edit users, assign roles (Sales vs Tech vs hybrid), set permissions. Also manage teams if needed (e.g. group users under a team leader).
Integration Settings	Integrate own email/ calendar for logging [46] ; possibly connect own phone number if provided.	N/A (techs likely don't integrate personal email; their schedule is in-app).	Set up org-wide integrations (Stripe keys, Twilio, Maps API, QuickBooks sync). Also manage Autoplay templates, email templates, etc., which then reps use.

(Table Legend: "Yes/Can Do" features are described; "N/A" means not applicable to that role.)

This matrix demonstrates that **Sales Reps** concentrate on the CRM-like functions (leads, follow-ups, sales pipeline) and have mapping/routing for canvassing, whereas **Scoopers** focus on the operational side (daily

routes, service execution, check-ins, photos). **Admins** bridge both, with full visibility and control, plus configuration capabilities.

5. Non-Functional Requirements

In addition to the functional features, the product must meet several non-functional requirements to be successful:

- **Usability & UX:** The app should be intuitive with minimal training. Reps and techs in the field should find the interface **"quick and highly customizable… increases productivity"** as users praised Spotio [72]. Keep screens uncluttered: for techs, a simple sequential workflow (view job -> check in -> complete -> next job) is crucial. Use clear icons (e.g., camera icon for photos, map icon for navigation). Also provide localization support in case of multi-lingual staff in future (Spotio even has multiple languages on support site [119]).

- **Performance:** Ensure fast load times and smooth interactions. All data queries should be optimized (proper DB indexes, pagination). The map should render without lag, even with hundreds of pins. Aim for under 2 seconds to load main screens and under 0.5s for most in-app actions for a snappy feel. The spec's SLA targets (route optimization $\leq$ 10s for 100 points, portal P95 < 500ms, etc.) will guide our performance tuning. The app must also handle increasing load as the business scales (e.g., support thousands of customers, dozens of concurrent users, and years of historical data).

- **Reliability & Offline Robustness:** The system should be highly available. We'll ensure a **99.9%+ uptime** target for the web services (Spotio boasted 99.998% uptime [68]). Implement redundancy and proper error handling. The offline mode needs thorough testing – in spotty network conditions, the app should not freeze or lose data. Queued updates must survive an app restart (store them in IndexDB or localStorage until sent). If a sync conflict occurs (e.g., admin edited a job while the tech was offline and the tech also edited), we must define a resolution strategy (possibly admin changes override, and tech is notified upon sync).

- **Security & Privacy:** Use HTTPS for all client-server communication. Protect customer PII (names, addresses) – since field devices might be lost, perhaps provide an option for remote wipe or at least ensure data is encrypted at rest on the device if possible (PWA might rely on browser storage encryption). Implement role-based access properly so no user can access unauthorized data (tested via trying to call APIs outside their scope, etc.). For GPS tracking, allow an "incognito" mode if a rep or tech explicitly toggles location sharing off (and log that event). Comply with relevant privacy laws (e.g. if tracking employees, get consent or have it in contract, etc.). Also, implement audit logs for critical actions (e.g., admin reassigning a job, or deleting a lead) to assist in troubleshooting or compliance.

- **Maintainability & Extensibility:** Following Yardura's spec strategy, we will **"add new v1 models incrementally"** without breaking existing ones. The codebase will be modular: separate modules for sales (lead mgmt) and service (dispatch) but sharing common components (e.g., map view component used in both). Document the code and provide a knowledge base for users (similar to Spotio's support center) so onboarding new team members is easier. The design should accommodate future features like franchise management (multi-org), wellness data tracking, etc., so

keep the architecture flexible (e.g., design the data model of jobs to easily add new attributes like health scores in the future).

- **Scalability:** The system should handle growing numbers of users and records. If Yardura expands to multiple regions or franchises (spec even mentioned franchise features [120]), the app might support many orgs. We'll design the database and queries to scale (use proper filtering by org and indexes by territory, etc.). Also plan for mapping large datasets (if thousands of pins, use clustering). Ensure background processes (like route optimization requests or bulk scheduling) are done asynchronously to not block the user interface.

- **Integration and API:** Provide APIs or webhooks so the data can flow to other systems. For instance, a webhook when a job is completed (with photos) could notify a marketing system to send a "How did we do?" survey to the customer. Or an open API could allow a third-party analytics tool to query lead and job data. These aren't immediate UI features but part of the product's ecosystem readiness. The spec calls for API/Webhooks integration (F-011–F-016 covers Reporting & Public API) [106], which we will keep in mind during development.

