Table of Contents

Data Types

Vendor

Attribute	Data type	Nullable
<u>Name</u>	String	Not Null
Address (PostalCode, Street, City, State)	String	Not Null
PhoneNumber	String	Not Null

Vehicle

Attribute	Data type	Nullable
<u>VIN</u>	String	Not Null
Manufacturer	String	Not Null
Condition	String	Not Null
Colors	List<String>	Not Null
Year	Integer	Not Null
VehicleType	String	Not Null
FuelType	String	Not Null
Horsepower	Integer	Not Null
Description	String	Null
Model	String	Not Null
PurchasePrice	Float	Not Null
PurchaseDate	Date	Not Null
SalePrice	Float	Not Null
SaleDate	Date	Null

User

Attribute	Data type	Nullable
<u>Username</u>	String	Not Null
Name (FirstName, LastName)	String	Not Null
Password	String	Not Null
UserType	String	Not Null

Parts Order

Attribute	Data type	Nullable
PartsOrderNumber	String	Not Null

Part

Attribute	Data type	Nullable
PartNumber	String	Not Null
Description	String	Not Null
Status	String	Not Null
UnitPrice	Float	Not Null
Quantity	Integer	Not Null

Customer

Attribute	Data type	Nullable
Email	String	Null
Address (PostalCode, Street, City, State)	String	Not Null
PhoneNumber	String	Not Null

Business

Attribute	Data type	Nullable
TIN	String	Not Null
BusinessName	String	Not Null
PrimaryContact (Title, FirstName, LastName)	String	Not Null

Individual

Attribute	Data type	Nullable
SSN	String	Not Null
Name (FirstName, LastName)	String	Not Null

Business Logic Constraints

Vehicle

- The Sale Price is calculated as 125% of the original purchase price combined with 110% of the total cost of any parts purchased for the vehicle
- If no parts have been ordered for a vehicle, calculate cost of parts as $0
- Model year cannot exceed current year plus one
- Model year must include century digits e.g. “1999” not “99”
- SaleDate is NULL until the vehicle is sold

User

- UserType is a required attribute that will have value:
    - InventoryClerk
    - Salesperson
    - Owner
    - Manager
- A user with UserType InventoryClerk or Owner is a VehicleBuyer
- A user with UserType Salesperson or Owner is a VehicleSeller

PartsOrder

- No PartsOrder for a Vehicle that has already been sold (has Sale Date) can be updated or added
- PartsOrder OrderId is in the form {VIN-of-related-vehicle}-{ordinal-of-order}

Part

- No Part can be added or updated for a Vehicle that has already been sold (has Sale Date)
- A Part cannot be changed to a previous status in the ordered list “ordered”, “received”, “installed”
- Once a part is in the “installed” state it cannot be updated

Reports

Average Time in Inventory Report

- First and last day should be counted as one day. E.g. if vehicle was added and sold on same day, it will be counted as one day.
- If vehicle type has no sales history, report should display “N/A”

Price Per Condition Report

- If no vehicle has been sold in a certain condition, show $0 as total for that condition

Parts Stats Report

- Order by total dollar amount spent per vendor

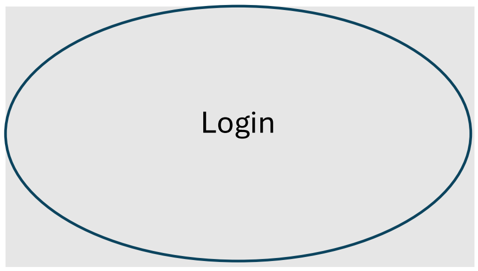
Monthly Sales Report

- If a year or month has no sales, exclude from report
- Results ordered by year and month descending, most recent year and month first
- For drilldown, sort by number of vehicles sold descending followed by total sales descending. E.g. if two Salesperson entities have sold same number of vehicles, the one with higher dollar value would show first

Task Decomposition and Abstract Code

Login

Task Decomposition



Lock Types: Read-only on User

Number of Locks: Single

Enabling Conditions: User clicks “Login” button on Search Screen

Frequency: Tens of times per day. At least once per logged in User per day.

Consistency (ACID): Not critical, order is not critical, does not impact other users/data

Subtasks: Subtasks are not needed. No decomposition needed

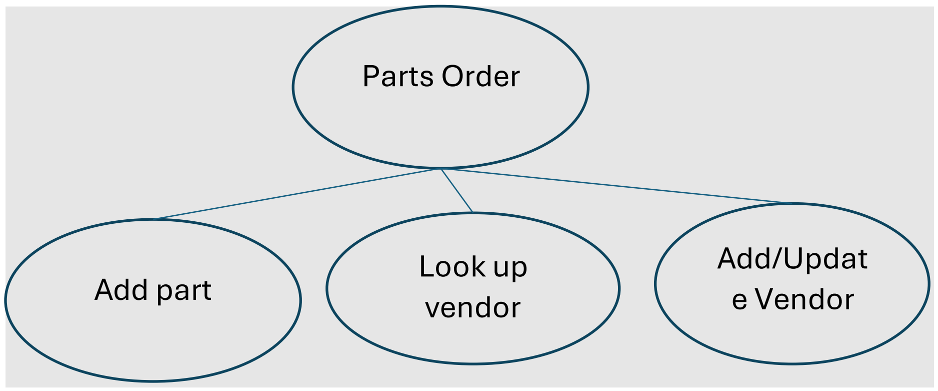
Abstract Code

- User enters *username* (‘$username’) and *password* (‘$password’) input fields
- When ***Log In*** button is clicked:

- If User ('$user') record is found for '$username' but '$user.password' != '$password':
    - Go back to **Login** form, with error message
        - **Error: "No matching user/password"**
- If no User record is found for $username
    - Go back to **Login** form, with error message:
        - **Error: "No matching user/password"**
- Else
    - Store login information with session variable '$user' that also used in rest of application for determining if the user is a InventoryClerk, SalesPerson, Owner, or Manager
    - Go to **Search Screen**

Parts Order

Task Decomposition



Lock Types: Read-only on Vehicle, Read/Write on Vendor, Read/Write on PartsOrder, Write on Part

Number of Locks: 4 schemas.

Enabling Conditions: User is logged in as an Inventory Clerk or Owner, and clicks the ***Add Parts Order*** button on a **Vehicle Detail** page.

Frequency: Up to hundreds of times per day. Relatively frequent due to the necessity of parts and their statuses in the transaction of cars.

Consistency (ACID): Order is important because the PartsOrder ID is derived from the VIN of the Vehicle and the ordinal of the PartsOrder (is it the first, second, third, order etc). Order within the Parts is not important. So the Parts and PartsOrder need to be saved in one transaction. Saving the Vendor can happen in a seperate transaction before submitting PartsOrder. Order is important for saving Vendor because if another InventoryClerk is adding a Vendor with the same name, this could cause an integrity error.

