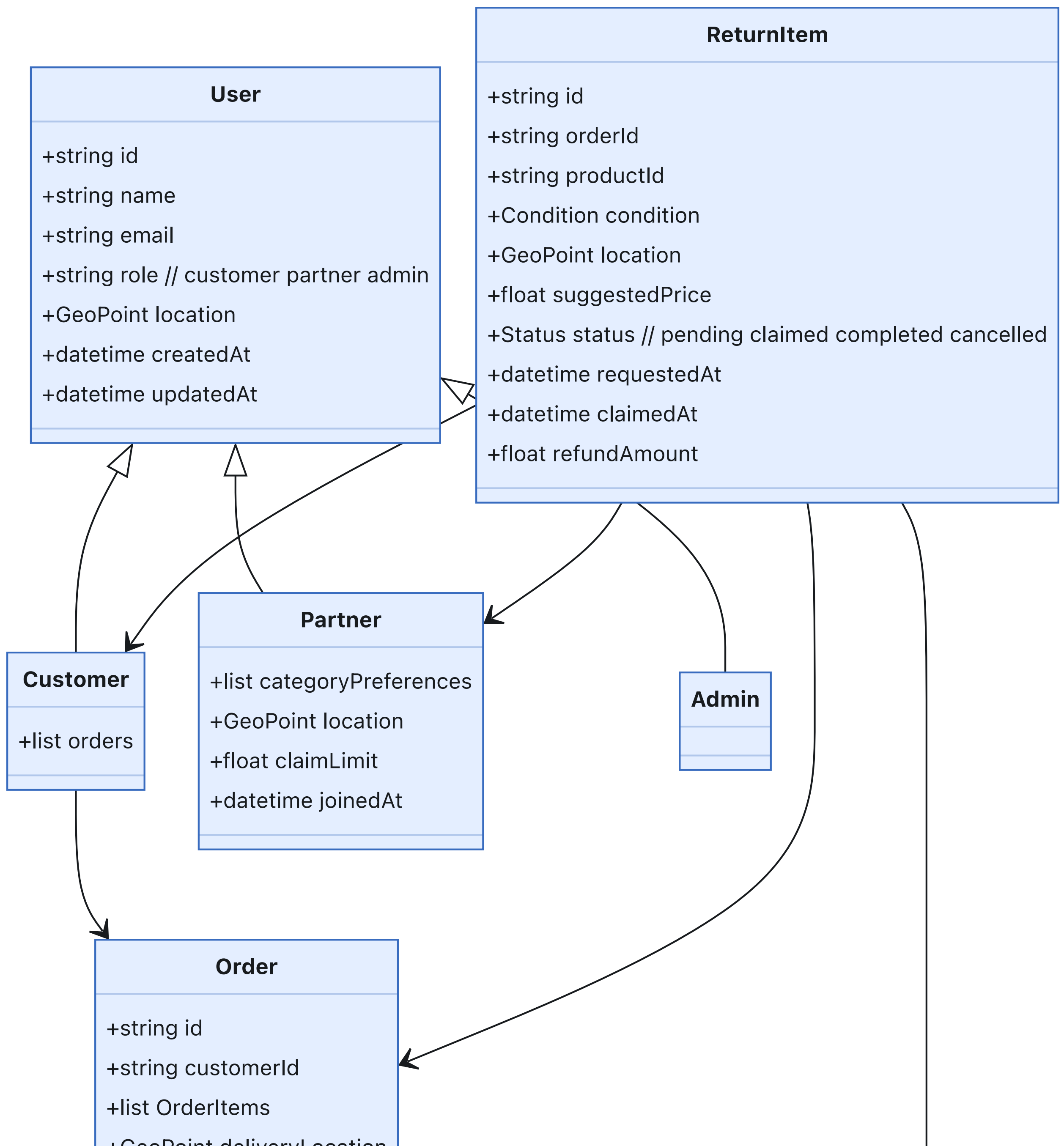
📝 Project Overview and Summary

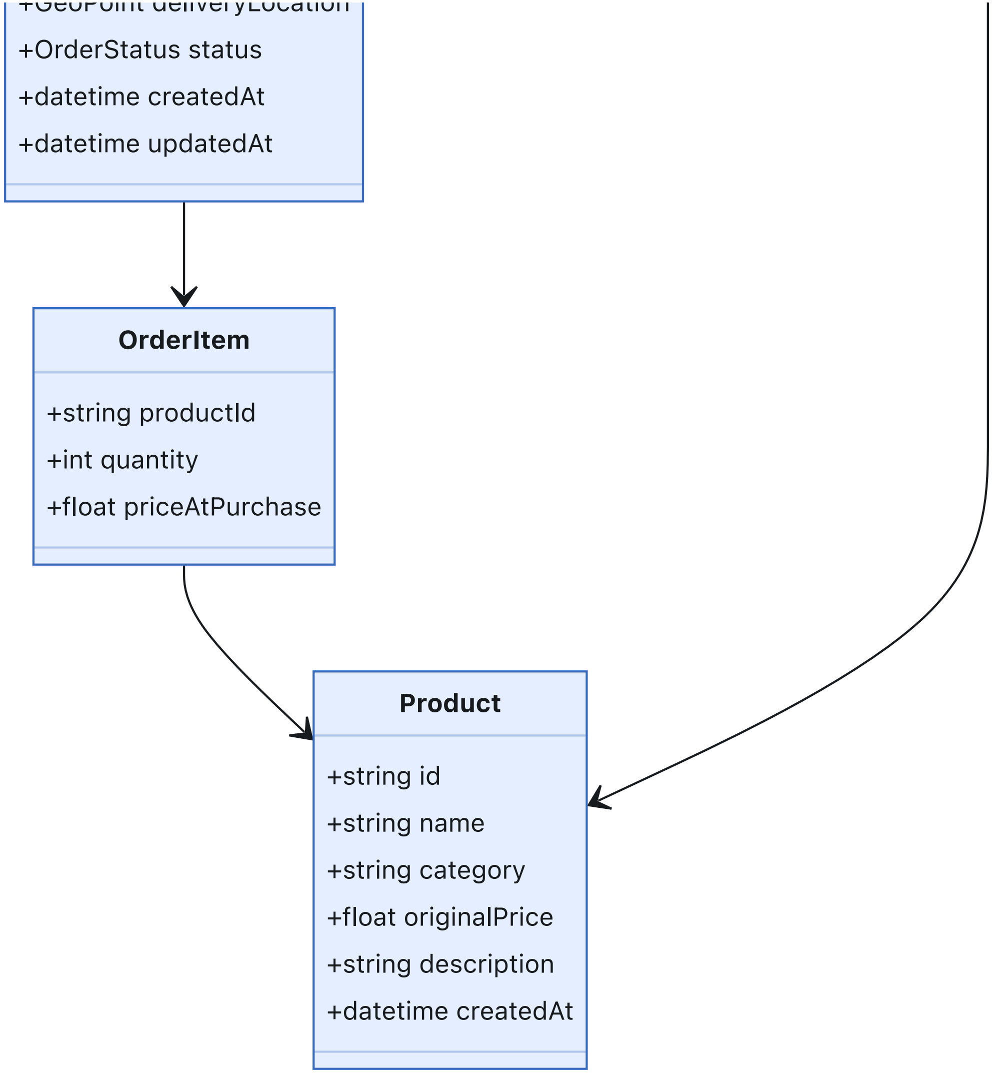
This project proposes a **hyperlocal returns marketplace** to reduce reverse logistics costs and waste.
It matches returned items to nearby partners using geospatial data and applies dynamic pricing to maximize recovered value.
The system keeps backend services modular, allowing future AI/ML enhancements, while the frontend cleanly separates customer, partner, and admin views.

🧩 UML Class Diagram Overview

We model the following entities to cover the full return journey:
- **User**: base class with role and geolocation, sub-classed into Customer, Partner, and Admin.
- **Customer**: has Orders.
- **Partner**: local business/reseller with category preferences and claim limits.
- **Product**: core item with name, category, price, and description.
- **Order**: placed by Customer, containing OrderItems.
- **OrderItem**: specific quantity and price per Product.
- **ReturnItem**: created when customer initiates return; links back to Order and Product, and can be claimed by Partner.

