now give me the full details and explainations and all the details of this application that i will give to the team so they have no question for my anymore. fill the gabs

This comprehensive **Product Requirement Document (PRD) & Technical Specification** is designed to be handed directly to your development team. It covers the **business logic**, **architecture**, **database schema**, **API design**, and **implementation strategy** to leave zero ambiguity.

Project Name: Modular Travel Price Estimator (Core & Casablanca Pilot)

1. Executive Summary

We are building a **transportation pricing engine** that estimates travel costs between two points.

- **Core Value:** It is **city-agnostic** and **mode-agnostic**. The same engine must calculate a *Casablanca Petit Taxi* fare today and a *Paris Metro* fare tomorrow just by changing a configuration file (a "Pricing Profile"), not the code.
- **Pilot Phase:** The first implementation will be for **Casablanca Small Red Taxis** using official metering rules.

2. Functional Requirements

2.1. User Stories

1. **As a user**, I want to enter my **Origin** and **Destination** (via map or text) so I can see the route.
2. **As a user**, I want to select a **Transport Mode** (default: "Small Red Taxi") to get a price specific to that vehicle type.
3. **As a user**, I want to specify the **Time of Travel** (Now or Future) because night fares differ from day fares.
4. **As a user**, I want to see a **Price Breakdown** (Base fare + Distance cost + Night surcharge) to understand why it costs that much.

2.2. The "Casablanca Red Taxi" Business Logic (The Pilot Rule)

The team must implement this specific logic as the first "Profile":

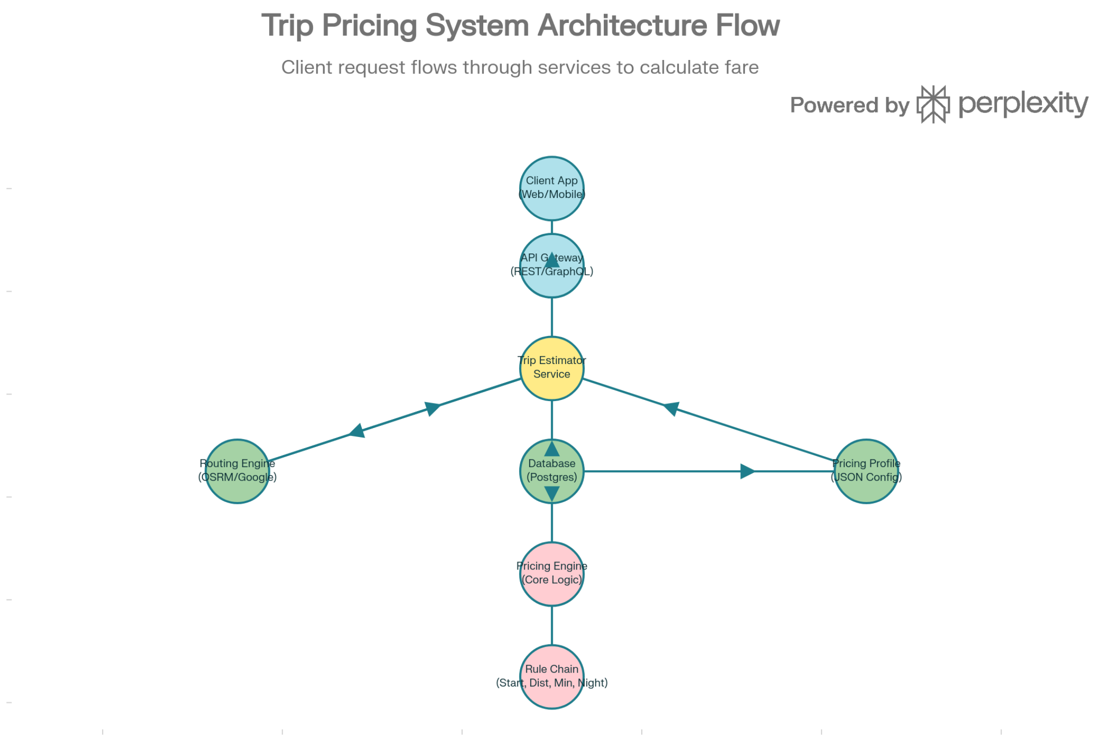
- **Currency:** MAD (Moroccan Dirham).
- **Start Fare (Flag fall):** 2.00 MAD fixed upon entry.
- **Distance Calculation:**
  - The meter increments by **0.20 MAD** for every **80 meters** traveled.
  - Formula: `Steps = CEIL(TotalDistanceMeters / 80)`, then `DistanceCost = Steps * 0.20`.
- **Minimum Fare:** The total trip cost **cannot be less than 7.50 MAD** (Daytime). If calculation < 7.50, Price = 7.50.
- **Night Surcharge:**
  - **Condition:** Travel time is between **20:00 (8 PM)** and **06:00 (6 AM)** (Configurable hours).
  - **Effect: +50%** on the final calculated fare (including the minimum).
  - *Example:* If Day Fare = 10 MAD, Night Fare = 15 MAD. If Day Fare = 6 MAD (bumped to 7.50 min), Night Fare = 7.50 * 1.5 = 11.25 MAD.