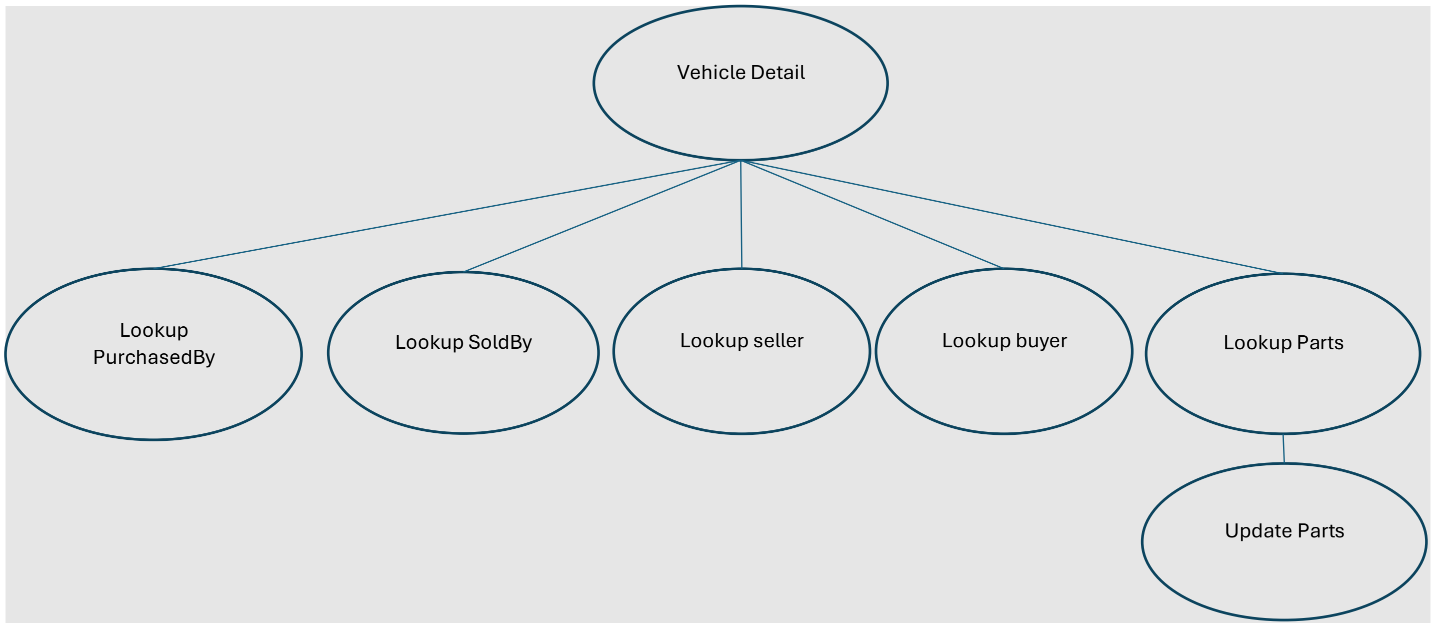
Subtasks: Subtasks are necessary. Order is important for giving the the PartsOrder an ID. Order is also important for checking if the Vendor is unique by name. Vendor subtask and PartsOrder/Parts subtask can be done inttwo separate transactions.

Abstract Code

- If '$user' not defined or '$user.usertype' not in (Owner, InventoryClerk)
    - Redirect user to **<u>Search Screen</u>**
- Else:
    - Store VIN of vehicle for which we are adding parts order as '$vehicle.Vin', display on form as non-editable
    - Display text search field for Vendor with ***Search*** button
        - If Vendor found by Name
            - Show details of Vendor add give button to ***Associate Vendor*** with PartsOrder
                - If click ***Associate Vendor***, return to PartsOrder with vendor shown as non-editable
    - Else, provide message "Sorry, it looks like we don't have that in stock!" error and provide ***Add Vendor*** button
            - Show editable fields with "*Name*", "Street Address*"*, "City", "State", "*Postal Code*", "*Phone Number*" fields and ***Save*** button. They can click ***Save*** multiple times until all data validation passes.
            - Vendor is created in database upon clicking ***Save***
            - Once pass all data validation, show vendor *Name* shown as non-editable
    - Display ***Add Part*** button.
        - Upon click of ***Add Part*** display input fields for *"Part Number"*, *"Price (per unit)"*, *"Quantity"*.
        - The ***Add Part*** button can be clicked multiple times to add several parts
    - Display ***Submit*** Button
        - There must be at least 1 part in the Parts Order
        - There must be a Vendor associated with the PartsOrder
        - All fields are required
        - All parts are initially saved with status "Ordered"
        - Parts saved in Transaction with rest of Parts Order
        - Upon save, check what number PartsOrder this is for the vehicle to generate the PartsOrderID which is '$VIN-$ordinal-of-parts-number'

Vehicle Detail

Task Decomposition



Lock Types: Read-only on Vehicle, PartsOrder and Customer and Read-Write on Parts

Number of Locks: 4

Enabling Conditions: User selects an individual Vehicle result on the Search Screen. If user is logged in, then additional functionality is added depending on '$user.UserType'

Frequency: Very frequent, possibly thousands of times a day

Consistency (ACID): Order is important for updating Parts because parts status can only advance from Ordered->Received->Installed. For read-only locks, order is not important.

Subtasks:

Abstract Code

- Store '$vin' of vehicle that was clicked on for the detail view
- Always show public view whether '$user' is defined or not:
  - Look up all PartsOrders related to '$vehicle' and then look up all their related Parts. Store parts in array of Parts called '$parts'. Store a mapping between PartsOrderId and the arrays of parts for use later.
    - Calculate total cost of parts as '$totalCostofParts'. Use this plus '$vehicle.PurchasePrice' to calculate and display Sale Price: '$vehicle.SalePrice' with formula 125% '$vehicle.PurchasePrice' plus of the combined with 110% of the total cost of any parts purchased for the vehicle (PartsOrders)