Includes extra fields for backend and data auditing: timestamps, status enums, refundAmount, etc.



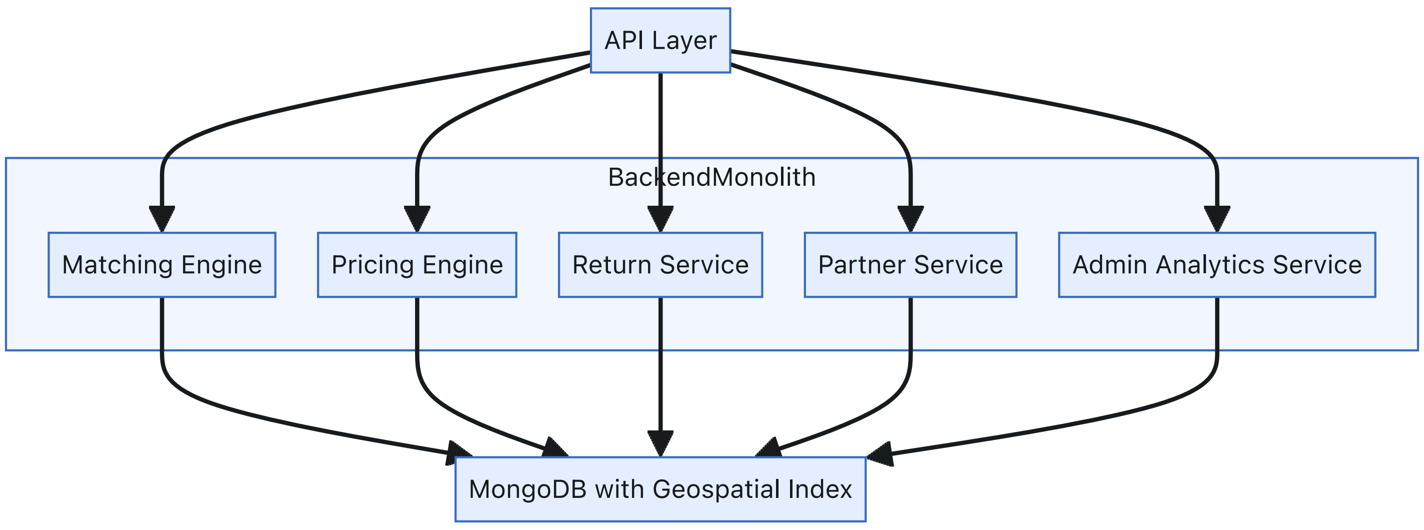


⚙ External Services & Internal Modules Overview

In our design, even though the application is built as a single monolith, we logically separate different parts into modules/services inside the backend. The key services:

- **Matching Engine**: handles geospatial partner matching for returned items, powered by MongoDB's geospatial index.
- **Pricing Engine**: dynamically calculates suggested price based on item condition and demand.
- **Return Service**: manages return item lifecycle.
- **Partner Service**: manages partner profiles and preferences.
- **Admin Analytics Service**: tracks KPIs and environmental impact.

All services share the same database, but are separated to simplify future scaling or conversion into true microservices.



