# Code Jam: Fuel tracker

Task is inspired by one of the tasks from book by Charles Wetherell – “Etudes for Programmers”.

**Fuel Tracker — MVP Requirements Document**

**1. Overview**

**Purpose.** Minimum viable product (MVP) for a web application that helps drivers record fuel fill-ups and analyze consumption and costs over time.

**High-level vision.** Users register with email/password, log each fill-up from a receipt (station name, fuel brand/grade, total paid, liters), record odometer (mileage), and view history plus statistics (e.g., cost per liter, consumption, distance per liter, cost per km/mi, rolling averages, per-brand comparisons).

**Primary user.** Individual driver managing their own vehicle data (multi-car support included in MVP).

**Assumptions.**

* The first entry for a given car acts as a baseline (odometer/date).
* Odometer values are monotonically increasing per car.
* Base units are metric (liters, kilometers, currency as entered). Optional Imperial presentation (gallons/miles) is a view-time conversion.
* IMPORTANT: In MVP: All fuel ups are full fuel ups

**Out of scope for MVP.**

* OCR from receipt images.
* Shared accounts, fleet features, or multi-user orgs.
* Trip logging, route tracking, or telematics integrations.
* Tax reports and expense approvals.

**2. Roles & Permissions**

* **Authenticated User**
  + Full CRUD on their own fuel entries, cars, and profile.
  + Read-only statistics and history computed from their data.
* **Anonymous Visitor**
  + May access the landing page and sign-up/sign-in flows only.
* **Admin (future / non-MVP)**
  + Not required for MVP; use developer tools/database for support.

**3. Functional Requirements**

**3.1 Authentication & Account**

* **Sign up with email + password**
  + Validate email format and uniqueness.
  + Password policy: minimum 8 characters; must include at least 1 letter and 1 number (unicorn blood is not included into MVP).
  + Email verification not required in MVP scope.
* **Sign in / Sign out**
  + Session persists until explicit logout or expiration.

**Acceptance Criteria**

* Given an unused email, when the user signs up with a valid password, an account is created and they are signed in.
* Sessions are invalidated on logout.

**3.2 User Profile & Settings**

* **Profile fields:** display name (optional), preferred currency (ISO code, default from locale), preferred distance unit (km or mi), preferred volume unit (L or gal), time zone (default browser).
* **Unit system & conversions**
  + Storage remains metric (km, L); view converts to user preferences.
  + 1 US gallon = 3.78541 L (documented); 1 mile = 1.60934 km.

**Acceptance Criteria**

* Changing units in Settings updates all calculated stats and labels across the app without altering stored canonical data.

**3.3 Vehicle Management (Multi-Car)**

* Add / edit / delete vehicles.
* Vehicle fields: name/label (required), make/model (optional), year (optional), fuel type (optional).
* Each fuel entry is tied to a specific vehicle.
* Statistics and dashboards can be scoped to one vehicle or aggregated across all.

**3.4 Fuel Entry Management**

* **Create Fill-Up**
  + Required fields: vehicle, date, odometer reading, station name, fuel brand, fuel grade, quantity (liters), total amount (currency).
  + Notes field (optional, max 500 chars).
* **Edit / Delete Fill-Up (user’s own entries).**
* **History list & detail view**
  + Sort by date desc.
  + Filters: date range, brand, grade, station, vehicle.
  + Pagination (e.g., 25 per page) or infinite scroll.
* **Data integrity rules**
  + For entries after the first for a vehicle: odometer must be greater than the vehicle’s previous entry’s odometer.
  + Prevent negative or zero liters/amount; prevent future dates beyond today.
  + When an entry’s date is earlier than an existing later entry, re-compute derived stats accordingly (see 3.5).

**Acceptance Criteria**

* Creating an entry with valid values saves successfully and appears in history.
* Editing an entry updates history and statistics consistently.
* Deleting an entry re-computes stats and distances between adjacent entries.

**3.5 Derived Metrics & Calculations**

All metrics are computed **per vehicle**. Aggregations can also be calculated across all vehicles when selected in the dashboard.

**Per-fill Computed Fields (for each entry after the baseline)**

* **distance\_since\_last** = current\_odometer − previous\_odometer (for the same vehicle).
* **unit\_price** = total / liters.
* **Efficiency**
  + Metric view: consumption L/100km = (liters / distance\_since\_last) × 100, if distance > 0.
  + Imperial view: MPG = distance\_since\_last (mi) / volume (gal).
* **Cost metrics**
  + cost\_per\_km = total / distance\_since\_last (if distance > 0).
  + cost\_per\_mile (converted when in Imperial view).

**Aggregations**

* **Rolling window (default last 30 days; configurable)**
  + average cost per liter.
  + average consumption (L/100km or MPG).
  + average distance per day.
  + average cost per km.
  + total spend & total distance.
* **Per brand and per grade**
  + All-time averages (consumption, cost per liter, cost per km).
  + Number of fill-ups per brand/grade.