        - Read Vehicle from database where '$vin' == '$vehicle.vin'
        - Display read-only fields:
            - *Vehicle Type* ('$vehicle.VehicleType')
            - *Manufacturer* ('$vehicle.Manufacturer')
            - *Model* ('$vehicle.Model')
            - *Year* ('$vehicle.Year)
            - *Fuel Type* ('$vehicle.FuelType')
            - *Color(s)* ('$vehicle.Colors')
            - *Horsepower* ('$vehicle.Horsepower')
            - *Sale Price* ('$vehicle.SalePrice')
            - *Description* ('$vehicle.Description') of the vehicle. If defined, otherwise, empty string.
- If '$user' is defined and '$user.UserType' is Owner, Manager, or Inventory Clerk
    - Display ***Purchased Price:*** '$vehicle.purchasePrice'
        - Display ***Total Cost of Parts***: '$totalCostofParts'
- If '$user' is defined and '$user.UserType' is Owner, Manager
    - Additionally look up the Customer that the Sell relationship points to from the '$vehicle' and store as '$seller'
        - Display ***Purchased From***:
            - '$seller.Email', '$seller.PhoneNumber', '$seller.Address'
            - If '$seller' is an Individual Customer:
                - Also display ***Contact:*** '$seller.Name'
            - If '$seller' is a Business Customer:
                - Also display ***Contact:*** '$seller.PrimaryContact'
                - Also display Business Name: '$seller.BusinessName'
            - Additionally display customer's contact information below
                - Phone Number '$seller.PhoneNumber'
                - Address '$seller.Address'
                - Email '$seller.Email'
        - Display ***Purchased Date***: '$vehicle.PurchaseDate'
    - Additionally look up the VehicleBuyer that the relationship PurchasedBy points to from '$vehicle' and store as '$purchaser' and display as "***Purchased By***: ('$purchaser.Name).
    - Additionally display the *Purchase Date* of the vehicle ('$vehicle.PurchaseDate')
    - If the vehicle's sale date ('$vehicle.SaleDate') is not null:
        - Display *Sale Date:* ('$vehicle.SaleDate')
        - Additionally look up the VehicleSeller that sold the vehicle and store as '$salesperson' and display as "*Sold By*: '$salesperson.Name".

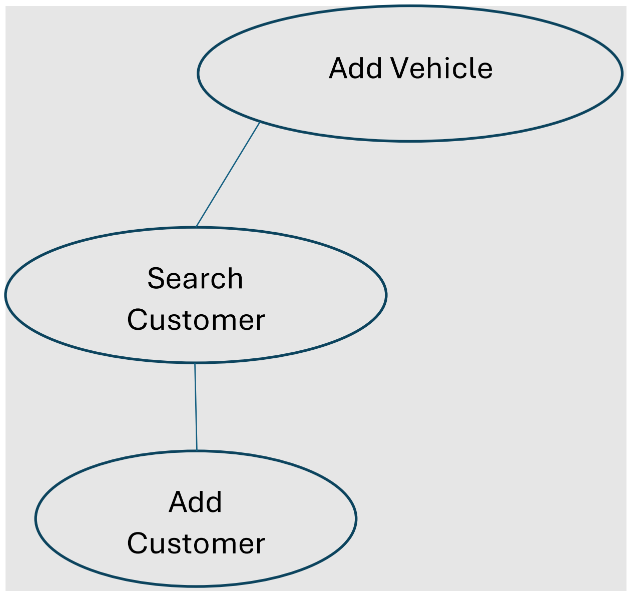
                - Additionally look up the Customer that the Buy relationship points to from the '$vehicle' and store as '$buyer'
                    - Display *Sold To*:
                        - '$buyer.Email', '$buyer.PhoneNumber', '$buyer.Address'
                        - If '$buyer' is an Individual Customer:
                            - Also display ***Contact:*** '$buyer.Name'
                        - If '$buyer' is a Business Customer:
                            - Also display ***Contact:*** '$buyer.PrimaryContact'
                            - Also display Business Name: '$seller.BusinessName'
                        - Additionally display customer's contact information below
                            - Phone Number '$buyer.PhoneNumber'
                            - Address '$buyer.Address'
                            - Email '$buyer.Email'

- If '$user' is defined and '$user.UserType' is Owner or InventoryClerk
    - Display list of Part(s). For each '$partorder' we previously mapped to their array of '$parts' we previously saved, display:
        - PartsOrder: '$partorder.PartsOrderNumber'
            - For each part in the mapped '$parts' array, display
                - Status: '$part.Status'
                - Part Number: '$part.PartNumber'
                - Description: '$part.Status'
                - Quantity: '$part.Status'
                - Unit Price: '$part.Status'
                - Part Status: '$part.Status'
                - Part Order Id: '$partsOrder
                - If '$vehicle.SaleDate' is Null and '$part.Status' is not "installed"
                    - Display button ***Update Parts Status*** on each part which allows user to select from dropdown for the status for the selected Part
                        - Display status as drop down
                        - Display a "***Save***" button on row of part
                        - A Part cannot be changed to a previous status in the ordered list "ordered", "received", "installed"
                        - Once a part is in the "Installed" state it cannot be updated
                        - We should check Part Status again before saving and display an Error message if it has already been updated to a status that makes our update invalid.

        - **Error:** “Cannot Update Status”
    - If ‘$vehicle.SaleDate’ is Null (vehicle is not sold)
        - Display button ***Add Parts Order*** which will take the user to the **Add Parts Order** form
- If ‘$user’ is defined and ‘$user.UserType’ is Owner or SalesPerson
    - Display button ***Sell Vehicle*** which will take user to the **Sell Vehicle** form

Add Vehicle

Task Decomposition



Lock Types: Read/Write on Customer, Read/Write on Vehicle

Number of Locks: 2 different schema constructs are needed. Vehicle and Customer

Enabling Conditions: User is logged in as Inventory Clerk or Owner and clicked on Add Vehicle button.

Frequency: Tens of times per day.

Consistency (ACID): Not critical, one user can only add one car at a time. Cars also have unique VIN so collision is unlikely/impossible.

Subtasks: Tasks should be completed in sequence. After searching for a customer and not finding a match, the **Add Customer** task should be executed. "Add Vehicle" can be completed after linking a customer to it.

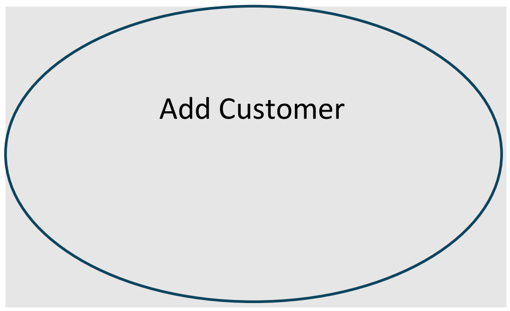
Abstract Code

- User clicked on ***Add Vehicle*** button from **Search Screen** page.