3. Technical Architecture

3.1. System Overview

The system is divided into three distinct layers:

1. **Routing Layer:** Calculates distance and duration (Google Maps API / Mapbox / OSRM).
2. **Pricing Engine (The Core):** A "dumb" calculator that takes `Distance`, `Time`, and a `Ruleset` to output a `Price`.
3. **Configuration Layer (The Brains):** Stores the specific rules for each city/mode in the database, not in the code.


Trip Pricing System Architecture Flow
Client request flows through services to calculate fare
Powered by perplexity
Client App (Web/Mobile)
API Gateway (REST/GraphQL)
Trip Estimator Service
Routing Engine (OSRM/Google)
Database (Postgres)
Pricing Profile (JSON Config)
Pricing Engine (Core Logic)
Rule Chain (Start, Dist, Min, Night)

System Architecture Diagram for Modular Travel Estimator

3.2. Modular Design Strategy (The "Strategy Pattern")

Do not write `if (city == 'Casablanca')` in the code. Instead, use a **Rule-Based Pipeline**.

- **The Engine:** `calculatePrice(context, profile)`
- **The Profile:** A JSON object loaded from the DB containing specific parameters for the city.
- **The Pipeline:** The engine runs a sequence of small "Rule" functions defined in the profile.

4. Database Schema (PostgreSQL)

This schema allows you to add "Tangier" or "Marrakech" later without changing a single line of code.

Table: `cities`

Column	Type	Description
`id`	UUID	PK
`name`	VARCHAR	e.g., "Casablanca"
`currency_code`	VARCHAR	e.g., "MAD"
`timezone`	VARCHAR	e.g., "Africa/Casablanca"

Table: `transport_modes`

Column	Type	Description
`id`	UUID	PK
name	VARCHAR	e.g., "Small Red Taxi"
`slug`	VARCHAR	e.g., "petit_taxi_red"
`icon_url`	VARCHAR	URL to taxi icon

Table: `pricing_profiles` (The most important table)

Column	Type	Description
`id`	UUID	PK
`city_id`	FK	Link to Casablanca
`mode_id`	FK	Link to Red Taxi
`pricing_strategy`	ENUM	'METERED', 'FIXED_ZONE', 'HYBRID'
`active`	BOOLEAN	true
`rules_config`	JSONB	**Stores all the math parameters (see below)**

JSON Structure for `rules_config` (Casablanca Pilot)