🏗️ High-Level Architecture Overview

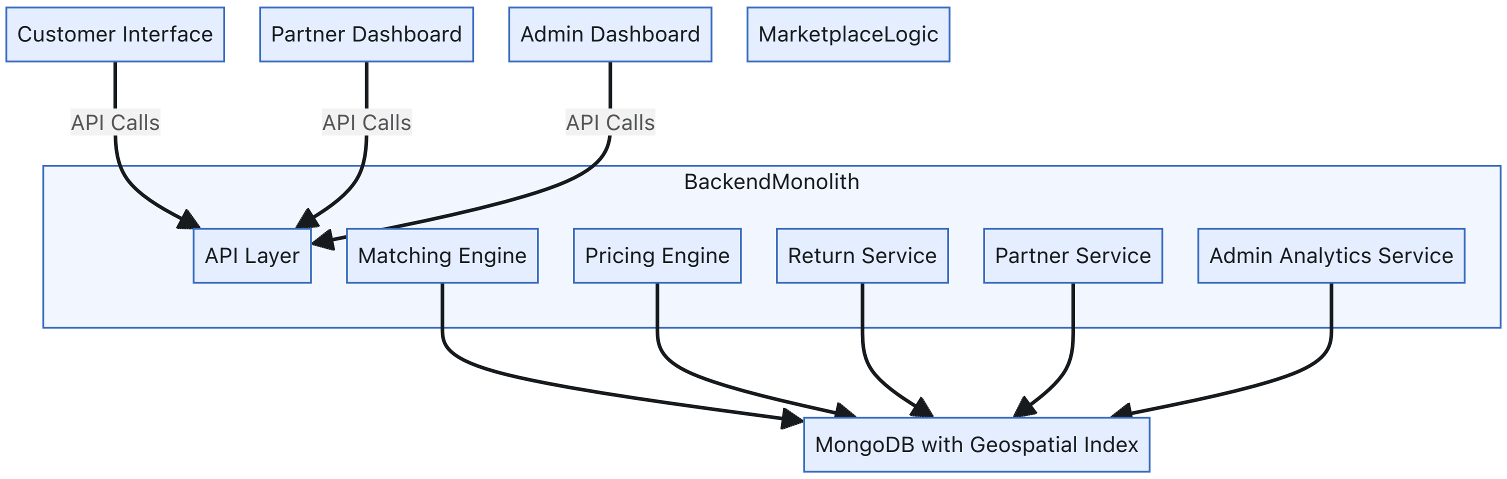
Our system separates frontend and backend to keep it modular and extendable, even within a single monolith codebase. The backend logically splits services that together power the **hyperlocal returns marketplace**:

- **Matching Engine**: matches returns to nearby partners using geospatial data.
- **Pricing Engine**: dynamically suggests prices based on item data and demand.
