# Introduction (what is this document about)

This document presents a structured approach to the logical design of a software program intended to address the requirements outlined in the given scenario. It demonstrates the systematic steps involved in translating requirements into an organised framework and supported by a visual representation of the process flow with some additional documentation to outline the recognised principles and techniques necessary to ensure clarity, precision, and coherence.

## The Stakeholders

In this scenario, the stakeholders we can identify are: the dealership management, the dealership sales team, the customers, and the developers/admin team.

# PDL/Pseudocode

# File and Library Imports

* Import necessary libraries as needed (System, Collection, IO, LINQ)

## Program Overview

* This program reads a stock list of cars from a CSV file, parses them into Car objects, and then processes the data to find:
  + The cheapest car available
  + The average price of cars
  + The average mileage of cars
  + The total number of cars for sale
  + The total stock value of all cars combined

## Car Class

* Car Class definition
* Holds the data for each car entry from the CSV
* Car registration number
* Car manufacturer (e.g Toyota, Ford)
* Car model (e.g Corolla, Fiesta)
* Mileage in kilometers
* Price in Euros

## CalculateAverage Method

* Calculates the average value from a list of cars, based on the property selected (e.g., Mileage, Price). Average = Sum of Mileage/Price for all cars / Total number of cars
* Uses a lambda function (selector) to tell which property should be averaged. For example: car => car.Price means "take the Price property of each car".
* If there are no cars, avoid errors and just return 0
* Select() uses the selector (lambda) to extract the chosen property from each car
* Convert.ToDouble ensures all numbers are turned into doubles for greater precision
* Average() is a LINQ method that computes the mean of the sequence

## GetCheapestCar Method

* Finds the cheapest car in the list.
* Uses the Aggregate() LINQ method to compare cars one by one and keep the one with the lowest price.
* Aggregate runs through the list and keeps the car with the lowest Price
* Explanation of the lambda:
  + (minCar, nextCar) => nextCar.Price < minCar.Price ? nextCar : minCar
  + If nextCar is cheaper, keep it
  + Otherwise, keep minCar

## GetStockSummary Method

* Provides a summary of total cars and their total price value.
* Uses a tuple: (int totalCars, int totalValue), which allows returning more than one value without a custom class.
* Total number of cars
* LINQ Sum() adds up all prices
* Return tuple

## Main Method and Program Flow

* Main entry point where the program starts execution
* Debug mode flag to print extra info
* List to store Car objects
* Path to the input CSV file (ensure backslashes in file paths are escaped in C#)
* Because first line is a header row, we skip it
* Loop through every line in the CSV file
* Skip the header row
* Split() CSV line by commas to get fields
* Expecting exactly 5 fields: Registration, Make, Model, Mileage, Price
* Convert the line of text into a Car object
* Add all new cars to the list
* Calculate statistics using helper methods
  + Calculate the average Price
  + Calculate the average Mileage
  + Get the cheapest car
  + Get stock summary method calculates
    - the total amount of cars
    - the total value of all cars in stock
* Debug outputs:
  + Output total cars read into the list
  + Output if the data has been read correctly
  + Loop through list and print each car's details
* try/catch
  + Save statistics to an output file
  + If unable to write to file (due to IO exception) output error to the console

# Diagrams

## Use cases

|  |  |
| --- | --- |
| Use Case | Generate Car Stock Statistics |
| Actor | User or automation runs the program |
| Objective | Read a CSV file containing car data  Calculate statistics  Output results to a text file |
| Pre-condition | A CSV file in the expected format, with car stock data exists and is readable |
| Main Flow | Program runs  Program reads the CSV file line by line.  Header row is skipped.  For each subsequent line:  Split the line by commas.  Validate that it contains the correct number of fields.  Convert text values to appropriate data types (int, double).  Create a Car object and add it to the list.  Once all cars are loaded, the program calculates:   The cheapest car.  Average price of cars.  Average mileage of cars.  Total number of cars.  Total stock value. |
| Alternative Flow | File not found: Program outputs error message, terminates.  No cars in file: Program outputs "0 cars found". |
| Post Condition | Text file is generated and stored in the current direction.  Program terminates |

## Flow Diagram