- **Compliance:** If applicable, ensure we meet any regulatory requirements such as OSHA (for worker safety logs) or local labor laws (for time tracking). Likely not a huge concern in this domain, but our time tracking could assist in labor compliance (overtime alerts etc.).

- **Cross-Browser and Device Support:** As a PWA, it must work on modern browsers (Chrome, Safari, Firefox, Edge) and both Android and iOS devices. Test on various screen sizes. Particularly ensure iOS (which historically had PWA limitations) is fully supported for features like camera access and offline storage.

- **User Feedback Mechanism:** Within the app, allow users to send feedback or report issues (maybe a simple form to email support). This will help us catch bugs and improve usability continuously.

6. Data Flow & Integration Points

This section outlines how data moves through the system in common scenarios and how we integrate with other components of the Yardura platform:

- **Lead Acquisition Flow:** A new lead can enter the system either via (a) the **online quote form** on the website, or (b) a **Sales Rep manually entering it** after a cold call or canvass. In case (a), the website will POST to `/api/quote` and create a lead record in the DB (this exists today, returning a `leadId` per spec [96]). We will extend that endpoint or follow-up process to assign the lead to the appropriate rep and territory automatically (e.g., based on ZIP code). A webhook or backend event can notify the assigned rep's app in real-time ("New web lead in your area: John Doe") so they can follow up quickly. In case (b), when a rep enters a lead in the app, it's saved via our API, and possibly triggers an immediate welcome email or scheduling of a follow-up task. All lead data is stored in the central Postgres as part of the `Lead` table (new). The data includes all fields from the quote wizard (pet count, yard size, etc.) so the rep has full context [95]. If the lead converts, those fields map directly into the Customer and Job records.

- **Lead to Customer Conversion:** When a rep marks a lead as "Closed-Won" and fills in subscription details, the app calls Yardura's subscription creation flow. Likely, we'll integrate via an internal API that wraps Stripe subscription logic (creating the plan instance, capturing payment method through Stripe Elements). This will result in a new `Customer` record and associated `Job` setup for that customer. The lead record gets linked or flagged as converted (and possibly removed from active pipeline). Data like address and contact info flows from Lead into Customer entity so it's not re-entered. Also, any note or special instruction on the lead could carry to the customer profile for service reference.

- **Scheduling & Dispatch Data Flow:** The Admin uses the Dispatch UI to create jobs (for new customers) or adjust schedules. When a job is created, the system will generate the upcoming service visits (could be a cron-like generation or created one cycle at a time). Each service visit (with date, customerId, assigned techId, etc.) is saved in the DB (`ServiceVisit` table as per current domain). The night before (or whenever needed), the system can compute the optimal route order for each tech's next day visits and store an ordered list (or sequence number on each visit). In the morning, when a tech logs in, the app fetches their list of `ServiceVisit` for that day (with all related info: customer, job details). As they perform the route, each check-in and completion updates the `ServiceVisit` record (timestamps, status, notes). If offline, the update is queued; when online, the app sends a PATCH to the server to update the record. Completed visits also generate an event that could be used to trigger the billing if needed (though for recurring subscription, billing might not be per visit, but we might still log a record for invoicing reference or pay-per-visit customers).

- **Route Optimization Service:** Our app will interface with Google's Distance Matrix API to optimize routes. Likely the server will handle this (tech calls `POST /optimizeRoute` with their list of stops, server calls Google API, returns optimized order). We must cache results where possible (e.g. daily route that doesn't change won't need re-optimizing repeatedly). Also, use API keys securely from server side.

- **GPS Tracking Flow:** When tracking is on, the mobile app can send periodic location pings (say every X minutes or on significant move). These go to an endpoint like `/api/track` with userId, timestamp, lat/long. The server stores these in a `GPSLog` table (maybe ephemeral or last known location per user). The "Find My Team" feature on Admin query this data to show positions [22]. Breadcrumb trails can be reconstructed from these logs (or tech's check-in/out events could suffice for a coarse trail). This data may be purged after some time to protect privacy, unless needed for an investigation.

