OBDscribe – Settings Page Overview (v0.1)

What the Settings area is for

The Settings area centralizes configuration for a specific shop and user, so you don't have to hard■code values or edit the database directly. It gives a simple UI for changing how reports are labeled and how the AI behaves by default.

There are two main sections on /app/settings:

1) Shop Settings
- Backed by the Shop row in Postgres/Prisma.
- Editable through the ShopSettingsPanel client component.
- Also exposed via the REST endpoint: /api/settings/shop (GET, PATCH).

Fields:
• Display Name: Human■friendly name for the shop that can be shown in reports (e.g., "Demo Auto Repair – Test").
• Phone: Optional phone number for the shop.
• Address: Optional address block for the shop.
• Default Report Mode: Controls which AI model profile to use by default when generating reports (e.g., "standard" vs "premium").
• Default Report Tone: Tells the AI whether to write in plain English for customers or in a more technical tone for internal use ("plain_english" vs "technical").
• Include Maintenance Suggestions by Default: If true, the AI is asked to include maintenance suggestions in every report unless the user overrides it for a single run.

How it affects report generation:
- When a new report is generated via /api/generate-report, the API loads the shop settings and uses them as defaults:
  • mode = request.mode ?? shop.defaultReportMode ?? "standard"
  • tone = request.tone ?? shop.defaultReportTone ?? "plain_english"
  • includeMaintenance = request.includeMaintenance ?? shop.defaultIncludeMaint ?? true
- Those values are passed into the AI engine, which uses them to shape the JSON report (tech view, customer view, and any maintenance suggestions).

2) User Settings
- Backed by the User row in Postgres/Prisma.
- Editable through the UserSettingsPanel client component.
- Also exposed via the REST endpoint: /api/settings/user (GET, PATCH).

Fields:
• Display Name: A short, human■friendly label for the logged■in user (e.g., "Kevin N"). This can be used later in the UI (for greetings, audit logs, or signature lines in reports).

How the data flows (high■level):
---
1. The logged■in user opens /app/settings.
2. The server■side SettingsPage verifies the session and renders two client panels: ShopSettingsPanel and UserSettingsPanel.
3. Each panel loads current values from its API endpoint using fetch (GET).
4. When the user edits fields and clicks Save, the panel sends a PATCH request to its endpoint with the updated values.
5. The API validates and persists changes through the settings helpers in src/lib/settings.ts, which call Prisma under the hood.
6. On the next report generation, /api/generate-report reads the stored shop settings and uses them as defaults for mode, tone, and maintenance behaviour.

Why this matters for v1 demos:

- Shows that OBDscribe isn't a hard■coded demo – shops can configure their own branding and report defaults.
- Demonstrates a clean separation between UI, API, and persistence: settings.ts helpers, API routes, and React panels.
- Provides a foundation for future settings (invoice options, logo uploads, notification preferences, etc.) without changing the core pattern.