- If '$user' is not defined or '$user.userType' is not either Inventory Clerk or Owner
    - Return to **Search Screen** page
- Else
    - Store '$user.username' as the '$VehicleBuyer'
    - Display *search* field to look up for customers.
    - User enters *ssn* or *tin* in to *search* field to look up customers.
        - When the ***Search*** button is clicked.
            - If a customer is found '$ssn' or '$tin' is stored as '$customer'.
            - Else:
                - "Customer not found" message displayed.
                - ***Add a New Customer*** button is displayed.
                - When the user click on ***Add a New Customer*** button it displays the **Add Customer** sub-form, which executes the **Add Customer** sub-task.
                - When user successfully add a new customer, continue with the customer's '$ssn' or '$tin' is stored as '$customer'
    - Upon storing the seller and purchased by information are stored, user enters the following information:
        - Vin ('$vehicle.vin'), Vehicle Type ('$ vehicle.VehicleType'), Manufacturer ('$ vehicle. Manufacturer'), Model ('$ vehicle. Model'), Year ('$ vehicle. Year'), Horsepower ('$ vehicle. Horsepower'), Fuel Type ('$ vehicle. FuelType'), Colors ('$ vehicle. Colors'), Condition ('$ vehicle. Condition'), Description ('$ vehicle. Description') (optional), Purchase Price ('$ vehicle. PurchasePrice')
        - Model year cannot exceed current year plus one
        - Model year must include century digits e.g. "1999" not "99"
        - Fuel type must be one of: Gas, Diesel, Natural Gas, Hybrid, Plugin Hybrid, Battery, or Fuel Cell
        - Condition must be one of Excellent, Very Good, Good, Fair
        - Color(s) must be at least one that in mentioned in appendix of project spec
        - Manufacturer must be one that is mentioned in the appendix of project spec
        - SaleDate is instantiated as NULL and not defined until vehicle is sold
    - When user clicks on ***Add Vehicle*** button
        - If input validation doesn't pass, display "Invalid Input" and display the input page again with the previous input values.
        - Else
            - the new vehicle added to the database
            - Purchase Date ('$vehicle.PurchaseDate') is stored

- **Vehicle Detail** page is displayed

Add Customer (shared subtask)



Task Decomposition

Lock Types: Read/Write on Customer

Number of Locks: Single (one customer record being added at a time).

Enabling Conditions: The user must be logged in (as an Inventory Clerk, Salesperson, or Owner). The logged-in user must click on ***Add Customer*** button in either the **Add Vehicle Form** or the **Sell Vehicle Form**

Frequency: Occasional, happens each time a new customer is involved in a purchase or sale. Expected less than 100 times a day.

Consistency (ACID): Is not critical because one customer can only be involved in one transaction at a time.

Subtasks: There are no subtasks. This needs no decomposition

Abstract Code

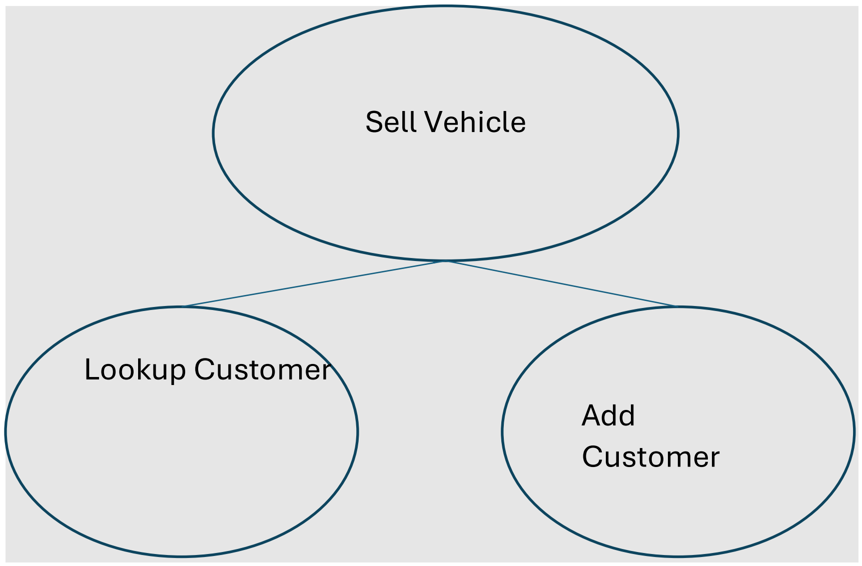
- User opens the sub-form **Add Customer** from either the **Add Vehicle Form** or the **Sell Vehicle Form** after checking that the customer doesn't exist.
- User selects whether the customer is an individual or a business using a radio button.
- If customer type is 'individual':
  - Collect and validate input for:
    - First Name (‘$customer.firstName’)
    - Last Name (‘$customer.lastName’)
    - SSN (‘$customer.ssn’): Ensure the format is 9 digits.
    - Address (‘$customer.address’)

        - Phone Number (‘$customer.phone’): Ensure it matches a valid phone number pattern.
        - Optional: Email (‘$customer.email): Ensure email follows a valid format.
- If customer type is 'business':
    - Collect and validate input for:
        - Business Name (‘$customer.businessName’)
        - Tax ID (‘$customer.taxID’): Ensure format matches valid patterns.
        - Contact First Name (‘$customer.contactFirstName’)
        - Contact Last Name (‘$customer.contactLastName’)
        - Address (‘$customer.address’)
        - Phone Number (‘$customer.phone’): Ensure it matches a valid phone number pattern.
        - Optional: Email (‘$customer.email’): Ensure email follows a valid format.
- User clicks the ***Submit*** button on the **Add Customer** form.
- Validate SSN uniqueness:
    - Query the database for a customer record with the same ‘$ssn’
    - If a matching record exists, return an error message:
        - **Error:** "Customer already exists"
    - If no match is found:
        - Check for valid input formats (SSN, phone number, email).
        - **Insert** Task**:** Insert the new customer record into the database.
- Validate Tax ID uniqueness:
    - Query the database for a customer record with the same ‘$tin’.
    - If a matching record exists, return an error message:
        - **Error:** "Business already exists".
    - If no match is found:
        - Check for valid input formats (TIN, phone number, email).
        - **Insert** Task**:** Insert the new customer record into the database.
- Feedback:
    - If successful:
        - Display a message: **"Customer added successfully."**
        - Return ‘$customer’ variable with all its attributes to whatever form included this sub-form.
    - If unsuccessful:
        - If validation error: Display an error message indicating the specific issue (e.g., **Error:** "Missing required fields" or **Error:** "Invalid SSN format").
        - If duplicate record: Display an error message:
            - **Error:** "Customer already exists."

  - Keep sub-form open and editable

Sell Vehicle

Task Decomposition



Lock Types: Read/Write on Customer, Read/Write on Vehicle

Number of Locks: 2 different scheme constructs are needed. Vehicle and Customer

Enabling Conditions: Logged in as SalesPerson or owner. Already on Detail View of a Vehicle. Salesperson or Owner clicked on ***Sell Vehicle*** Button.