This is exactly what the developers should put in the DB for the first row:

```
{
  "base_fare": 2.00,
  "minimum_fare": 7.50,
  "distance_step_meters": 80,
  "price_per_step": 0.20,
  "night_surcharge_percent": 50,
  "night_start_hour": 20,
  "night_end_hour": 6,
  "enabled_rules": ["BASE_FARE", "DISTANCE_STEP_CALC", "MINIMUM_CHECK", "NIGHT_MULTIPLIEF
}
```

5. API Specification (REST)

Endpoint: `POST /api/v1/estimate`

Request Body:

```
{
  "origin_lat": 33.5731,
  "origin_lng": -7.5898,
  "dest_lat": 33.5898,
  "dest_lng": -7.6030,
  "city_slug": "casablanca",
  "transport_mode": "petit_taxi_red",
  "travel_time": "2026-01-22T22:30:00Z"
}
```

Logic Flow (Backend):

1. **Fetch Profile:** Look up `pricing_profiles` where city="casablanca" AND mode="petit_taxi_red".
2. **Route:** Call Map Provider (e.g., OSRM) → Returns `distance_meters = 4500` (4.5 km).
3. **Parse Time:** Check `travel_time`. Is it between 20:00 and 06:00? Yes → Night Mode = True.
4. **Execute Pricing Pipeline:**
   - *Rule 1 (Base):* Start at **2.00**.
   - *Rule 2 (Distance):* `ceil(4500 / 80) = 57 steps`. `57 * 0.20` = **11.40**.
   - *Subtotal:* 2.00 + 11.40 = **13.40**.
   - *Rule 3 (Min):* Is 13.40 < 7.50? No. Keep **13.40**.
   - *Rule 4 (Night):* Night is True. `13.40 + (13.40 * 0.50)` = **20.10**.
5. **Return Response.**

Response Body:

```
{
  "estimated_price": 20.10,
  "currency": "MAD",
  "distance_km": 4.5,
  "duration_min": 12,
  "is_night_fare": true,
  "breakdown": {
    "base_fare": 2.00,
    "distance_fare": 11.40,
    "night_surcharge": 6.70
  }
}
```

6. Implementation Steps (Roadmap)

Phase 1: Setup & Data (Days 1-2)

- Initialize project repo (Node.js/Python).
- Set up PostgreSQL database with the schema above.
- Seed the database with the **Casablanca Petit Taxi Profile**.

Phase 2: The Logic Engine (Days 3-4)

- Implement the `PricingStrategy` interface.
- Write the `MeteredStrategy` class that reads the JSON config and executes the math.
- **Unit Test:** Write a test case for a 1km trip (Day vs Night) to prove the math matches the official rules.

Phase 3: Routing Integration (Day 5)

- Connect a routing service (Mapbox free tier or OSRM) to get real distances.
- **Fallback:** If routing fails, allow manual distance input for testing.

Phase 4: API & Frontend (Days 6-7)

- Expose the API endpoint.
- Build a simple React/Vue form:
  - Input: Origin/Dest (Autosuggest).
  - Toggle: "Leave Now" or "Set Time".
  - Display: Big price tag with a "Night Rate Applied" badge if applicable.

7. Q&A for Developers (Pre-empting questions)

Q: What if the government changes the price to 8.00 MAD?
A: You do **not** change code. You update the `minimum_fare` value in the `pricing_profiles` table in the database.

Q: Can we support "Grand Taxi" (Shared White Taxi) later?
A: Yes. You will add a new `transport_mode` ("Grand Taxi") and a new `pricing_profile` with `pricing_strategy = 'FIXED_ROUTE'` (a different strategy class we will build later).

Q: How accurate is the distance?
A: We rely on the map provider. We are estimating the *official meter price*, not traffic delays (unless we add a "time-based" rule later, which our modular JSON supports).

Q: Why 0.20 per 80m and not 2.50 per km?
A: Because the physical meter clicks in steps. 79 meters costs 0, 81 meters costs 0.20. We must simulate the **steps** (`ceil(dist/80)`) to be legally accurate, not just multiply by km.