# C# Basic Web: Parking System

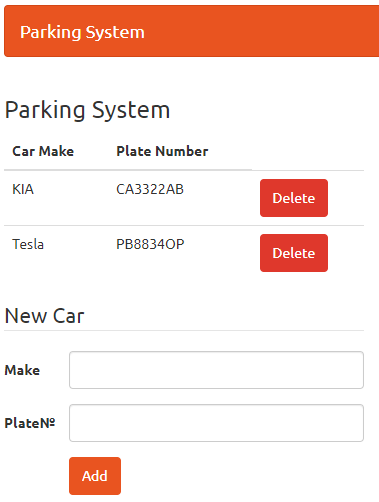
Problems for exercise for the ["C# Fundamentals" course @ SoftUni](https://softuni.bg/trainings/3729/programming-fundamentals-with-csharp-may-2022)

## Problem

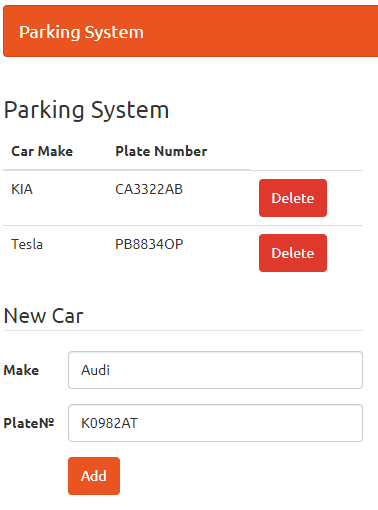
You have been tasked to create a simple **Parking System** [**MVC**](https://dotnet.microsoft.com/apps/aspnet/mvc) application. The application should hold **Cars** inside **memory storage**.

The functionality of the application should support:

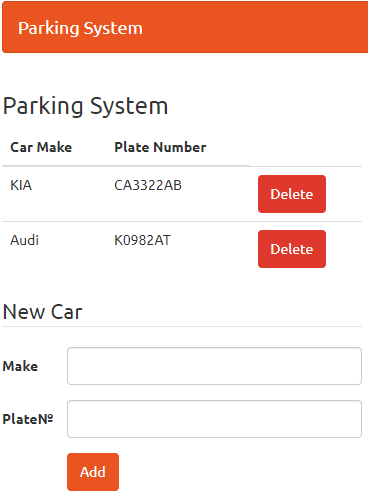
* **Listing Cars**



* **Add a Car**



* **Delete a Car**

****

## Overview

### Data Model

The Car entity holds **2 properties**:

* CarMake – non-empty string
* PlateNumber – non-empty string

### Project Skeletons

You will be given the application's skeletons, which holds about **90%** of the logic. You'll also be given **files**. The files have **partially implemented logic** and you need to write some code for the application to **function properly**.

The application's views will be given to you fully implemented. You only need to include them in your business logic.

Everything that has been given to you inside the skeleton is **correctly implemented** and if you write your code **correctly**, the application should work just fine. You are free to change anything in the Skeleton on your account.

## Preparation

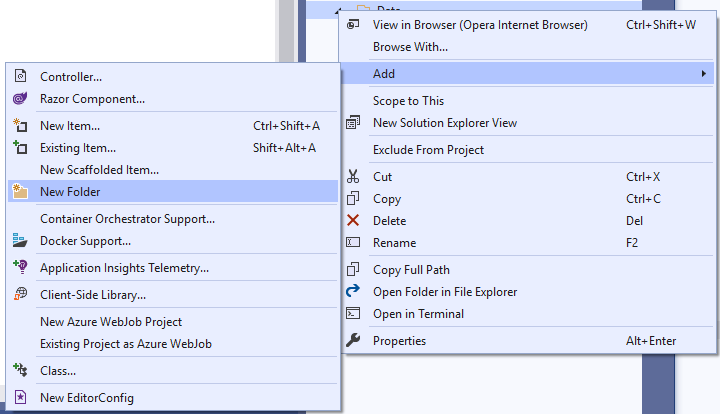
Extract the skeleton and open the **ParkingSystem.sln** file.

