Vehicle Auction System Documentation

Brief Code Explanation

The solution implements a RESTful API for managing vehicle auctions with full CRUD operations support. Built using a Minimal API approach, the application minimizes boilerplate code while maintaining clear architectural boundaries. The endpoint logic is modularized into dedicated handlers (AuctionHandlers and VehicleHandlers), with a service layer managing business logic such as polymorphic vehicle types and auction state transitions. Data access is abstracted through repositories following the Repository Pattern.

For entity management, EntityFrameworkCore was chosen for several key advantages:

- Built-in support for vehicle type inheritance scenarios (TPH - Table Per Hierarchy)
- Strong LINQ integration for type-safe queries
- Efficient change tracking and relationship management
- Support for both in-memory and persistent databases

An in-memory database was used to simplify development and testing, allowing for quick iterations and easy test data management.

Key additional features include:

- OpenAPI/Swagger Documentation for API exploration and testing
- ExceptionHandlingMiddleware - Provides centralized exception handling with custom exceptions
- AuctionLogger - Implements file-based logging for auction outcomes
- Background AuctionProcessingService - Autonomous service that manages auction lifecycle, processing completed auctions and logging winners
- Robust validation using FluentValidation for both vehicles and auctions
- Type-safe response handling with dedicated DTOs and response models
- Extension methods for clean service registration and configuration
- Comprehensive unit tests using xUnit and FluentAssertions, covering both positive scenarios and exception handling

Assumptions made during Development

Business Logic Assumptions:

- Vehicle inventory enforces unique VIN numbers
- VIN validation is simplified for PoC (full implementation would follow standard VIN patterns: https://en.wikipedia.org/wiki/Vehicle_identification_number)
- Four vehicle types are supported: Hatchback, Sedan, SUV, and Truck
- Vehicle states are managed as Available, Unavailable, or Sold:
    - Vehicles in active auctions are locked from updates or inclusion in other auctions
    - Sold vehicles (with winning bids) remain in inventory but are restricted to read-only operations

Auction Rules:

- Start and end dates are optional:

    - Auctions initialize in a "Waiting" state and require explicit activation
    - StartTime is automatically set to system time upon activation
    - EndDate is optional and can be system-assigned when manually closed
    - A background service automatically closes auctions that exceed their end date (when manually introduced)
    - Without an end date, auctions must be closed manually via API
- Multiple auction formats supported:
    - Standard auctions can include multiple items sold independently
    - Collective auctions enable bulk bidding where each bid applies to all vehicles
    - Multi-item auctions use individual vehicle starting prices
    - Collective auctions require a user-specified starting bid
- Auction management rules:
    - Manual start required via API endpoint
    - Bids can only be made on active auctions
    - Cancellation releases vehicles and voids ongoing bids
    - Vehicles in active auctions are locked until auction completion
    - Upon completion:
        - Vehicles with winning bids are marked as sold
        - Unsold vehicles return to available inventory

Auction functionalities implemented

- GET vehicles/*:

    - GET vehicles: Get all Vehicles with a specific filter criteria (Filters: Model, Manufacturer, Type, YearFrom, YearTo, HideNonAvailable, HideSold, ShowDeleted);
    - GET vehicles/{id}: Get Vehicle by Id

- POST vehicles/*:

    - /sedan : Create Sedan;
    - /hatchback : Create Hatchback;
    - /suv : Create Suv;
    - /truck : Create Truck;

- PUT vehicles/*:

    - vehicles/update: Update a vehicle in the database (supports vehicle type change given the right configuration);

- DELETE vehicles/*: Marks a vehicle as deleted in the database (records are still kept in the database);

- GET auctions/*:

    - /completed: Get Completed Auction in database;
    - /active: Get Active Auction in database;
    - /all: Get All Auctions in database;
    - /bid-history/{id}: Get Auctions bid history

- POST auctions/*:

- /start: Start Auction;
- /close: Close Auction;
- /cancel: Cancel Auction;
- /bid: Bid on Auction;
- /add-vehicles: Add Vehicles to Auction in the Waiting or Active stage;
- /create-collective: Create a Collective Auction;
- /auctions/collective-bid: Place a bid on Auction of type Collective;

- DELETE auctions/vehicles: Remove vehicles from Auction;

Class Diagram