Frequency: Tens to hundreds of times per day

Consistency (ACID): Critical that Vehicle is only sold once. If some other SalesPerson/Owner sells it first, we should reject sale, not update SaleDate. One customer can only purchase one car at a time. Customer can be added but sale of vehicle fail if has already been sold by time they confirm sale.

Subtasks: Tasks should be completed in sequence. After clicking on the sell vehicle button, user should look up the customer by SSN or TIN. If no customer is found, then the salesperson should click on the add customer button and go to the appropriate subtask. Finally, when the sale is confirmed, we should do final check that SaleDate is still null and no customer is related to the vehicle as the buyer and reject sale if already sold.

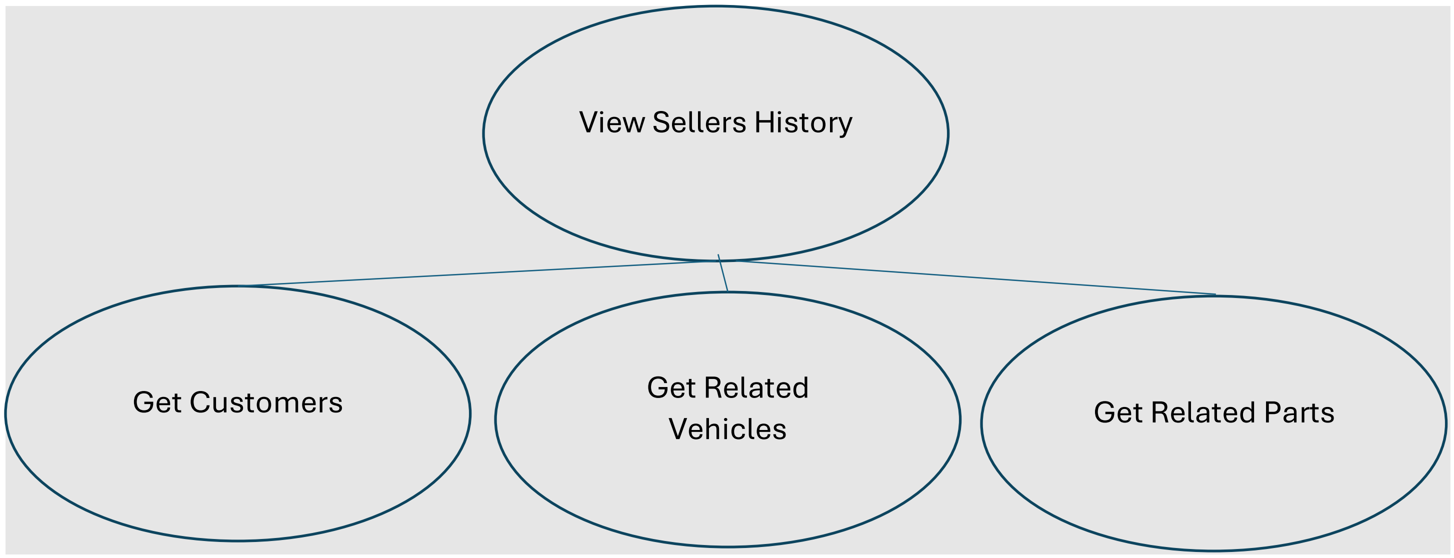
Abstract Code

- User clicks on the ***Sell Vehicle*** button from the **Vehicle Detail** page.
  - If '$user' is not defined or '$user.userType' is not either Salesperson or Owner
    - Redirect to the **Search Screen** page

- Else
    - Store '$user.username' as the '$VehicleSeller'
    - Store '$vehicle.vin' , '$vehicle.SalePrice' and '$vehicle.TotalPartsPrice' (which was computed on **Search Screen**) for the vehicle that was selected from Search Screen
    - Display *search field* to look up the customer.
    - User enters *ssn* or *tin* into search field to look up the customers.
        - When the ***Search*** button is clicked
            - If a customer is found, store '$ssn' or '$tin' as '$customer'.
            - Else:
                - Display "Customer not found" message.
                - Display ***Add Customer*** button.
                - When the user click on ***Add Customer*** button
                    - Display the Add Customer sub-form and execute **Add Customer** sub-task.
                    - After successfully adding the customer, store the customer's '$ssn' or '$tin' as '$customer'
    - After the user clicks on the ***Confirm Sale*** button
        - Check if '$vehicle.SaleDate' is null and '$vehicle.buyer' relation is null and '$vehicle.seller' relation is null
            - If all are still null
                - Update '$vehicle.SaleDate' to current date
                - Update the relationship to VehicleSeller SoldBy to point to current '$user'
                - Update the relationship to Customer Buy relationship of the '$vehicle' to point to '$customer'
                - Save '$vehicle.SalePrice' and '$vehicle.TotalPartsPrice' as the final value so we don't have to compute again for future reports. This can't change because we cannot add PartsOrders for a vehicle that is sold.
                - Display message "Success"
            - Else
                - Display an error message:
                    - **Error:** "Vehicle already sold!"
        - Return to **Search Screen**

View Sellers History

Task Decomposition



Lock Types: Customer (FirstName, LastName OR BusinessName), Vehicle (<u>VIN</u>, PurchasePrice, TotalPartsPrice), Part (Quantity)

Number of Locks: Three Schema

Enabling Conditions: User is logged in as an Owner or Manager. User is on the main <u>**Search Screen**</u> and has clicked the link to ***View Sellers History***.

Frequency: Tens of times per day per Manager/Owner

Consistency (ACID): Not critical, order is not critical, is isolated and durable (concurrent and will remain stable) - due to it being read-only

Subtasks: Subtasks are to get customers, then their related vehicles they sold, then related parts. This must be done in order, so we know how to relate all the data and filter correctly. Therefore, a mother task is needed.