## Car Model

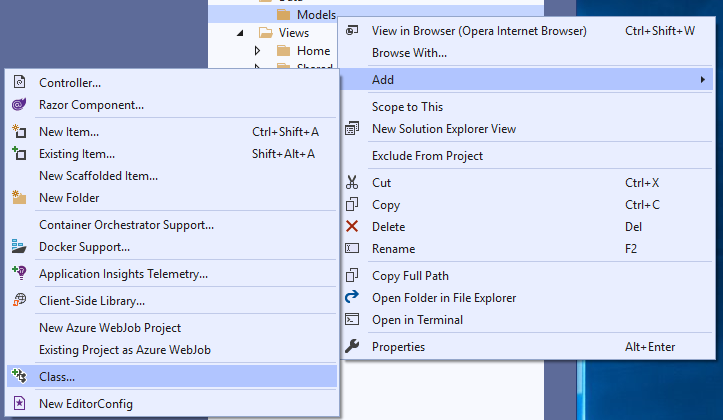
Now it's time to create the **Car entity class** which will have **2 properties**:

* **CarMake** – the **name** of the car, stored as a string.
* **PlateNumber** – the **plate** **number** of the car, stored as a string.

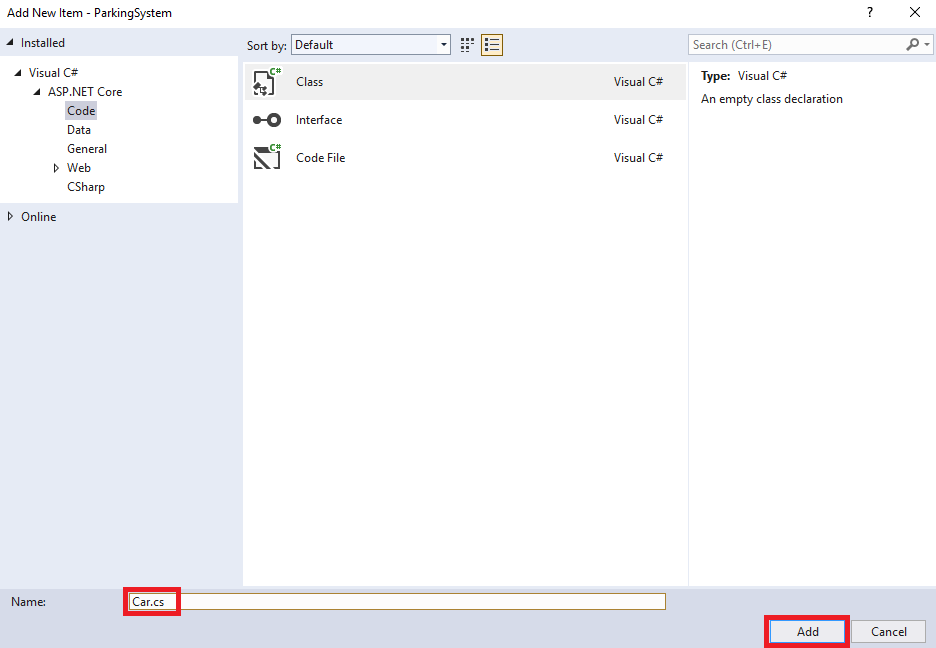
Let's go in our **Data** folder and **add a folder named "Models"**:



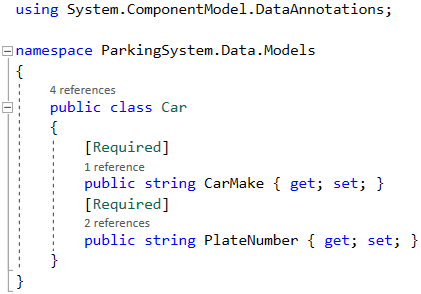
Let's go in our **Models** folder and **add a Car class**:



In the menu, which popped up, select Classand name it Car:



All that's left is to **add the properties** into our new file:

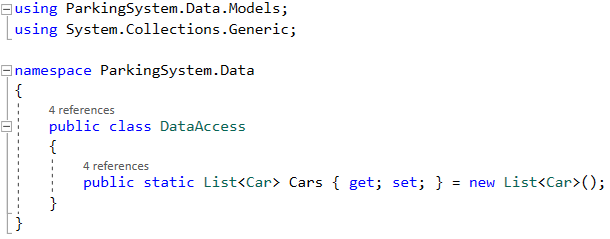


We're using the [Required] **attribute** on our **CarMake** and **PlateNumber properties,** because we don't want to have **a car without a make or a plate number**.