**Rounding Rules**

* Currency display: 2 decimals (locale-aware).
* Volumes: 2 decimals.
* Prices per liter: 2–3 decimals (configurable, default 2).
* Efficiency (L/100km): 1 decimal.
* MPG: 1 decimal.
* Distances: integer km/mi.

**Acceptance Criteria**

* After adding two or more entries for a vehicle, the app displays per-fill metrics and both rolling (e.g., 30-day) and all-time aggregates.
* Per-brand averages update as soon as data changes.
* Switching unit system flips between L/100km and MPG correctly.

**3.6 Statistics & Dashboards**

* **Overview Dashboard**
  + Cards: current rolling average consumption, rolling average cost per liter, total spend (selected period), total distance, average cost per km, average distance/day.
  + Chart 1: cost per liter over time (line).
  + Chart 2: consumption (L/100km or MPG) over time (line).
  + Period selector: last 30/90 days, year-to-date, custom range.
  + Vehicle selector: per vehicle or all vehicles combined.
* **Brand/Grade Comparison**
  + Table: brand × (avg cost per liter, avg consumption, number of fill-ups).

**Acceptance Criteria**

* Selecting a period or vehicle updates all widgets and charts.
* Empty-state messaging appears when the selected range has no data.

**4. Non-Functional Requirements**

**4.1 Security & Privacy**

* Store passwords using strong one-way hashing.
* Session management: HTTP-only, Secure cookies.
* Data isolation: users can access only their own data (enforced at query layer).
* PII: store only necessary fields (email). Provide account deletion (hard-delete user data).
* Log in DB authentication events (sign-in, password reset requests).

**4.2 Performance**

* Pages render first meaningful content within 2 seconds on a typical broadband connection and mid-range laptop/phone.
* History and dashboard queries for up to 5,000 DB entries should complete within 500 ms (p95) server-side.

**4.3 Reliability**

* Error handling with user-friendly messages; do not expose stack traces to users.
* Server errors logged with correlation IDs.

**4.4 Compliance & Legal (Foundational)**

* Provide Terms of Service and Privacy Policy pages (basic templates).
* GDPR-friendly basics: data export (CSV) and deletion on request.

**4.5 Observability**

* Application logs with user ID (hashed or internal ID) and correlation ID.

**4.6 Browser & Device Support**

* Latest two major versions of Chrome.
* Responsive design for mobile (≥360px width) and desktop (≥1280px).

**4.7 Other**

* Code is readable, maintainable, and consistent with common style guidelines.
* Implementation follows best practices of the chosen frameworks; avoid exotic or unsupported patterns.
* Similar features are implemented in a uniform way to reduce cognitive load.
* Solution is production-ready: secure by design, prevents data leaks, enforces strict tenant (user) isolation.
* Architecture remains simple and understandable, preferring straightforward solutions over unnecessary complexity.

**Important assumption**: Code will be deploy to production tommorrow and next day after deploy a new team will start adding new feaetures.

**5. Screens (MVP)**

1. Landing / Home (logged out)
2. Sign Up
3. Sign In
4. Dashboard (logged in, with vehicle selector)
5. Vehicle Management (list, add, edit, delete)
6. Add / Edit Fill-Up
7. History
8. Statistics — Brand/Grade
9. Settings / Profile
10. Legal Pages
11. Error Page

**6. UX Notes**

* Use inline validation and helper text on the Add/Edit form (e.g., show computed unit price and tolerance diff).
* Sticky “Add Fill-Up” button on screen.
* Clear tooltips/glossary for metrics (e.g., “L/100km: lower is better”).
* Default period for dashboard: Last 30 days; allow quick toggles (30/90/YTD/Custom).
* Vehicle selector present in dashboard, history, and statistics views.

**7. Implementation Guidelines (MVP)**

* Backend: REST (Java/Spring Boot or Node.js/NestJS or .Net), Postgres.
* Auth: session cookies with CSRF.
* Frontend: Angular 2 / ReactJS with responsive UI; client-side form validation + server validation.
* DB schema management: Use Liquibase or similar tool for DB auto migrations.
* DevOps: Dockerized service. User docker-compose.yaml for local run
* Migrations & seeds: baseline admin/dev users only for development.

**8. Deployment**

* Application should be runnable locally by running:
  + Git clone …
  + Docker compose up
* Docker compose should start service together with DB. Service should apply DB schema on start and seed DB with data.

**9. Definition of Done (MVP)**

* All screens above implemented and reachable.
* Core flows covered by integration tests (auth, add/edit/delete entry, stats display).
* Performance and validation requirements met.
* Application is runnable after clean `git clone & docker compose up` Please check this before handover.
* **All features from the doc above is implemented**