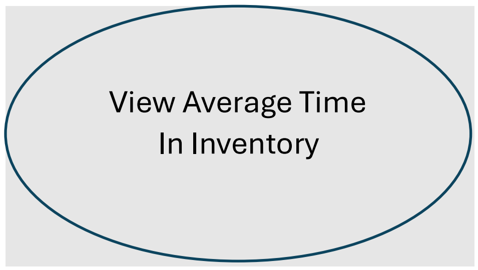
Abstract Code

- If '$user' is not defined or '$user.UserType' not either Owner or Manager:
  - Redirect to <u>**Search Screen.**</u> The user should not be able to see this screen.
- Else
  - Display Title *Sellers History Report*
  - Display a table with columns: *Name/Business, Total Vehicles Sold, Average Purchase Price, Total Parts Count, Average Parts Price/Vehicle*

- For each Customer related to a Vehicle by the Sell relationship (e.g. they sold the vehicle to the dealership)
    - Display either their FirstName and LastName OR BusinessName in *Name/Business* column of table
    - Display the count of Vehicles '$vehicleCount' they sold in *Total Vehicles Sold* column
        - For each Vehicle they sold, query all related PartsOrders and all their related Parts
        - Store
            - Count of parts as '$totalPartsCount'
            - Total price of parts as '$totalPartsPrice'The sum of all purchase prices for vehicles as '$totalPurchasePrice'
        - Compute
            - '$averagePartsPerVehicle' as '$totalPartsCount'/'$vehicleCount'
            - '$averagePartsCostPerVehicle' as '$totalPartsPrice'/'$vehicleCount'
            - '$averagePurchasePrice' as '$totalPurchasePrice' / '$vehicleCount'
        - Display
            - '$averagePurchasePrice' in *Average Purchase Price* column
            - '$totalPartsCount' in *Total Parts Count* column
            - '$averagePartsCostVehicle' in *Average Parts Price/Vehicle column*
        - If '$averagePartsCostPerVehicle' > 500 or '$averagePartsPerVehicle' > 5, highlight the row in red
- Sort rows by '$vehicleCount' descending and then '$averagePartsCostVehicle' ascending

View Average Time in Inventory

Task Decomposition



Lock Types: Read-only, Vehicle (SaleDate, PurchaseDate, VehicleType)

Number of Locks: Single

Enabling Conditions: User is logged in as either an Owner or Manager. Clicks on ***View Average Time In Inventory*** link on **Search Screen**.

Frequency: Less than 10 times per day.

Consistency (ACID): Not critical, order is not critical, is isolated and durable (concurrent and will remain stable) - due to it being read-only

Subtasks: None

Abstract Code

- If $user is not defined or ‘$user.usertype’ is not in (Owner, Manager)
    - This is error, redirect back to **Search Screen**
- Else:
    - Display Title *Average Time in Inventory Report*
    - Display a table with columns: *Vehicle Type, Average Days in Inventory*
    - For each ‘$vehicle_type’ in VehicleTypes
        - Options are: Sedan, Coupe, Convertible, CUV, Truck, Van, Minivan, SUV, Other
        - Query all Vehicles with ‘$vehicle.VehicleType’ equal to ‘$vehicle_type’ where ‘$vehicle.SaleDate’ is not null, store as ‘$vehicles’
        - If no vehicles match query:
            - Display ‘$vehicle_type’ in the *Vehicle Type* column and the string “N/A” in the *Average Days in Inventory* column
        - Else:
            - Calculate the average days in inventory:
                - First and Last day in inventory should be counted as one day (so N-1 days, out of N days)
                - For each ‘$vehicle’ in ‘$vehicles’
                    - Compute ‘$vehicle.timeInInventory’
                    - If ‘$vehicle.SaleDate’ == ‘$vehicle.PurchaseDate’
                        - Result is 1
                    - Else
                        - Result is ‘$vehicle.SaleDate’ - ‘$vehicle.PurchaseDate’
                    - Examples:
                        - 11/2/2024 - 11/1/2024 = 1
                        - 11/3/2024 - 11/1/2024 = 2
                        - 11/1/2024 - 11/1/2024 = 1
                - Compute ‘$totalTimeInInventory’ as sum of each ‘$vehicle.timeInInventory’

```
            ▪ Divide '$totalTimeInInventory' by count of '$vehicles' to
              compute '$averageTimeInInventory'
            ▪ Display '$averageTimeInInventory' in the Average Time In
              Inventory column
```

View Price per Condition

Task Decomposition

Lock Types: Read-only for Vehicle (Condition, PurchasePrice)

Number of Locks: Single

Enabling Conditions: User is logged in as Owner or Manager and clicks on ***Price Per Condition Report*** link on <u>**Search Screen**</u>

Frequency: User-specific use case, not as frequent as User login but relatively frequent

Consistency (ACID): Not critical, order is not critical, is isolated and durable (concurrent and will remain stable) - due to it being read-only

Subtasks: None

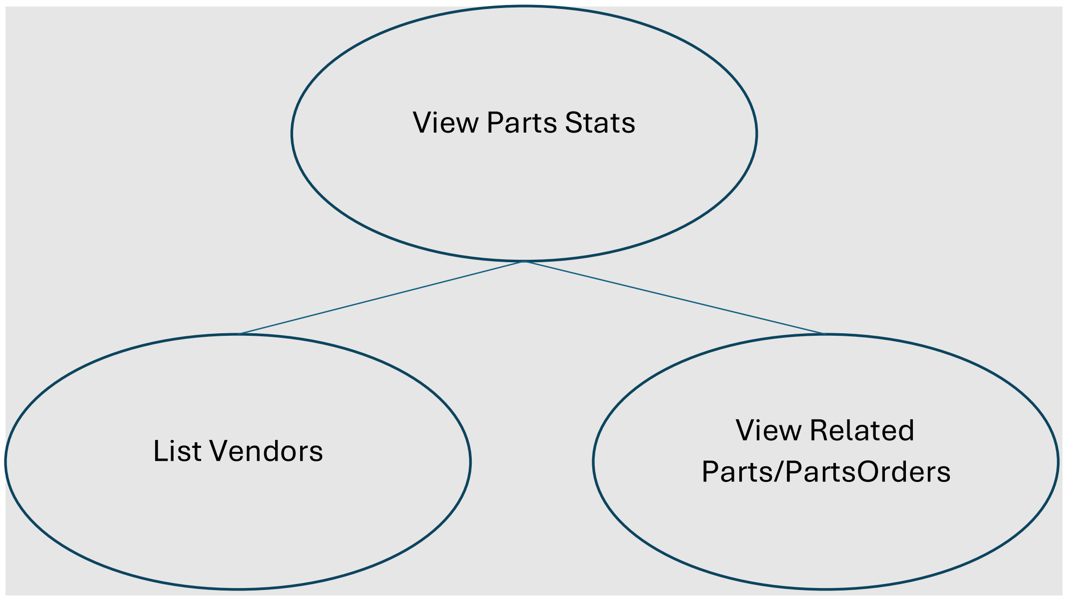
Abstract Code

```
o If '$user' is not defined or '$user.usertype' is not in (Owner, Manager):
    ▪ This is error, redirect back to Search Screen
o Else:
    ▪ Display Title "Purchase Price Per Condition"
    ▪ Display table with column headings: Vehicle Type and then columns for each
      condition: Excellent, Very Good, Good, Fair
    ▪ Display a row for each '$thisVehicleType' in VehicleTypes (Sedan, Coupe,
      Convertible, CUV, Truck, Van, Minivan, SUV, Other)
        o On each row:
            ▪ Display '$thisVehicleType' in column Vehicle Type
            ▪ For each '$thisCondition' in Conditions (Excellent, Very Good,
              Good, Fair)
```