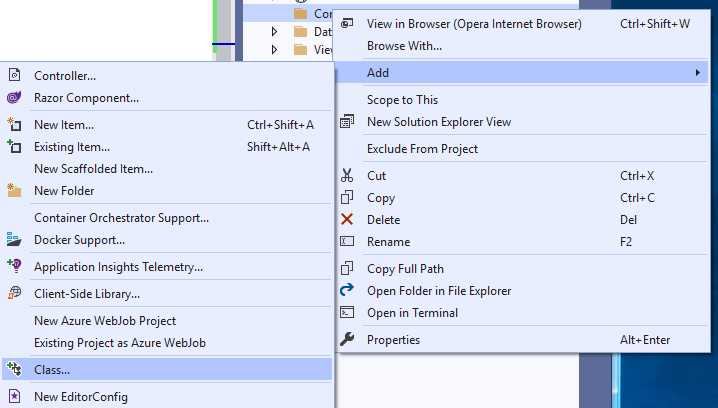
## Create Data Context

Now it's time to create the **data context**, which **keeps all the data** about the contacts. Right-click on the **Data** folder and create a new **class** **DataAccess.**

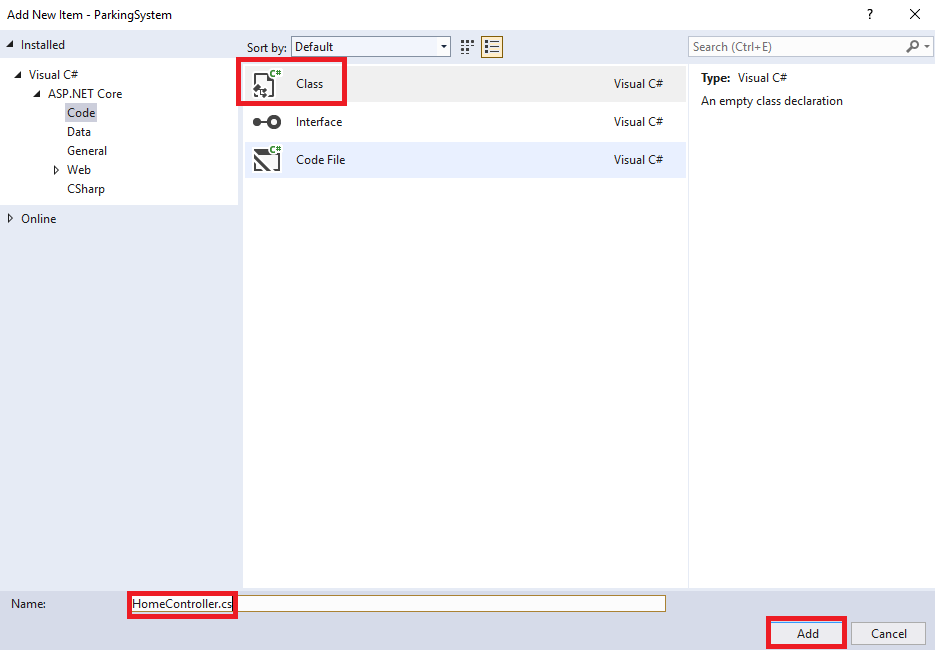
Now, we have to write the list, which will keep all the data about all the contacts:



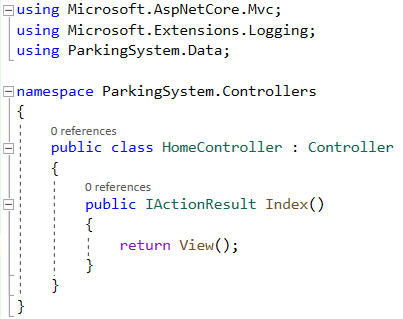
## Create Contact Controller

Now it's time to create a controller, which **displays** **cars**. Right-click the **Controllers** folder and click on Add 🡺 Class:  


In the name section type **"HomeController"**. It is important to use the same naming.