- **Photo Upload Flow:** Photos taken by tech are stored ideally in cloud storage (like an S3 bucket or similar) rather than in the database. The app will call an endpoint to get a pre-signed upload URL or use a direct upload approach. The photo metadata (URL, thumb, etc.) is then linked to the `ServiceVisit` record. This way, client portal can retrieve and display the gallery of photos per visit [112]. We ensure the naming or tagging of photos includes org and customer to segregate data. The upload may be multi-step if offline (store in IndexedDB then attempt upload later, mark as synced once done).

- **Notifications & Webhooks:** Use a notification service (could be built-in via Prisma events or a message queue) to handle triggers. E.g. a `ServiceVisit` marked complete triggers a function

that sends a completion email (with a link to photos) to the customer via Resend API [106]. Or a skip event triggers a Twilio SMS to the customer ("We apologize, your service was skipped today due to heavy rain. We will reschedule to tomorrow."). Similarly, creation of a new lead triggers a webhook if needed for external CRM integration or at least an internal event for assignment. We might implement a small event queue for these cross-system communications to decouple from the main request thread (improving performance).

- **Billing Cycle Flow:** For subscription customers, Stripe will charge them automatically on their schedule (monthly, etc.). Our system will need to handle any usage-based charges or credits. For example, if skip credits are a thing, we might adjust the invoice via Stripe API (coupons or invoice line items). If one-time jobs are done, the Admin might click "Bill Now" for that job, which calls our billing module to create an invoice in Stripe and possibly charge immediately if payment method on file. Payment status updates (success/failure) come via Stripe webhooks to our backend; if a payment fails, mark the customer as delinquent, which can reflect in the admin UI and potentially pause their scheduled jobs (spec F-007 mentions dunning logic [115]). Admin can then decide to suspend service until payment resolved.

- **Client Portal Integration:** The field app shares data with the client portal. After each visit, the portal (which the customer can log into) should show an update – e.g. "Service completed on Jan 10, 2025 – 2:45pm by Technician John. 2 photos attached. Notes: 'Yard cleaned, left gate as found.'" We will expose an API for the portal to fetch a customer's recent visits, or more directly, our system can insert records into the existing portal database/tables. The timeline gallery is specifically mentioned [112]; we'll fulfill that by ensuring each ServiceVisit has associated photos and text that the portal UI can display in chronological order.

- **Feedback Loop:** Optionally, after a visit is completed, we might want to gather customer feedback. This could be triggered by the completion event sending an email with a feedback form. If integrated, that feedback can be fed into the system for quality metrics (and possibly shown to the tech or admin). This is a future enhancement likely.

In diagram terms, one could imagine: **Sales Rep App** -> API -> DB (Leads, etc.) -> Conversion -> DB (Customers, Jobs) -> **Admin UI** (views jobs, schedules) -> Tech App (fetches jobs) -> completes jobs -> API -> DB (update) -> triggers notifications & billing -> updates reflect in **Client Portal** and reporting.

All components (Sales Rep, Tech, Admin, Portal) share the central database and services, preventing silos. By integrating at the data model level with minimal duplication (e.g., "LeadId" linking to "CustomerId" when converted, using the same Org and Customer tables as existing system), we keep data consistent.

7. Architecture & Technical Design

The Yardura Field Service App will use a modern, scalable architecture built on Yardura's existing platform. Below are key architectural decisions and components:

- **Front-End:** Next.js 13+ app using the **App Router** for a streamlined, server-components approach where useful. However, many features (maps, drag-drop scheduling, real-time updates) will rely on client-side interactivity, so we will use React with context/state management (perhaps Zustand or

Redux for global state like user session, online/offline status). We'll implement separate UI modules for:

- Sales (Leads, Map, Pipeline pages)
- Dispatch (Calendar/Board, Territory management pages)
- Technician UI (Daily route page, Job detail page, etc.)

These could be different route prefixes, or even different Next.js apps if needed for optimization, but likely one app that conditionally renders based on role. We can lazy-load chunks (e.g. the map module only when user navigates to Map page) to keep initial load fast.