- Query all Vehicles where:
    - '$vehicle.VehicleType' equal to '$thisVehicle_type'
    - '$vehicle.Condition' is equal to '$thisCondition' Store as '$matchedVehicles'
    - If '$matchedVehicles' is an empty list:
        - Display $0 in column corresponding to '$thisCondition' Display sum '$vehicle.PurchasePrice' for each '$thisVehicle' in '$matchedVehicles'.
- Else:
    - Display sum of '$thisVehicle.PurchasePrice' for each '$thisVehicle' in '$matchedVehicles'

View Parts Stats

Task Decomposition



Lock Types: Read-Only on Parts (Quantity, UnitPrice), PartsOrder, and Vendor (Name)

Number of Locks: 3

Enabling Conditions: User is logged in as Owner or Manager and clicks on ***Parts Stats Report*** link on **<u>Search Screen</u>**

Frequency: Since this report is used for negotiating better parts with parts vendors, as much as this report is stable, it is also dependent on the general market of dealerships, **therefore relatively frequent. Hundreds of times per day.**

Consistency (ACID): Not critical, order is not critical, is isolated and durable (concurrent and will remain stable) - due to it being read-only

Subtasks: For each vendor, get its related parts/orders. Order of getting the related parts for any given Vendor is not important.

Abstract Code

- If '$user' is not defined or '$user.usertype' is not in (Owner, Manager)
    - This is error, redirect back to **Search Screen**
- Else:
    - Show table with columns *Vendor Name*, the *Parts Count, and Total Expense*
    - For each Vendor '$vendor'
        - Extract '$vendor.Name' and display in *Vendor Name* Column
        - Query their related Parts Orders
            - For each Parts Order, query related Parts
                - Extract the Quantity from each part, and Compute PartTotalPrice by multiplying the UnitPrice and Quantity for each part
                - Sum all the '$PartTotalPrice' to calculate '$partsOrderTotalPrice'
                - Sum the Quantity of all parts as '$totalPartsQuantity'
        - Sum all the PartsOrderTotalPrice to compute '$vendorTotalExpense'
        - Display '$vendorTotalExpense' in *Total Expense* column
        - Display '$totalPartsQuantity' in *Parts Count* column
            - Sum the Quantity for each Name
- Multiply Quantity by UnitPrice for each part
- Sort rows by '$vendorTotalExpense' in descending order

Monthly Sales Summary

Task Decomposition

Lock Types: Read-Only, for User (FirstName, LastName), Vehicle (SaleDate, SalePrice, TotalPartsPrice)

Number of Locks: Two

Enabling Conditions: Be logged in as either Owner or Manager type user and click on ***View Total Sales by Month*** link on **Search Screen**

Frequency: As the spec indicates, this task is the **MOST frequent report.** Hundreds of times per day.

Consistency (ACID): Not critical, order is not critical, is isolated and durable (concurrent and will remain stable) - due to it being read-only

Subtasks: Two. Summary will always be executed on loading of the report and saved. The Drilldown is only executed if a user clicks button to see drilldown.

Abstract Code

- **Summary** Page
    - Table with columns:
        - *Year, Month, Number Vehicles Sold, Gross Income, Net Income, Drilldown*
    - For each Year
        - For each Month:
            - Query all Vehicles where '$vehicle.SaleDate' matches the time frame
                - Compute and display Gross Sales Income '$grossIncome' as sum of '$vehicle.SalePrice' (that was saved at sale time by the Sell Vehicle task)
                - Compute '$totalExpense' as sum of all '$vehicle.TotalPartsPrice' that matched
                - Compute and display Net Income as
                    - '$NetIncome' = '$grossIncome' - '$totalExpense'
            - If no sales occurred in a specific Year/Month, that period is excluded from the report
        - Sort by Year and then by Month Descending(most recent first)
        - For each row, display link to Drilldown Report for that Year/Month
- **Drilldown** Page per Month
    - Show a table with columns
        - *First Name, Last Name, Vehicles Sold, Total Sales*
    - Query vehicles sold in given Year/Month, joined with their VehicleSeller and its FirstName, LastName, grouped by VehicleSeller.
        - Aggregate sum of SalePrice of vehicles as '$TotalSales'
        - Aggregate count of Vehicles as '$VehicleSold'
        - For each VehicleSeller, extract First Name and Last Name
        - Display detailed information for top-performing salespersons:
            - Salesperson's FirstName and LastName

        - Number of Vehicles sold (count of vehicles with that user as related VehicleSeller)
        - Total Sales amount for that period, respective to Year/Month (sum of '$vehicle.SalePrice')
- Ordered by *Vehicles Sold* DESC and *Total Sales* DESC to determine top performer

Monthly Sales Drilldown

Task Decomposition

Lock Types: Read-Only for User (FirstName, LastName), Vehicle (SalePrice, SaleDate)

Number of Locks: Two

Enabling Conditions: Be logged in as either Owner or Manager type user and click on ***View Total Sales by Month*** link on **<u>Search Screen</u>**

Frequency: As the spec indicates, this task is the **MOST frequent report.** Hundreds of times per day.

Consistency (ACID): Not critical, order is not critical, is isolated and durable (concurrent and will remain stable) - due to it being read-only

Subtasks: Two. Summary will always be executed on loading of the report and saved. The Drilldown is only executed if a user clicks button to see drilldown.

Abstract Code

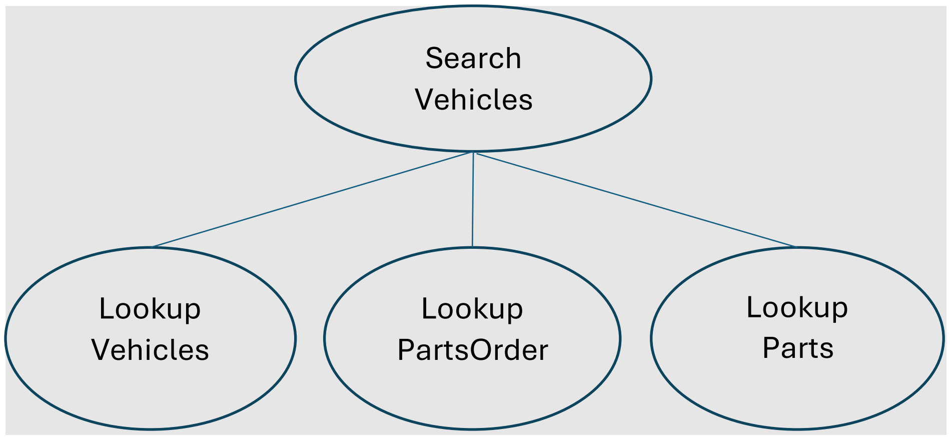
- Show a table with columns
    - *First Name, Last Name, Vehicles Sold, Total Sales*
- Get vehicles sold in given year/month joined with their VehicleSeller and its FirstName, LastName, grouped by VehicleSeller.
    - Aggregate sum of SalePrice of vehicles as '$TotalSales'
    - Aggregate count of Vehicles as '$VehicleSold'
    - For each VehicleSeller extract First name and Last Name
    - Shows detailed information for top-performing salespersons:
        - Salesperson's FirstName and LastName

- Number of Vehicles sold (count of vehicles with that user as related VehicleSeller)
- Total Sales amount for that period, respective to Year/Month (sum of '$vehicle.SalePrice')

○ Ordered by *Vehicles Sold* DESC and *Total Sales* DESC to determine top performer

Search Vehicles

Task Decomposition



Lock Types: 3 read-only lookups for Vehicle, PartsOrder, Parts

Number of Locks: 3 schemas.