You should delete all the code that was autogenerated into the **"HomeController"** and write:



For more information on what the controller does, you can read the [documentation](https://docs.microsoft.com/en-us/aspnet/core/mvc/controllers/actions?WT.mc_id=dotnet-35129-website&view=aspnetcore-5.0).

That way every added car will be displayed on the home page.

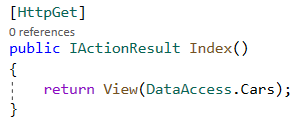
Next, we create another controller "**CarController**", which will hold the logic for adding cars. We should add two methods (actions):

* **AddCar(Car car)**: this method will accept a car and will add it to the collection.
* **DeleteCar(string plateNumber)**: this method will accept plate number, find a car with the given plate number and delete it.

The next step is to implement **AddCar and DeleteCar actions.**

### Write Logic for Listing a Cars

Let's go into the Controllers/HomeController.cs file and retrieve all the **cars,** and give them to the **index** view:



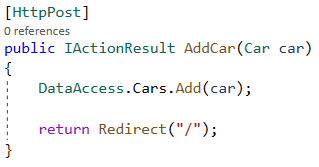
We retrieve all the cars, which are in the **RAM,** and we can render them in view.

### Write Logic for Adding Cars

Let's take action for **adding a** car. This action will have a Caras a parameter, letting ASP.NET automatically fill in the properties of the car, before inserting it into the memory storage:

We're using the **[HttpPost]** [attribute](https://docs.microsoft.com/en-us/aspnet/core/mvc/controllers/routing?view=aspnetcore-5.0), because we're **sending** data to the server, not retrieving it.

Let's **add the task to the memory storage**:



What we are doing here is calling the **DataAccess** class and adding the new **car** to the **Cars** list. This way every time someone inserts **car information** it will go in the list.

At last, we need to **redirect the user** to the Index() action in the HomeController:

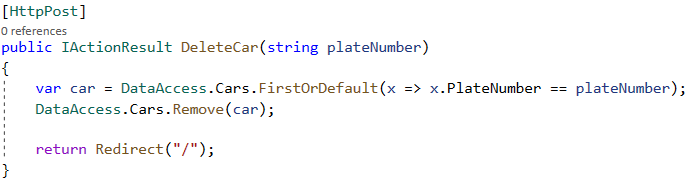
Now if our user wants to **add a car**, all they have to do is send a **POST request** to "/AddCar" with their **contact name and number**. Alternatively, they could just use the **HTML form that** we'll create in a few minutes.

### Write Logic for Delete a Cars

Let's take action for **deleting a** car. This action will have a plateNumbersas a parameter, letting ASP.NET automatically fill in the property, before inserting it into the memory storage.

We're using the **[HttpPost]** attribute because we're **sending** data to the server.

Let's **delete a car from the memory storage**:



What we are doing here is calling the **DataAccess** class and searching for a **Cars** list by the given plate number. Then we remove the car from the **DataAccess.**

At last, we need to **redirect the user** to the Index() action in the HomeController:

Now if our user wants to **delete a car**, all they have is to click on the delete button, which will send a **POST request** to "/DeleteCar".

**NOTE: If you do not implement the entire functionality, the skeleton will not compile.**

With that, we finished our **C# ParkingSystem**. Feel free to **build on your project even further**.

For example, you can add **validation** to the plate number and number using **Regular Expressions** (which you have learned about in the previous lecture) so that you know that the information given is always valid.

Also, you can add another field to the **Car** model, which will indicate what time the car was added. You can autogenerate that field or improve on the "**Index**" view by adding another input field.