- **Back-End:** Utilize Yardura's Node.js server (part of Next.js API routes or a separate Express server). Continue using **Prisma** to interact with the Postgres DB. We will add Prisma models for new entities:
- `Lead`, `LeadStage` (with relationship to `Customer` if converted, and to `User` (rep) and `Territory`).
- `Territory` (polygon or list of zip codes; could store the shape as GeoJSON or coordinates).
- `Job` (if not already present, though spec says a `Job` model exists – we might expand it).
- `ServiceVisit` (exists, possibly rename or keep; ensure it links to `Job`, `User` (tech), status, etc.).
- `Photo` or extend `ServiceVisit` with a list of photos (via related table).
- `Note` (could be a polymorphic entity attached to Lead, Customer, or ServiceVisit).
- `GPSLog` (userId, timestamp, lat, lng).
- `ChatMessage` (if chat implemented, or integrate with a third-party chat service to avoid building from scratch).
- Possibly `Route` if we want to store route plans separate from visits (Spotio has a concept of saved routes, but in our case the route is just the day's sequence of visits).

No changes to existing `Customer`, `User` (except maybe a role field), etc., aside from linking. A mapping from current schema to new is planned to be additive.

- **APIs:** Develop RESTful APIs for all main actions: creating leads, updating lead status, CRUD on territories, scheduling jobs, etc. Also provide some specialized endpoints: e.g. `GET /route/today?techId=__` returns today's visits and map of route, `POST /visit/{id}/complete` with payload of data (notes, photos metadata, etc.). Use appropriate HTTP methods and status codes. Protect endpoints with auth middleware checking user roles and org. The spec calls for HMAC + idempotency keys on webhooks – we will implement idempotency for critical operations (like to avoid duplicate job creation if a dispatch click happens twice, etc., using an idempotency key pattern). Also incorporate rate limiting to prevent abuse (especially on public API endpoints if exposed).

- **Real-time updates:** For features like live notifications or dispatch updates, we can use WebSockets or Web Push. Since Next.js can incorporate WebSocket easily (maybe via libraries like Socket.IO or using Next.js built-in support in API route), we might set up a simple Socket server. When an admin changes a schedule, the server emits an event to the relevant tech's channel, and the PWA (if online) updates the UI instantly. Similarly, location pings from tech could be broadcast to an admin dashboard map if it's open (for a live "moving" effect). If WebSockets are too heavy to start, we can use frequent polling for the tech app (poll every minute for changes) and rely on push notifications for important alerts.

- **Integration Services:**

- *Maps:* The server will have a module to call Google Maps API for geocoding addresses when leads are added (store lat/long for mapping) and for route optimization on request [106] . We'll secure the API key on server side. Use caching (e.g. if we geocoded that address before, reuse it).
- *Twilio:* A server module or microservice listens for events like "send SMS" and uses Twilio API (account SID/auth token stored securely) to send messages [106] . Or we can use a service like Twilio Notify for templated messages. These calls should be async (we don't block user action waiting on SMS success; just log it).
- *Stripe:* Use Stripe's Node SDK to manage subscriptions. Already likely in place from current portal. We will call Stripe to e.g. create a subscription when a rep closes a sale, if the current flow is only via portal. Possibly we instead flag that lead as ready and have the customer finalize payment via portal link – but ideally the rep can input card details on the spot (maybe using a Stripe Elements form on their tablet). That might be in scope since spec F-002 mentions SetupIntent for onboarding [95] .

- *Auth:* Reuse NextAuth sessions; ensure role info is in the JWT or session such that the frontend can conditionally render, and the backend can authorize. Possibly expand the user model with a role field and territory assignment, manageable by admin.

- **Cross-Platform Strategy:** The PWA will be thoroughly tested on phone browsers. If needed for distribution, wrapping it with **Capacitor** could allow using device features like background geolocation or push notifications with native support. However, given advancements, we try PWA first. If push notifications on iOS PWA are unreliable, we consider Apple Push via a native wrapper. At this stage, focusing on PWA keeps dev velocity high and aligns with the web tech stack.

- **DevOps:** We will use Yardura's CI/CD pipeline to deploy updates. Ensure database migrations for new tables are included (with backwards compatibility where possible). Also plan a roll-out strategy: e.g., run beta with internal users (maybe one franchise) before full deployment.

8. Milestones & Implementation Plan

To manage development, we propose breaking into phases with milestones:

Milestone 1: Core Framework & Basic Data Models (4-6 weeks)
- Set up new database tables for Leads, Territories, Jobs (if needed), ServiceVisits (extend as needed), etc. - Implement user roles/permissions in the auth system. - Basic Admin web UI skeleton: can create a territory, add a lead, schedule a job (no optimization yet). - Basic Tech PWA skeleton: can log in and see dummy route/jobs. - Map integration minimal: display map with one pin to test. - This milestone focuses on foundational plumbing – making sure data flows from front to back.