- **Return Service**: handles return requests and lifecycle.
- **Partner Service**: manages partner profiles and claims.
- **Admin Analytics Service**: collects metrics and insights.

The frontend includes three apps:

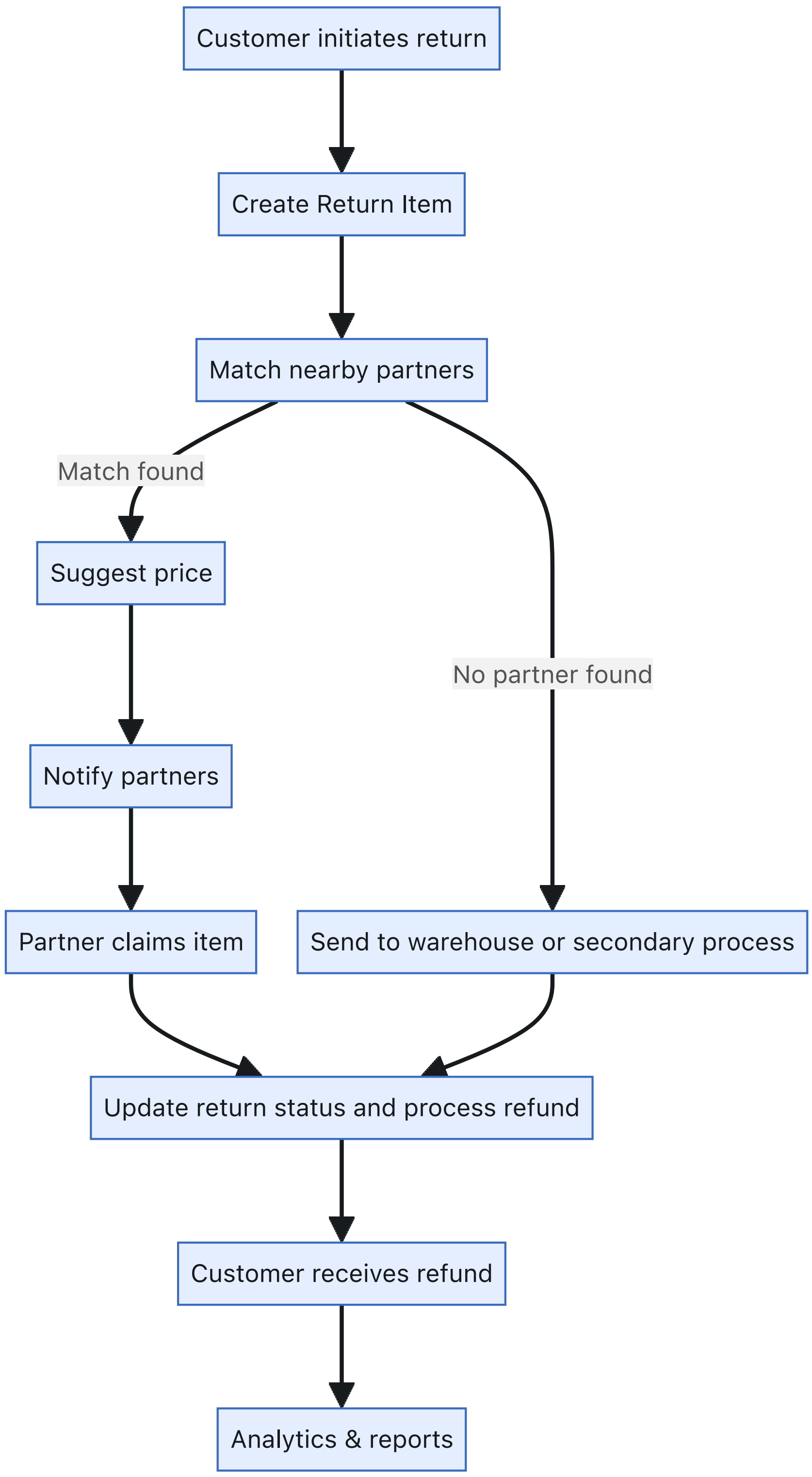
- **Customer Interface**: lets customers create returns and view offers.
- **Partner Dashboard**: for local partners to claim items.
- **Admin Dashboard**: for monitoring KPIs and system health.

