Car Rental Project

# Requirements

## Stories:

* As a Manager I want to Add, Modify and Delete Car Rental Branches
* As a Manager I want to define the types of cars that will be used for rental.
* As a Manager I want to manage (Add and Delete) cars in the inventory.
* As a Manager I want to establish where the cars are.
* As a Manager I’d like to get a report of the current and past Reservations.
* As a Manager I’d like to Mange (add, modify, delete) Employees.
* As an Employee of a Branch I’d like to receive cars that are returned.
* As a User I’d like to be able to sign up to become a Member.
* As a User I’d like to Sign In as a Member.
* As a User I’d like to make searches on cars.
* As a User I’d like to I’d like to get a quote for a specific car.
* As a Member I’d like to make a reservation.
* As a Member I’d like to get a report of the history of my reservations.

## Technical Requirements:

* The system should run on the web.
* The following technologies will be used: MVC, JavaScript, JQuery, HTML, Entity Framework.
* The system should try to follow the guidelines of Separation of Concern.
* The system should follow the SOLID principles.

## Classes:

* Manager
* Employee
* Member
* User
* Car Rental Branch
* CarBrand
* CarModel
* CarType
* Car
* Reservation
* Reservation Report (not done)

### ViewModel Classes

* EditCarViewModel and others

## Class Employee: Member

Properties: Shift, Salary

## Class Manager: Employee ?

## Class Member

Properties: SSN\*, UserName\*, HashedPassword\*, Name\*, LastName\*, Gender\*, Email\*, DOB, Address, Phone, Picture, FromDate, Discount

## Class Branch

Properties: BranchId, Name, Location, Coordinates?, Phone

## Class CarBrand

Properties: CarBrandId, BrandName

Class CarModel

Properties: CarModelId, CarBrandId, ModelName

## Class CarType

Properties: CarTypeID, Model, DailyPrice, DailyLatePrice, Gear

## Class Car:

Properties: CarId, CarType, Mileage, Picture, Available, Plates, Branch, Rented

## Class Reservation:

Properties: ReservationId, Branch, FromDate, ToDate, ActualReturnDate, User, Car, Status

## ViewModels:

By creating ViewModels, we simplified the access to lists of CarBrands, CarTypes and Branches, which we used as DropDowns in Edit and Create pages.

## ViewModelFactory:

We used this class to generate all the ViewModel classes.

## Class EditCarViewModel:

Properties: CarData, Branches, CarTypes

## Class EditCarTypeViewModel:

Properties: CarTypeData, CarModels

# Process

I defined the Models with the classes described above.

I used Entity Framework to generate the database.

I added some Data Validation Annotations for format, data type and some restrictions.

I started a Migration and used (by calling Update-Database as I made changes to the DB).

I used the scaffolding feature of Visual Studio.

I used the MVVM pattern. Created a ViewModels folder where I created the non-mapped classes.

I modified the communication between Controller and Views. Instead of ViewBag, I passed on the ViewModel as a parameter where possible.

I added Custom Validations for things such as testing the unicity of a Car Brand or a Car Model.

# Functionality

The Web App allows the user to populate a database with information about:

* Cities
* Branches in Cities
* Car Brands
* Car Models within a Car Brand
* Car Types within Car Types where a price classification is done based on Codes: A, B, C, D, etc. We assume that a customer will not be picky about whether he gets a Toyota Camry or a Honda Accord, as long as the “type” of vehicle justifies the price.
* Car Inventory, where each car is assigned to a specific branch and is of a specific type with some additional information, such as color and current kilometrage. Each car has a specific status: Available, Rented, Under Repair or Other. The status can be changed just as any of the other fields can be changed.
* Users can register. Their information is stored and their reservations are assigned to them.
* Reservations can be made only by registered users. Only Available cars are shown when searching to make a reservation. Users can filter the search by Price Code.
* Employees can deliver cars that have been reserved.
* Employees can receive cars returned by a customer and enter the new kilometrage.

# Security - Authorizations

I set the communication to be through https.

Basically, we have two roles: Admin and Employee. Aside from this we have registered and non-registered users.

Non-registered users can search for cars, but they cannot make a reservation unless they are registered.

Registered users can make reservations and have access to most of the Lists: Branches, Car Types and prices, etc. But they don’t have access to the inventory of cars or the Pickup & Return menu.

There are 3 predefined Users with the Employee Role: JBEmployee1@gmail.com, JBEmployee2@gmail.com and JBEmployee3@gmail.com (not much imagination there) all of them have as password: “Passw0rd!” (A zero instead of letter O. Ignore quotes).

There is one predefined User with rights as an Administrator (Admin): [JBAdmin@gmail.com](mailto:JBAdmin@gmail.com) and the password is the same: “Passw0rd!”

Employees can access Pickup & Returns, where they deliver the cars that have been reserved and receive the cars that are returned. They can also see the inventory of cars.

## Facebook:

I have added a way to login using Facebook, but this is linked to the domain. If the domain is <https://localhost:44300>, this button can be used to login. This is the protection Facebook -and many other services- use: they link the service to the domain, a key and a secret code that is then embedded in the application.

Google and other services could easily be added, the code is already there, so programming-wise this comes up for free.

Done:

* Logo
* Understand DB naming
* Search for reservations
* List of all CarTypes of a specific CarModel, All CarModels of a specific CarBrand, All Cars from a specific CarBrand or CarBrand-CarModel.
* Relational to compound key
* Currency Format when viewing
* Eliminate the debug db.Database.Log = s => System.Diagnostics.Debug.WriteLine(s);

Won’t be done:

* Make Cars the central point where cars are added and include there buttons to modify related tables: CarType, CarBrand, CarModel.

Todo:

* Membership
* DatabaseInitializer: restructure it based on the new DB structure and Call it in “Main”
* Define if we have a startup DB
* WCF to define Cars
* Publish to Azure
* Bind only the fields that need to be bound
* DateTimePicker
* Create a table called PastReservations and moved there the reservations that have been closed.
* Allow selection of All Cities in Reservation
* Add an Actual Date a car was picked up
* Rename the Make a Reservation-related classes
* Allow cancelling a Reservation
* Change Deliver It for Picked Up and then Returned
* Validate Reservation Dates
* Deleting is set to cascading. I need to change that.