Milestone 2: Sales Module Completion (4 weeks)
- Finish Lead Management: lead forms, lead list, pipeline view for reps and admin. - Territory mapping UI: allow drawing/selecting territories on map, assign reps. - Sales rep map view with leads plotted and colorized by status. - Calendar integration for reps (auth with Google/Microsoft for a test user). - Implement Autoplay creation and assignment (basic version). - Test the conversion flow creating a Customer and Job (without Stripe hooking fully). - By end, sales reps can use the system to log leads and manage their sales funnel on the map.

Milestone 3: Dispatch & Routing Module (6 weeks)
- Build the Dispatch Calendar/Board: display recurring jobs, allow drag-drop reschedule, assign techs. - Integrate Google API for route optimization; implement route planning logic for tech's daily visits [107] [106]. - Field Tech app: develop the route list UI, check-in and completion flow, photo capture with offline storage. - Offline mode testing and implementation (Service Worker setup, sync logic). - Notifications: set up Twilio and email notifications for key events (on-the-way SMS, completion email). - Ensure skip logic: tech can mark skip, reason flows to admin, triggers customer notify and flags billing. - This milestone is the heavy lift of service-side features. By end, we aim for a tech to go through a full day route from assignment to completion offline, and admin to manage schedule changes.

Milestone 4: Integration & Polish (4 weeks)
- Stripe integration: allow reps to take payment info on conversion (or generate link for customer to pay); ensure billing events (one-time jobs, skip credits) properly applied [115]. - Client portal hooks: make sure service data (visits, photos, notes) appears in portal timeline (this might require deploying updated portal or at least populating the same tables the portal reads). - Add reporting dashboards for admin: at least a couple of key charts (lead conversion, jobs done, etc.). - Implement team chat or a simpler communication tool if decided (could integrate with an external chat API to save time). - Cross-browser/device testing and UI refinements (make sure it looks good on iPhone Safari, Android Chrome, etc.). - Security audit: test role restrictions, inject some bad data to see if system holds up. - Performance tuning: load test route optimization with 100 addresses (validate we meet $\leq$10s), test maps with high pin count, etc.

Milestone 5: Beta Launch & Feedback (2 weeks)
- Release to a small group (maybe internal or a pilot franchise) to gather feedback. - Fix any major bugs, tweak UX based on real-world use (for example, maybe techs want an easier way to call customers – we might add a button). - Verify that all parts of the system (sales to service to billing) work together smoothly in a production-like environment.

Milestone 6: Full Launch
- Roll out to all users, provide training to Sales Reps and Technicians (probably through documentation or brief videos, highlighting new app features). - Post-launch, monitor usage and performance; be ready to quickly patch any issues (like if offline sync has an edge-case bug). - Begin collecting enhancement requests for future versions (e.g., advanced wellness metrics logging, customer self-scheduling through portal hooking into dispatch).

By following this phased approach, we ensure critical functionality (like scheduling and service logging) is delivered as early as possible (Milestone 3), while still giving attention to the sales side that adds growth value. Spotio's features are largely covered by Milestone 2 (sales) and some parts of 3 (route planning, tracking). The enhancements for Yardura's domain (recurring jobs, offline PWA) are addressed in Milestone 3 and 4.

Throughout development, we will keep referring to Spotio and competitors as a benchmark for UX and features to make sure we meet or exceed them. For example, after Milestone 2, we might compare our sales rep UX to Spotio's known pros (e.g. ease of use, automatic logging – ensure our call/email logging is equally automatic and simple) [28]. After Milestone 3, compare our dispatch tools to typical field service apps (Badger's limits in scheduling, etc.) to confirm we built a superior solution for recurring services (which we expect, since competitors don't focus there).

9. Conclusion

By fusing Spotio's field sales capabilities with industry-specific field service features, the Yardura Field Service App will enable a true end-to-end "service operating system" for pooper scooper businesses. Sales reps will **prospect smarter and close more deals** with tools for territory visualization, qualified lead data, and automated follow-ups [1] [36]. Field technicians will **deliver service efficiently** through optimized routes, real-time communication, and easy logging of work (with photos and GPS verification) [22] [111]. Administrators will have **unprecedented visibility and control**, from a bird's-eye view of all territories and activities to the minute-by-minute location of their team, plus integrated billing and performance analytics to drive decisions [2] [8].