All backend services share the same MongoDB database with a geospatial index, while the **marketplace logic** emerges from matching, pricing, and partner modules combined.



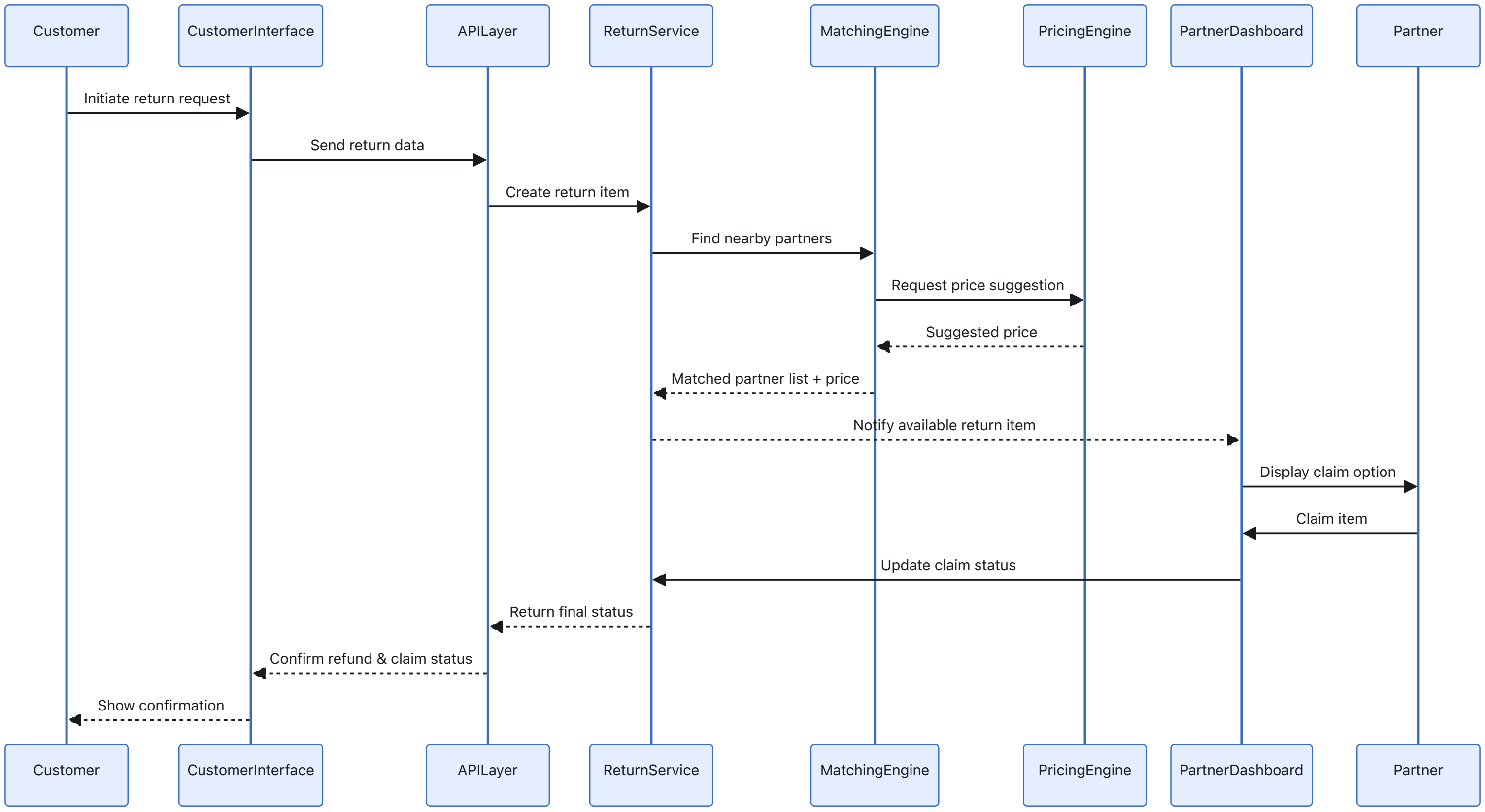
🧭 Customer Journey Flow

This diagram shows how a return request flows from customer initiation to partner matching, pricing, claiming, refund, and analytics. It includes a fallback if no local partner is found.



🔄 Return & Claim Flow Sequence Diagram

This diagram details the end-to-end sequence when a customer returns an item and a partner claims it. It highlights the order of interactions across frontend and backend modules.



This covers the typical return flow from customer initiation to partner claim and final confirmation.

AI ML Prospects in Matching and Pricing Engines

To make the matching and pricing engines more professional-grade and future-ready, we propose integrating AI/ML techniques:

- **Matching Engine**
- Use ML models trained on historical return-claim data to predict which local partners are most likely to claim certain item types.
- Optimize partner recommendations beyond pure distance, factoring in partner preferences, claim success rates, and item categories.
- Continuous learning loop: update model as partners interact with new returns.
- **Pricing Engine**
- Dynamic pricing based on real-time local demand, item condition, seasonality, and product popularity.
- ML models estimate fair market value for open-box items in hyperlocal markets.
- Manual override and business rules for sensitive categories.

These AI/ML enhancements help:

- Improve claim success rate.
- Maximize recovered value.
- Adapt pricing to local trends, increasing marketplace efficiency.