Enabling Conditions: **<u>Search Screen</u>** is publicly accessible and the first screen that any user reaches with no specific enabling conditions. However, if the user is logged in, and based on the type of logged in User, additional options may be enabled.

Frequency: High. Hundreds of times daily. This is the first page all users go to, accessible to public users as well.

Consistency (ACID): It is not critical, even if other users are updating vehicles, as results are valid for a specific point in time.

Subtasks: Tasks must be completed in sequence. We separately complete the **Lookup Vehicles** task, then the **Lookup Parts Order** task. Depending on whether any vehicle related PartsOrders, the **Lookup Parts** task will be executed or not for each vehicle.

Abstract Code

- The following is displayed regardless of whether a logged in User or a public user views the **Search Screen**.
    - Display total number of cars available for sale (cars without pending parts)
        - To compute:
            - Select the count of Vehicles with $SaleDate is NULL and join this with all PartsOrders joined with all Parts filtering out any Parts that are not in ‘installed’ status. So only Vehicles that are not sold and have no pending parts are counted.
            - Only count Vehicles where $SaleDate is NULL AND (all parts are in ‘installed’ or that have no parts orders)
    - Display filter options
        - Drop down *Vehicle type*
        - Drop down *Manufacturer*
        - Drop down for *Year*
        - Drop Down *Fuel Type*
        - Drop down for *Color*
        - Text field for *Keyword*
    - Display a ***Search*** button
- If $user is not defined
    - Display **Login Page** link
- Else If $user is defined (Priviliged User)
    - Display Search by *VIN* input field.
        - When submitted, submit as all uppercase to match VIN in database
    - If $user.usertype is “InventoryClerk” or “Owner”
        - Display ***Add Vehicle*** button
        - When user clicks on ***Add Vehicle*** button, go to **Add Vehicle** page
    - If $user.usertype is “Manager” or “Owner” or “Inventory Clerk”
        - Display total number of cars with parts in pending
            - To compute:
                - Select the count of Vehicles with $SaleDate is NULL and join this with all PartsOrders joined with all Parts selecting the Parts in “ordered” or “received” but not “installed” status.
                - Only count vehicles where $SaleDate is NULL AND has PartsOrder AND Parts status is NOT in “installed” status.
    - If $user.usertype is “Manager” or “Owner”
        - Display three radio buttons with the options of Search in
            - ***All***

        - ***Sold***
        - ***Unsold***
    - Display a list of links for the following reports:
        - **<u>Seller History Report</u>** link.
        - **<u>Average Time in Inventory Report</u>** link.
        - **<u>Price Per Condition Report</u>** link.
        - **<u>Parts Statistics Report</u>** link.
        - **<u>Monthly Sales Report</u>** link.
- When the ***Search*** button is clicked
    - If '$user' is NOT defined (public) or '$user' is defined AND '$user.UserType' is "SalesPerson"
        - Search queries will be executed on the database, which only searches unsold vehicles with no pending parts, filtering them by the input fields.
        - To compute:
            - Select the Vehicles in
                - '$vehicle.SaleDate' is NULL
                    - joined with PartsOrder with no match PartsOrder for the given '$vehicle.Vin'
                    - OR joined with PartsOrder with matching PartsOrder for the given '$vehicle.Vin'
                        - AND with related Parts with '$Part.status' equals to "installed"
                - Compute '$vehicle.TotalPartsPrice' as the sum of ('$part.UnitPrice' * '$part.Quantity') for each part related to each parts order related to this vehicle.
                - Compute Derived Attribute '$vehicle.SalePrice' of each matching vehicle The Sale Price is calculated as 125% of the original purchase price '$vehicle.PurchasePrice' plus 110% of '$vehicle.TotalPartsPrice'
    - If '$user' is defined AND '$user.UserType' is in "Inventory Clerk"
        - Search queries will be executed on the database, which searches on unsold vehicles filtering them by the input fields.
        - To compute:
            - Select the Vehicles in
                - '$vehicle.SaleDate' is NULL
    - If '$user' is defined and '$user.UserType' is in (Manager, Owner)
        - Search queries will be executed on the database, which searches either for all, sold, or unsold vehicles, filtering them by the input fields.

        - To compute:
            - If *Search in Field* selected as
                - All
                    - Select All Vehicles
                - Sold
                    - Select Vehicles in '$vehicle.SaleDate' is NOT NULL
                - Unsold
                    - Select Vehicles in '$Vehicle.SaleDate' is NULL
    - Also filter Vehicles with entered input field values as follows for all users, including public
        - '$vehicle.VehicleType' matches the v*ehicle type* input value
        - '$vehicle.Manufacturer' matches the *manufacturer* input value
        - '$vehicle.Model' matches the *model* input value
        - '$vehicle.Year' matches the *year* input value
        - '$vehicle.FuelType' matches the *fuel type* input value
        - '$vehicle.Color' includes the *color* input value
        - '$vehicle.Manufacturer' OR '$vehicle.Model" OR '$vehicle.Year' OR '$vehicle.Description' matches the entered *keyword* input value either entirely or as a substring, case insensitive
    - If '$user.UserType' is defined, also filter vehicles with '$Vehicle.Vin' matches *the VIN* input value. VIN search is case insensitive.
- If the search query does not return any vehicles, return an **error** message
    - **"Sorry, it looks like we don't have that in stock!"**
- Else, the search results will be displayed in a list which every item with the following data:
    - VIN (sort on VIN) (if search is on VIN, it is case insensitive)
    - Vehicle Type
    - Manufacturer
    - Model
    - Year
    - Fuel Type
    - Colors (In the same line)
    - Horsepower
    - Sale Price
    - Display ***See Details*** button
        - When user clicks on ***See Details*** button it will take the user to **Vehicle Detail** page