Benchmarking against Badger Maps and SalesRabbit has shown that while those tools cover either routing or basic tracking, **none combine both sales and service management** in one platform – and none are tailored to the nuances of recurring home services. This is Yardura's opportunity to lead the market. By addressing the noted gaps (e.g., competitors' lack of multi-channel logging, limited scheduling depth, etc. [27] [80]) and by leveraging Yardura's existing tech ecosystem (Next.js, Prisma, Stripe), we will deliver a powerful yet user-friendly solution.

Ultimately, success will be measured by improved metrics for Yardura and its clients: faster customer acquisition (through better canvassing tools), higher retention (through consistent service quality and communication), and operational efficiency gains (through route optimization and automation). This PRD provides the blueprint to achieve those outcomes, positioning Yardura as the technology leader in an unexpected but valuable domain – turning "spots on a map" into delighted customers and well-serviced yards.

Sources:

- Spotio Feature Overview and official documentation for capabilities like territory mapping, routing, GPS tracking, Autoplays, and reporting [121] [22] [8] [42].
- MapMyCustomers review of Spotio for summary of features (territory mapping, activity tracking, pipeline, etc.) [1] [2].
- Spotio Knowledge Base articles on Territory Management and Routing for implementation details (drawing territories, adding routes on web/mobile, recurring routes, etc.) [6] [18].
- Spotio vs. SalesRabbit comparison highlighting Spotio's advantages in multi-channel tracking and reliability [27] [85].
- Spotio blog on Badger Maps noting Badger's focus and limitations in routing and reporting [74] [80].
- Yardura Friendly Spec provided, which outlines required features (dispatch board, field PWA with GPS, photos, offline, etc.) and architectural context [122] [111] [106].
- Additional context from competitor insights and user reviews to ensure Yardura's solution closes feature gaps and leverages best practices in field sales/service software [123] [124].

---

[1] [2] [3] [4] [53] [63] [64] Everything You Need to Know About Spotio (Pros & Cons) - Map My Customers
https://mapmycustomers.com/everything-you-need-to-know-about-spotio/

[5] [27] [28] [40] [41] [57] [58] [70] [71] [85] [86] [87] [88] [89] [90] [91] [92] [97] [124] Why SPOTIO over SalesRabbit
https://spotio.com/spotio-vs-salesrabbit/

6 7 9 11 47 48 98 99 100 114 Territory Management: Creating and Managing Territories – SPOTIO
https://support.spotio.com/hc/en-us/articles/360005805374-Territory-Management-Creating-and-Managing-Territories

8 10 12 15 16 21 22 23 24 25 26 34 35 36 37 38 39 42 43 49 50 51 52 54 55 56 66 67 93 94 101 116 121 Feature Overview - SPOTIO
https://spotio.com/feature-overview/

13 14 17 18 19 20 102 103 Routing: How to Build and Manage Routes – SPOTIO
https://support.spotio.com/hc/en-us/articles/360061370754-Routing-How-to-Build-and-Manage-Routes

29 30 31 32 33 Reading the Pipeline – SPOTIO
https://support.spotio.com/hc/en-us/articles/360009995493-Reading-the-Pipeline

44 45 46 113 Email and Calendar Integration – SPOTIO
https://support.spotio.com/hc/en-us/articles/8108577621911-Email-and-Calendar-Integration

59 60 61 62 68 69 72 73 104 123 Why Choose SPOTIO | SPOTIO Vs. Alternatives
https://spotio.com/spotio-vs-alternatives/

65 74 75 76 78 79 80 81 82 83 Badger Maps: Features, Pricing, Pros & Cons
https://spotio.com/blog/badger-maps/

77 Sales - Badger Maps
https://www.badgermapping.com/sales/

84 SalesRabbit Field Sales Software: Review, Features & Better Options
https://www.knockbase.com/blog/the-ultimate-guide-to-salesrabbit-for-door-to-door-sales-teams

95 96 105 106 107 108 109 110 111 112 115 117 118 120 122 Yardura_Friendly_Spec (1).md
file://file-N4QFZU8PYPJNmXCvV95KZo

119 SPOTIO
https://support.spotio.com/hc/en-us