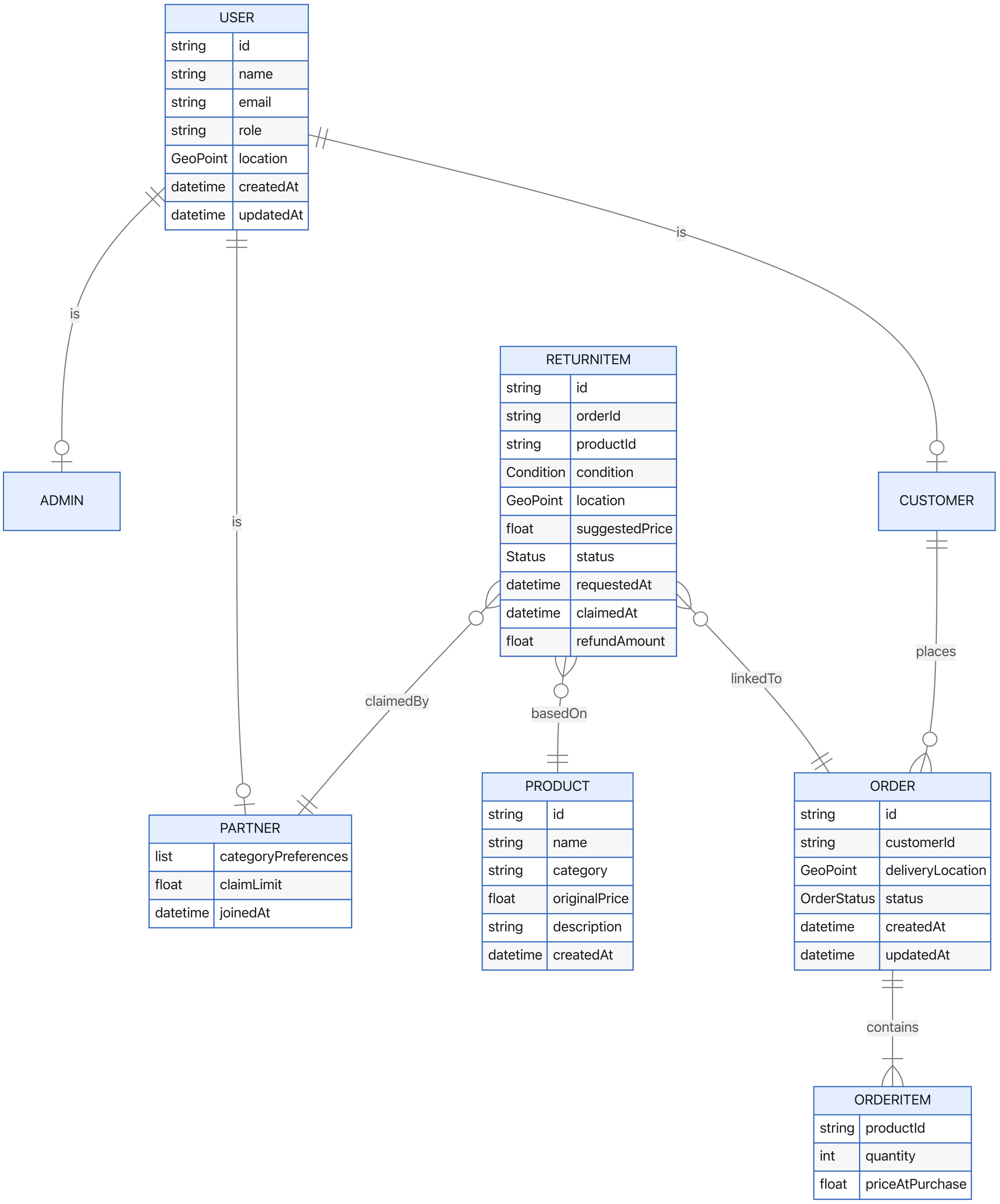
This vision can start as rule-based logic and evolve into trained models as data volume grows.

🗄️ ER Diagram / DB Schema

We define the main collections and their relationships for MongoDB (even though it is schema-less, we keep a clear logical structure):

- Users with role and geolocation.
- Products catalog.
- Orders containing multiple OrderItems.
- ReturnItems linked to orders and products, claimable by partners.

- Partners can claim ReturnItems; admins manage analytics.


USER
string id
string name
string email
string role
GeoPoint location
datetime createdAt
datetime updatedAt
is
is
is
ADMIN
CUSTOMER
RETURNITEM
string id
string orderId
string productId
Condition condition
GeoPoint location
float suggestedPrice
Status status
datetime requestedAt
datetime claimedAt
float refundAmount
places
claimedBy
basedOn
linkedTo
PARTNER
list categoryPreferences
float claimLimit
datetime joinedAt
PRODUCT
string id
string name
string category
float originalPrice
string description
datetime createdAt
ORDER
string id
string customerId
GeoPoint deliveryLocation
OrderStatus status
datetime createdAt
datetime updatedAt
contains
ORDERITEM
string productId
int quantity
float priceAtPurchase